Final Report for AOARD Grant FA2386-11-1-4112

"Second Generation of Mass estimation"

Date: September 1, 2013

Name of Principal Investigators: Kai Ming Ting

E-mail: kaiming.ting@monash.edu

Institution: Monash University

Mailing Address: Gippsland School of Information Technology, Monash University, Victoria 3842, Australia

Phone: +61 3 990 26241

Fax: +61 3 99026879

Period of Performance: 01/September/2011 - 31/August/2013

Abstract

We made three progresses in the field through mass estimation: First, we propose the first adaptive version of mass estimation using a new nearest neighbor procedure which runs significantly faster than existing nearest neighbor procedures, and it needs no indexing schemes. Second, we propose the first mass-based Bayesian classifier which estimates the likelihood directly in multi-dimensional space; unlike existing Bayesian classifiers which estimate simplified surrogates of likelihood (e.g., one-dimensional likelihood). Third, we have created the first mass-based similarity measure and show that it is an effective alternative to distance-based similarity measure in content-based information retrieval problems.

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **10 SEP 2013**	2. REPORT TYPE **Final**	3. DATES COVERED **15-09-2011 to 14-09-2013**

4. TITLE AND SUBTITLE
Second Generation of Mass Estimation

5a. CONTRACT NUMBER
FA2386-11-1-4112

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Kai Ming Ting

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Gippsland School of Information Technology, ,Monash University,Victoria,Australia,AU,3842

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AOARD, UNIT 45002, APO, AP, 96338-5002

10. SPONSOR/MONITOR’S ACRONYM(S)
AOARD

11. SPONSOR/MONITOR’S REPORT NUMBER(S)
AOARD-114112

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Three progresses were made in the field through mass estimation: 1) the first adaptive version of mass estimation using a new nearest neighbor procedure which runs significantly faster than existing nearest neighbor procedures and needs no indexing schemes, 2) the first mass-based Bayesian classifier which estimates the likelihood directly in multi-dimensional space; unlike existing Bayesian classifiers which estimate simplified surrogates of likelihood (e.g., one-dimensional likelihood), and 3) the first mass-based similarity measure which can be an effective alternative to distance-based similarity measure.

15. SUBJECT TERMS
Information diffusion, Opinion formation, Social network, Burst detection, Influential nodes, Network dynamics, Knowledge discovery from network

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**173**	

In addition, we have extended the two previous works on mass estimation and published them in Machine Learning Journal and Journal of Knowledge and Information Systems.

1 Introduction

The project deepens the research achieved in the pioneering mass estimation project (supported by AOARD Grant FA2386-10-1-4052) to yield the next generation of mass estimation and to enable mass estimation to be applied more widely to data mining tasks. The project has been refined to achieve the following specific aims:

1. **Create the second generation of mass estimation**
2. **Develop a new Bayesian Classifier using mass estimation**
3. **Propose the first mass-based similarity measure**

In the previous project, mass estimation has been established to be a new paradigm in data mining. Like density estimation in the existing paradigm, mass estimation has influence in many aspects of data mining. We have previously shown that mass-based approaches work more effectively and efficiently than density-based approaches in four data mining tasks : information retrieval, regression, anomaly detection and clustering.

The second generation of mass estimation consolidates mass estimation's fundamental role in data mining in terms of algorithmic design, and demonstrate that the new paradigm based on mass is applicable and effective in areas of data mining as widely as the existing density paradigm has applied now.

This project broadens its application to an additional data mining task: classification in the Bayesian framework. The resultant mass-based Bayesian classifiers can deal with very large data sets which are infeasible with existing Bayesian classifiers.

The unexpected result of this project is the creation of the mass-based similarity measure. The new measure is unique because it does not compute distance and it is a non-metric measure. It could potentially enhance similarity modelling envisaged by psychologists.

The ultimate goal of the work is to produce a complete theory of mass estimation that enables mining of big data in all data mining tasks. This project has contributed to the generalisation of the initial formulation of mass estimation and has affirmed its fundamental data modelling mechanism which has generic applicability to many areas of data mining tasks.

We provide the theoretical analyses in Section 2, the results and discussion in Section 3, and the final remark in Section 4.

2 Theoretical Analyses

This section provides the description of the theoretical analyses for the abovementioned three aims in the following three subsections.

2.1 Create the second generation of mass estimation

We establish the properties of a good mass estimation method in guiding us to create the second generation of mass estimation. The first generation of mass estimation is based on a tree structure to partition the feature space into local regions. The work is guided by answering the following questions:

- What are the characteristics of local regions necessary for good mass estimation?
- Are pair-wise calculations, as in the case of pair-wise distance calculations in existing approaches, required to achieve good task-specific performance?
- What is the alternative to the tree-based approach for mass estimation?

We explored different ways to partition the space and identify the characteristics of the local regions required for a specific task. We also created the first non-tree-based approach for mass estimation.

As most existing algorithms are density based, we have designed our work to evaluate the efficacy of mass estimation by first building new density estimators based on mass, and then replacing the density estimators in existing algorithms with the new density estimators. In each case, the same algorithm is used; only the core modelling mechanism used is different, i.e., either existing density estimator or the new density estimator based on mass. We focus our investigation in two existing algorithms in two separate tasks, i.e., LOF in anomaly detection and DBSCAN in clustering.

2.2 Develop a new Bayesian Classifier using mass estimation

We have proposed the first mass-based classifier, a new Bayesian classifier called **Mass-Bayes** which has constant training time complexity and constant space complexity. Unlike existing Bayesian classifiers which must make some assumption that allows them to estimate simplified surrogates of likelihood $p(\mathbf{x}|y)$, the new Bayesian classifier estimates the likelihood directly without any assumptions. This is made possible by the use of mass estimation to estimate the likelihood directly in multi-dimensional space. It aggregates the multi-dimensional likelihoods estimated from random subsets of the training data using varying size random feature subsets.

The symbols used in the following description are provided in Table 1.

In Bayesian classifier learning, estimating the joint probability distribution $p(\mathbf{x}, y)$ or the likelihood $p(\mathbf{x}|y)$ directly from training data is considered to be difficult, especially in large multi-dimensional data sets. In order to circumvent this difficulty, existing Bayesian classifiers such as Naive Bayes, BayesNet and AηDE have focused on estimating simplified surrogates of $p(\mathbf{x}, y)$ from different forms of one-dimensional likelihoods.

Naive Bayes (NB) is the simplest generative approach that estimates $p(\mathbf{x}, y)$ by assuming that the attributes are statistically independent given y:

$$\hat{p}(\mathbf{x}, y)_{NB} = p(y) \prod_{i=1}^{d} p(x_i|y) \tag{1}$$

BayesNet learns a network of probabilistic relationship among the attributes including the class attribute and learned the conditional probabilities from the training data. The

Table 1: List of symbols used in Sections 2.2 and 3.2.

D	Set of training instances
n	Number of training instances in D
d	Number of attributes
c	Number of classes
$\lvert \cdot \rvert$	Cardinality of a set
$p(\cdot)$	Probability
$\mathbf{x}$	d-dimensional vector representing a data instance
y	Class label of a data instance
x_i	The value of the i^{th} attribute of the instance $\mathbf{x}$
π_i	The set of parents of x_i in a Bayesian network
π_y	The set of parents of y in a Bayesian network
η	Number of privileged attributes in AηDE
v	Average number of discrete values for an attribute
S^{η}	The collection of all subsets of size η of the set of d attributes
$\mathbf{s}$	A subset of attributes of size η
$\mathbf{x_a}$	$\lvert\mathbf{a}\rvert$-dimensional vector of values of $\mathbf{x}$ defined by $\mathbf{a} \subset \{1, \cdots, d\}$
$\binom{d}{\eta}$	A binomial coefficient of η out of d
t	Number of trees in MassBayes
ψ	Sub-sample size
G_t	A collect of t subsets of varying sizes of d attributes.
$\mathbf{g}$	A random subset of attributes
$\mathcal{D}$	A subset of D of size ψ
$T(\cdot)$	A function that divides the feature space into non-overlapping regions
$T(\mathbf{x})$	The region where $\mathbf{x}$ falls into
$\mathcal{D}_y$	The set of instances belonging to class y in $\mathcal{D}$
$\mathcal{D}_{y,\mathbf{x_g}}$	The set of instances belonging to class y and having values $\mathbf{x_g}$ in $\mathcal{D}$

joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y)_{BayesNet} = p(y|\pi_y) \prod_{i=1}^{d} p(x_i|\pi_i) \tag{2}$$

In another simplification of BayesNet, AηDE relaxes the independence assumption by allowing dependency between y and a fixed number of privileged attributes or super-parents. The other attributes are assumed to be independent given the η super-parents and y. AηDE avoids the expensive searching in learning probabilistic dependencies by constructing an ensemble of η-dependence estimators. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y)_{A\eta DE} = \sum_{s \in S^\eta} p(\mathbf{x_s}, y) \prod_{j \in \{1,2,\cdots,d\} \backslash \mathbf{s}} p(x_j | \mathbf{x_s}, y) \tag{3}$$

where S^η is the collection of all subsets of size η of the set of d attributes $\{1, 2, \cdots, d\}$; and $\mathbf{x_s}$ is a η-dimensional vector of values of $\mathbf{x}$ defined by $\mathbf{s}$.

It has been shown that A1DE and A2DE produce better predictive accuracy than the other state-of-the-art generative classifiers. However, it only allows dependencies on a fixed number of attributes and y. Because of the high time complexity of $O\left(n\binom{d}{\eta+1}\right)$ and space complexity of $O\left(c\binom{d}{\eta+1}v^{\eta+1}\right)$, where v is the average number of values for an attribute, only A2DE or A3DE is feasible even for a moderate number of attributes. Furthermore, selecting an appropriate value of n for a particular data set requires a search.

AηDE and many other implementations of BayesNet require all the attributes to be discrete. The continuous-valued attributes must be discretised using a discretisation method before building a classifier.

Rather than aggregating an ensemble of η-dependence single-dimensional likelihood estimators, we propose to aggregate an ensemble of t multi-dimensional likelihood estimators where each likelihood is estimated using different random subsets of d attributes from data. The likelihood $p(\mathbf{x}|y)$ is estimated as:

$$\hat{p}(\mathbf{x}|y) = \frac{1}{t} \sum_{\mathbf{g} \in G_t} p(\mathbf{x_g}|y) \tag{4}$$

where G_t is a collection of t subsets of varying sizes of d attributes; and $\mathbf{x_g}$ is a $|\mathbf{g}|$-dimensional vector of values of $\mathbf{x}$ defined by $\mathbf{g}$; and $1 \leq |\mathbf{g}| \leq d$.

Each $p(\mathbf{x_g}|y)$ is estimated using a random subset of training instances $\mathcal{D} \subset D$, where $|\mathcal{D}| = \psi < n$.

$$\hat{p}(\mathbf{x}_g|y) = \frac{|\mathcal{D}_{y,\mathbf{x_g}}|}{|\mathcal{D}_y|} \tag{5}$$

where $|\mathcal{D}_{y,\mathbf{x_g}}|$ is the number of instances having attribute values $\mathbf{x_g}$ belonging to class y in $\mathcal{D}$ and $|\mathcal{D}_y|$ is the number of instances belonging to class y in $\mathcal{D}$.

Rather than relying on a specific discretisation method in the preprocessing step, we propose to build a model directly from data, akin to an adaptive multi-dimensional histogram, to determine $\mathbf{x_g}$ which adapts to the local data distribution. Let $T(\cdot)$ be a function that divides the feature space into non-overlapping regions and $T(\mathbf{x})$ be the region where $\mathbf{x}$ falls. In a multi-dimensional space, each instance in $\mathcal{D}$ can be isolated by splitting only on few dimensions i.e., only a subset of d attributes ($\mathbf{g} \subset \{1, 2, \cdots, d\}$) is used to define $T(\mathbf{x})$. Hence, $|\mathcal{D}_{y,\mathbf{x_g}}|$ is the number of instances belonging to class y in the region $T(\mathbf{x})$. Let $p(T(\mathbf{x})|y)$ be the probability of region $T(\mathbf{x})$ when only class y instances in $\mathcal{D}$ are considered.

$$p(T(\mathbf{x})|y) = \hat{p}(\mathbf{x_g}|y) = \frac{|\mathcal{D}_{y,\mathbf{x_g}}|}{|\mathcal{D}_y|} \tag{6}$$

The new generative classifier, called *MassBayes*, estimates the joint distribution as:

$$\hat{p}(\mathbf{x}, y)_{MassBayes} = p(y)\ \frac{1}{t} \sum_{\mathbf{g} \in G_t} p(\mathbf{x_g}|y) = p(y)\ \frac{1}{t} \sum_{i=1}^{t} p(T_i(\mathbf{x})|y) \tag{7}$$

The average probability of t different regions $T_i(\mathbf{x})$ ($i = 1, 2, \cdots, t$), constructed using $\mathcal{D}_i \subset D$, provides a good estimate for $p(\mathbf{x}|y)$ as it estimates the multi-dimensional likelihood by considering the distribution in different local neighbourhood of $\mathbf{x}$ in the data space. An illustrative example is provided in Figure 1.

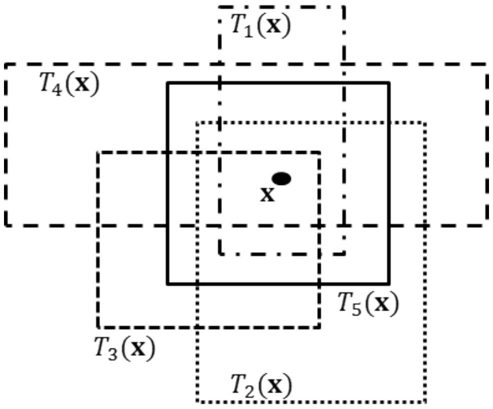


Figure 1: Different regions from different $T_i(\cdot)$ ($i = 1, 2, \cdots, 5$) that cover $\mathbf{x}$.

MassBayes has the following characteristics in comparison with AηDE:

1. In each estimator, AηDE estimates one-dimensional likelihood given a fixed number of super-parents and y, whereas MassBayes estimates multi-dimensional likelihood using varying number of attributes.

2. In AηDE, the ensemble size is fixed to $\binom{d}{\eta}$. But, MassBayes allows the flexibility for users to set the ensemble size.

3. AηDE requires continuous-valued attributes to be discretised before building the model. The performance of AηDE is affected by the discretisation technique used. In contrast, MassBayes builds models directly from data. It can be viewed as a dynamic multi-dimensional discretisation where the information loss is minimised by averaging over multiple models.

4. Each model in MassBayes is built with training subset of size $\psi < n$ which gives rise to the constant training time. In contrast, each model in AηDE is trained using the entire training set.

5. AηDE is a deterministic algorithm whereas MassBayes is a randomised algorithm.

6. Like AηDE, MassBayes is a generative classifier without search.

2.3 Propose the first mass-based similarity measure

Data mining algorithms have traditionally relied on similarity measures to gauge the similarity between two instances as the core operation to solve various data mining problems. For example, anomaly detection requires ranking of instances in a database according to their degrees of anomaly; an information retrieval task ranks instances in a database which are most similar to a query. These ranking tasks are traditionally accomplished by computing the distance between two instances as the key step to calculate the ranking.

The first mass-based similarity measure is motivated by our recent content-based multimedia information retrieval (CBMIR) system called ReFeat [8].

ReFeat is unique in two aspects. First, it uses a similarity measure which is primarily based on data distribution in the local region. In contrast, commonly used distance measures are solely based on the positions of instances in the feature space. Second, at the heart of ReFeat is an anomaly detector which provides a ranking score $\ell(\mathbf{x})$ for an instance $\mathbf{x}$, independently of other instances. This is fundamentally different from most ranking measures that rely on a distance measure to compute the distance of an instance relative to another instance, $dist(\mathbf{x}, \mathbf{y})$ (e.g., ORCA, Qsim).

The use of such a unique similar measure is the key reason why ReFeat has produced better retrieval performance than state-of-the-art CBMIR systems including manifold learning method MRBIR, Bayesian learning method BALAS, query-sensitive ranking methods such as InstRank and Qsim.

Despite its unique approach and demonstrably excellent retrieval performance, the ReFeat paper [8] does not provide a satisfactory explanation as to why a unary score function could produce an appropriate ranking of database instances for a query which requires a binary function. More to the point: ReFeat does not guarantee that two 'similar' instances, having a similar ranking score $\ell(\cdot)$, are in the same local neighbourhood.

We investigate the source of the power of ReFeat. Note that the anomaly detector used in ReFeat is a special case of mass-based approach, revealed in our journal paper on mass estimation [1].

From a foundation in mass estimation, we derive a new mass-based similarity measure that enables a new CMBIR system to significantly improve the retrieval performance of ReFeat.

3 Results and Discussion

This section provides the results and discussion for each of the three aims in the following three subsections.

3.1 Create the second generation of mass estimation

3.1.1 Properties of a good mass estimation method

The most successful method for mass estimation thus far is to aggregate mass from multiple local regions. There are different ways in which local regions can be constructed, using trees in our previous work [1, 2] and nearest neighbours in the current project [4]. They all have been shown to perform well in a number of different data mining tasks. The trees could be built using random splits, equal-volume splits or equal-data-size split; or the region could be grid-based or non-grid-based.

However, certain tasks demand some properties that need to be met. For example, in clustering tasks, the local regions must be close regions, usually defined using all dimensions. This is to avoid points, which are near in some dimensions but far in others, from linking into one cluster. On the other hand, in classification tasks, the local regions must be open to cover sufficiently large regions in order to provide a good estimation of probabilities in the Bayesian classification context. In anomaly detection tasks, either open or close local regions are found to work well.

The mass-based methods show that local regions are sufficient to determine the "neighbours" required to accomplish classification, clustering and anomaly detection tasks, without calculating pair-wise distance. This eliminates the need to compute pair-wise distance; and it is the key contributor to the significant speedup and less memory requirement in mass-based methods in comparison with existing methods which require distance calculations.

3.1.2 A new approach to nearest neighbour density estimator based on the second generation of mass estimation

The first mass-based approach we have developed under this project has advanced the mass-based approaches developed in the previous AOARD supported project in one key important aspect: mass estimation is adaptive to the local data distributions. Previous mass estimation approaches (published in IEEE ICDM-2010 and IEEE ICDM-2011) are unable to adapt to differing local data distributions in a single data set.

The new approach produces the first nearest neighbour procedure having $O(n)$ time complexity and constant space complexity. In contrast, existing nearest neighbor algorithms have $O(n^2)$ time complexity and $O(n)$ space complexity. It is the first nearest neighbour density estimator to have linear time complexity and constant space complexity, as far as we know.

Unlike existing algorithms which require some indexing scheme to speed up the nearest neighbor search, the new approach has no such requirement. Replacing the nearest neighbor procedure in existing algorithms with the new procedure, we have shown that this enables two nearest neighbour algorithms, otherwise infeasible, to scale up to data sets with millions of instances in anomaly detection and clustering tasks.

The new density estimator called **LiNearN**, for linear time nearest neighbour algorithm, has the following distinctive nearest neighbor features:

- By rejecting the premise that a nearest neighbour algorithm must find the nearest neighbour for every instance in the given data set, LiNearN finds the nearest neighbour for every instance in a subsample.
- LiNearN defines local neighbourhood using nearest neighbours in each of the many subsamples by building a region centered at each instance. This differs from the existing nearest neighbour density estimators where the local neighbourhoods are defined based on either k nearest neighbours or a fixed radius.

Our asymptotic analysis reveals that the new density estimator has a parameter which trades off between bias and variance, as in existing density estimators such as k-nearest neighbour density estimators.

We assess LiNearN in anomaly detection and clustering tasks and compare with three state-of-the-art nearest neighbour algorithms, ORCA, LOF and DBSCAN. LiNearN produces similar results compared with these algorithms in terms of task-specific performance measures, but it runs orders of magnitude faster than these algorithms in large data sets. The advantages of the new nearest neighbour approach shown in these two tasks imply that it can potentially be adopted, in place of existing nearest neighbour algorithms, to solve other data mining tasks.

The full details of the LiNearN results can be found in the attached paper [4].

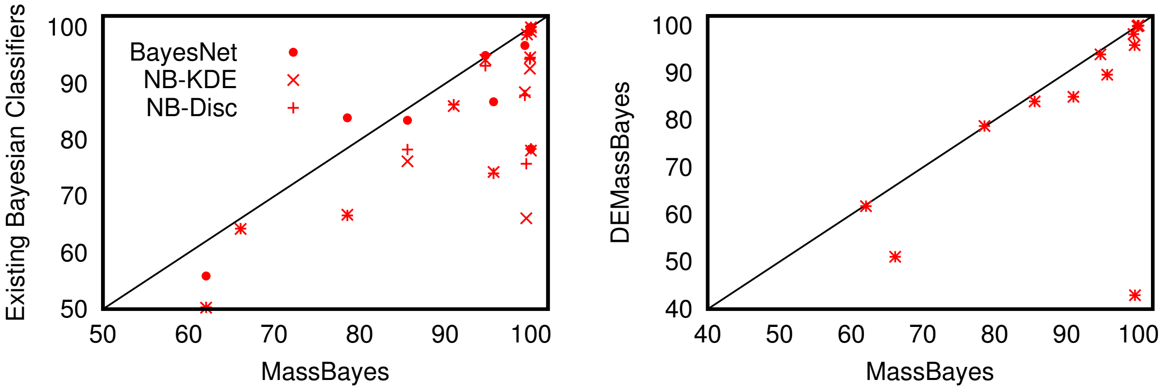
3.2 Develop a new Bayesian Classifier using mass estimation

The new Bayesian classifier based on mass called MassBayes has been shown to have better predictive accuracy than existing state-of-the-art Bayesian classifiers such as BayesNet and AηDE, especially in large data sets. It also scales better than these classifiers in large data sets because of its constant training time complexity. Unlike DEMassBayes [2], the mass-based Bayesian classifier does not make use of a density estimator to estimate the likelihood.

The average classification accuracies of MassBayes against the existing generative classifiers and DEMassBayes over a 10-fold cross-validation are plotted in Figure 2. A total of fifteen data sets were used in this experiment. The coordinate values of each point in the plot are the accuracies of each pair of classifiers in a data set. If both the classifiers had produced the same accuracies in a data set, the point representing that data set would lie on the diagonal. In both the plots, many points lie below the diagonal line and only a few points are above. This shows that MassBayes is better than the existing Bayesian classifiers.

The runtime of MassBayes was an order of magnitude faster than some contenders (such as A2DE and NB-KDE) in large data sets and it was competitive to many existing Bayesian classifiers in many data sets. The runtime of MassBayes and the existing Bayesian classifiers in the three largest data sets - KDDCup99, YearPrediction and CoverType was presented in Figure 3. It is interesting to note that BayesNet, A2DE and A3DE in the KDDCup99 data set and BayesNet and A3DE in YearPrediction did not complete the task because they required memory space more than 20GB. The memory requirement in BayesNet and AηDE increases with d and c. ENNBayes is a version of MassBayes implemented nearest neighbour; that is why it runs slower than MassBayes.

Note that the reported runtime results for AηDE, BayesNet and NB-Disc did not include



(a) MassBayes versus BayesNet and NB

(b) MassBayes versus DEMassBayes

Figure 2: Scatter plot of accuracies of MassBayes versus those of BayesNet, two variants of Naive Bayes (NB-KDE and NB-Disc) and DEMass-Bayes.

the discretisation time that must be done as a preprocessing step, which give AηDE, BayesNet and NB-Disc an unfair advantage over MassBayes. The discretisation cost is significantly large in large and moderately high dimensional data sets. For examples, the discretisation took 1290 seconds in the KDDCup99 data set, and 467 seconds in the YearPrediction data set. The discretisation time itself was of the same order of magnitude as the runtime of MassBayes.

Figures 4 and 5 show the increase in training time and space required to store the classification model of MassBayes and three variants of AηDE when the number of training instances (n) and the number of attributes (d) are increased by factors (5, 10, 50, 100, 500) and (2, 4, 8, 12 16) respectively. In MassBayes, both the runtime and memory requirement were constant when the training size was increased, whereas they varied sub-linearly when number of attributes were increased. In contrast, the runtime and memory requirement of AηDE increase exponentially as training size increases. With respect to the increase in the number of attributes, the runtime and memory requirement of AηDE also increase exponentially.

The details of the results are reported in the attached papers [3, 6].

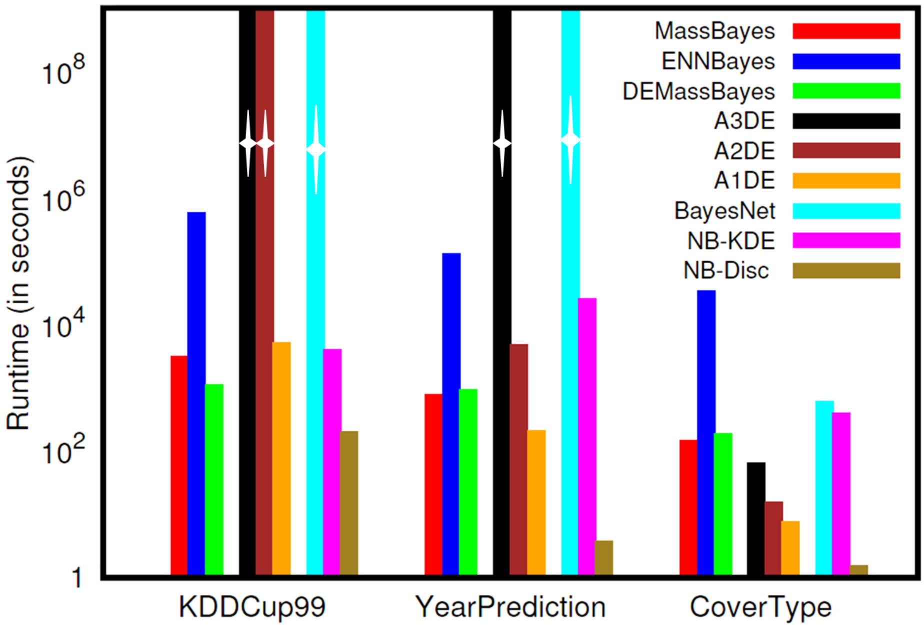
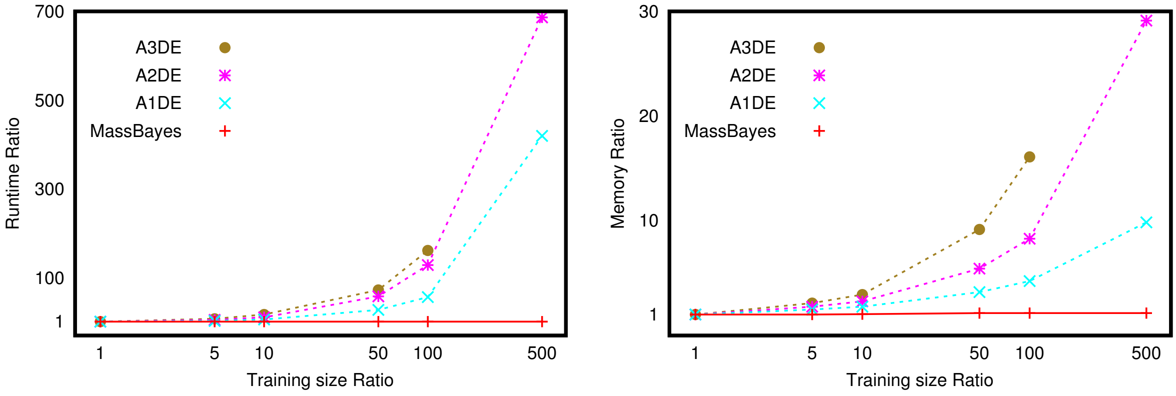


Figure 3: Runtime of MassBayes, ENNBayes and the other contenders in the three largest data sets: KDDCup99, CoverType and YearPrediction. The vertical axis is on a logarithmic scale of base 10. For ease of reading, the classifiers are organised into groups of three: the first group has three classifiers (MassBayes, ENNBayes and DEMassBayes) based on the proposed ensemble approach; the second group has three variants of AηDE (A3DE, A2DE and A1DE); and the last group has BayesNet, NB-KDE and NB-Disc. Note that the discretisation time was not included in the runtime of AηDE, BayesNet and NB-Disc. Histograms with star which have the maximum height indicate that the classifiers did not complete the tasks.

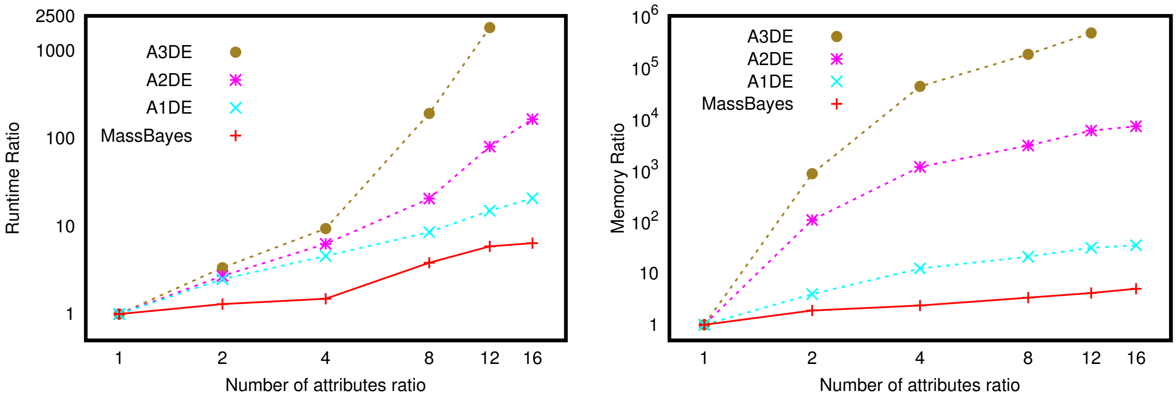
.



(a) Training Time

(b) Memory requirement

Figure 4: Increase in training time and memory requirement to learn a classification model with the increase in training size in a subset of KDDCup99 data set with three largest classes (i.e., $c = 3, d = 32$). The horizontal axes are on a logarithmic scale of base 10. A3DE did not complete when the training size was increased to five million. The base of the training size ratio is 10000 instances.



(a) Training Time

(b) Memory requirement

Figure 5: Increase in training time and memory requirement to learn a classification model with the increase in the number of attributes in a subset of KDDCup99 data set with three largest classes (i.e., $c = 3, n = 5125369$). The number of attributes is increased from 2 to 32. The horizontal and vertical axes in Figures (a) and (b) are on a logarithmic scale of base 2 and 10, respectively. A3DE did not complete when the number of attributes was increased to 32.

.

3.3 Propose the first mass-based similarity measure

This section describes the preliminary result of the first mas-based similarity measure. The key results at this point in time are summarised as follows:

- Introducing a unique similarity measure, **Massim**, and establishing its theoretical foundation based on mass.
- Creating a new CMBIR system called **MassIR** based on this similarity measure.
- Empirically evaluating MassIR in comparison with ReFeat and systems which employ commonly used similarity measures, and showing its superiority in image and music information retrievals.

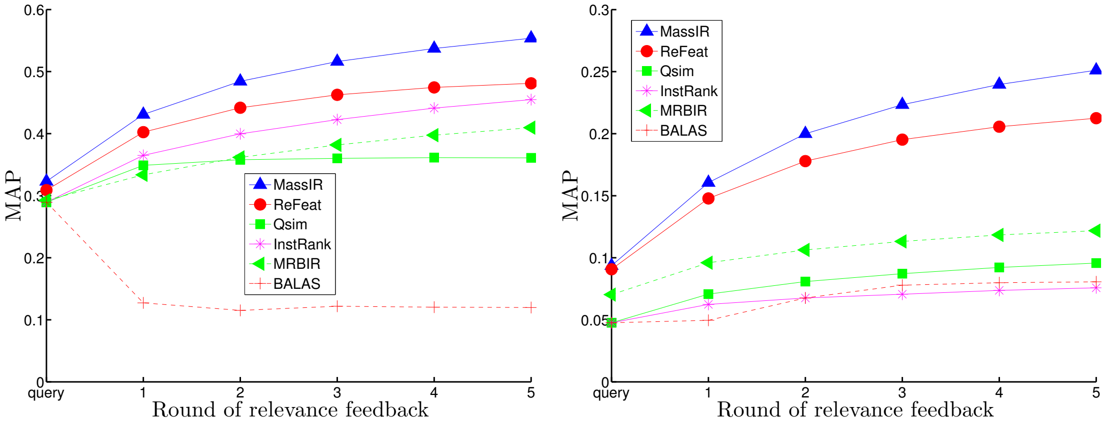
Massim has the following characteristics:

- Unlike the similarity measure used in ReFeat, Massim guarantees that two similar instances are in the same local neighbourhood.
- Unlike distance-based similarity measures, it does not compute distance and primarily based on data distribution in the local region.
- It is a generalisation of mass estimation. Under certain conditions, it reduces to mass estimation.

The key result of the evaluation is provided here, and the aim is to compare MassIR with state-of-the-art systems ReFeat, Qsim, InstRank, MRBIR and BALAS, all in the context of content-based information retrieval.

Two databases, which were previously used in other studies: GTZAN music database and COREL image database, are employed in this experiment.

Figure 6 shows the comparison of MassIR with ReFeat, Qsim, InstRank, MRBIR and BALAS. Results of Qsim, InstRank, MRBIR and BALAS were taken from [8]. The result shows that MassIR performs significantly better than all other existing methods. Also note that the performance gap increases as the number of feedback rounds increases, indicating that MassIR utilizes feedback instances more effectively than other methods.



(a) GTZAN music database

(b) COREL image database

Figure 6: Comparison of retrieval performance in terms of MAP(Mean Average Precision).

The full description of Massim and MassIR and the complete evaluation results are provided in the attached paper [5].

This preliminary result provides the foundation for further investigation in a new project to establish the theory and evidence that non-metric similarity measures can work better than commonly used similarity measures based on a metric in data mining tasks.

4 Final Remark

The successful creation of the second generation of mass estimation using nearest neighbour approach represents a milestone that will have significant influence in the future development of mass estimation. While developing the new Bayesian classifier using the first generation of mass estimation reported here, we had also spent a substantial amount of time to generalise the approach which incorporates the second generation of mass estimation. This has yielded ENNBayes reported in paper [6].

The work on the first mass-based similarity measure, though envisaged and described briefly in the initial research proposal of this project, was not one of the original aims of this project. However, I am glad that the progress in this project has enabled the research team to produce the preliminary findings of this new similarity measure. I have the opinion that the finding represents the beginning of the next phase of mass estimation research—the research team has shown that the new similarity measure is a generalisation of mass estimation. This is an exciting development.

The work on the generalisations of both mass-based Bayesian classifier approach and mass estimation mentioned above had unfortunately taken time away from the research initially planned for data streams and high dimensional issues. However, I believe that the results the research team has produced form a better foundation for researching these and other issues in the future.

5 List of Publications and Significant Collaborations that resulted from AOARD supported projects

5.1 List of peer-reviewed journal publications:

[1] Kai Ming Ting, Guang-Tong Zhou, Fei Tony Liu and Tan Swee Chuan (2013). Mass Estimation. *Machine Learning Journal.* Vol. 90, Issue. 1, pp. 127-60.

[2] Kai Ming Ting, Takashi Washio, Jonathan R. Wells, Fei Tony Liu and Sunil Aryal (2013). DEMass: A New Density Estimator for Big Data. *International Journal of Knowledge and Information Systems.* Vol. 35, Issue. 3, pp. 493-524

5.2 List of peer-reviewed conference publications

[3] Sunil Aryal and Kai Ming Ting (2013). MassBayes: A new generative classifier with multi-dimensional likelihood estimation. *In Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining.* Lecture Notes in Computer Science. Springer Berlin Heidelberg. pp. 136-148

5.3 Papers currently submitted for review

[4] Jonathan R. Wells, Kai Ming Ting and Takashi Washio. LiNearN: A New Approach to Nearest Neighbour Density Estimator. Submitted to *Pattern Recognition*

[5] Kai Ming Ting, Thilak Laksiri Fernando and Geoffrey I. Webb. Mass-based Similarity Measure: An Effective Alternative to Distance-based Similarity Measures. Submitted to *2013 IEEE International Conference on Data Mining*

[6] Sunil Aryal and Kai Ming Ting. An ensemble approach to estimate multi-dimensional likelihood in Bayesian classifier learning. Submitted to *Machine Learning*

5.4 Significant collaboration with industry and research institutions

Collaboration with two colleagues, Takashi Washio (Osaka University) and Geoff Webb (Monash University) have resulted three papers in mass estimation [2, 4, 5].

As a result of the work on mass estimation, the following collaborations have produced one industry grant and two publications:

A. Toyota InfoTechnology Centre (Japan) has provided a grant in 2012 to work on an application to a vehicle warning system and produced the following publication:

[7] Jonathan R. Wells, Kai Ming Ting and Naiwala P. Chandrasiri (2012). A non-time series approach to vehicle related time series problems. *In Proceedings of the Tenth Australasian Data Mining Conference.* Volume 134 in the Conferences in Research and Practice in Information Technology Series. Australian Computer Society.

B. Research collaboration with Shandong University (China) has produced paper [1] and the following publication:

[8] Guang-Tong Zhou, Kai Ming Ting, Fei Tony Liu and Yilong Yin (2012). Relevance Feature Mapping for Content-based Multimedia Information Retrieval. *Pattern Recognition.* Vol.45, pp. 1707-1720.

Note

Paper [1] is an extension of the work previously published in KDD-2010. It includes an extension from single-dimensional mass estimation to multi-dimensional mass estimation; and their empirical evaluations in three tasks in regression, information retrieval and anomaly detection; and an in-depth analysis and comparison with a closely related method called data depth.

Paper [2] extends the work previously published in IEEE ICDM-2011 to include (i) a contrast between point-based definitions and set-based definitions of the proposed mass-based density estimator; and (ii) an application of mass-based density estimator to Bayesian classifier and an in-depth comparison with existing state-of-the-art Bayesian classifiers.

Paper [6] extends the work previously published in PAKDD-2013 [3] to present the first generic approach to estimate multi-dimensional likelihood $p(\mathbf{x}|y)$ directly by aggregating $p_i(\mathbf{x}|y)$ estimated from an ensemble of estimators where each estimator is constructed from a small fixed-size random sub-sample of data $\mathcal{D}_i \subset D$ ($i = 1, 2, ..., t$). This is a generic approach because $p_i(\mathbf{x}|y)$ can be estimated using different data modelling methods. DEMass-Bayes [2] and MassBayes [3] are two realisations of the proposed generic approach. In this paper, we introduce an additional realisation of the proposed generic approach called ENNBayes along with MassBayes. ENNBayes estimates $p_i(\mathbf{x}|y)$ from $\mathcal{D}_i$ using a nearest neighbour density estimator.

Software Downloads

The source codes of multi-dimensional mass estimation, DEMass-DBSCAN, DEMass-Bayes and MassBayes, algorithms proposed in papers [1, 2, 3], are made available at
`http://sourceforge.net/projects/mass-estimation/`

Attachments

Papers [1, 2, 3, 4, 5, 6] are attached.

Mach Learn
DOI 10.1007/s10994-012-5303-x

Mass estimation

Kai Ming Ting · Guang-Tong Zhou · Fei Tony Liu · Swee Chuan Tan

Received: 17 June 2011 / Revised: 20 May 2012 / Accepted: 1 June 2012

Abstract This paper introduces mass estimation—a base modelling mechanism that can be employed to solve various tasks in machine learning. We present the theoretical basis of mass and efficient methods to estimate mass. We show that mass estimation solves problems effectively in tasks such as information retrieval, regression and anomaly detection. The models, which use mass in these three tasks, perform at least as well as and often better than eight state-of-the-art methods in terms of task-specific performance measures. In addition, mass estimation has constant time and space complexities.

Keywords Mass estimation · Density estimation · Information retrieval · Regression · Anomaly detection

1 Introduction

> 'Estimation of densities is a universal problem of statistics (knowing the densities one can solve various problems).' —Vapnik (2000).

Density estimation has been the base modelling mechanism used in many techniques designed for tasks such as classification, clustering, anomaly detection and information

Editor: Tong Zhang.

K.M. Ting (✉) · F.T. Liu · S.C. Tan
Gippsland School of Information Technology, Monash University, Melbourne, Vic 3842, Australia
e-mail: kaiming.ting@monash.edu

F.T. Liu
e-mail: tony.liu@monash.edu

S.C. Tan
e-mail: james.tan@monash.edu

G.-T. Zhou
School of Computing Science, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada
e-mail: zhouguangtong@gmail.com

Published online: 26 June 2012

retrieval. For example in classification, density estimation is employed to estimate the class-conditional density function (or likelihood function) $p(x|j)$ or posterior probability $p(j|x)$—the principal function underlying many classification methods; e.g., mixture models, Bayesian networks, Naive Bayes. Examples of density estimation include kernel density estimation, k-nearest neighbours density estimation, maximum likelihood procedures and Bayesian methods.

Ranking data points in a given data set in order to differentiate core points from fringe points in a data cloud is fundamental in many tasks, including anomaly detection and information retrieval. Anomaly detection aims to rank anomalous points higher than normal points; information retrieval aims to rank points similar to a query higher than dissimilar points. Many existing methods (e.g., Bay and Schwabacher 2003; Breunig et al. 2000; Zhang and Zhang 2006) have employed density to provide the ranking; but density estimation is not designed to provide a ranking.

We show in this paper that a new base modelling mechanism called *mass estimation* possesses different properties from those offered by density estimation:

- A mass distribution stipulates an ordering from core points to fringe points in a data cloud. In addition, this ordering accentuates the fringe points with a concave function derived from data, resulting in fringe points having markedly smaller mass than points close to the core points.
- Mass estimation is more efficient than density estimation because mass is computed by simple counting and it requires only a small sample through an ensemble approach. Density estimation (often used to estimate $p(x|j)$ and $p(j|x)$) requires a large sample size in order to have a good estimation and is computationally expensive in terms of time and space complexities (Duda et al. 2001).

Mass estimation has two advantages in relation to efficacy and efficiency. First, the concavity property mentioned above ensures that fringe points are ‘stretched’ to be farther from the core points in a mass space—making it easier to separate fringe points from those points close to core points. This property in mass space can then be exploited by a machine learning algorithm to achieve a better result for the intended task than applying the same algorithm in the original space without this property. We show the efficacy of mass in improving the task-specific performance of four existing state-of-the-art algorithms in information retrieval and regression tasks. The significant improvements are achieved through a simple mapping from the original space to a mass space using the mass estimation mechanism introduced in this paper.

Second, mass estimation offers to solve a ranking problem more efficiently using the ordering derived from data directly—without expensive distance (or related) calculations. An example of inefficient application is in anomaly detection tasks where many methods have employed distance or density to provide the required ranking. An existing state-of-the-art density-based anomaly detector `LOF` (Breunig et al. 2000) (which has quadratic time complexity) completes a job involving half a million data points in more than five hours; yet the mass-based anomaly detector we have introduced here completes it in less than 20 seconds! Section 6.3 provides the details of this example.

The rest of the paper is organised as follows. Section 2 introduces mass and mass estimation, together with their theoretical properties. We also describe methods for one-dimensional mass estimation. We extend one-dimensional mass estimation to multi-dimensional mass estimation in Sect. 3. We provide an implementation of multi-dimensional mass estimation in Sect. 4. Section 5 describes a mass-based formalism which serves as a basis of applying mass to different data mining tasks. We realise the formalism in three different tasks: information retrieval, regression and anomaly detection, and report the empirical evaluation results in Sect. 6. The relations to kernel density estimation, data depth and other related work are described in Sects. 7, 8 and 9, respectively. We provide conclusions and suggest future work in the last section.

2 Mass and mass estimation

Data mass or *mass*, in its simplest form, is defined as the number of points in a region. Any two groups of data in the same domain have the same mass if they have the same number of points, regardless of the characteristics of the regions they occupy (e.g., density, shape or volume). Mass in a given region is thus defined by a rectangular function which has the same value for the entire region in which the mass is measured.

To estimate the mass for a point and thus the mass distribution of a given data set, a more sophisticated form is required. The intuition is based on the simplest form described above, but multiple (overlapping) regions covering a point are generated. The mass for the point is then derived from an average of masses from all regions covering the point. We show two ways to define these regions. The first is to generate all possible regions through binary splits from the given data points; and the second is to generate random axis-parallel regions within the confine covered by a data sample. The first is described in this section and the second is described in Sect. 3.

Each region can be defined in multiple levels where a higher level region covering a point has a smaller volume than that of a lower level region covering the same point. We show that the mass distribution has special properties: (i) the mass distribution defined by level-1 regions is a concave function which has the maximum mass at the centre of the data cloud, irrespective of its density distribution, including uniform and U-shape distributions; and (ii) higher level regions are required to model multi-modal mass distributions.

Note that *mass is not a probability mass function, and it does not provide a probability*, as the probability density function does through integration.

In Sect. 2.1, we show (i) how to estimate a mass distribution for a given data set through binary splits and (ii) the theoretical properties of mass estimation. Section 2.2 describes an approximation to the theoretical mass estimation which works more efficiently in practice. The symbols and notations used are provided in Table 1.

Table 1 Symbols and notations

$\mathcal{R}^u$	A real domain of u dimensions
x	A one-dimensional instance in $\mathcal{R}$
$\mathbf{x}$	An instance in $\mathcal{R}^u$
D	A data set of $\mathbf{x}$, where $\lvert D \rvert = n$
$\mathcal{D}$	A subset of D, where $\lvert \mathcal{D} \rvert = \psi$
$\mathbf{z}$	An instance in $\mathcal{R}^t$
D'	A data set of $\mathbf{z}$
c	The ensemble size used to estimate mass
h	Level of mass distribution
t	Number of mass distributions in $\widetilde{\mathbf{mass}}(\cdot)$
$m_i(\cdot)$	Mass base function defined using binary split s_i
$mass(\cdot)$	Mass function which returns a real value in one-dimensional mass space
$\widetilde{\mathbf{mass}}(\cdot)$	Mass function which returns a vector of t values in t-dimensional mass space

2.1 Mass distribution estimation

In this section, we first show in Sect. 2.1.1 a mass distribution estimation that uses binary splits in the one-dimensional setting, where each binary split separates the one-dimensional space into two non-empty regions. In Sect. 2.1.2, we then generalise the treatment using multiple levels of binary splits.

2.1.1 Mass distribution estimation using binary splits

Here, we employ a binary split to divide the data set into two separate regions and compute the mass in each region. The mass distribution at point x is estimated to be the sum of all 'weighted' masses from regions occupied by x, as a result of $n-1$ binary splits for a data set of size n.

Let $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line,[1] $x_i \in \mathcal{R}$ and $n > 1$. Let s_i be the binary split between x_i and x_{i+1}, yielding two non-empty regions having two masses m_i^L and m_i^R.

Definition 1 Mass base function: $m_i(x)$ as a result of s_i, is defined as

$$m_i(x) = \begin{cases} m_i^L & \text{if } x \text{ is on the left of } s_i \\ m_i^R & \text{if } x \text{ is on the right of } s_i \end{cases}$$

Note that $m_i^L = n - m_i^R = i$.

Definition 2 Mass distribution: $mass(x_a)$ for a point $x_a \in \{x_1, x_2, \ldots, x_{n-1}, x_n\}$ is defined as a summation of a series of mass base functions $m_i(x)$ weighted by $p(s_i)$ over $n-1$ splits as follows, where $p(s_i)$ is the probability of selecting s_i.

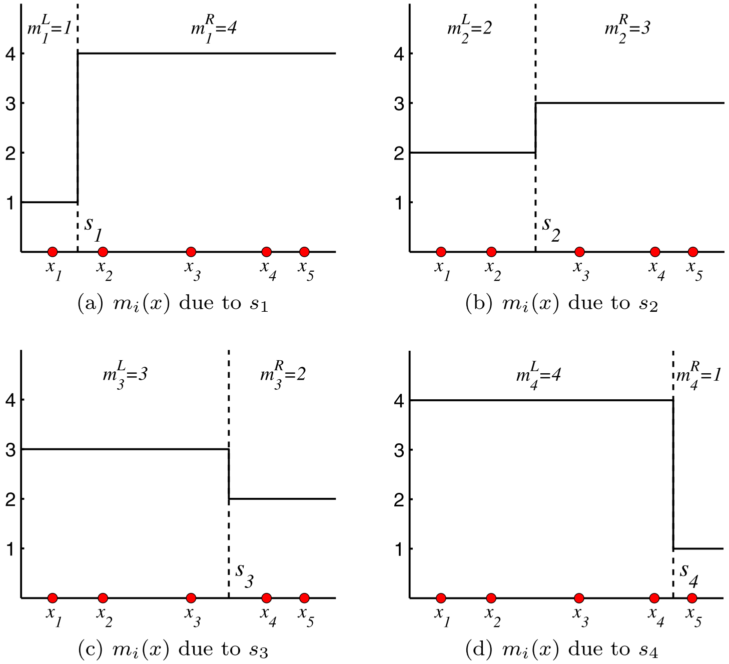
$$\begin{aligned} mass(x_a) &= \sum_{i=1}^{n-1} m_i(x_a)p(s_i) \\ &= \sum_{i=a}^{n-1} m_i^L p(s_i) + \sum_{j=1}^{a-1} m_j^R p(s_j) \\ &= \sum_{i=a}^{n-1} i p(s_i) + \sum_{j=1}^{a-1} (n-j)p(s_j) \end{aligned} \tag{1}$$

Note that we have defined $\sum_{i=q}^{r} f(i) = 0$, when $r < q$ for any function f.

Example For an example of five points $x_1 < x_2 < x_3 < x_4 < x_5$, Fig. 1 shows the resultant $m_i(x)$ due to each of the four binary splits s_1, s_2, s_3, s_4; and their associated masses over four splits are given below:

$$\begin{aligned} mass(x_1) &= 1p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_2) &= 4p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_3) &= 4p(s_1) + 3p(s_2) + 3p(s_3) + 4p(s_4) \\ mass(x_4) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 4p(s_4) \\ mass(x_5) &= 4p(s_1) + 3p(s_2) + 2p(s_3) + 1p(s_4) \end{aligned}$$

[1] In data having a pocket of points of the same value, an arbitrary order can be 'forced' by adding increasing multiples of an insignificant small value ϵ to each subsequent point of the pocket, without changing the general distribution.



(a) $m_i(x)$ due to s_1 (b) $m_i(x)$ due to s_2

(c) $m_i(x)$ due to s_3 (d) $m_i(x)$ due to s_4

Fig. 1 Examples of mass base function $m_i(x)$ due to each of the four binary splits: s_1, s_2, s_3, s_4

For a given data set, $p(s_i)$ can be estimated on the real line as $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1) > 0$, as a result of a random selection of splits based on a uniform distribution.[2]

For a point $x \notin \{x_1, x_2, \ldots, x_{n-1}, x_n\}$, $mass(x)$ is defined as an interpolation between two masses of adjacent points x_i and x_{i+1}, where $x_i < x < x_{i+1}$.

Theorem 1 *$mass(x_a)$ is maximised at $a = n/2$ for any density distribution of $\{x_1, \ldots, x_n\}$; and the points x_a, where $x_1 < x_2 < \cdots < x_{n-1} < x_n$ on the real line, can be ordered based on mass as follows.*

$$mass(x_a) < mass(x_{a+1}), \quad a < n/2$$

$$mass(x_a) > mass(x_{a+1}), \quad a > n/2$$

Proof The difference in mass between two consecutive points x_a and x_{a+1} differs in only one term, i.e., the mass associated with $p(s_a)$; and $\forall i \neq a$, the terms for $p(s_i)$ have the same mass.

[2] The estimated $mass(x)$ values can be calibrated to a finite data range Δ by multiplying a factor $(x_n - x_1)/\Delta$.

$$\begin{aligned} mass(x_a) - mass(x_{a+1}) &= \sum_{i=a}^{n-1} ip(s_i) + \sum_{j=1}^{a-1}(n-j)p(s_j) \\ &\quad - \sum_{i=(a+1)}^{n-1} ip(s_i) - \sum_{j=1}^{a}(n-j)p(s_j) \\ &= ap(s_a) - (n-a)p(s_a) \\ &= (2a-n)p(s_a) \end{aligned} \tag{2}$$

Thus,

$$sign\big(mass(x_a) - mass(x_{a+1})\big) = \begin{cases} \text{negative} & \text{if } a < n/2 \\ 0 & \text{if } a = n/2 \\ positive & \text{if } a > n/2 \end{cases}$$

The point $x_{n/2}$ can be regarded as the median. Note that the number of points with the maximum mass depends on whether n is odd or even: When n is an odd integer, only one point has the maximum mass at x_{median}, where $median = \lceil n/2 \rceil$; when n is an even integer, two points have the maximum mass at $a = n/2$ and $a = 1 + n/2$. □

Theorem 2 *$mass(x_a)$ is a concave function defined w.r.t. $\{x_1, x_2, \dots, x_n\}$, when $p(s_i) = (x_{i+1} - x_i)/(x_n - x_1)$ for $n > 2$.*

Proof We only need to show that the gradient of x_a is non-increasing, i.e., $g(x_a) > g(x_{a+1})$ for each a.

Let $g(x_a)$ be the gradient between x_a and x_{a+1}, and from (2):

$$g(x_a) = \frac{mass(x_{a+1}) - mass(x_a)}{x_{a+1} - x_a} = \frac{n-2a}{x_n - x_1}$$

The result follows: $g(x_a) > g(x_{a+1})$ for $a \in \{1, 2, \dots, n-1\}$. □

Corollary 1 *A mass distribution estimated using binary splits stipulates an ordering, based on mass, of the points in a data cloud from $x_{n/2}$ (with the maximum mass) to the fringe points (with the minimum mass at either side of $x_{n/2}$), irrespective of the density distribution including uniform density distribution.*

Corollary 2 *The concavity of mass distribution stipulates that fringe points have markedly smaller mass than points close to $x_{n/2}$.*

The implication from Corollary 2 is that fringe points are 'stretched' to be farther away from the median in a mass space than in the original space—making it easier to separate fringe points from those points close to the median. The mass space is mapped from the original space through $mass(x)$. This property in mass space can then be exploited by a machine learning algorithm to achieve a better result for the intended task than applying the same algorithm in the original space without this property. We will show that this simple mapping significantly improves the performance of four existing algorithms in information retrieval and regression tasks in Sects. 6.1 and 6.2.

Equation (1) is sufficient to provide a mass distribution corresponding to a unimodal density function or a uniform density function. To better estimate multi-modal mass distributions, multiple levels of binary splits need to be carried out. This is provided in the following.

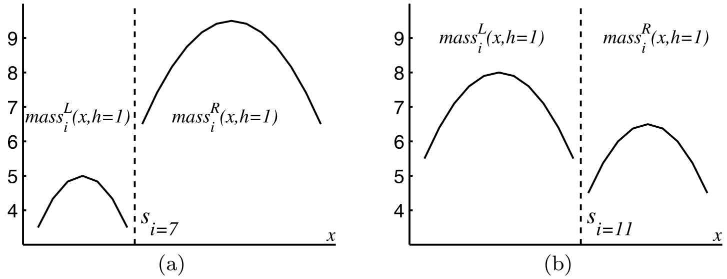


Fig. 2 Two examples of $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$ due to $s_{i=7}$ and $s_{i=11}$ in the process to get $mass(x, h = 2)$ from a data set of 20 points with uniform density distribution. The resultant $mass(x, h = 2)$ is shown in Fig. 3(a)

2.1.2 Level-h mass distribution estimation

If we treat the mass estimation defined in the last subsection as level-1 estimation, then level-h estimation can be viewed as localised versions of the basic level-1 estimation.

Definition 3 The level-h mass distribution for a point $x_a \in \{x_1, \dots, x_n\}$, where $h < n$, is expressed as

$$\begin{aligned} mass(x_a, h) &= \sum_{i=1}^{n-1} mass_i(x_a, h\text{-}1)\, p(s_i) \\ &= \sum_{i=a}^{n-1} mass_i^L(x_a, h\text{-}1)\, p(s_i) \\ &\quad + \sum_{j=1}^{a-1} mass_j^R(x_a, h\text{-}1)\, p(s_j) \end{aligned} \tag{3}$$

Here a high level mass distribution is computed recursively by using the mass distributions obtained at lower levels. A binary split s_i in a level-$h(> 1)$ mass distribution produces two level-(h-1) mass distributions: (a) $mass_i^L(x, h\text{-}1)$—the mass distribution on the left of split s_i which is defined using $\{x_1, \dots, x_i\}$; and (b) $mass_i^R(x, h\text{-}1)$—the mass distribution on the right which is defined using $\{x_{i+1}, \dots, x_n\}$. Equation (1) is the mass distribution at level-1.

Figure 2 shows two (out of 19 splits) required to compute level-2 mass estimation, $mass(x, h = 2)$, from a data set of 20 points. Each split produces two level-1 mass estimations: $mass_i^L(x, h = 1)$ and $mass_i^R(x, h = 1)$. Note that level-1 mass distribution is concave, as proven in Theorem 2. This example shows the results of two splits $s_{i=7}$ and $s_{i=11}$, where each level-1 mass distribution is concave.

Using the same analysis as in the proof for Theorem 1, the above equation can be re-expressed as:

$$mass(x_{a+1}, h) = mass(x_a, h) + \begin{cases} [mass_a^R(x_a, h\text{-}1) - mass_a^L(x_a, h\text{-}1)]p(s_a), & h > 1 \\ (n - 2a)p(s_a), & h = 1 \end{cases} \tag{4}$$

As a result, only the mass for the first point x_1 needs to be computed using (3). Note that it is more efficient to compute the mass distribution from the above equation which has time complexity $O(n^{h+1})$; the computation using (3) has complexity $O(n^{h+2})$.

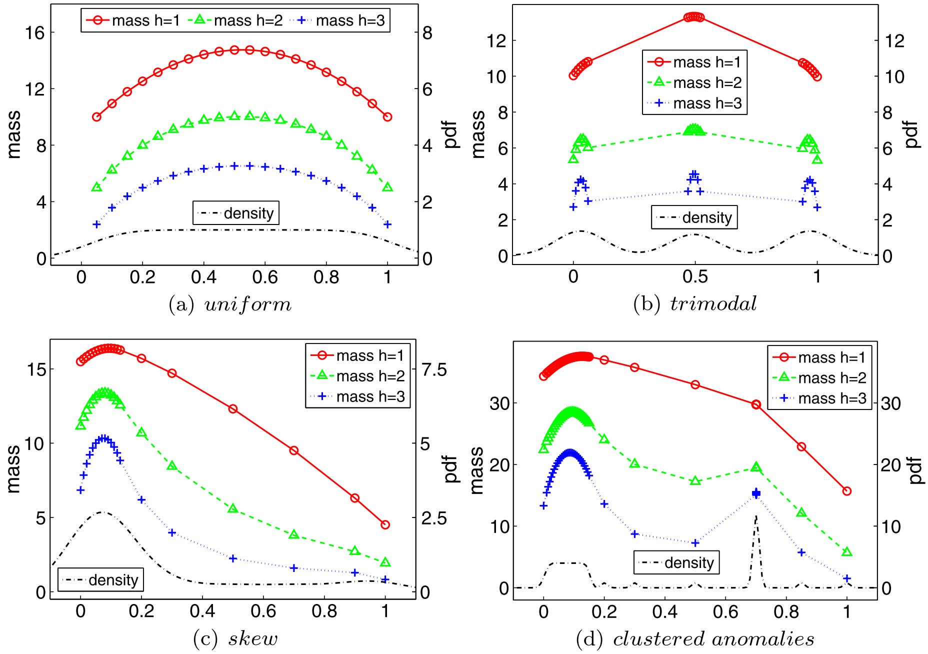


Fig. 3 Examples of level-h mass distribution for $h = 1, 2, 3$ and density distribution from kernel density estimation: Gaussian kernel with bandwidth $= 0.1$ (for the first three figures) and 0.01 (for the last figure in order to show the density spike). The data sets have 20 points each for the first three figures, and the last one has 50 points

Definition 4 A level-h mass distribution stipulates an ordering of the points in a data cloud from α-core points to the fringe points. Let α-neighbourhood of a point x be defined as $N_\alpha(x) = \{y \in D | dist(x, y) \leq \alpha\}$ for some distance function $dist(\cdot, \cdot)$. Each α-core point x^* in a data cloud has the highest mass value $\forall x \in N_\alpha(x^*)$. A small α defines local core point(s); and a large α, which covers the entire value range for x, defines global core point(s).

Examples of level-h mass estimation in comparison with kernel density estimation are provided in Fig. 3. Note that $h = 1$ mass estimation looks at the data as a group, and it produces a concave function. As a result, an $h = 1$ mass estimation always has its global core point(s) at the median, regardless of the underlying density distribution—see the four examples of $h = 1$ mass estimation in Fig. 3.

For $h > 1$ mass distribution, though there is no guarantee for a concave function any more as a whole, our simulation shows that each cluster within the data cloud (if they exist) exhibits a concave function and it becomes more distinct (as a concave function) as h increases. An example is shown in Fig. 3(b) which has a trimodal density distribution. Notice that the $h > 1$ mass distributions have three α-core points for some α, e.g., 0.2. Other examples are shown in Figs. 3(c) and 3(d).

Traditionally, one can estimate the core-ness or the fringe-ness of non-uniformly distributed data to some degree by using density or distance (but not in uniform density distribution). Mass allows one to do that in any distribution without density or distance calculation—the key computational expense in all methods that employ them. For example in Fig. 3(c) which has a skew density distribution, the distinction between near fringe points and far fringe points are less obvious using density, unless distances are computed to reveal the difference. In contrast, mass distribution depicts the relative distance from x_{median} using the fringe points' mass values, without further calculation.

Figure 3(d) shows an example where there are clustered anomalies which are denser than the normal points (shown in the bigger cluster on the left of the figure). Anomaly detection based on density will identify all these clustered anomalies as more 'normal' than the normal points because anomalies are defined as points having low density. In sharp contrast, $h = 1$ mass estimation will correctly rank them as anomalies which have the third lowest mass values. These points are interpreted as points at the fringe of the data cloud of normal points which have higher mass values.

This section has described properties of mass distribution from a theoretical perspective. Though it is possible to estimate mass distribution using (1) and (3), they are limited by its high computational cost. We suggest a practical mass estimation method in the next subsection. We use the term 'mass estimation' and 'mass distribution estimation' interchangeably hereafter.

2.2 Practical one-dimensional level-h mass estimation

Here we devise an approximation to (3) using random subsamples from a given data set.

Definition 5 $mass(x, h|\mathcal{D})$ is the approximate mass distribution for a point $x \in \mathcal{R}$, defined w.r.t. $\mathcal{D} = \{x_1, \ldots, x_\psi\}$, where $\mathcal{D}$ is a random subset of the given data set D, and $\psi \ll |D|$, $h < \psi$.

Here we employ a rectangular function to define mass for a region encompassing each point $x \in \mathcal{D}$. $mass(x, h|\mathcal{D})$ is implemented using a lookup table where a region for each point $x_i \in \mathcal{D}$ covers a range $(x_{i-1} + x_i)/2 \leq x < (x_{i+1} + x_i)/2$ and has the same $mass(x_i, h|\mathcal{D})$ value for the entire region. The range for each of the two end-points is set to have equal length on either side of the point. An example is provided in Fig. 4(a).

In addition, a number of mass distributions needs to be constructed from different samples in order to have a good approximation, that is,

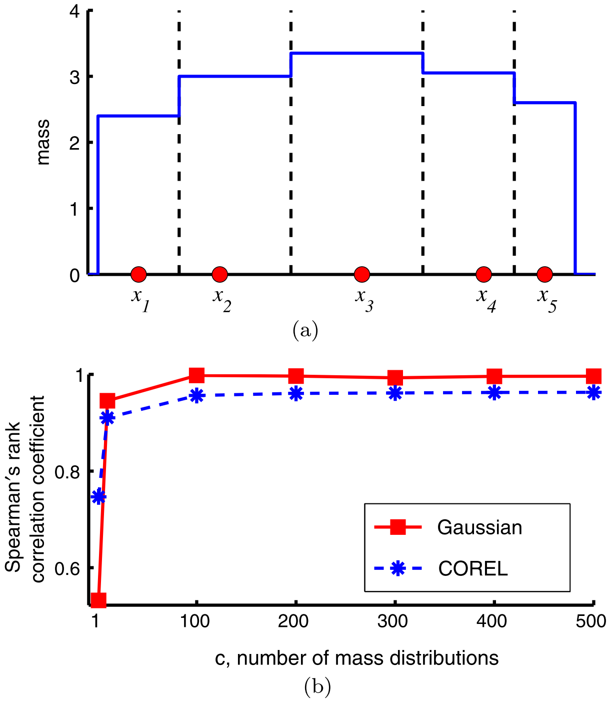
$$\overline{mass}(x, h) \approx \frac{1}{c} \sum_{k=1}^{c} mass(x, h|\mathcal{D}_k) \tag{5}$$

The computation of $mass(x, h)$ using the given data set D costs $O(|D|^{h+1})$ in terms of time complexity; whereas $mass(x, h|\mathcal{D})$ costs $O(\psi^{h+1})$.

Only relative, not absolute, mass is required to provide an ordering between instances. For $h = 1$, because the relative mass is w.r.t. the median and the median is a robust estimator (Aloupis 2006)—that is why small subsamples produce a good estimator for ordering. While this reason cannot be applied to $h > 1$ (and multi-dimensional mass estimation to be discussed in the next section) because the notion of median is undefined, our empirical results in Sect. 6 show that all these mass estimations using small subsamples produce good results.

In order to show that relative performance of $mass(x, 1)$ and $mass(x, 1|\mathcal{D})$, we compare the ordering results based on mass values in two separate data sets: the one-dimensional Gaussian density distribution and the COREL data set; each of the data sets has 10000 data points. Figure 4(b) shows the correlation (in terms of Spearman's rank correlation coefficient) between the orderings provided by $mass(x, 1)$ using the entire data set and

Fig. 4 (**a**) An example of practical mass distribution $mass(x, h|\mathcal{D})$ for 5 points, assuming a rectangular function for each point. (**b**) Correlation between the orderings provided by $mass(x, 1)$ and $mass(x, 1|\mathcal{D})$ for two data sets: one-dimensional Gaussian density distribution and the COREL data set used in Sect. 6.1 (whose result is averaged over 67 dimensions)



(a)

(b)

$mass(x, 1|\mathcal{D})$ using $\psi = 8$. They achieve very high correlations when $c \geq 100$ for both data sets.

The ability to use a small sample, rather than a large sample, is a key characteristic of mass estimation.

3 Multi-dimensional mass estimation

Ting and Wells (2010) describe a way to generalise the one-dimensional mass estimation we have described in the last section. We reiterate the approach in this section but the implementation we employed (to be described in Sect. 4) differs. Section 9 provides the details of these differences.

The approach proposed by Ting and Wells (2010) eliminates the need to compute the probability of a binary split, $p(s_i)$; and it gives rise to randomised versions of (1), (3) and (5).

The idea is to generate multiple random regions which cover a point, and then the mass for that point is estimated by averaging all masses from all those regions. We show that random regions can be generated using axis-parallel splits called half-space splits. Each half-space split is performed on a randomly selected attribute in a multi-dimensional feature space. For a h-level split, each half-space split is carried out h times recursively along every path in a tree structure. Each h-level (axis-parallel) split generates 2^h non-overlapping regions. Multiple h-level splits are used to estimate mass for each point in the feature space.

The multi-dimensional mass estimation requires two functions. First, it needs a function that generates random regions covering each point in the feature space. This function is a generalisation of the binary split into half-space splits or 2^h-region splits when h levels of half-space splits are used. Second, a generalised version of the mass base function is used to define mass in a region. The formal definition follows.

Let $\mathbf{x}$ be an instance in $\mathcal{R}^d$. Let $T^h(\mathbf{x})$ be one of the 2^h regions in which $\mathbf{x}$ falls into; $T^h(\cdot)$ is generated from the given data set D, and $T^h(\cdot|\mathcal{D})$ is generated from $\mathcal{D} \subset D$; and m be the number of training instances in the region.

The generalised mass base function: $\mathbf{m}(T^h(\mathbf{x}))$ is defined as

$$\mathbf{m}\big(T^h(\mathbf{x})\big) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{ is in a region of } T^h \text{ having } \mathsf{m} \text{ instances} \\ 0 & \text{otherwise} \end{cases}$$

In one-dimensional problems, (1), (3) and (5) can now be approximated as follows:

$$\sum_{i=1}^{n-1} m_i(x)p(s_i) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}\big(T_k^1(x)\big) \tag{6}$$

$$mass(x,h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}\big(T_k^h(x)\big) \tag{7}$$

$$\overline{mass}(x,h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}\big(T_k^h(x|\mathcal{D}_k)\big) \tag{8}$$

where $c > 0$ is the number of random regions to be used to define the mass for x.

Here every T_k^h is generated randomly with equal probability. Note that $p(s_i)$ in (1) has the same assumption.

Since T^h is defined in multi-dimensional space, the multi-dimensional mass estimation is the same as (8) by simply replacing x with $\mathbf{x}$:

$$\overline{mass}(\mathbf{x},h) \approx \frac{1}{c}\sum_{k=1}^{c} \mathbf{m}\big(T_k^h(\mathbf{x}|\mathcal{D}_k)\big) \tag{9}$$

Like its one-dimensional counterpart, the multi-dimensional mass estimation stipulates an ordering from core points (having high mass) to fringe points (having low mass) in a data cloud, regardless of its density distribution. While we do not have a proof of this property for multi-dimensional mass estimation, empirical results suggest that it is. This property is shown in Fig. 5(a) using $h = 1$, where the highest mass value is at the centre of the entire data cloud, when the four clusters are treated as a single data cloud; while the four clusters are scattered in each of the four quadrants. Mass values become lower as they move away from the centre. Note that though this implementation of multi-dimensional mass estimation does not guarantee concavity, the approximation of the ordering is sufficiently close to a concave function (in regions with data) to produce the required ranking for different purposes (see Sect. 6).

Figure 5(b) shows the contour map for $h = 32$ on the same data set. It demonstrates that multi-dimensional mass estimation can use a high h level to model multi-modal distribution.

We show in Sect. 6 that both $mass(x, h|\mathcal{D})$ and $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ (in (5) and (9), respectively) can be employed effectively for three different tasks: information retrieval, regression and anomaly detection, through the mass-based formalism described in Sect. 5. We shall describe the implementation of multi-dimensional mass estimation in the next section.

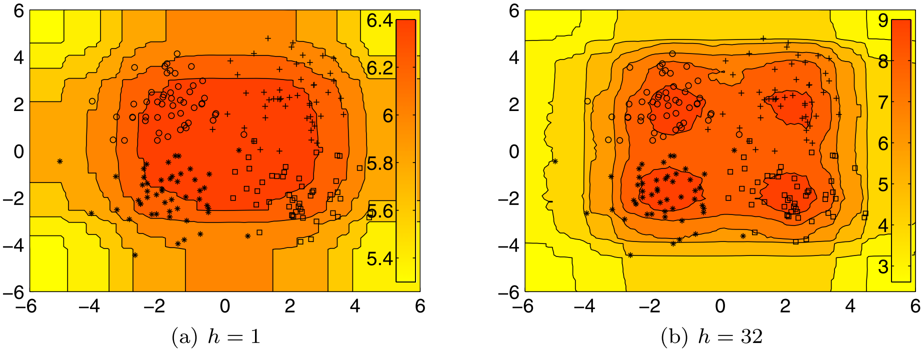


Fig. 5 Contour maps of multi-dimensional mass distribution for a two-dimensional data set with four clusters (each containing 50 points), where points in each cluster are marked with a distinct marker. The points are randomly drawn from Gaussian distributions with unit standard deviation and means located at (2; 2), (−2; 2), (−2; −2) and (2; −2), respectively. The two figures are produced using $h = 1$ and $h = 32$, respectively. Other parameters are set as follows: $c = 1000$ and $\psi = |\mathcal{D}| = 64$. The algorithm used to generate these contour maps will be described in Sect. 4.2. The legend indicates the colour-coded mass values

4 Half-Space Trees for mass estimation

This section describes the implementation of T^h using Half-Space Tree. Two variants are provided. We have used the second variant of Half-Space Tree to implement the multi-dimensional mass estimation.

4.1 Half-Space Tree

The motivation of the proposed method, Half-Space Tree, comes from the fact that equal-size regions contain the same mass in a space with uniform mass distribution, regardless of the shapes of the regions. This is shown in Fig. 6(a), where the space enveloped by the data is split into equal-size half-spaces recursively three times into eight regions. Note that the shapes of the regions may be different because the splits at the same level may not use the same attribute.

The binary half-space split ensures that every split produces two equal-size half-spaces, each containing exactly half of the mass before the split under a uniform mass distribution. This characteristic enables us to compute the relationship between any two regions easily. For example, the mass in every region shown in Fig. 6(a) is the same, and it is equivalent to the original mass divided by 2^3 because three levels of binary half-space splits have been applied. A deviation from the uniform mass distribution allows us to rank the regions based on mass. Figure 6(b) provides such an example in which a ranking of regions based on mass provides an order of the degrees of anomaly in each region.

Definition 6 Half-Space Tree is a binary tree in which each internal node makes a half-space split into two equal-size regions, and each external node terminates further splits. All nodes record the mass of the training data in their own regions.

Let $T^h[i]$ be a Half-Space Tree with depth level i; and $\mathbf{m}(T^h[i])$ or short for $\mathbf{m}[i]$ be the mass in one of the regions at level i.

The relationship between any two regions is expressed using mass with reference to $\mathbf{m}[0]$ at depth level $= 0$ (the root) of a Half-Space Tree.

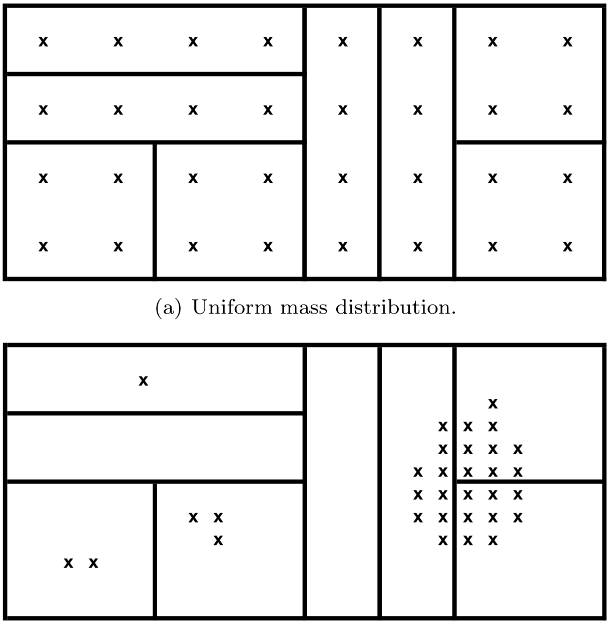


Fig. 6 Half-space subdivisions of: (**a**) uniform mass distribution; and (**b**) non-uniform mass distribution

(a) Uniform mass distribution.

(b) Non-uniform mass distribution.

Under uniform mass distribution, the mass at level i is related to mass at level 0 as follows:

$$\mathbf{m}[0] = \mathbf{m}[i] \times 2^i,$$

or mass values between any two regions at levels i and j, $\forall i \neq j$, are related as follows:

$$\mathbf{m}[i] \times 2^i = \mathbf{m}[j] \times 2^j.$$

Under non-uniform mass distribution, the following inequality establishes an ordering between any two regions at different levels:

$$\mathbf{m}[i] \times 2^i < \mathbf{m}[j] \times 2^j.$$

We employ the above property to rank instances and define the (augmented) mass for Half-Space Tree as follows.

$$s(\mathbf{x}) = \mathbf{m}[\ell] \times 2^\ell, \tag{10}$$

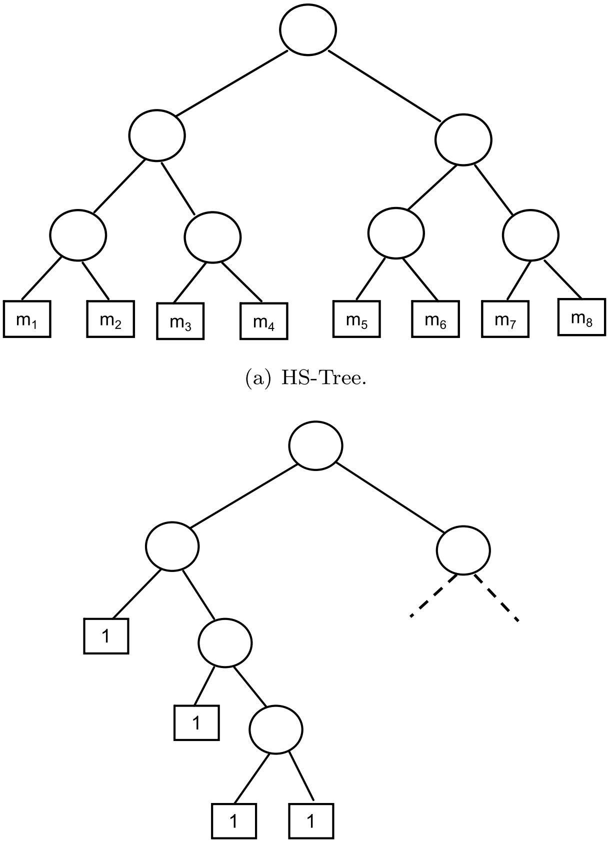
where ℓ is the depth level of an external node with $\mathbf{m}[\ell]$ instances in which a test instance $\mathbf{x}$ falls into.

Mass is estimated using $\mathbf{m}[\ell]$ only if a Half-Space Tree has all external nodes at the same depth level. The estimation is based on augmented mass, $\mathbf{m}[\ell] \times 2^\ell$, if the external nodes have differing depth levels. We describe two such variants of Half-Space Tree below.

HS-Tree: *based on mass only*. The first variant, HS-Tree, builds a balanced binary tree structure which makes a half-space split at each internal node and all external nodes have the same depth. The number of training instances falling into each external node is recorded and it is used for mass estimation. An example of HS-Tree is shown in Fig. 7(a).

HS-Tree*: *based on augmented mass*. Unlike HS-Tree, the second variant, HS*-Tree, whose external nodes have differing depth levels. The mass estimated from HS*-Tree is

Fig. 7 Half-Space Tree: (**a**) HS-Tree: An HS-Tree for the data shown in Fig. 6(a) has $\mathsf{m}_i = 4, \forall i$, which are $\mathbf{m}[\ell = 3]$ (i.e., mass at level 3). (**b**) HS*-Tree: An example of a special case of HS*-Tree when the size limit is set to 1



(a) HS-Tree.

(b) HS*-Tree.

defined in equation (10) in order to account for different depths. We call this *augmented mass* because the mass is augmented in the calculation by the depth level in HS*-Tree, as opposed to mass only in HS-Tree.

In a special case of HS*-Tree, the tree growing process at a branch will only terminate to form an external node if the training data size at the branch is 1 (i.e., the size limit is set to 1). Here the mass estimated depends on depth level only, i.e., 2^ℓ or simply ℓ. In other words, *the depth level becomes a proxy for mass in HS*-Tree* when the size limit is set to 1. An example of HS*-Tree, when the size limit is set to 1, is shown in Fig. 7(b).

Since the two variants have similar performance, we focus on HS*-Tree only in this paper because it builds a smaller-sized tree than HS-Tree which may grow many branches with zero mass—this saves on training time and memory space requirements.

4.2 Algorithm to generate Half-Space Trees

Half-Space Trees estimate a mass distribution efficiently, without density or distance calculations or clustering. We first describe the training procedure, then the testing procedure, and finally the time and space complexities.

Training. The procedure to generate a Half-Space Tree is shown in Algorithm 1. It starts by defining a (random) range for each dimension in order to form a work space which

Algorithm 1 $T^h(\mathcal{D}, S, h)$

```
Inputs: $\mathcal{D}$—input data, $S$—data size limit at external node, $h$—maximum depth limit
Output: a Half-Space Tree
1: $SizeLimit \leftarrow S$
2: $MaxDepthLimit \leftarrow h$
3: $(min, max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
4: return SingleHalf-SpaceTree($\mathcal{D}, min, max, 0$)
```

covers all the training data. The *InitialiseWorkSpace*$(\cdot)$ function in Algorithm 1 is carried out as follows. For each attribute q, a random split value (z_q) is chosen within the range $[\mathcal{D}min_q, \mathcal{D}max_q]$, i.e., the minimum and maximum values of q in the subsample. Then, attribute q of the work space is defined to have the range $[min_q, max_q] = [z_q - r, z_q + r]$, where $r = 2 \cdot \max(z_q - \mathcal{D}min_q, \mathcal{D}max_q - z_q)$. The ranges of all dimensions define the work space used to generate a Half-Space Tree. The work space defined by $[min_q, max_q]$ is then passed over to Algorithm 2 to construct a Half-Space Tree.

Constructing a single Half-Space Tree is almost identical to constructing an ordinary decision tree[3] (Quinlan 1993), except that no splitting selection criterion is required at each node.

Given a work space, an attribute q is randomly selected to form an internal node of an Half-Space Tree (line 4 in Algorithm 2). The split point of this internal node is simply the mid-point between the minimum and maximum values of attribute q (i.e., min_q and max_q), defined by the work space (line 5). Data are filtered through one of the two branches depending on which side of the split the data reside (lines 6–7). This node building process is repeated for each branch (lines 9–12 in Algorithm 2) until a size limit or a depth limit is reached to form an external node (lines 1–2 in Algorithm 2). The training instances at the external node at depth level ℓ form the mass $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ to be used during the testing process for $\mathbf{x}$. The parameters are set as follows: $S = \log_2(|\mathcal{D}|) - 1$ and $h = |\mathcal{D}|$ for all the experiments conducted in this paper.

Ensemble. The proposed method uses a random subsample $\mathcal{D}$ to build one Half-Space Tree (i.e., $T^h(\cdot|\mathcal{D})$), and multiple Half-Space Trees are constructed from different random subsamples (using sampling without replacement) to form an ensemble.

Testing. During testing, a test instance $\mathbf{x}$ traverses through each Half-Space Tree from the root to an external node, and the mass recorded at the external node is used to compute its augmented mass (see (11) below). This testing is carried out for all Half-Space Trees in the ensemble, and the final score is the average score from all trees, as expressed in (12) below.

The mass, augmented by depth ℓ of the region of Half-Space Tree T^h in which $\mathbf{x}$ falls into, is given as follows.

$$s(\mathbf{x}, h) = \mathbf{m}\big(T^h(\mathbf{x}|\mathcal{D})\big) \times 2^{\ell} \tag{11}$$

Mass needs to be augmented with depth ℓ of a Half-Space Tree in order to 'normalise' the masses from different depths in the tree.

The mass for point $\mathbf{x}$ estimated from an ensemble of c Half-Space Trees is given as follows.

$$\overline{mass}(\mathbf{x}, h) \approx \frac{1}{c}\sum_{k=1}^{c} s_k(\mathbf{x}, h) \tag{12}$$

[3] However, they are for different tasks: Decision trees are for supervised learning tasks; Half-Space trees are for unsupervised learning tasks.

Algorithm 2 SingleHalf-SpaceTree($\mathcal{D}, min, max, \ell$)

```
Inputs: $\mathcal{D}$—input data, $min$ & $max$—arrays of minimum and maximum values for all attributes in a work space, $\ell$—current depth level
Output: a Half-Space Tree
if ($|\mathcal{D}| \leq SizeLimit$) or ($\ell \geq MaxDepthLimit$) then
    return $exNode(Size \leftarrow |\mathcal{D}|)$
else
    randomly select an attribute $q$
    $mid_q \leftarrow (max_q + min_q)/2$ {$mid_q$ is the mid-point value between the current $min_q$ and $min_q$ values of $q$.}
    $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < mid_q)$ {Extract data satisfying condition: $q < mid_q$.}
    $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq mid_q)$ {Extract data satisfying condition: $q \geq mid_q$.}
    {Build two nodes: Left and Right as a result of a split into two equal-volume half-spaces.}
    $temp \leftarrow max_q$; $max_q \leftarrow mid_q$
    $Left \leftarrow$ SingleHalf-SpaceTree($\mathcal{D}_l, min, max, \ell + 1$)
    $max_q \leftarrow temp$; $min_q \leftarrow mid_q$
    $Right \leftarrow$ SingleHalf-SpaceTree($\mathcal{D}_r, min, max, \ell + 1$)
    return $inNode(Left, Right, SplitAtt \leftarrow q,\ SplitValue \leftarrow mid_q)$
end if
```

Time and Space complexities. Because it involves no evaluations or searches, a Half-Space Tree can be generated quickly. In addition, a good performing Half-Space Tree can be generated using only a small subsample (size ψ) from a given data set of size n, where $\psi \ll n$. An ensemble of Half-Space Trees has training time complexity $O(ch\psi)$ which is constant for an ensemble with fixed subsample size ψ, maximum depth level h and ensemble size c. It has time complexity $O(chn)$ during testing. The space complexity for Half-Space Trees is $O(ch\psi)$ and is also a constant for an ensemble with fixed subsample size, maximum depth level and ensemble size.

5 Mass-based formalism

The data ordering expressed as a mass distribution can be interpreted as a measure of relevance with respect to the concept underlying the data, i.e., points having high mass are highly relevant to the concept and points having low mass are less relevant. In tasks whose primary aim is to rank points in a database with reference to a data profile, mass provides the ideal ranking measure without distance or density calculations. In anomaly detection, high mass signifies normal points and low mass signifies anomalies; in information retrieval, high (low) mass signifies that a database point is highly (less) relevant to the query. Even in tasks whose primary aim is not ranking, the transformed mass space can be better exploited by existing algorithms because the transformation stretches concept-irrelevant points farther away from relevant points in the mass space.

We introduce a formalism in which mass can be applied to different tasks in this section, and provide the empirical evaluation in the following section.

Let $\mathbf{x}_i = [x_i^1, \ldots, x_i^u]$; $\mathbf{x}_i \in D$ of u dimensions; and $\mathbf{z}_i = [z_i^1, \ldots, z_i^t]$; $\mathbf{z}_i \in D'$ in the transformed mass space of t dimensions. The proposed formalism consists of three components:

Algorithm 3 Mass_Estimation(D, ψ, h, t)

```
Inputs: $D$—input data; $\psi$—data size for $\mathcal{D}_k$; $h$—level of mass distribution; $t$—number of
    mass distributions.
Output: $\widetilde{\mathbf{mass}}(\mathbf{x}) \to \mathcal{R}^t$—a function consists of $t$ mass distributions, using either one-
    dimensional or multi-dimensional mass estimation: $mass(x^d, h|\mathcal{D}_k)$ or $\mathbf{m}(T_k^h(\mathbf{x}|\mathcal{D}_k))$.
1: for $k = 1$ to $t$ do
2:     $\mathcal{D}_k \leftarrow$ a random subset of size $\psi$ from $D$;
3:     $d \leftarrow$ a randomly selected dimension from { $1, \ldots,$ u };
4:     Build $mass(x^d, h|\mathcal{D}_k)$;
5: end for
   Note: For multi-dimensional mass estimation, steps 3 and 4 are replaced with one step:
   Build $\mathbf{m}(T_k^h(\mathbf{x}|\mathcal{D}_k))$;
```

Algorithm 4 Mass_Mapping($D, \widetilde{\mathbf{mass}}$)

```
Inputs: $D$—input data; $\widetilde{\mathbf{mass}}$—a function consists of $t$ mass distributions.
Output: $D'$—a set of mapped instances $\mathbf{z}_i$ in $t$ dimensions.
for $i = 1$ to $|D|$ do
    $\mathbf{z}_i \leftarrow \widetilde{\mathbf{mass}}(\mathbf{x}_i)$;
end for
```

C1 The first component constructs a number of mass distributions in a mass space. A mass distribution $mass(x^d, h|\mathcal{D})$ for dimension d in the original feature space is obtained using our proposed one-dimensional mass estimation, as given in Definition 5. A total number of t mass distributions is generated which forms $\widetilde{\mathbf{mass}}(\mathbf{x}) \to \mathcal{R}^t$, where $t \gg u$. This procedure is given in Algorithm 3. Multi-dimensional mass estimation $\mathbf{m}(T^h(\mathbf{x}|\mathcal{D}))$ (replacing one-dimensional mass estimation $mass(x^d, h|\mathcal{D})$) can be used to generate the mass space similarly; see note in Algorithm 3.

C2 The second component maps the data set D in the original space of u dimensions into a new data set D' in t-dimensional mass space using $\widetilde{\mathbf{mass}}(\mathbf{x}) = \mathbf{z}$. This procedure is described in Algorithm 4.

C3 The third component employs a decision rule to determine the final outcome for the task at hand. It is a task-specific decision function applied to $\mathbf{z}$ in the new mass space.

The formalism becomes a blueprint for different tasks. Components **C1** and **C3** are mandatory in the formalism, but component **C2** is optional, depending on the task.

For information retrieval and regression, the task-specific **C3** procedure is simply using an existing algorithm for the task except that the process is carried out in the new mapped mass space, instead of the original space. The `MassSpace` procedure is given in Algorithm 5.

The task-specific **C3** procedure for anomaly detection is shown in steps 2–5 in Algorithm 6: `MassAD`. Note that anomaly detection requires **C1** and **C3** only; whereas the other two tasks require all three components.

In our experiments described in the next section, the mapping from u dimensions to t dimensions using Algorithm 3 is carried out one dimension at a time when using one-dimensional mass estimation; and all u dimensions at a time when using multi-dimensional mass estimation. Each such mapping produces one dimension in mass space and is repeated t times to get a t-dimensional mass space. Note that randomisation gives different variations

Algorithm 5 Perform task in `MassSpace`(D, ψ, h, t)

```
Inputs: $D$—input data; $\psi$—data size for $\mathcal{D}$; $h$—level of mass distribution; $t$—number of mass distributions.
Output: Task-specific model in mass space.
$\widetilde{\textbf{mass}}(\cdot) \leftarrow$ Mass_Estimation$(D, \psi, h, t)$;
$D' \leftarrow$ Mass_Mapping$(D, \widetilde{\textbf{mass}})$;
Perform task (information retrieval or regression) in the mapped mass space using $D'$;
```

Algorithm 6 for Anomaly Detection: `MassAD`(D, ψ, h, t)

```
Inputs: $D$—input data; $\psi$—data size for $\mathcal{D}$; $h$—level of mass distribution; $t$—number of mass distributions.
Output: Ranked instances in $D$.
$\widetilde{\textbf{mass}}(\cdot) \leftarrow$ Mass_Estimation$(D, \psi, h, t)$;
for $i = 1$ to $|D|$ do
    $\mathsf{M}_i \leftarrow$ Average of $t$ masses from $\widetilde{\textbf{mass}}(\mathbf{x}_i)$;
end for
Rank instances in $D$ based on $\mathsf{M}_i$, where low mass denotes anomalies and high mass denotes normal points;
```

to each of the t mappings. The first randomisation occurs at step 2 in Algorithm 3 in selecting a random subset of data. Additional randomisation is applied to attribute selection at step 3 in Algorithm 3 for one-dimensional mass estimation, or at step 4 in Algorithm 2 for multi-dimensional mass estimation.

6 Experiments

We evaluate the performance of `MassSpace` and `MassAD` for three tasks in the following three subsections. We denote an algorithm `A` using one-dimensional and multi-dimensional mass estimations as `A`$'$ and `A`$''$, respectively.

In information retrieval and regression tasks, the mass estimation uses $\psi = 8$ and $t = 1000$. These settings are obtained by examining the rank correlation as shown in Fig. 4(b)—having a high rank correlation between $mass(x, 1)$ and $mass(x, 1|\mathcal{D})$. Note that this is done before any method is applied, and no further tuning of the parameters is carried out after this step. In anomaly detection tasks, $\psi = 256$ and $t = 100$ are used so that they are comparable to those used in a benchmark method for a fair comparison. In all tasks, $h = 1$ is used for one-dimensional mass estimation, and it cannot afford to use a high h because of its high cost $O(\psi^h)$. $h = \psi$ is used for multi-dimensional mass estimation in order to reduce one parameter setting.

All the experiments were run in Matlab and conducted on a Xeon processor which ran at 2.66 GHz and with 48 GB memory. The performance of each method was measured in terms of task-specific performance measure and runtime. Paired t-tests at 5 % significance level were conducted to examine whether the difference in performance is significant between two algorithms under comparison.

Note that we treated information retrieval and anomaly detection as unsupervised learning tasks. Classes/labels in the original data were used as ground truth for evaluation of performance only; they were not used in building mass distributions. In regression, only the training set was used to build mass distributions in step 1 of Algorithm 5; the mapping in step 2 was conducted for both the training and testing sets.

6.1 Content-based image retrieval

We use a Content-Based Image Retrieval (CBIR) task as an example of information retrieval. The `MassSpace` approach is compared with three state-of-the-art CBIR methods that deal with relevance feedbacks: a manifold based method `MRBIR` (He et al. 2004), and two recent techniques for improving similarity calculation, i.e., `Qsim` (Zhou and Dai 2006) and `InstR` (Giacinto and Roli 2005); and we employ the Euclidean distance to measure the similarity between instances in these two methods. The default parameter settings are used for all these methods.

Our experiments were conducted using the COREL image database (Zhou et al. 2006) of 10000 images, which contains 100 categories and each category has 100 images. Each image is represented by a 67-dimensional feature vector, which consists of 11 shape, 24 texture and 32 color features. To test the performance, we randomly selected 5 images from each category to serve as the queries. For a query, the images within the same category were regarded as relevant and the rest were irrelevant. For each query, we continued to perform up to 5 rounds of relevance feedback. In each round, 2 positive and 2 negative feedbacks were provided. This relevance feedback process was also repeated 5 times with 5 different series of feedbacks. Finally, the average results with one query and in different feedback rounds were recorded. The retrieval performance was measured in terms of Break-Even-Point (BEP) (Zhou and Dai 2006; Zhou et al. 2006) of the precision-recall curve. The online processing time reported is the time required in each method for a query plus the stated number of feedback rounds. The reported result is an average over 5 × 100 runs for query only; and an average over 5 × 100 × 5 runs for query plus feedbacks. The offline costs of constructing the one-dimensional mass estimation and the mapping of 10000 images were 0.27 and 0.32 seconds, respectively. The multi-dimensional mass estimation and the corresponding mapping took 1.72 and 5.74 seconds, respectively.

The results are presented in Table 2 where the retrieval performance better than that conducted in the original space at each round has been boldfaced. The results are grouped for ease of comparison.

The BEP results clearly show that the `MassSpace` approach achieves a better retrieval performance than that using the original space in all three methods `MRBIR`, `Qsim` and `InstR`, for one query and all rounds of relevance feedbacks. Paired t-tests with 5 % significance level also indicate that the `MassSpace` approach significantly outperforms each of the three methods in all experiments, without exception. These results show that the mass space provides useful additional information that is hidden in the original space.

The results also show that the multi-dimensional mass estimation provides better information than the one-dimensional mass estimation—`MRBIR`$''$, `Qsim`$''$ and `InstR`$''$ give better retrieval performance than `MRBIR`$'$, `Qsim`$'$ and `InstR`$'$, respectively; only some exceptions occur in the higher feedback rounds for `InstR`$'$, with minor differences.

The processing time for the `MassSpace` approach is expected to be longer than each of the three methods because the number of dimensions in the mass space is significantly higher than those in the original space, where $t = 1000$ and $u = 67$. Despite that, Table 3 shows that `MRBIR`$''$, `MRBIR`$'$ and `MRBIR` all have similar level of runtime.

Figure 8 shows an example of performance for `InstR`$'$—BEP increases as t increases until it reaches a plateau at some t value; and the processing time is linear w.r.t. the number of dimensions of the mass space, t.

Table 2 CBIR results (in BEP × 10^{-2}). An algorithm `A` using one-dimensional and multi-dimensional mass estimations are denoted as `A`$'$ and `A`$''$, respectively. Note that a high BEP is better than a low BEP

	`MRBIR`$''$	`MRBIR`$'$	`MRBIR`	`Qsim`$''$	`Qsim`$'$	`Qsim`	`InstR`$''$	`InstR`$'$	`InstR`
One query	**12.65**	**10.70**	9.69	**12.38**	**10.35**	7.78	**12.38**	**10.35**	7.78
Round 1	**16.58**	**14.24**	12.72	**19.18**	**15.46**	10.59	**13.88**	**13.33**	9.40
Round 2	**18.41**	**16.05**	13.90	**21.98**	**17.58**	11.81	**15.12**	**14.95**	9.99
Round 3	**19.69**	**17.34**	14.75	**23.67**	**18.71**	12.59	**16.19**	**16.07**	10.36
Round 4	**20.48**	**18.20**	15.33	**24.65**	**19.50**	13.16	**16.88**	**16.93**	10.78
Round 5	**21.15**	**19.86**	15.71	**25.42**	**19.96**	13.55	**17.49**	**17.58**	11.05

Table 3 CBIR results (online time cost in seconds)

	`MRBIR`$''$	`MRBIR`$'$	`MRBIR`	`Qsim`$''$	`Qsim`$'$	`Qsim`	`InstR`$''$	`InstR`$'$	`InstR`
One query	0.714	0.785	0.364	0.715	0.822	0.093	0.715	0.822	0.093
Round 1	0.762	0.893	0.696	0.207	0.208	0.035	0.197	0.198	0.026
Round 2	0.763	0.893	0.696	0.228	0.231	0.058	0.200	0.200	0.028
Round 3	0.763	0.893	0.696	0.257	0.259	0.086	0.200	0.200	0.028
Round 4	0.764	0.893	0.696	0.291	0.294	0.122	0.200	0.200	0.028
Round 5	0.764	0.893	0.697	0.335	0.341	0.167	0.200	0.200	0.028

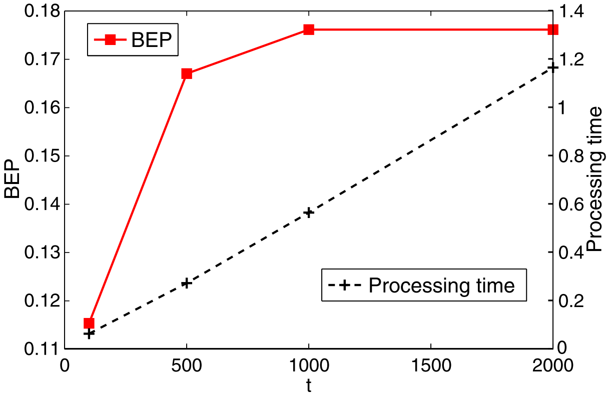


Fig. 8 An example of CBIR round 5 result: The retrieval performance and the processing time as t increases for `InstR`$'$

6.2 Regression

In this experiment, we compare support vector regression (Vapnik 2000) that employs the original space (`SVR`) with that employs the mapped mass space (`SVR`$''$ and `SVR`$'$). `SVR` is the ϵ-SVR algorithm with RBF kernel, implemented by LIBSVM (Chang and Lin 2001). `SVR` is chosen here because it is one of the top performing models.

We utilize five benchmark data sets including four selected from UCI repository (Asuncion and Newman 2007) and one earthquake data (Simonoff 1996) from www.cs.waikato.ac.nz/ml/weka/distribution. The data sizes are shown in the second column of Table 4. We only considered data sets with more than 1000 data points, that contained only real-valued at-

Table 4 Regression results (the smaller the better for MSE)

	Data size	MSE ($\times 10^{-2}$)			W/D/L	
		SVR″	SVR′	SVR	SVR″	SVR′
tic	9822	**5.56**	**5.58**	5.62	18/0/2	17/0/3
wine_white	4898	**1.08**	**1.21**	1.36	20/0/0	20/0/0
quake	2178	**2.87**	**2.86**	2.92	17/0/3	18/0/2
wine_red	1599	**1.50**	1.62	1.62	19/0/1	11/0/9
concrete	1030	**0.28**	**0.33**	0.57	20/0/0	20/0/0

Table 5 Regression results (time in seconds)

	#Dimension	Processing time			Factor increase		
		SVR″	SVR′	SVR	time(SVR″)	time(SVR′)	#dimension
tic	85	23.4	26.6	**11.9**	2.0	2.2	12
wine_white	11	8.2	9.2	**4.2**	2.0	2.2	91
quake	3	2.5	3.4	**1.0**	2.5	3.4	333
wine_red	11	1.7	2.6	**1.0**	1.6	2.5	91
concrete	8	1.2	2.3	**0.9**	1.3	2.6	125

tributes and that had no missing values. we did this in order to get a result with a higher confidence than those obtained from small data sets.

In each data set, we randomly sampled two-thirds of the instances for training and the remaining one-third for testing. This was repeated 20 times and we report the average result of these 20 runs. The data set, whether in the original space or the mass space, was min-max normalized before an ϵ-SVR model was trained. To select optimal parameters for the ϵ-SVR algorithm, we conducted a 5-fold cross validation based on mean squared error using the training set only. The kernel parameter γ was searched in the range $\{2^{-15}, 2^{-13}, 2^{-11}, \dots, 2^3, 2^5\}$; the regularization parameter C in the range $\{0.1, 1, 10\}$, and ϵ in the range $\{0.01, 0.05, 0.1\}$. We measured regression performance in terms of mean squared error (MSE) and runtime in seconds. The runtime reported is the runtime for SVR only. The total cost of mass estimation (from the training set) and mapping (of training and testing sets) in the largest data set, tic, was 1.8 seconds for one-dimensional mass estimation, and 8.5 seconds for multi-dimensional mass estimation. The cost of normalisation and the parameter search using 5-fold cross-validation was not included in the reported result for all SVR″, SVR′ and SVR.

The result is presented in Table 4. SVR′ performs significantly better than SVR in all data sets in MSE measure; the only exception is in the wine_red data set. SVR″ performs significantly better than SVR in all data sets, without exceptions. SVR″ generally performs better than SVR′.

Although both SVR″ and SVR′ take more time to run because each of them runs on the data with a significantly higher dimension, yet the factor of increase in time (shown in the last three columns of Table 5) ranges from 1.3 to 3.4 only, when the factor of increase in the number of dimensions ranges from 12 to over 300. This is because the time complexity in the key optimisation process in SVR is not dependent on the number of dimensions.

Table 6 Data characteristics of the data sets in anomaly detection tasks. The percentage in brackets indicates the percentage of anomalies

	Data size	#Dimension	Anomaly class
Http	567497	3	Attack (0.4 %)
Forest	286048	10	Class 4 (0.9 %) vs class 2
Mulcross	262144	4	2 clusters (10 %)
Smtp	95156	3	Attack (0.03 %)
Shuttle	49097	9	Classes 2, 3, 5, 6, 7 (7 %) vs class 1
Mammography	11183	6	Class 1 (2 %)
Annthyroid	7200	6	Classes 1, 2 (7 %)
Satellite	6435	36	3 Smallest classes (32 %)

6.3 Anomaly detection

This experiment compares MassAD with four state-of-the-art anomaly detectors: isolation forest or iForest (Liu et al. 2008), a distance-based method ORCA (Bay and Schwabacher 2003), a density-based method LOF (Breunig et al. 2000), and one-class support vector machine (or 1-SVM) (Schölkopf et al. 2000). MassAD was built with $t = 100$ and $\psi = 256$, the same default settings as used in iForest (Liu et al. 2008), which also employed a multi-model approach. The parameter settings employed for ORCA, LOF and 1-SVM were as stated by Liu et al. (2008).

All the methods were tested on the eight largest data sets used by Liu et al. (2008). The data characteristics are summarized in Table 6, which include one anomaly data generator Mulcross (Rocke and Woodruff 1996) and the other seven are from UCI repository (Asuncion and Newman 2007). The performance was evaluated in terms of averaged AUC (area under ROC curve) and processing time (a total of training time and testing time) over ten runs (following Liu et al. 2008).

MassAD and iForest were implemented in Matlab and tested on a Xeon processor ran at 2.66 GHz. LOF was written in Java in ELKI platform version 0.4 (Achtert et al. 2008); and ORCA was written in C++ (www.stephenbay.net/orca/). The results for ORCA, LOF and 1-SVM were conducted using the same experimental setting but on a slightly slower 2.3 GHz machine, the same machine used by Liu et al. (2008).

The AUC values of all the compared methods are presented in Table 7 where the figures boldfaced are the best performance for each data set. The results show that MassAD using the multi-dimensional mass estimation achieves the best performance in four data sets, and close to the best (the difference which is less than 0.03 AUC) in two data sets; MassAD using the one-dimensional mass estimation achieves the best performance in three data sets, and close to the best in one data set. iForest performs best in four data sets. The results are close for these three algorithms because they share many similarities (see Sect. 9 for details).

Again, the multi-dimensional version of MassAD generally performs better than the one-dimensional version, with five wins, one draw and two losses. Most importantly, the worst performance in the Mulcross data set can be easily 'corrected' using a better parameter

Table 7 AUC values for anomaly detection

	MassAD		iForest	ORCA	LOF	1-SVM
	Mass″	Mass′				
Http	**1.00**	**1.00**	**1.00**	0.36	0.44	0.90
Forest	0.90	**0.92**	0.87	0.83	0.56	0.90
Mulcross	0.26	**0.99**	0.96	0.33	0.59	0.59
Smtp	**0.91**	0.86	0.88	0.87	0.32	0.78
Shuttle	**1.00**	0.99	**1.00**	0.55	0.55	0.79
Mammography	0.86	0.37	**0.87**	0.77	0.71	0.65
Annthyroid	0.75	0.71	**0.82**	0.68	0.72	0.63
Satellite	**0.77**	0.62	0.71	0.65	0.52	0.61

Table 8 Runtime (second) for anomaly detection

	MassAD		iForest	ORCA	LOF	1-SVM
	Mass″	Mass′				
Http	168	18	74	9487	18913	35872
Forest	63	10	39	6995	10853	9738
Mulcross	52	10	38	2512	5432	7343
Smtp	27	4	13	267	540	987
Shuttle	20	3	8	157	368	333
Mammography	21	1	3	4	39	11
Annthyroid	7	1	3	2	9	4
Satellite	13	1	3	9	10	9

setting—by using $\psi = 8$, instead of 256, the multi-dimensional version of MassAD improves its detection performance from 0.26 to 1.00 in terms of AUC.[4]

It is also noteworthy that the multi-dimensional MassAD significantly outperforms the traditional density-based, distance-based and SVM anomaly detectors in all data sets, except two: one in Annthyroid when compared to ORCA; the poor performance in Mulcross was discussed earlier. The above observations validate the effectiveness of our proposed mass estimation on anomaly detection tasks.

Table 8 shows the runtime result. Although MassAD was run on a slightly faster machine, the result still shows that it has a significant advantage in term of processing time over ORCA, LOF and 1-SVM. The comparison with iForest is presented in Table 9 with a breakdown of training time and testing time. Note that one-dimensional MassAD took the same time as iForest in training, but it only took about one-tenth of the time required by iForest in testing. On the other hand, the multi-dimensional MassAD took slightly more time than iForest in training, but it took up to three times the time required by iForest in testing.

The time and space complexities for five anomaly detection methods are given in Table 10. The one-dimensional MassAD and iForest have the best time and space complexities due to their ability to use small $\psi \ll n$ and $h = 1$. Note that the one-dimensional MassAD ($h = 1$) is faster by a factor of $\log(\psi = 256) = 8$ which shows up in the testing time—ten times faster than iForest given in Table 9. The training time disadvan-

[4] Mulcross produces anomaly clusters rather than scattered anomalies. Detecting anomaly clusters are more effective using a low ψ setting when the multi-dimensional version of MassAD is employed.

Table 9 Training time and testing time (second) for MassAD and iForest, using $t = 100$ and $\psi = 256$

	Training time			Testing time		
	MassAD		iForest	MassAD		iForest
	Mass$''$	Mass$'$		Mass$''$	Mass$'$	
Http	16.2	14.3	14.4	151.8	3.3	59.6
Forest	10.3	8.2	8.6	53.1	2.0	30.8
Mulcross	9.1	7.9	8.1	42.8	2.1	29.4
Smtp	5.4	3.9	3.5	21.9	0.6	9.9
Shuttle	6.1	3.1	2.8	14.1	0.3	5.6
Mammography	8.4	1.3	1.2	12.8	0.1	1.8
Annthyroid	3.1	1.3	1.1	3.4	0.1	1.5
Satellite	6.6	1.2	1.6	5.9	0.0	1.9

Table 10 A comparison of time and space complexities. The time complexity includes both training and testing. n is the given data set size and u is the number of dimensions. For MassAD and iForest, the first part of the summation is the training time and the second the testing time

	Time complexity	Space complexity
MassAD (multi-dimensional)	$O(t(\psi + n)h)$	$O(t\psi h)$
MassAD (one-dimensional)	$O(t(\psi^{h+1} + n))$	$O(t\psi)$
iForest	$O(t(\psi + n) \cdot log(\psi))$	$O(t\psi \cdot log(\psi))$
ORCA	$O(un \cdot log(n))$	$O(un)$
LOF	$O(un^2)$	$O(un)$

tage, compared to iForest, did not show up because of small ψ. The one-dimensional MassAD also has an advantage over iForest in space complexity by a factor of $\log(\psi)$. The multi-dimensional MassAD has similar order of worst-case time and space complexities as iForest, though it might have a larger constant.

In contrast to ORCA and LOF (distance-based and density-based methods), the time complexity (and the space complexity) for both MassAD and iForest are independent of the number of dimension u.

6.4 Constant time and space complexities

In this section, we show that $mass(x, h|\mathcal{D})$ (in step 4 of Algorithm 3) takes only constant time, regardless of the given data size n, when the algorithmic parameters are fixed. Table 11 reports the runtime time for sampling (to get a random sample of size ψ from the given data set—steps 2 and 3 of Algorithm 3) and the runtime for one-dimensional mass estimation—to construct $mass(x, h|\mathcal{D})$ t times, for five data sets which include the largest and smallest data sets in regression and anomaly detection tasks.

The results show that the sampling time increased linearly with the size of the given data set, and it took significantly longer (in the largest data set) than the time to construct the mass distribution—which was constant, regardless of the given data size. Note that the training time provided in Table 9 includes both the sampling time and mass estimation time, and it is dominated by the sampling time for large data sets.

The memory required for each construction of $mass(x, h|\mathcal{D})$ is to store one lookup table of size ψ which is constant.

The constant time and space complexities apply to multi-dimensional mass estimation too.

Table 11 Runtime (second) for sampling, $mass(x, 1|\mathcal{D})$ and $mass(x, 3|\mathcal{D})$, where $t = 1000$ and $\psi = 8$

	Data size	Sampling	$mass(x, 1\|\mathcal{D})$	$mass(x, 3\|\mathcal{D})$
Http	567497	138.30	0.33	10.96
Shuttle	49097	16.16	0.39	10.97
COREL	10000	1.23	0.27	11.03
tic	9822	1.09	0.43	11.14
concrete	1030	0.18	0.31	10.95

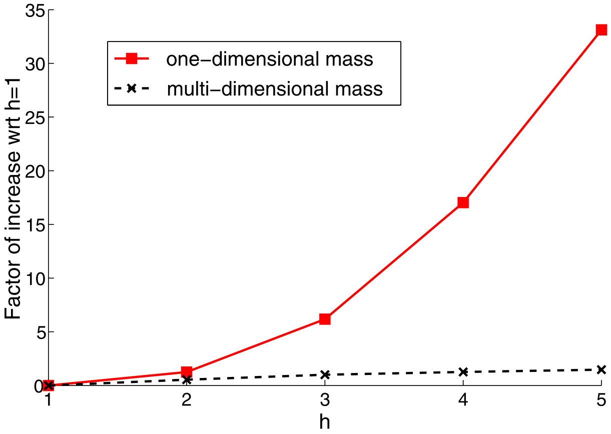


Fig. 9 Runtime comparison: One-dimensional mass estimation versus multi-dimensional mass estimation for different values of h in the COREL data set, where both are using $\psi = 8$ and $t = 1000$. In this experiment, we set h to the required value for multi-dimensional mass estimation, rather than $h = \psi$ which was used in all experiments reported in the previous sections

6.5 Runtime comparison between one-dimensional and multi-dimensional mass estimations

In terms of runtime, the comparison so far in the experiments might give the impression that multi-dimensional mass estimation is worse than one-dimensional mass estimation. In fact, the opposite is true because the above results are obtained from $h = 1$ for one-dimensional mass estimation and $h = \psi$ for multi-dimensional mass estimation. Figure 9 shows the head-to-head comparison for different h values in the COREL data set. When h increases from 1 to 5, the runtime for the one-dimensional mass estimation increases by a factor of 33. In contrast, the runtime for the multi-dimensional mass estimation increases by a factor of 1.5 only.

6.6 Summary

The above results in all three tasks show that the orderings provided by mass distributions deliver additional information about the data that would otherwise hidden in the original features. The additional information, which accentuates fringe points with a concave function (or an approximation to a concave function in the case of multi-dimensional mass estimation), improves the task-specific performance significantly, especially in the information retrieval and regression tasks.

Using Algorithm 5 for the information retrieval and regression tasks, the runtime is expected to be higher because the new space has much higher dimensions than the original space ($t \gg u$). It shall be noted that the runtime increase (linearly or worse) is solely a characteristic of the existing algorithms used, and is not due to the mass space mapping which has constant time and space complexities.

Table 12 A comparison of kernel density estimation and mass estimation. Kernel density estimation requires two parameter settings: kernel function $K(\cdot)$ and bandwidth h_w; mass estimation has one: h

Kernel density$(x) = \frac{1}{nh_w} \sum_{i=1}^{n} K(\frac{x-x_i}{h_w})$
$mass(x,h) = \begin{cases} \sum_{i=1}^{n-1} mass_i(x,h\text{-}1)p(s_i), & h > 1 \\ \sum_{i=1}^{n-1} m_i(x)p(s_i), & h = 1 \end{cases}$

We believe that a more tailored approach that better integrates the information provided by mass (into the **C3** component in the formalism) for a specific task can potentially further improve the current level of performance in terms of either task-specific performance measure or runtime. We have demonstrated this 'direct' application using Algorithm 6 for the anomaly detection task, in which `MassAD` performs equally well or significantly better than four state-of-the-art methods in terms of task-specific performance measure, and the one-dimensional mass estimation executes faster than all other methods in terms of runtime.

Why does one-dimensional mapping work when tackling multi-dimensional problems? We conjecture that if there is no or little interaction between features, then the one-dimensional mapping will work because the ordering that accentuates the fringe points for each original dimension making it easy for existing algorithms to exploit. When there are strong interactions between features, then one-dimensional mapping might not achieve good results. Indeed, our results in all three tasks show that multi-dimensional mass estimation does perform better than one-dimensional mass estimation in general, in terms of task-specific performance measures.

The ensemble method for mass estimation usually needs only a small sample to build each model in an ensemble. In addition, in order to build all t models for an ensemble, $t\psi$ could be more than n when $\psi > n/t$.

The key limitation of the one-dimensional mass estimation is its high cost when a high value of h is applied. This can be avoided by implementing it using a tree structure rather than a lookup table, as we have done using Half-Space Trees which reduces the time complexity to $O(th(\psi + n))$ from $O(t(\psi^{h+1} + n))$.

7 Relation to kernel density estimation

A comparison of mass estimation and kernel density estimation is provided in Table 12.

Like kernel estimation, mass estimation at each point is computed through a summation of a series of values from a mass base function $m_i(\cdot)$, equivalent to a kernel function $K(\cdot)$. The two methods differ in the following ways:

- *Aim*: Kernel estimation is aimed to do probability density estimation; whereas mass estimation is to estimate an order from the core points to the fringe points.
- *Kernel function*: While kernel estimation can use different kernel functions for probability density estimation; we doubt that mass estimation requires a different base function for two reasons. First, a more sophisticated function is unlikely to provide a better ordering than a simple rectangular function. Second, the rectangular function keeps the computation simple and fast. In addition, a kernel function must be fixed (i.e., having user-defined values for its parameters); e.g., the rectangular kernel function has fixed width or fixed per unit size. But the rectangular function used in mass has no parameter and no fixed width.
- *Sample size*: Kernel estimation or other density estimation methods require a large sample size in order to estimate the probability accurately (Duda et al. 2001). Mass estimation

Table 13 CBIR results (in BEP × 10^{-2})

(a) Compare with $\texttt{Qsim}^K$ (using kernel density estimation), $\texttt{Qsim}^D$ (using data depth), $\texttt{Qsim}^{LD}$ (using local data depth)

	$\texttt{Qsim}''$	$\texttt{Qsim}'$	$\texttt{Qsim}^K$	$\texttt{Qsim}^D$	$\texttt{Qsim}^{LD}$	Qsim
One query	**12.38**	10.35	2.90	10.39	7.60	7.78
Round 1	**19.18**	15.46	3.01	15.02	10.95	10.59
Round 2	**21.98**	17.58	2.74	17.16	12.50	11.81
Round 3	**23.67**	18.71	2.54	18.37	13.42	12.59
Round 4	**24.65**	19.50	2.42	19.20	14.03	13.16
Round 5	**25.42**	19.96	2.34	19.74	14.36	13.55

(b) Compare with $\texttt{InstR}^K$, $\texttt{InstR}^D$ and $\texttt{InstR}^{LD}$

	$\texttt{InstR}''$	$\texttt{InstR}'$	$\texttt{InstR}^K$	$\texttt{InstR}^D$	$\texttt{InstR}^{LD}$	InstR
One query	**12.38**	10.35	2.90	10.39	7.60	7.78
Round 1	**13.88**	13.33	2.91	13.05	8.71	9.40
Round 2	**15.12**	14.95	2.55	14.73	9.68	9.99
Round 3	**16.19**	16.07	2.25	15.98	10.28	10.36
Round 4	16.88	**16.93**	2.06	16.82	10.78	10.78
Round 5	17.49	**17.58**	1.99	17.50	11.17	11.05

using $mass(x, h|\mathcal{D})$ needs only a small sample size in an ensemble to accurately estimate the ordering.

Here we present the results using a Gaussian kernel density estimation, replacing the one-dimensional mass estimation, using the same subsample size in an ensemble approach. The bandwidth parameter is set to be the standard deviation of the subsample; and all the other parameters are the same.

The results for information retrieval and anomaly detection are provided in Tables 13 and 15. Compared to mass, density performed significantly worse in information retrieval tasks in all experiments using Qsim and InstR, denoted as $\texttt{Qsim}^K$ and $\texttt{InstR}^K$, respectively. They were even worse than those run in the original space. In anomaly detection, DensityAD, which used a Gaussian kernel density estimation, performed significantly worse than MassAD in six out of eight data sets in the anomaly detection tasks, and better in the other two data sets.

8 Relation to data depth

There is a close relationship between the proposed mass and data depth (Liu et al. 1999): they both delineate the centrality of a data cloud (as opposed to compactness in the case of the density measure). The properties common to both measures are: (a) the centre of a data cloud has the maximum value of the measure; (b) an ordering from the centre (having the maximum value) to the fringe points (having the minimum values).

However, there are two key differences. First, not until recently (see Agostinelli and Romanazzi 2011) data depth always models a given data with one centre, regardless whether

Table 14 CBIR results (online time cost in seconds)

(a) Compare with $\mathtt{Qsim}^K$, $\mathtt{Qsim}^D$, $\mathtt{Qsim}^{LD}$

	$\mathtt{Qsim}''$	$\mathtt{Qsim}'$	$\mathtt{Qsim}^K$	$\mathtt{Qsim}^D$	$\mathtt{Qsim}^{LD}$	$\mathtt{Qsim}$
One query	0.715	0.822	0.820	0.840	0.829	0.093
Round 1	0.207	0.208	0.224	0.237	0.226	0.035
Round 2	0.228	0.231	0.279	0.288	0.276	0.058
Round 3	0.257	0.259	0.348	0.355	0.343	0.086
Round 4	0.291	0.294	0.435	0.438	0.425	0.122
Round 5	0.335	0.341	0.547	0.543	0.531	0.167

(b) Compare with $\mathtt{InstR}^K$, $\mathtt{InstR}^D$ and $\mathtt{InstR}^{LD}$

	$\mathtt{InstR}''$	$\mathtt{InstR}'$	$\mathtt{InstR}^K$	$\mathtt{InstR}^D$	$\mathtt{InstR}^{LD}$	$\mathtt{InstR}$
One query	0.715	0.822	0.820	0.840	0.829	0.093
Round 1	0.197	0.198	0.203	0.215	0.206	0.026
Round 2	0.200	0.200	0.205	0.216	0.206	0.028
Round 3	0.200	0.200	0.206	0.217	0.207	0.028
Round 4	0.200	0.200	0.207	0.218	0.208	0.028
Round 5	0.200	0.200	0.207	0.218	0.208	0.028

Table 15 Anomaly detection: `MassAD` vs `DensityAD` and `DepthAD` (AUC)

	`MassAD`		`DensityAD`	`DepthAD`	
	$\mathtt{Mass}''$	$\mathtt{Mass}'$		`Depth`	`LDepth`
Http	**1.00**	**1.00**	0.99	0.98	0.52
Forest	0.90	**0.92**	0.70	0.85	0.49
Mulcross	0.26	0.99	**1.00**	0.99	0.93
Smtp	0.91	0.86	0.59	0.92	**0.93**
Shuttle	**1.00**	0.99	0.90	0.87	0.72
Mammography	**0.86**	0.37	0.27	0.36	0.79
Annthyroid	0.75	0.71	0.80	0.58	**0.86**
Satellite	**0.77**	0.62	0.61	0.59	0.69

the data is unimodal or multi-modal; whereas mass can model both unimodal and multi-modal data by setting $h = 1$ or $h > 1$. Local data depth (Agostinelli and Romanazzi 2011) has a parameter (τ) which allows it to model multi-modal data as well as unimodal data. However, the performance of local data depth appears to be sensitive to the setting of τ (see a discussion of the comparison below). In contrast, a single setting of h in mass estimation had produced good task-specific performance in three different tasks in our experiments.

Second, mass is a simple and straightforward measure, and has efficient estimation methods based on axis-parallel partitions only. Data depth has many different definitions, depending on the construct used to define depth. The constructs could be Mahalanobis, Convex Hull, simplicial, halfspace and so on (Liu et al. 1999), all of which are expensive to compute (Aloupis 2006)—this has been the main obstacle in applying data depth to real applications in multi-dimensional problems. For example, Ruts and Rousseeuw (1996) compute the contour of data depth of a data cloud for visualization, and employ depth as the anomaly score to identify anomalies. Because of its computational cost, it is limited to small data size only. In contrast to the axis-parallel partitions used in mass estimation, halfspace data depth[5] (Tukey 1975), for example, requires to consider all halfspaces which demands high computational time and space.

To provide a comparison, we replace the one-dimensional mass estimation (defined in Algorithm 3) with data depth (defined by simplicial depth Liu et al. 1999) and local data depth (defined by simplicial local depth Agostinelli and Romanazzi 2011). We repeat the experiments by employing both the data depth and local data depth implementation in R by Agostinelli and Romanazzi (2011) (accessible from r-forge.r-project.org/projects/localdepth). Both data depths are carried out in the same approach by using sample size ψ to build each of the t models in an ensemble.[6] The number of simplices used to do the empirical estimation is set to 10000 for all runs. Default settings are used for all other parameters (i.e., the membership of a data point in simplices is evaluated in the "exact" mode rather than the approximate mode, and the tolerance parameter is fixed to 10^{-9}). Note that local depth uses an additional parameter τ to select candidate simplices, where a simplex having volume larger than τ is excluded from consideration. As the performance of local depth is sensitive to τ, we employ the quantile order of τ of 10 %, the low value of the range 10 %–30 % suggested by Agostinelli and Romanazzi (2011). Because both data depth and local data depth are estimated using the same procedure, their runtimes are the same.

The task-specific performance result for information retrieval is provided in Table 13. Note that local data depth could produce worse retrieval results than those in the original feature space. Data depth performed close to that achieved by the one-dimensional mass estimation, but it was significantly worse than the multi-dimensional mass estimation.

Figure 10 shows a scale up test in the information retrieval task using `Qsim` with one query and feedback round 5. It is interesting to note both mass and data depth performed better using small rather than large subsampling size. As expected, KDE produced better results with increasing subsampling sizes; but even with $\psi = 8196$ in the COREL data set of 10000 instances, KDE still performed the worst compared to mass and data depth.

Table 15 shows the result in anomaly detection. Data depth performed worse than both versions of mass estimation in six out of eight data sets; local data depth performed worse than multi-dimensional mass estimation in five out of eight data sets; local data depth versus one-dimensional mass estimation have four wins and four losses. Note that though local data depth achieved the best result in two data sets, it also produced the worst in three data sets which were significantly worse than others (in http, forest and shuttle).

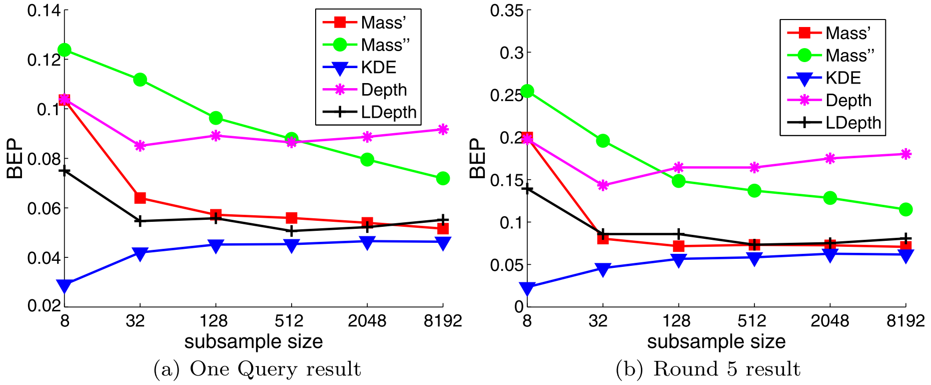
The runtime results are provided in Tables 14 and 16. These results do not reveal the time complexities of the algorithms because of small ψ (and the CBIR results do not include the offline time cost). We conducted a scale up test using the Mulcross data set by increasing

[5] Zuo and Serfling (2000) define halfspace data depth (HD) of a point x in $\mathcal{R}^u$ w.r.t. a probability measure P on $\mathcal{R}^u$ as the minimum probability mass carried by any closed halfspace containing x:

$$HD(x; P) = inf\{P(H) : H \text{ a closed halfspace}, x \in H\}, \quad x \in \mathcal{R}^u$$

In the language of data depth, the one-dimensional mass estimation may be interpreted as a kind of average probability mass of halfspaces containing x, weighted by mass covered by halfspace. But the one-dimensional mass estimation defined in (1) allows mass to be computed by a summation of $n - 1$ components from the given data set of size n, whereas data depth does not. In addition, our implementation of multi-dimensional mass estimation using a tree structure with axis-parallel splits cannot be interpreted using any of the constructs employed by data depth.

[6] Our experiments indicate that using the entire data set to estimate data depth or local data depth produces worse results than those using an ensemble approach. This result is shown in Appendix.



(a) One Query result

(b) Round 5 result

Fig. 10 Scale up test in information retrieval using `Qsim`. Subsampling data size is increased from $\psi = 8$ to $\psi = 8196$ in the COREL image data set containing 10000 instances. The same experimental setting as reported in Sect. 6.1 is used

Table 16 Anomaly detection: `MassAD` vs `DensityAD` and `DepthAD` (time in seconds)

	`MassAD`		`DensityAD`	`DepthAD`	
	`Mass''`	`Mass'`		`Depth`	`LDepth`
Http	168	18	17	38	38
Forest	63	10	10	31	31
Mulcross	52	10	10	31	31
Smtp	27	10	10	26	26
Shuttle	20	4	4	25	25
Mammography	21	3	3	24	24
Annthyroid	7	1	1	23	23
Satellite	13	1	1	23	23

the subsampling size. Using the runtime at $\psi = 8$ as the base, runtime ratio is computed for all other subsampling sizes. The result is presented in Fig. 11. It shows that data depth or local data depth had the worst runtime ratio which increased its runtime 58 times when ψ was increased by a factor of 512. The multi-dimensional mass estimation had the best runtime ratio of 6.6, followed by KDE (24) and one-dimensional mass estimation (34) when ψ ratio = 512. The actual runtimes in seconds were 126.6 (Mass''), 166.7 (KDE), 239.4 (Mass'), and 600.5 (Data Depth). This result is not surprising because the multi-dimensional mass estimation has time complexity $O(\psi)$, KDE has $O(\psi^2)$, the one-dimensional mass estimation has $O(\psi^{h+1})$, and data depth using simplices has $O(\psi^4)$ (Aloupis 2006).

9 Other work based on mass

`iForest` (Liu et al. 2008) and `MassAD` share some common features: Both are ensemble methods which build t models, each from a random sample of size ψ, and they both combine the outputs of the models through averaging during testing. Although `iForest` (Liu et al. 2008) employs path length—an instance traverses from the root of a tree to its leaf—as the anomaly score, we have shown that the path length used in `iForest` is in

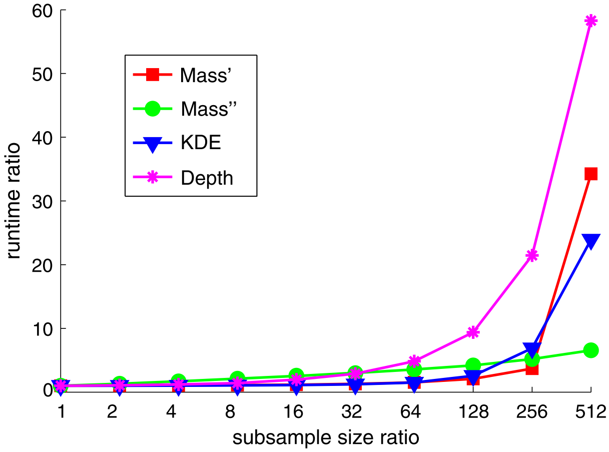


Fig. 11 Scale up test using the Mulcross data set. The base subsampling data size (ψ) is 8; doubling at each step until $\psi = 4096$. Each point in the graph is an average over 10 runs

fact a proxy to mass (see Sect. 4.1 for details). In other words, `iForest` is a kind of mass-based method—that is why `MassAD` and `iForest` have similar detection accuracy. Multi-dimensional `MassAD` has the closest resemblance to `iForest` because of the use of tree. The key difference is that `MassAD` is just one application of the more fundamental concept of mass introduced here, whereas `iForest` is for anomaly detection only. In terms of implementation, the key difference is how the cut-off value is selected at each internal node of a tree: `iForest` selects the cut-off value randomly whereas a Half-Space Tree selects a mid point deterministically (see step 5 in Algorithm 2).

How easily can the proposed formalism be applied to other tasks? In addition to the tasks we have applied in this paper, we have applied mass estimation 'directly', using the proposed formalism, to solve problems in content-based multimedia information retrieval (Zhou et al. 2012) and clustering (Ting and Wells 2010). While the 'indirect' application is straightforward which simply uses the existing algorithms in the mass space, a 'direct' application requires a complete rethink of the problem and produces a totally different algorithm. However, this rethink of a problem in terms of mass often results a more efficient and sometimes more effective algorithm than existing algorithms. We provide a brief description of the two applications in the following two paragraphs.

In addition to the mass-space mapping we have shown here (i.e., components **C1** and **C2**), Zhou et al. (2012) present a content-based information retrieval method that assigns a weight (based on `iForest`, thus, mass) to each new mapped feature w.r.t. a query; and then it ranks objects in the database according to their weighted average feature values in the mapped space. The method also incorporates relevance feedback which modifies the ranking based on the feedbacks through reweighted features in the mapped space. This method forms the third component of the formalism stated in Sect. 5. This 'direct' application of mass has been shown to be significantly better than the 'indirect' approach we have shown in Sect. 6.1, in terms of both task-specific measure and runtime (Zhou et al. 2012). It is interesting to note that, unlike existing retrieval systems which rely on a metric, the new mass-based method does not employ a metric—it is the first information retrieval system that does not use a metric, as far as we know.

Ting and Wells (2010) use a variant of Half-Space Trees we have employed here and apply mass directly to solve clustering problems. It is the first mass-based clustering algorithm, and it is unique because it does not use any distance and density measure. In this task, like in the case of anomaly detection, only two components are required. After building a mass model (in the **C1** component), the **C3** component consists of linking instances with non-zero mass connected by the mass model and making each group of connected instances a separate cluster; and all other unconnected instances are regarded as noise. This mass-based clustering algorithm has been shown to perform equally well as DBSCAN (Ester et al. 1996) in terms of clustering performance, but it runs orders of magnitude faster (Ting and Wells 2010).

The earlier version of this paper (Ting et al. 2010) establishes the properties of mass estimation in the one-dimensional setting only; and use it in all three tasks. This paper extends one-dimensional mass estimation to multi-dimensional mass estimation using the same approach as described by Ting and Wells (2010), and implements multi-dimensional mass estimation using Half-Space Trees. This paper reports new experiments using the multi-dimensional mass estimation, and shows the advantage of using multi-dimensional mass estimation over one-dimensional mass estimation in the three tasks reported earlier (Ting et al. 2010). These related works show that mass estimation can be implemented in different ways using tree-based or non-tree-based methods.

10 Conclusions and future work

This paper makes two key contributions. First, we introduce a base measure, mass, and delineate its three properties: (i) a mass distribution stipulates an ordering from core points to fringe points in a data cloud; (ii) this ordering accentuates the fringe points with a concave function—a property that can be easily exploited by existing algorithms to improve their task-specific performance; and (iii) the mass estimation methods have constant time and space complexities. Density estimation has been the base modelling mechanism employed in many techniques thus far. Mass estimation introduced here provides an alternative choice, and it is better suited for many tasks which require an ordering rather than probability density estimation.

Second, we present a mass-based formalism which forms a basis to apply mass to different tasks. The three tasks (i.e., information retrieval, regression and anomaly detection) to which we have successfully applied are just examples of its application. Mass estimation has potentials in many other applications.

There are potential extensions to the current work. First, the algorithms for the three tasks and the formalism can be improved or extended to include more tasks. Second, because the purposes and their properties differ, mass estimation is not intended to replace density estimation—it is thus important to identify areas in which each is best suited for. This will ascertain (i) areas in which density has been a mismatch, unbeknown up to now, and (ii) areas in which mass estimation is weak. Third, the proposed approach to multi-dimensional mass estimation is an approximation and it does not guarantee concavity. It will be interesting to explore a version that has such a guarantee and to examine whether it will further improve the task-specific performance in all three tasks reported here. Fourth, the current implementation of multi-dimensional mass estimation using Half-Space Trees limits its applications to low dimensional problems because it suffers the same problem as in all other grid oriented methods. We will explore non-grid oriented implementations of mass which have potential to tackle high dimensional problems more effectively than existing density-based and distance-based methods.

The Matlab source codes of both one-dimensional and multi-dimensional mass estimations are available at http://sourceforge.net/projects/mass-estimation/.

Acknowledgements This work is supported by the Air Force Research Laboratory, under agreement numbers FA2386-09-1-4014, FA2386-10-1-4052 and FA2386-11-1-4112. The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The anonymous reviewers have provided many helpful comments to improve the clarity of this paper.

Appendix: Anomaly detection using data depth that builds a single model from the entire data set

This appendix provides the results in anomaly detection task where data depth and local data depth built a single model from the entire data set, i.e., $\texttt{DepthAD}_s$. This is in contrast to `DepthAD` which employed an ensemble approach in Sect. 8.

Table 17 shows that `MassAD` generally has higher AUC than $\texttt{DepthAD}_s$ which employed either data depth or local data depth. The only exception is the Annthyroid data set. Note that these results are generally worse than those employing an ensemble approach, reported in Table 15.

Table 17 AUC values for anomaly detection, comparing `MassAD` with $\texttt{DepthAD}_s$ (which employed either data depth or local data depth) that build a single model from the entire data set

	`MassAD`		$\texttt{DepthAD}_s$	
	$\texttt{Mass}''$	$\texttt{Mass}'$	`Depth`	`LDepth`
Http	**1.00**	**1.00**	0.84	0.50
Forest	0.90	**0.92**	0.50	0.55
Mulcross	0.26	**0.99**	0.88	0.61
Smtp	**0.91**	0.86	0.86	0.76
Shuttle	**1.00**	0.99	0.51	0.70
Mammography	**0.86**	0.37	0.73	0.62
Annthyroid	0.75	0.71	0.59	**0.85**
Satellite	**0.77**	0.62	0.50	0.70

References

Achtert, E., Kriegel, H.-P., & Zimek, A. (2008). ELKI: a software system for evaluation of subspace clustering algorithms. In *Proceedings of the 20th international conference on scientific and statistical database management* (pp. 580–585).

Agostinelli, C., & Romanazzi, M. (2011). Local depth. *Journal of Statistical Planning and Inference*, *141*, 817–830.

Aloupis, G. (2006). Geometric measures of data depth. *DIMACS Series in Discrete Math and Theoretical Computer Science*, *72*, 147–158.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Bay, S. D., & Schwabacher, M. (2003). Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of ACM SIGKDD* (pp. 29–38).

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. In *Proceedings of ACM SIGKDD* (pp. 93–104).

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: a library for support vector machines.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of ACM SIGKDD* (pp. 226–231).

Giacinto, G., & Roli, F. (2005). Instance-based relevance feedback for image retrieval. In *Advances in NIPS* (pp. 489–496).

He, J., Li, M., Zhang, H., Tong, H., & Zhang, C. (2004). Manifold-ranking based image retrieval. In *Proceedings of ACM multimedia* (pp. 9–16).

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. In *Proceedings of IEEE ICDM* (pp. 413–422).

Liu, R., Parelius, J. M., & Singh, K. (1999). Multivariate analysis by data depth. *The Annals of Statistics*, *27*(3), 783–840.

Quinlan, J. (1993). *C4.5: programs for machine learning*. San Mateo: Morgan Kaufmann.

Rocke, D. M., & Woodruff, D. L. (1996). Identification of outliers in multivariate data. *Journal of the American Statistical Association*, *91*(435), 1047–1061.

Ruts, I., & Rousseeuw, P. (1996). Computing depth contours of bivariate point clouds. *Computational Statistics and Data Analysis*, *23*(1), 153–168.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in NIPS* (pp. 582–588).

Simonoff, J. S. (1996). *Smoothing methods in statistics*. Berlin: Springer.

Ting, K. M., & Wells, J. R. (2010). Multi-dimensional mass estimation and mass-based clustering. In *Proceedings of IEEE ICDM* (pp. 511–520).

Ting, K. M., Zhou, G.-T., Liu, F. T., & Tan, S. C. (2010). Mass estimation and its applications. In *Proceedings of ACM SIGKDD* (pp. 989–998).

Tukey, J. W. (1975). Mathematics and picturing data. In *Proceedings of the international congress on mathematics* (Vol. 2, pp. 525–531).

Vapnik, V. N. (2000). *The nature of statistical learning theory* (2nd ed.). Berlin: Springer.

Zhang, R., & Zhang, Z. (2006). BALAS: empirical Bayesian learning in the relevance feedback for image retrieval. *Image and Vision Computing*, *24*(3), 211–223.

Zhou, G.-T., Ting, K. M., Liu, F. T., & Yin, Y. (2012). Relevance feature mapping for content-based multimedia information retrieval. *Pattern Recognition*, *45*, 1707–1720.

Zhou, Z.-H., Chen, K.-J., & Dai, H.-B. (2006). Enhancing relevance feedback in image retrieval using unlabeled data. *ACM Transactions on Information Systems*, *24*(2), 219–244.

Zhou, Z.-H., & Dai, H.-B. (2006). Query-sensitive similarity measure for content-based image retrieval. In *Proceedings of IEEE ICDM* (pp. 1211–1215).

Zuo, Y., & Serfling, R. (2000). General notion of statistical depth function. *The Annal of Statistics*, *28*, 461–482.

Knowl Inf Syst
DOI 10.1007/s10115-013-0612-3

REGULAR PAPER

DEMass: a new density estimator for big data

Kai Ming Ting · Takashi Washio · Jonathan R. Wells · Fei Tony Liu · Sunil Aryal

Received: 7 December 2011 / Revised: 28 August 2012 / Accepted: 20 October 2012

Abstract Density estimation is the ubiquitous base modelling mechanism employed for many tasks including clustering, classification, anomaly detection and information retrieval. Commonly used density estimation methods such as kernel density estimator and k-nearest neighbour density estimator have high time and space complexities which render them inapplicable in problems with big data. This weakness sets the fundamental limit in existing algorithms for all these tasks. We propose the first density estimation method, having average case sub-linear time complexity and constant space complexity in the number of instances, that stretches this fundamental limit to an extent that dealing with millions of data can now be done easily and quickly. We provide an asymptotic analysis of the new density estimator and verify the generality of the method by replacing existing density estimators with the new one in three current density-based algorithms, namely DBSCAN, LOF and Bayesian

The source codes of DEMass-DBSCAN and DEMass-Bayes are available at http://sourceforge.net/projects/mass-estimation/.

The preliminary version of this paper appeared in Proceedings of the 2011 IEEE International Conference on Data Mining [33].

K. M. Ting · J. R. Wells · F. T. Liu · S. Aryal (✉)
Gippsland School of Information Technology, Monash University,
Gippsland Campus, Churchill, VIC 3842, Australia
e-mail: sunil.aryal@monash.edu

K. M. Ting
e-mail: kaiming.ting@monash.edu

J. R. Wells
e-mail: jonathan.wells@monash.edu

F. T. Liu
e-mail: tony.liu@monash.edu

T. Washio
The Institute of Scientific and Industrial Research, Osaka University,
8-1 Mihogaoka, Ibarakishi, Osaka 5670047, Japan
e-mail: washio@ar.sanken.osaka-u.ac.jp

Published online: 09 February 2013

classifiers, representing three different data mining tasks of clustering, anomaly detection and classification. Our empirical evaluation results show that the new density estimation method significantly improves their time and space complexities, while maintaining or improving their task-specific performances in clustering, anomaly detection and classification. The new method empowers these algorithms, currently limited to small data size only, to process big data—setting a new benchmark for what density-based algorithms can achieve.

Keywords Density estimation · Density-based algorithms

1 Introduction

Density estimation is ubiquitously applied to various tasks such as clustering, classification, anomaly detection and information retrieval. Despite its pervasive use (‘estimation of densities is a universal problem of statistics’ [36]), there are no efficient density estimation methods thus far. Most existing methods such as kernel density estimator (KDE) and k-nearest neighbour (k-NN) density estimator cannot be applied to problems with big data. This paper is motivated to introduce the first efficient density estimation method for big data. We show that two existing density-based algorithms for clustering and anomaly detection, when employing the new density estimator, set a new runtime benchmark that is orders of magnitude faster. For example, the clustering algorithm now takes only hours instead of more than one month to complete a task involving one million of instances, after the existing density estimator is replaced with the new one.

We make five contributions in this paper:

1. Propose a new density estimation method which has a significant advantage over existing methods in terms of time and space complexities.
2. Establish the asymptotic behaviour of the method through a bias-variance analysis.
3. Verify the generality of the method by replacing existing density estimators with the new one in three current density-based algorithms.
4. Significantly simplify and speed up the two current algorithms in anomaly detection and clustering tasks using set-based definitions instead of the common point-based definitions.
5. Introduce the first Bayesian classifier with constant training time complexity in the number of instances. The proposed Bayesian classifier estimates the multidimensional likelihood directly from the training data, unlike most existing Bayesian classifiers which estimate single-dimensional likelihood.

The new density estimation method distinguishes itself from existing methods by:

- Employing no distance measures in the density estimation process.
- Having constant space complexity and average case sub-linear time complexity in the number of instances in unsupervised learning tasks and constant training time complexity in the number of training instances in supervised learning tasks. Thus, it can be applied to big data in which current methods such as kernel and k-NN density estimators are infeasible because they are expensive to compute.

Three existing density estimators are presented in Sect. 2, in order to contrast with the new density estimator we introduce in Sect. 3. We analyse the error produced by the new estimator by a bias-variance analysis and provide a comparison of the estimation results between the new estimator and KDE in Sect. 4. Sections 5 and 6 describe how the new estimator can replace the existing density estimators in three current state-of-the-art density-based algorithms and their empirical evaluation results, respectively. A discussion of the related issues and the conclusions are provided in the last two sections.

2 Density estimation

This section describes probably the three most commonly used density estimation methods, namely KDE, k-nearest neighbour density estimator and ϵ-neighbourhood density estimator.

2.1 Kernel density estimator

Let $\mathbf{x}$ be an instance in a d-dimensional space $\mathcal{R}^d$. The KDE defined by a kernel function $K(\cdot)$ and bandwidth b is given as follows [29]:

$$\bar{f}_{\text{KDE}}(\mathbf{x}) = \frac{1}{nb^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{b}\right)$$

The difference $\mathbf{x}-\mathbf{x}_i$ requires some form of distance measure, and n is the number of instances in the given data set D. An example of $K(\cdot)$, as a rectangular function, is given as follows:

$$K(\mathbf{x}) = \begin{cases} \frac{1}{2} & \text{if } |\mathbf{x}| < 1 \\ 0 & \text{otherwise.} \end{cases}$$

2.2 k-NN density estimator

A k-NN density estimator can be expressed as follows [30]:

$$\bar{f}_{\text{kNN}}(\mathbf{x}) = \frac{|N(\mathbf{x},k)|}{n\sum_{\mathbf{x}'\in N(\mathbf{x},k)} \text{distance}(\mathbf{x},\mathbf{x}')}$$

where $N(\mathbf{x}, k)$ is the set of k nearest neighbours to $\mathbf{x}$, and the search for nearest neighbours is conducted over D of size n.

2.3 ϵ-Neighbourhood density estimator

The ϵ-neighbourhood density estimator [12] is defined as follows:

$$\bar{f}_{\epsilon}(\mathbf{x}) = \frac{|N_\epsilon(\mathbf{x})|}{n\epsilon}$$

where $N_\epsilon(\mathbf{x}) = \{\mathbf{x}' \in D | \text{distance}(\mathbf{x},\mathbf{x}') \leq \epsilon\}$.

All the three density estimators have $O(n^2)$ time complexity and $O(n)$ space complexity in order to estimate the densities of n instances. Although there are various indexing schemes to speed up the search for nearest neighbour in order to aid the k-NN and ϵ-neighbourhood density estimators, they are not satisfactory in terms of dealing with high-dimensional problems and large data sets. We will provide further discussion of this issue in Sect. 7.

3 Density estimator based on mass

A recently introduced base measure called mass [35] has demonstrated its wide application to solve various data mining tasks such as regression, information retrieval, clustering and anomaly detection, including one in data stream [31,34,35].

Because mass is more fundamental than density, we show in this paper that a density estimator can be constructed from mass. The key advantage of mass is that it can be computed very quickly. The new density estimator based on mass (DEMass) inherits this advantage and executes significantly faster than existing density estimators such as KDE, k-NN and ϵ-neighbourhood. It raises the capability of density-based algorithms to handle big data to a new high level.

A mass base function is defined as follows by [34]:

$$\mathbf{m}(T(\mathbf{x})) = \begin{cases} \mathsf{m} & \text{if } \mathbf{x} \text{is in a region of } T(\cdot), \\ 0 & \text{otherwise,} \end{cases}$$

where $T(\cdot)$ is a function which subdivides the feature space into non-overlapping regions based on the given data set D and m is the number of samples in a region of $T(\mathbf{x})$ in which $\mathbf{x}$ falls into.

Ting and Wells [34] show that mass can also be effectively estimated using data subsets $\mathcal{D}_i \subset D (i = 1, \ldots, t)$ and its associated $T_i(\mathbf{x}|\mathcal{D}_i)$, where $|\mathcal{D}_i| = \psi \ll n$. Each $\mathcal{D}_i$ is sampled without replacement from D. The mass estimated using sub-samples is defined as follows:

$$\overline{\text{mass}}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i)).$$

We now introduce the new density estimators based on mass (DEMass) and describe its implementation in the next two subsections.

3.1 DEMass

Once mass is estimated, density can be estimated as a ratio of mass and volume. Thus, the new density estimators based on mass functions $\mathbf{m}(T(\mathbf{x}))$ and $\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))$ are defined, respectively, as:

$$f_{\mathbf{m}}(\mathbf{x}) = \frac{\mathbf{m}(T(\mathbf{x}))}{nv}. \tag{1}$$

$$\bar{f}_{\mathbf{m}}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \frac{\mathbf{m}(T_i(\mathbf{x}|\mathcal{D}_i))}{\psi v_i}. \tag{2}$$

where v and v_i are the volumes of regions $T(\mathbf{x})$ and $T_i(\mathbf{x}|\mathcal{D}_i)$, respectively.

We use the term DEMass to refer to the new density estimator constructed from $T_i(\mathbf{x}|\mathcal{D}_i)$ in the rest of this paper. DEMass has two key differences/advantages when compared to the one based on a kernel method, k-NN or ϵ-neighbourhood:

- $\bar{f}_{\mathbf{m}}$ is estimated from $t\psi$ instances only which are significantly smaller than D in a large data set. It sums over t number of randomly generated regions, whereas $\bar{f}_{\text{KDE}}$ sums over n number of instances in D, and $\bar{f}_{\text{kNN}}$ and $\bar{f}_{\epsilon}$ require a search on the entire data set. For a large data set, $\bar{f}$ is prohibitively expensive to compute in these three methods.[1]

[1] While there are ways to reduce the computational cost of KDE, k-NN and ϵ-neighbourhood, they are usually limited to low-dimensional problems or incur significant preprocessing cost. See Sect. 7 for a discussion.

- $\bar{f}_{\mathbf{m}}$ needs no distance measures.

3.2 Implementation

Mass estimation can be implemented in different ways [31,34,35]. We use the implementation of $T(\cdot|\mathcal{D})$ as described by [34] using a binary tree (called *h*:*d*-*Trees* by [34]) as the basis to build density estimator $\bar{f}_{\mathbf{m}}$. Algorithm 1 generates t trees from a given data set D. Algorithm 2 generates a single tree using a subset $\mathcal{D} \subset D$, where $|\mathcal{D}| = \psi$.

When $T(\cdot|\mathcal{D})$ is implemented as a binary tree, the volumes of regions in $T(\cdot|\mathcal{D})$ are controlled by a parameter h which defines the level of binary subdivision. Let Δ_i be a workspace in $\mathcal{R}^d$ which envelops $\mathcal{D}_i$, and Δ_i has its length along each dimension j as $\Delta_{ij} = \max(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i) - \min(x_{kj}|\mathbf{x}_k \in \mathcal{D}_i)$. The workspace Δ_i is adjusted to become Ω_i using a random perturbation conducted as follows. For each dimension j, a split point v_j is chosen randomly within the range Δ_{ij}. Then, the new range Ω_{ij} along dimension j is defined as $[v_j - r, v_j + r]$, where $r = \max(v_j - \min_j(\Delta_i), \max_j(\Delta_i) - v_j)$. The new ranges on all dimensions define the adjusted workspace Ω_i for the tree-building process. This random initialisation of the workspace is done in line# 5 of Algorithm 1.

A subset $\mathcal{D}_i$ is constructed by sampling ψ instances without replacement from D. It is used to construct $T_i(\cdot)$. If there are not enough instances to sample, all the instances are used (i.e. $\psi = n$, if $\psi \geq n$). The random adjustment of the work space as described earlier ensures that no two trees are identical even if they are constructed from the same set of instances. The random sampling is done in line# 4 of Algorithm 1.

The dimension to split is selected from a randomised set of d dimensions in a round-robin manner at each level of a tree. At each level, the workspace is divided into two equal-volume half-spaces by splitting at mid-point of the selected dimension. The process is then repeated recursively on each non-empty half-space until the maximum height ($h \times d$) is reached. Hence, each path from the root to a leaf has $h \times d$ nodes such that each of the d attributes appears exactly h times.

Each $T_i(\cdot|\mathcal{D}_i)$ is constructed within workspace Ω_i, resulting in potentially 2^{hd} hyper-rectangular regions where every region has an equi-width $\delta x_{ij} = \Omega_{ij}/2^h$ on each dimension j and a volume $v_i = \delta x_{i1} \times \cdots \times \delta x_{id}$. For example, in a one-dimensional space with workspace Ω_i derived from $\mathcal{D}_i$ and $h = 3$, $T_i(\cdot|\mathcal{D}_i)$ subdivides the workspace into 2^3 regions. We use T_i to denote T_i^h, unless h is required in the context, and T_i is built from $\mathcal{D}_i$, for each i.

Let $\ell = h \times d$, and m_k be the mass of region k. There is a total of 2^ℓ regions which have a total mass: $|\mathcal{D}| = \sum_{k=1}^{2^\ell} \mathsf{m}_k$, where $\mathsf{m}_k = \mathbf{m}(T(\mathbf{x}|\mathcal{D}))$, and $\mathbf{x}$ is in region k of T.

Our implementation reduces the number of regions generated, usually less than 2^{hd}, for $\psi \ll n$. At each node, if one of the two child nodes is empty, the range of the node is reduced to the range of the non-empty child node instead of creating the empty node (line# 7–12 in Algorithm 2). This avoids creating unnecessary empty regions and reduces memory requirement.

The height of each tree is hd. At each level of a tree, each instance in $\mathcal{D}$ has to be assigned to either of the two child nodes. Thus, the time complexity of constructing t trees is $O(thd\psi)$.

There are a total of $\min(2^{hd}, \psi)$ leaf nodes in each tree. In general, $\psi < 2^{hd}$ with moderate d and h. Thus, the space complexity is $O(td\psi + n)$ during construction. After the trees are built, the data set is discarded, yielding $O(td\psi)$.

To estimate the density of a given instance $\mathbf{x}$, only these trees are used according to Eq. (2).

In the next section, we will show that the bias between $\bar{f}_{\mathbf{m}}(\mathbf{x})$ and the true probability density function $p_d(\mathbf{x})$ converges asymptotically through a bias-variance analysis.

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data with d attributes, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t h:d-$Trees$

```
$MaxHeightLimit \leftarrow h \times d$
**Initialise** $F$
**for** $i = 1$ to $t$ **do**
    $\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
    $(min,max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
    $A \leftarrow$ Randomised list of $d$ attributes.
    $F \leftarrow F \cup$ SingleTree($\mathcal{D}, min, max, 0$)
**end for**
```

Algorithm 2 : SingleTree($\mathcal{D}, min, max, \ell$)

Inputs: $\mathcal{D}$ - input data, min & max - arrays of minimum and maximum values for each of the d attributes that define a work space, ℓ - current height level.
Output: an h:d-$Trees$

```
Initialise $Node(\cdot)$
**while** $(\ell < MaxHeightLimit)$ **do**
    $q \leftarrow nextAttribute(A, \ell)$ {Retrieve an attribute from $A$ based on height level.}
    $mid_q \leftarrow (max_q + min_q)/2$
    $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < mid_q)$
    $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq mid_q)$
    **if** $(|\mathcal{D}_l| = 0)$ or $(|\mathcal{D}_r| = 0)$ **then** {Reduce range for single-branch node.}
        **if** $(|\mathcal{D}_l| > 0)$ **then** $max_q \leftarrow mid_q$
        **else** $min_q \leftarrow mid_q$
        **end if**
        $\ell \leftarrow \ell + 1$
        continue at the start of while loop
    **end if**
    {Build two nodes: $Left$ and $Right$ as a result of a split into two equal-volume half-spaces.}
    $temp \leftarrow max_q$; $max_q \leftarrow mid_q$
    $Left \leftarrow$ SingleTree($\mathcal{D}_l, min, max, \ell + 1$)
    $max_q \leftarrow temp$; $min_q \leftarrow mid_q$
    $Right \leftarrow$ SingleTree($\mathcal{D}_r, min, max, \ell + 1$)
    terminate while loop
**end while**
**return** $Node(Left, Right, SplitAtt \leftarrow q, SplitValue \leftarrow mid_q, Size \leftarrow |\mathcal{D}|)$
```

4 Error analysis through bias-variance decomposition

The DEMass $\bar{f}_{\mathbf{m}}(\mathbf{x})$ can be thought of as a random variable because of its dependence on D and its random sub-samples $\mathcal{D}_i$ $(i = 1, \ldots, t)$. Accordingly, we analyse mean squared error (MSE) of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ from its true probability density $p_d(\mathbf{x})$. It is defined as:

$$\mathrm{MSE}(\bar{f}_{\mathbf{m}}(\mathbf{x})) = E\left[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - p_d(\mathbf{x})\}^2\right]$$

where the expectation $E[\cdot]$ is taken over the distribution of $\bar{f}_{\mathbf{m}}(\mathbf{x})$. This is rewritten by introducing the expectation of $\bar{f}_{\mathbf{m}}(\mathbf{x})$: $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ as follows [29]:

$$\mathrm{MSE}(\bar{f}_{\mathbf{m}}(\mathbf{x})) = \left\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\right\}^2 + E\left[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\right]$$

The first term on the rhs is called '*square bias* ' and the second '*variance*'. We evaluate the magnitude of each of these two terms in the following.

To simplify notations for the rest of the paper, we have used $T_i(\mathbf{x})$ to denote $T_i(\mathbf{x}|\mathcal{D}_i)$ and $p(T_i(\mathbf{x}))$ to denote $p(\mathbf{x}_k \in T_i(\mathbf{x})|\ \mathbf{x}_k \in \mathcal{D}_i)$.

Let $\mathbf{c}_i$ be the centre of a region of $T_i(\mathbf{x})$ where each element c_{ij} of $\mathbf{c}_i$ is a middle point of the interval on each dimension j. The second-order Taylor approximation of $p_d(\mathbf{x})$ around $\mathbf{c}_i$ for $T_i(\mathbf{x})$ is given as:

$$\begin{aligned} p_d(\mathbf{x})|_{\mathbf{c}_i \in T_i(\mathbf{x})} \approx\ & p_d(\mathbf{c}_i) + (\mathbf{x}-\mathbf{c}_i)^{\top} \nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \\ & + \frac{1}{2}\{(\mathbf{x}-\mathbf{c}_i)^{\top}\nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}, \end{aligned} \tag{3}$$

where $\nabla = [\partial/\partial x_1, \ldots, \partial/\partial x_d]^{\top}$.

Note that $\mathbf{m}(T_i(\mathbf{x}))$ follows a binomial distribution[2] $B(\psi, p(T_i(x)))$. Therefore, $E[\bar{f}_{\mathbf{m}}(\mathbf{x})]$ is expressed by substituting $E[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))$ in Eq. (2).

$$\begin{aligned} E[\bar{f}_{\mathbf{m}}(\mathbf{x})] &= \frac{1}{t}\sum_{i=1}^{t} \frac{E[\mathbf{m}(T_i(\mathbf{x}))]}{\psi v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t} \frac{p(T_i(x))}{v_i} \\ &= \frac{1}{t}\sum_{i=1}^{t} \frac{1}{v_i} \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*) d\mathbf{x}_*. \end{aligned} \tag{4}$$

Accordingly, the square bias is evaluated as follows by applying Eq. (3) and the fact that the integral of an odd function over $[c_{ij} - \delta x_j/2, c_{ij} + \delta x_j/2]$ for each dimension j is zero.

$$\begin{aligned} & \left\{E[\bar{f}_{\mathbf{m}}(\mathbf{x})] - p_d(\mathbf{x})\right\}^2 \\ & \approx \left[\frac{1}{t}\sum_{i=1}^{t}\left\{\frac{1}{24}\sum_{j=1}^{d} \left.\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\right|_{\mathbf{x}=\mathbf{c}_i} \delta x_{ij}^2 - (\mathbf{x}-\mathbf{c}_i)^{\top}\nabla p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i} \right.\right. \\ & \qquad \left.\left. - \frac{1}{2}\{(\mathbf{x}-\mathbf{c}_i)^{\top}\nabla\}^2 p_d(\mathbf{x})|_{\mathbf{x}=\mathbf{c}_i}\right\}_{\mathbf{c}_i \in T_i(\mathbf{x})}\right]^2 \\ & \le \left[\frac{1}{t}\sum_{i=1}^{t}\left\{\frac{1}{24}\left|\sum_{j=1}^{d} \left.\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\right|_{\mathbf{x}=\mathbf{c}_i}\right| \Delta_{ij}^2 2^{-2h} + \sum_{j=1}^{d}\left|\left.\frac{\partial p_d(\mathbf{x})}{\partial x_j}\right|_{\mathbf{x}=\mathbf{c}_i}\right| \Delta_{ij} 2^{-h} \right.\right. \\ & \qquad \left.\left. + \frac{1}{2}\sum_{j=1}^{d}\sum_{k=1}^{d}\left|\left.\frac{\partial^2 p_d(\mathbf{x})}{\partial x_j \partial x_k}\right|_{\mathbf{x}=\mathbf{c}_i}\right| \Delta_{ij}\Delta_{ik} 2^{-2h}\right\}_{\mathbf{c}_i \in T_i(\mathbf{x})}\right]^2 \\ & = O(4^{-h}) \end{aligned}$$

This result shows that the square bias diminishes as level h increases, i.e., as the size of the regions decreases. Though this analysis uses the second-order approximation of $p_d(\mathbf{x})$, the result using the higher-order approximation is the same since the first-order term dominates in the above formula.

[2] The implementation of $T(\cdot)$ used in this paper is a tree-based nonparametric method. The binomial distribution is required for the error analysis only.

Because $\mathbf{m}(T_i(\mathbf{x}))$ follows the binomial distribution $B(\psi, p(T_i(x)))$, the variance of $\mathbf{m}(T_i(\mathbf{x}))$ is:

$$\mathrm{var}[\mathbf{m}(T_i(\mathbf{x}))] = \psi p(T_i(x))(1 - p(T_i(x))).$$

In concert with Eq. (2), the variance of $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is represented as follows:

$$\begin{aligned}
&E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big] \\
&= \frac{1}{t^2}\sum_{i=1}^{t} \frac{p(T_i(\mathbf{x}))(1 - p(T_i(\mathbf{x})))}{\psi v_i^2} \\
&= \frac{1}{t^2}\sum_{i=1}^{t} \frac{1}{\psi v_i^2} \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*)\mathrm{d}\mathbf{x}_* \left(1 - \int_{T_i(\mathbf{x})} p_d(\mathbf{x}_*)\mathrm{d}\mathbf{x}_*\right).
\end{aligned}$$

Using the similar calculus as applied to the square bias, we obtain the variance as follows where $\mathbf{c_i}$ is a centre of $T_i(\mathbf{x})$.

$$\begin{aligned}
&E\big[\{\bar{f}_{\mathbf{m}}(\mathbf{x}) - E[\bar{f}_{\mathbf{m}}(\mathbf{x})]\}^2\big] \\
&\approx \frac{1}{t^2}\sum_{i=1}^{t} \frac{1}{\psi} \left\{ p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\bigg|_{\mathbf{x}=\mathbf{c}_i} \delta x_{ij}^2 \right\} \\
&\quad \times \left\{ \frac{1}{v_i} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\bigg|_{\mathbf{x}=\mathbf{c}_i} \delta x_{ij}^2 \right\}. \\
&= \frac{1}{t^2}\sum_{i=1}^{t} \frac{1}{\psi} \left\{ p_d(\mathbf{c}_i) + \frac{1}{24}\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\bigg|_{\mathbf{x}=\mathbf{c}_i} \Delta_{ij}^2 2^{-2h} \right\} \\
&\quad \times \left\{ \frac{2^{dh}}{\prod_{j=1}^{d} \Delta_{ij}} - p_d(\mathbf{c}_i) - \frac{1}{24}\sum_{j=1}^{d} \frac{\partial^2 p_d(\mathbf{x})}{\partial x_j^2}\bigg|_{\mathbf{x}=\mathbf{c}_i} \Delta_{ij}^2 2^{-2h} \right\}. \\
&= O(2^{dh})
\end{aligned}$$

This result indicates that the variance increases when level h increases. Also, the result does not change even if we use the higher-order approximation because the term $p_d(\mathbf{c}_i)/v_i$ dominates in the above formula.

The property of DEMass, revealed from this error analysis, is similar to that of the conventional KDE which shows a bias-variance trade-off—the bias decreases as the kernel bandwidth b decreases, but this increases the variance, and the reverse is true if the kernel bandwidth is increased [29]. The parameter k in k-NN density estimator has the same effect.

In conclusion, DEMass has a comparable estimation of density with the KDE if both trade-off bias and variance are equally well, and it is indeed the case in practice. Figure 1 shows the estimation result of a normal distribution using KDE and DEMass. It demonstrates that DEMass produces similar result to that generated by KDE, for different data sizes. Smoothing can be applied by increasing b for KDE or decreasing h for DEMass which produces the estimation results as shown in Fig. 2. The parameters used for DEMass are the following: $t = 1{,}000$ and $\psi = n$ when $n = 10, 100$; and $\psi = 1{,}000$ when $n = 1{,}000{,}000$.

Note that in either setting shown in Figs. 1 and 2, the estimations of both KDE and DEMass approach the true distribution as the number of instances increases.

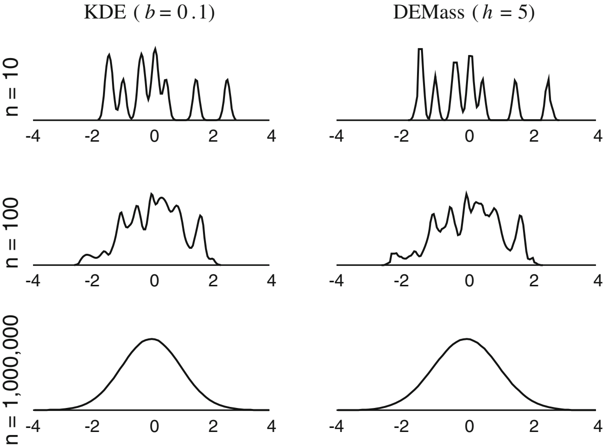


Fig. 1 Example estimations of Kernel density estimator (with Gaussian kernel) using $b = 0.1$ and DEMass using $h = 5$ for different data sizes, $n = 10, 100, 1{,}000{,}000$. The true data distribution is a normal distribution

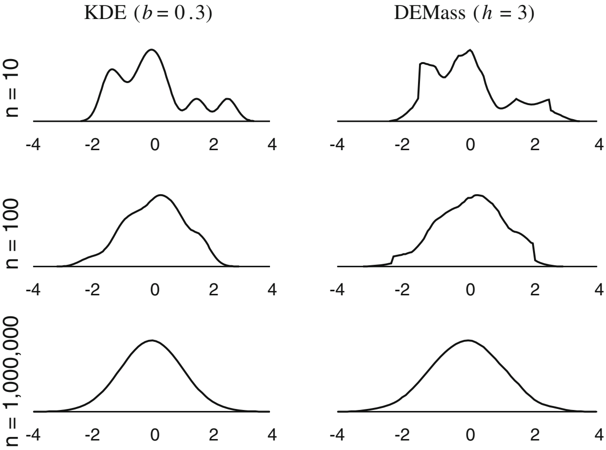


Fig. 2 Example estimations of Kernel density estimator (with Gaussian kernel) using $b = 0.3$ and DEMass using $h = 3$ for the same data used in Fig. 1

5 Using DEMass in existing density-based algorithms

This section describes how DEMass can be applied to three current density-based algorithms, DBSCAN [12], LOF [6] and Bayesian classifiers, in place of their existing density estimators. DBSCAN, LOF and Bayesian classifiers are one of the best algorithms for clustering, anomaly detection and classification, respectively.

Using DEMass automatically carries the two advantages mentioned in Sect. 3: (i) the estimation requires no distance measures; thus, it completely saves the cost of distance calculations for every pair of instances and (ii) DEMass enables small samples to construct the required regions $T(\cdot|\mathcal{D})$, overcoming the key limitation of DBSCAN and LOF in handling big data. We will discuss further advantages specific to individual algorithms in the following subsections.

Table 1 Algorithms for DBSCAN and DEMass-DBSCAN

Step	DBSCAN	DEMass-DBSCAN
1	Label all points as core, border or noise points, based on $\bar{f}_\epsilon(\mathbf{x})$	Label all $T(\mathbf{x})$ satisfying Definition S1 in Table 2 as core regions, based on $f_\mathbf{m}(\mathbf{x})$. Points not covered by core regions are noise
2	Eliminate noise points	Eliminate noise points
3	Connect all core points that are within ϵ of each other	Connect all core regions that have non-zero intersections
4	Make each group of connected core points into a separate cluster	Make each group of connected core regions into a separate cluster
5	Assign each border point to one of the clusters of its associated core points	
Time complexity	$O(dn^2)$	$O(thdn)$
Space complexity	$O(dn)$	$O(td\psi)$

Note that border points are not required with DEMass-DBSCAN; thus step 5 is not needed. Both versions of DBSCAN could include an additional cluster size threshold to eliminate small size clusters in the last step

5.1 DEMass-DBSCAN

The principal steps of DEMass-DBSCAN are the same as DBSCAN, except that no border points and their associated step are required. A comparison of the two algorithms is provided in Table 1. The algorithm for DBSCAN is adapted from [30].

Following its principal steps, the use of DEMass simplifies DBSCAN in two ways, in addition to the two advantages already mentioned above. First, DEMass enables regions to be labelled instead of individual points. Because the number of regions is significantly less than the number of points, labelling and linking required in steps 1 and 3 become significantly faster. Second, no border points need to be defined because the connections within a cluster are established via core regions only when DEMass is used. The first simplification is the key reason for the significant speed up achieved by DEMass-DBSCAN, which we will show in Sect. 6.1.

The time complexity to construct t trees is $O(thd\psi)$. In order to assign a cluster to each instance, each of the t trees is traversed from the root to the respective leaf node. The time complexity of that searching is $O(thdn)$. Thus, the overall time cost of DEMass-DBSCAN is $O(thd\psi + thdn)$. Since $\psi \ll n$, the first term can be ignored. Hence, the total time complexity of DEMass-DBSCAN is $O(thdn)$. But, the time complexity of DBSCAN is $O(dn^2)$ as it requires pairwise distance calculation between n instances in order to search for ϵ-neighbours. Also, DBSCAN needs to store all the instances in order to cluster a future instance, yielding space complexity of $O(dn)$. But, DEMass-DBSCAN requires space to store t h:d-$Trees$ only, which is $O(td\psi)$. The space complexity of DEMass-DBSCAN is constant (independent of n).

The DEMass-DBSCAN algorithm shown in Table 1 is derived from set-based definitions, and the DBSCAN algorithm is derived from point-based definitions. We can use point-

Table 2 Point-based and set-based definitions for DEMass-DBSCAN. Note that $T(\cdot)$ is used to denote $T(\cdot|\mathcal{D})$

Point-based definitions	Set-based definitions
Definition P1: The h-**neighbourhood** of a point p, denoted by $N_h(p)$, is defined by $N_h(p) = \{q \in D \mid q \in T^h(p)\}$ In contrast, the ϵ-neighbourhood for DBSCAN is defined as $N_\epsilon(p) = \{q \in D \mid dist(p, q) \leq \epsilon\}$, which requires a distance function $dist(\cdot, \cdot)$ No distance functions are required for $N_h(\cdot)$	
Definition P2: A point p is **directly density-reachable** from a point q wrt h and $MinPts$ if (i) $p \in N_h(q)$ and (ii) $\frac{\lvert N_h(q)\rvert}{v} \geq \frac{MinPts}{v_{max}}$ (core point condition)	*Definition S1:* $T(\mathbf{x})$ is a **core region** of point $\mathbf{x}$ wrt h and $MinPts$ if $\frac{\mathbf{m}(T(\mathbf{x}))}{v} \geq \frac{MinPts}{v_{max}}$, where $v_{max} = \max_i v_i$ and v is the volume of region $T(\mathbf{x})$
Definition P3: A point p is **density-reachable** from a point q wrt h and $MinPts$ if there is a chain of points $p_1, \ldots, p_n$, where $p_1 = p$ and $p_n = q$ such that p_{i+1} is directly density-reachable from p_i	
Definition P4: A point p is **density-connected** to a point q wrt h and $MinPts$ if there is a point o such that both p and q are density-reachable from o wrt h and $MinPts$	*Definition S2:* $T_r(\cdot)$ is **density-connected** to $T_s(\cdot)$ wrt h and $MinPts$ if there is a chain of regions $T_1(\cdot), \ldots, T_g(\cdot)$ where $r = 1$ and $s = g$ such that $T_\imath(\cdot) \cap T_{\imath+1}(\cdot) \neq \emptyset$ and $T_\imath(\cdot)$ is a core region for $\imath \in \{1, \cdots, g\}$ wrt h and $MinPts$
Definition P5: Let D be a data set of points A **cluster** C wrt h and $MinPts$ is a non-empty subset of D satisfying the following conditions: (i) $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt h and $MinPts$, then $q \in C$ (Maximality) (ii) $\forall p, q \in C$: p is density-connected to q wrt h and $MinPts$ (Connectivity)	*Definition S3:* An **arbitrary-shape cluster** C wrt h and $MinPts$ is a non-empty subset of a data set D satisfying the following conditions: $\forall r, s$; $T_r(\cdot), T_s(\cdot) \subset C$: $T_r(\cdot)$ is density-connected to $T_s(\cdot)$ wrt h and $MinPts$
Definition P6: Let $C_1, \ldots, C_k$ be the clusters of the data set D wrt h and $MinPts$ Then we define **noise** as the set of points in the data set D not belonging to any cluster C_j, i.e., noise = $\{p \in D \mid \forall j : p \notin C_j\}$	*Definition S4:* Let $C_1, \ldots, C_k$ be the clusters of D wrt h and $MinPts$ **Noise** is the set of points in D not belonging to any cluster $C_\jmath$, i.e., noise = $\{\mathbf{x} \in D \mid \forall \jmath : \mathbf{x} \notin C_\jmath\}$

The point-based definitions are adopted from those defined for DBSCAN [12]

based definitions for DEMass-DBSCAN, except that the neighbourhood definition needs to be adapted to DEMass. Although point-based definitions can be defined as in DBSCAN [12], set-based definitions are simpler. The formal point-based and set-based definitions for DEMass-DBSCAN are given in Table 2.

A comparison between DBSCAN and DEMass-DBSCAN is provided using two examples showed in Fig. 3a, b. They show how core points and non-core points are labelled in DBSCAN

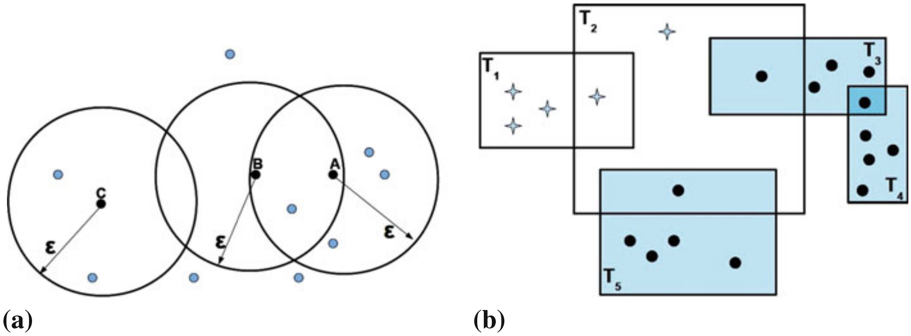


Fig. 3 An example for DBSCAN and DEMass-DBSCAN for $MinPts = 5$. **a** An example for DBSCAN for $MinPts = 5$. A is a core point, B is a border point, and C is a noise point. **b** An example for DEMass-DBSCAN for $MinPts = 5$. *The circle symbol* indicates core points, and *the star symbol* indicates noise points. T_3, T_4 and T_5 are core regions. T_3 and T_4 are linked by a common core point

Table 3 Algorithms for LOF and DEMass-LOF

Step	LOF	DEMass-LOF
1	Compute density distribution: $\bar{f}_{kNN}(\mathbf{x})$	Compute density distribution: $\bar{f}_{\mathbf{m}}(\mathbf{x})$
2	Compute $LOF(\mathbf{x})$ using $\dfrac{\sum_{\mathbf{x}' \in N(\mathbf{x},k)} \frac{\bar{f}_{kNN}(\mathbf{x}')}{\lvert N(\mathbf{x},k)\rvert}}{\bar{f}_{kNN}(\mathbf{x})}$	Compute $LOF_p(\mathbf{x})$ using: $\dfrac{\frac{1}{t}\sum_{i=1}^{t} \frac{\mathbf{m}(\breve{T}_i(\mathbf{x}))}{\breve{\psi}\breve{v}_i}}{\bar{f}_{\mathbf{m}}(\mathbf{x})}$
3	Rank all instances based on their LOF values in descending order	Rank all instances based on their LOF_p values in descending order
Time complexity	$O(dn^2)$	$O(thdn)$
Space complexity	$O(dn)$	$O(td\psi)$

$\bar{f}_{kNN}(\mathbf{x})$ and $N(\mathbf{x}, k)$ are defined in Sect. 2.2; $\bar{f}_{\mathbf{m}}(\mathbf{x})$ is defined in Sect. 3. $\breve{T}_i(\mathbf{x}) \supset T_i(\mathbf{x})$ correspond to the parent and child nodes in our tree implementation; $\breve{\psi}$ and $\breve{v}_i$ are the data size and volume of $\breve{T}_i(\mathbf{x})$, respectively. Note that $\breve{T}_i(\mathbf{x})$, the next superset of $T_i^h(\mathbf{x})$, is not necessarily $T_i^{h-1}(\mathbf{x})$ because there are d levels in the tree for each increment of h and the implementation allows single-branch extensions if there are no data in other branches. See Sect. 3.2 for details of the implementation

and DEMass-DBSCAN. One superficial difference in these examples is that DBSCAN uses hyper-spheres and DEMass-DBSCAN uses hyper-rectangles. This difference can be easily eliminated by using L^∞-norm (instead of L^2-norm) in DBSCAN.

5.2 DEMass-LOF

Table 3 compares the algorithms for LOF and DEMass-LOF which have three identical principal steps: compute density distribution and LOF, and then rank all instances based on their LOF values. The key difference is the density estimator used in step 1 which changes the computation of LOF in step 2.

In addition to the two advantages due to the use of DEMass mentioned in Sect. 3, the advantage specific to LOF is that DEMass enables the computation of the relative density to be substantially simplified, changing from nearest-neighbour-based to set-based. Instead of finding the neighbours of **x** and then computing the density of each neighbour, the modified ranking measure LOF_p is computed based on the region $T(\mathbf{x})$ and its immediate larger region $\breve{T}(\mathbf{x}) \supset T(\mathbf{x})$. In the tree implementation of $T(\cdot)$, this corresponds to computing the density of the node in which x falls into, relative to the density of its parent node.

In steps 1 and 2, the time complexities of DEMass-LOF and LOF are $O(thdn)$ and $O(dn^2)$, respectively. Since DEMass-LOF does not need to perform neighbourhood search as in LOF, it is much faster, especially in large data sets. Unlike LOF that stores all n instances, DEMass-LOF needs space to store t trees only. Hence, the space complexity of DEMass-LOF is $O(td\psi)$.

The parameter k in LOF has an inverse relationship with h in DEMass-LOF, i.e., high h corresponds to low k (which covers a smaller region than that using low h or high k). A larger k increases LOF's processing time so as a larger h increases DEMass-LOF's processing time.

Note that both LOF and LOF_p are relative density scores, which range from 0 to $+\infty$, indicating the degree of anomaly; the higher the score, the higher the degree of anomaly.

5.3 DEMass-Bayes

Bayesian classifiers require density estimation in order to estimate the class conditional probability of test instance $\mathbf{x}$ given class y, i.e., $p(\mathbf{x}|y)$. Because it is difficult to compute $p(\mathbf{x}|y)$ directly even in problems with a moderate number of dimensions, a number of assumptions have been made to simplify the computation. We describe those used in Naive Bayes (NB) [21], Bayesian networks (BayesNet) [15] and Aggregating One-Dependence Estimators (AODE) [38].

Naive Bayes assumes class conditional independence and estimates density distribution on each dimension separately [21].

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y) \tag{5}$$

For continuous-valued attributes, $p(x_i|y)$ can be computed either through discretisation or using a density estimator. Naive Bayes with discretisation (NB-Disc) [11] estimates $p(x_i|y)$ through discretisation. Naive Bayes with Gaussian distribution (NB-GD) [21] estimates $p(x_i|y)$ through normal probability estimation. Naive Bayes with kernel density estimation (NB-KDE) [22] estimates $p(x_i|y)$ through a kernel density estimator.

The assumption made by Naive Bayes is often violated in the real world where attributes are related in some way. Other Bayesian classifiers such as BayesNet [15] and AODE [38] employ less restrictive assumptions.

BayesNet learns probabilistic relationships among attributes in the form of directed acyclic graph (DAG) from the training data. In a graph, each node is probabilistically independent of its non-descendants given the state of its parents. At each node, joint probabilities with respect to its parents are learned from the training data. In many implementations, the continuous-valued attributes are discretised. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$p(\mathbf{x}, y) = p(y|\pi_y) \prod_{j=1}^{d} p(x_j|\pi_j) \tag{6}$$

where π_j is $parent(x_j)$ and π_y is $parent(y)$.

AODE allows conditional dependence with class and one 'privileged' attribute, and other attributes are conditionally independent given the class label y and a privileged attribute x_i.

The conditional probabilities are computed as follows:

$$p(\mathbf{x}|x_i, y) = \prod_{j=1}^{d} p(x_j|x_i, y) \tag{7}$$

As AODE cannot handle continuous-valued attributes, they are discretised, and the conditional probabilities are estimated by the proportion of instances having values in an interval belonging to class y.

In contrast to the existing implementation of Bayesian classifiers, the implementation based on DEMass estimates $p(\mathbf{x}|y)$ directly, without any assumptions. In order to use DEMass in classification, we made the following adjustments to estimate $p(\mathbf{x}|y)$.

- Instead of constructing t trees in total, t trees per class are constructed to estimate the density distribution of each class separately. The instances in each data subset $\mathcal{D}_i \subset D$ are separated according to their class labels into c subsets where c is the number of classes, yielding $\mathcal{D}_{i,y}$ $(y = 1, 2, \ldots, c)$; $\bigcup_y \mathcal{D}_{i,y} = \mathcal{D}_i$ and $|\mathcal{D}_i| = \psi$. A tree is constructed from $\mathcal{D}_{i,y}$ to represent regions $T_{i,y}(\cdot)$. Hence, a total of ct trees are constructed.
- Instead of growing the tree to the maximum height, we stop growing when the number of instances in a node is less than or equal to one. This provides a smoother estimation.

We compute the class conditional probability based on DEMass as:

$$p(\mathbf{x}|y) \equiv \bar{f}_{\mathbf{m}}(\mathbf{x}|y) = \frac{1}{t}\sum_{i=1}^{t} \frac{\mathbf{m}(T_{i,y}(\mathbf{x}))}{|\mathcal{D}_{i,y}|v_{i,y}} \tag{8}$$

where $\mathcal{D}_{i,y}$ is a subset of samples belonging to class y in $\mathcal{D}_i$ and $v_{i,y}$ is the volume of the region $T_{i,y}(\mathbf{x})$.

The rest of the steps in DEMass-Bayes are the same as existing Bayesian classifiers. The prior probabilities $p(y)$ are calculated from the training data. Finally, Bayes rule is used to predict the class which has the maximum posterior $p(y|\mathbf{x})$.

$$\hat{y} = \arg\max_{y} \quad p(y) \times p(\mathbf{x}|y) \tag{9}$$

In case of a tie, a random prediction is made between the classes yielding the equal maximum posterior probabilities.

The decision rules of existing Bayesian classifiers and DEMass-Bayes are provided in Table 4. The existing Bayesian classifiers estimate the conditional probability as the product of one-dimensional likelihoods as shown in Table 4. In contrast, the proposed Bayesian classifier, DEMass-Bayes, does not make any explicit assumptions and estimates the multidimensional likelihood directly from the training data.

The time and space complexities of existing Bayesian classifiers and DEMass-Bayes are provided in Table 5. DEMass-Bayes uses subsets of the given training set to construct h:d-$Trees$ in order to estimate mass. Its training time is constant as shown in Table 5. But, the training time of the other Bayesian classifiers is linear in n. Also, the proposed method has constant space complexity. Hence, the proposed method scales better than the existing Bayesian classifiers in big data.

6 Empirical evaluation

The evaluations in clustering and anomaly detection tasks are conducted in the unsupervised learning setting, whereas classification task in the supervised setting. We compare DBSCAN

Table 4 Decision rules of existing Bayesian classifiers and DEMass-Bayes

Classifier	Decision rule	Remarks
NB-GD	$\arg\max_y \quad p(y) \prod_{i=1}^{d} p(x_i \mid y)$	$p(x_i \mid y)$ is estimated with normal distribution
NB-KDE		$p(x_i \mid y)$ is estimated with KDE
NB-Disc		$p(x_i \mid y)$ is estimated through discretisation
BayesNet	$\arg\max_y \quad p(\pi_y, y) \prod_{i=1}^{d} p(x_i \mid \pi_i, y)$	$\pi_i = parent(x_i), \pi_y = parent(y)$ Joint probabilities are estimated through discretisation
AODE	$\arg\max_y \sum_{i=1}^{d} p(x_i, y) \prod_{j=1}^{d} p(x_j \mid x_i, y)$	$p(x_j \mid x_i, y)$ in Eq. (7) is estimated through discretisation
DEMass-Bayes	$\arg\max_y \quad p(y) \quad p(\mathbf{x} \mid y)$	$p(\mathbf{x} \mid y)$ is estimated using Eq. (8)

Note that with the exception of DEMass-Bayes, all existing Bayesian classifiers estimate one-dimensional likelihoods

Table 5 Time and space complexities of existing Bayesian classifiers and DEMass-Bayes

Classifier	Time complexity		Space complexity
	Training	Testing	
NB-GD[a]	$O(nd)$	$O(cd)$	$O(cd)$
NB-KDE[a]	$O(nd)$	$O(cmd)$	$O(cmd)$
NB-Disc[b]	$O(nd)$	$O(cd)$	$O(cdu)$
AODE[b]	$O(nd^2)$	$O(cd^2)$	$O(c(du)^2)$
DEMass-Bayes	$O(cthd\varphi)$	$O(cthd)$	$O(ctd\varphi)$

n: total number of training instances, m: average number of training instances in a class, d: number of dimensions, c: number of classes, u: average number of discrete values of an attribute, t: number of trees, and φ: average number of samples per class in $\mathcal{D}_i$

[a] Langley and John [22], [b]Webb et al. [38]

with DEMass-DBSCAN in the first subsection and then compare LOF with DEMass-LOF in the second subsection. Finally, we compare DEMass-Bayes with five existing Bayesian classifiers, namely NB-GD, NB-KDE, NB-Disc, BayesNet and AODE in the last subsection.

All experiments for clustering and anomaly detection tasks were conducted as single-thread jobs processed at 2.3 GHz in a Linux cluster (www.vpac.org) using a node with 32 GB memory, whereas all the experiments for classification task were conducted as single-thread jobs using a node in Linux cluster with 2.27 GHz and 120 GB memory.

All DEMass-based algorithms were written in JAVA in WEKA platform [39], so as DBSCAN and existing Bayesian classifiers. LOF was written in Java in ELKI platform version 0.4 [1].

The data sets used are from UCI Machine Learning Repository [14], unless stated otherwise. Only data sets more than 10,000 instances are used in order to examine the algorithms' capability to deal with large data sets.

The clustering result was reported in terms of CPU runtime (in seconds), number of clusters identified, number of unassigned instances and F-measure which was calculated based on assigned instances only. F-measure $= 1$ when all assigned instances are in the correct clusters, i.e., perfect clustering, and F-measure $= 0$ if all instances are assigned to

Table 6 Data sets used for the clustering task for comparing DEMass-DBSCAN with DBSCAN

Data sets	Size *n*	#*d*	#Clusters
RingCurve-Wave-TriGaussian3D	70,000	3	7
RingCurve-Wave-TriGaussian48D	70,000	48	7
OneBig	68,000	20	9
Pendigits	10,992	16	10

wrong clusters. The anomaly detection result was reported in terms of CPU runtime and AUC (Area Under ROC Curve) based on the ranked result. The classification result was reported in terms of classification accuracy and CPU runtime (in seconds). We tuned the parameters of each algorithm in the unsupervised learning setting and reported the best result. In the supervised learning, the default parameter settings were used for all the classifiers unless specified otherwise and reported the average classification accuracy and average runtime over a 10-fold cross-validation.

6.1 DEMass-DBSCAN versus DBSCAN

DEMass-DBSCAN had $\psi = 256$ and $t = 1{,}000$ as default, and both DEMass-DBSCAN and DBSCAN used $MinPts = 6$ in all experiments. As a result, only one parameter needed to be tuned for a particular data set: h for DEMass-DBSCAN and ϵ for DBSCAN.

Table 6 provides the properties of the four data sets used. RingCurve-Wave-TriGaussian3D, RingCurve-Wave-TriGaussian48D and OneBig are the three largest data sets as used in [34], and we use an additional data set, Pendigits, which has more than 10,000 instances.

RingCurve-Wave-TriGaussian consists of three two-dimensional synthetic data: RingCurve, Wave and TriangularGaussian as shown in Fig. 10 in the 'Appendix', embedded in either a 3-dimensional data set or a 48-dimensional data set (where 42 dimensions are irrelevant with a constant value). There are a total of seven clusters with 10,000 instances in each cluster. In OneBig [26], the biggest cluster has 50,011 instances, and each of the other eight clusters has approximately 1,000 instances. In addition, there are 10,000 noise instances randomly distributed in the feature space. Each of the ten clusters in Pendigits has approximately 1,000 instances.

The clustering results from DEMass-DBSCAN and DBSCAN are shown in Table 7. DEMass-DBSCAN ran faster than DBSCAN by a factor more than 17 in both 3-dimensional and 48-dimensional RingCurve-Wave-TriGaussian data sets. In terms of #clusters and #unassigned, DEMass-DBSCAN performed slightly worse than DBSCAN in the 3-dimensional data set, but better in the 48-dimensional data set. DEMass-DBSCAN decreased its number of unassigned instances from 535 to 61 when the number of dimensions was increased from 3 to 48, whereas DBSCAN had the same 332 unassigned instances in both cases. DEMass-DBSCAN performs either similarly to or better than DBSCAN in terms of F-measure in these two data sets.

DEMass-DBSCAN and DBSCAN for OneBig had the same clustering result in terms of F-measure and number of clusters, but DEMass-DBSCAN ran faster than DBSCAN by a factor of 7. Note that DEMass-DBSCAN had correctly identified all but one of the 10,000 noise instances, whereas DBSCAN correctly identified all of the noise instances. In Pendigits, the result showed that although DEMass-DBSCAN had a lower F-measure than DBSCAN,

Table 7 Clustering results of DEMass-DBSCAN ($h = 7$ for RingCurve-Wave-TriGaussian3D, $h = 6$ for RingCurve-Wave-TriGaussian48D, $h = 3$ for OneBig and $h = 2$ for Pendigits) and DBSCAN ($\epsilon = 0.01$ for RingCurve-Wave-TriGaussian3D and RingCurve-Wave-TriGaussian48D, $\epsilon = 0.1$ for OneBig and $\epsilon = 0.2$ for Pendigits)

	RingCurve-Wave-TriGaussian3D		RingCurve-Wave-TriGaussian48D	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	135	2,391	1,261	21,906
#Cluster	9	8	7	8
#Unassigned	535	332	61	332
F-measure	0.9999	0.9999	1.0000	0.9999
	OneBig		Pendigits	
	DEMass-DBSCAN	DBSCAN	DEMass-DBSCAN	DBSCAN
Runtime	1,145	8,544	91	204
#Cluster	9	9	47	65
#Unassigned	10,021	10,005	2,166	6,251
F-measure	1.00	1.00	0.65	0.75

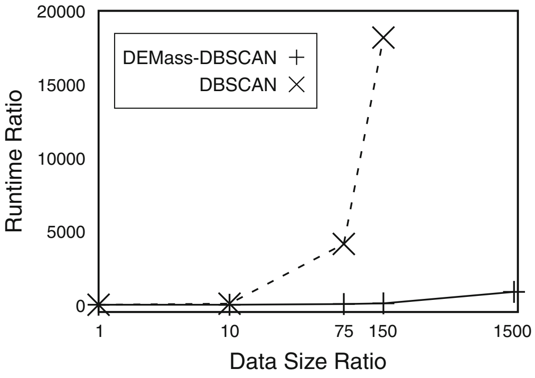


Fig. 4 Scale-up test: DEMass-DBSCAN vs DBSCAN in the 48-dimensional RingCurve-Wave-TriGaussian data set. Note that DBSCAN completed the task of the one million data set (at data size ratio =150) in 36 days versus DEMass-DBSCAN 4.5 h. Even with the 10 million data set, DEMass-DBSCAN completed it in 38 h, but it is infeasible to run DBSCAN

it was better than DBSCAN in all other measures: it had only 20 % instances unassigned, whereas DBSCAN had 57 % instances unassigned; DEMass-DBSCAN found 47 cluster, whereas DBSCAN detected 65.

In order to examine how well the algorithms scale up to large data sets, we used the 48-dimensional RingCurve-Wave-TriGuassian data set and increased the data size from 7,000 to 70,000, half-a-million, 1 million and 10 million. Figure 4 plotted runtime ratio versus data size ratio (1, 10, 75, 150 and 1,500) by using 7,000 as the base. The result showed that DEMass-DBSCAN had a sub-linear increase in runtime: the runtime ratio increased from 1 to 101 when the data size ratio increased from 1 to 150. In contrast, DBSCAN's runtime ratio increased from 1 to 18,000 with the same increase in data size ratio. DEMass-DBSCAN was faster than DBSCAN by a factor of 193 when the one million data set is used. Even the data size was increased by a factor of 1,500, and the runtime of DEMass-DBSCAN increased by a factor of 862 only.

Table 8 Data sets used for the anomaly detection task for comparing DEMass-LOF with LOF

Data sets	Size *n*	#*d*	Anomaly class
Http	567,497	3	Attack (0.4 %)
ForestCover (FC)	286,048	10	Class 4 (0.9 %) versus class 2
Mulcross	262,144	4	2 Clusters (10 %)
Smtp	95,156	3	Attack (0.03 %)
Shuttle	49,097	8	Classes 2,3,5,6,7 (7 %)

Table 9 Compare LOF and DEMass-LOF in terms of AUC (Area Under ROC Curve) and time (in seconds)

	AUC				Time (s)			
	DEMass-LOF		LOF		DEMass-LOF		LOF	
	$h = 1$	$h = 4$	$k = 10$	$k = 60$	$h = 1$	$h = 4$	$k = 10$	$k = 60$
Http	0.99	0.93	0.44	0.35	19	42	18,913	19,818
FC	0.74	0.77	0.57	0.58	39	40	10,835	11,147
Mulcross	0.96	0.09	0.59	0.59	12	53	5,432	5,486
Smtp	0.29	0.89	0.32	0.85	2	5	540	552
Shuttle	0.94	0.71	0.55	0.62	5	12	368	380

AUC = 1 is the perfect detection performance and AUC = 0 is the worst. The default settings for DEMass-LOF were $h = 1$, $\psi = 256$ and $t = 100$ which were used for all data sets. The parameters k (for LOF) and h (for DEMass-LOF) were changed in order to explore a better result

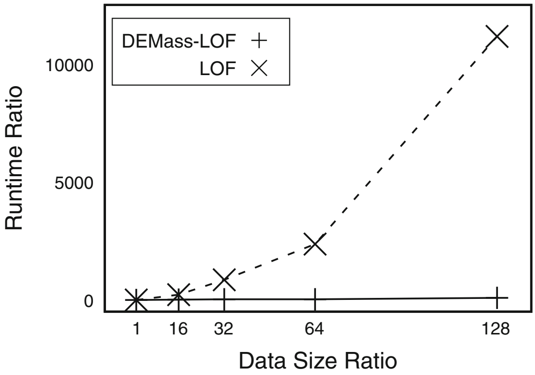
6.2 DEMass-LOF versus LOF

For anomaly detection tasks, we compare LOF with DEMass-LOF in this section. Table 8 provides the properties of the data sets used. These are the five largest data sets used by [35]. Note that Http and Smtp are subsets of the network intrusion data set used in KDDCUP 99 [40] and an anomaly data generator, ‘Mulcross’ [27], is used to generate a synthetic data set. All the data sets used have nearly fifty thousand or more instances, with the largest up to half-a-million instances. The default settings for DEMass-LOF were $\psi = 256$ and $t = 100$.

Table 9 compares LOF with DEMass-LOF in terms of detection performance AUC and time. DEMass-LOF using either h=1 or 4 obtained better AUC results than LOF. It is interesting to note that DEMass-LOF achieved extreme results in the Smtp and Mulcross data sets between the two h settings, and it behaved differently in these two data sets, where a low h setting is better in Mulcross but a high h setting is better in Smtp. This is because the two data sets have two different types of anomalies: clustered and scattered anomalies [24,25]. Mulcross has clustered anomalies, i.e., outlying clusters with high density but a small number of instances. DEMass-LOF with a high h setting (i.e., h =4) regarded these anomaly clusters more ‘normal’ than normal instances, which was reflected in the result: AUC = 0.09. In contrast, the Smtp data set has scattered anomalies which are isolated outlying instances around normal clusters. This scenario requires a high h setting in order for DEMass-LOF to compute the right densities for these anomalies.

LOF was not competitive, and the AUC results did not change much from the presented results even other k values were used (we had tried k =30, 40, 50, 80, 100, 120.)

Fig. 5 Scale-up test: LOF versus DEMass-LOF in Mulcross. The base for data size ratio is 8,192 instances, and the base for runtime ratio is the runtime on 8,192 instances



However, it shall be noted that LOF could achieve good detection accuracy with an appropriate k. For example, LOF obtained AUC = 0.99 when $k = 4,000$ was used in the shuttle data set. But similar search in the largest three data sets failed with out-of-memory problem even though the computer system was allocated 32 GB memory! This result reveals two universal problems with k-NN approaches like LOF: (i) an extensive parameter search is required to obtain good detection accuracy; this search adds a significant cost to the already long-runtime process. The total time cost is often prohibitive; and (ii) high memory requirement.

Table 9 also compares these detectors in terms of processing time. DEMass-LOF was one to three orders of magnitude faster than LOF in these data sets.

Figure 5 shows the runtime of both algorithms when scaling from 8,192 instances up to a million instances in the Mulcross data set. The data size was increased by a factor of 16, 32, 64, 128 from 8,192 instances. DEMass-LOF increased its runtime by a factor of 11, 23, 25 and 86, respectively. In contrast, LOF increased its runtime by a factor of 217, 845, 2,371 and 11,173, respectively. At data size ratio = 128, which has a million instances, LOF completed the task in 28 h whereas DEMass-LOF accomplished it in 45 seconds!

6.3 DEMass-Bayes versus Bayesian classifiers

In this subsection, we compare the performance of DEMass-Bayes with five existing Bayesian classifiers: NB-GD, NB-KDE, NB-Disc, BayesNet and AODE.

For better estimation of multidimensional density, we need sufficient training data. Hence, we chose large data sets with size $n > 10{,}000$. We tested on 10 data sets with different sizes, dimensions, number of classes and class distributions. The properties of the data sets are provided in Table 10.

Out of 10 data sets used, Wave, RingCurve and OneBig [26] are synthetic and the rest are the real data sets from UCI Machine Learning Repository [14]. RingCurve and Wave are the subsets of RingCurve-Wave-TriGaussian data set shown in the 'Appendix'. OneBig and Pendigits are the same data sets as used in Sect. 6.1. In OneBig, noise in the data set is treated as a separate class; hence, it has 10 classes. Magic04 has two classes with 12,332 and 6,688 instances, and Mammography has two classes with 10,923 and 260 instances. Letters is a data set of 26 characters (classes) with approximately 750 data instances in each class. Out of seven classes in Shuttle, approximately 80 % of the data belongs to the first class whereas the smallest class has 10 instances only. MiniBooNE is a dataset from an experiment to distinguish electron neutrinos (signal) from muon neutrinos (background). The

Table 10 Data sets used in classification task to compare the performance of DEMass-Bayes with five existing Bayesian classifiers

Data sets	Size n	$\#d$	$\#c$
CoverType	581,012	10	7
MiniBooNE	129,596	50	2
OneBig	68,000	20	10
Shuttle	58,000	8	7
Letters	20,000	16	26
RingCurve	20,000	2	2
Wave	20,000	2	2
Magic04	19,020	10	2
Mammography	11,183	6	2
Pendigits	10,992	16	10

class distribution is approximately 7:3. CoverType is a data set of forest cover type with seven classes. It is the biggest data set used, having more than half-a-million instances. The class distribution is unbalanced as two classes have more than two hundred thousand instances each and the smallest class has 2,747 instances.

In classification, it is important to estimate the density distribution of classes in the data space as good as possible in order to separate them. To achieve better classification accuracy, more samples are required to grow trees further to capture the detailed information about the local class distributions. The larger the sample size to build $h{:}d\text{-}Trees$, the better the density estimation. Hence, two variants of DEMass-Bayes are used: one employs a fixed-size sub-sample, and the other uses the entire training data.

1. DEMass-Bayes: A sub-sample $\mathcal{D} \subset D$ ($|\mathcal{D}| = \psi < n$) was used to build each $h{:}d\text{-}Tree$. The sub-sampling size (ψ) was set to 10,000 as default.
2. DEMass-Bayes$'$: The entire training set ($\psi = n$) was used to build each $h{:}d\text{-}Tree$.

The other two parameters h and t were set as default to 10 and 100, respectively.

All the other algorithms were executed with the default parameter settings except BayesNet. For BayesNet, the parameter '*maximum number of parents*' was set to 100 to examine whether a large number of parents produces better accuracy, and the parameter '*initialise as naive Bayes*' was set to '*false*' to initialise an empty network structure. The other parameters were set to default values.

We normalised the data in the range of [0–1] to avoid attributes with large values affecting volume calculation.

Since AODE cannot handle continuous-valued attributes, we discretised the attributes using the method proposed in [13]. BayesNet does discretisation before building the classification model.

We performed 10-fold cross-validation and reported the average accuracy and average runtime. The classification accuracies (%) and runtime (seconds) are provided in Tables 11 and 13, respectively.

A statistical test based on two standard errors was performed to examine whether the difference in classification accuracies between two classifiers is significant. The win:loss:draw counts of both variants of DEMass-Bayes against the existing Bayesian classifiers are reported in Table 12. A win or loss was counted if the difference was significant; otherwise, it was a draw.

Table 11 Average classification accuracies (%) over a 10-fold cross-validation for DEMass-Bayes′, DEMass-Bayes and existing Bayesian classifiers: BayesNet, AODE, NB-KDE, NB-GD and NB-Disc

Data sets	DEMass-Bayes′	DEMass-Bayes	Bayes Net	AODE	NB-KDE	NB-GD	NB-Disc
CoverType	**92. 05***	81.42†	87.54	72.89	66.72	63.05	66.56
MiniBooNE	88.13†	85.50†	**90.19**	89.58	86.06	83.40	86.18
OneBig	99.93†	99.93†	**99.99**	99.69	99.98	99.89	99.98
Shuttle	99.86†	99.86†	**99.92**	99.85	92.67	85.66	94.40
Letters	**92.27***	91.71*	86.76	88.81	74.20	64.01	74.01
RingCurve	**100.00***	**100.00***	99.96	99.98	99.26	90.11	99.38
Wave	**100.00***	**100.00***	78.27	78.50	77.91	66.80	78.27
Magic04	**84.17***	83.23	83.36	83.00	76.12	72.69	77.78
Mammography	**98.64**	**98.64**	98.48	98.42	97.86	95.68	97.50
Pendigits	**98.87***	**98.87***	96.56	97.84	88.64	85.75	87.78
Average	**95.39**	93.92	92.10	90.86	85.94	80.70	86.18

* (†) represents the significantly better (worse) predictive accuracy of DEMass-Bayes′ or DEMass-Bayes over the best accuracy of the existing Bayesian classifiers based on the two-standard-error significant test
Boldface represents the best accuracy achieved in each data set

Table 12 Win:Loss:Draw counts of DEMass-Bayes′ and DEMass-Bayes against the other contenders in terms of accuracy based on the two-standard-error significance test

Contenders	DEMass-Bayes′	DEMass-Bayes
BayesNet	6:3:1	4:4:2
AODE	7:1:2	6:1:3
NB-KDE	9:1:0	8:2:0
NB-GD	10:0:0	10:0:0
NB-Disc	9:1:0	8:2:0

The results in Tables 11 and 12 show that DEMass-Bayes′ yielded better classification accuracies in most of the data sets. Even the sub-sample version DEMass-Bayes produced competitive classification accuracies to the existing Bayesian classifiers. DEMass-Bayes′ had six wins, three losses and one draw against BayesNet; seven wins, one loss and two draws against AODE; nine wins and one loss against NB-KDE and NB-Disc; and all ten wins over NB-GD. Similarly, DEMass-Bayes had four wins and four losses against BayesNet; six wins and one loss against AODE; eight wins and two losses against NB-KDE and NB-Disc; and all ten wins against NB-GD.

Both variants of DEMass-Bayes outperformed existing Bayesian classifiers in four data sets namely Pendigits, Wave, RingCurve and Letters. In case of Wave and Letters, they had large improvement in accuracy over existing Bayesian classifiers by 20 and 5 %, respectively. They had slightly poorer accuracy than the best existing Bayesian classifier (BayesNet) in three data sets—MiniBooNE, OneBig and Shuttle.

Wave is a synthetic data set of two parallel waves as shown in Fig. 10b in the ‘Appendix’. In this data set, the prediction decision of NB-GD and NB-KDE depends on the likelihood on the y-dimension only as the classes are equally likely everywhere on the x-dimension. The same applies to NB-Disc, AODE and BayesNet which used a supervised discretisation method; as a result, the values in the x-dimension cannot be discretised, and they are grouped

Table 13 Average runtime (in seconds) over a 10-fold cross-validation for DEMass-Bayes′, DEMass-Bayes and existing Bayesian classifiers: BayesNet, AODE, NB-KDE, NB-GD and NB-Disc

Data sets	DEMass-Bayes′	DEMass-Bayes	Bayes Net	AODE	NB-KDE	NB-GD	NB-Disc
CoverType	1021.5	163.0	403.6	10.8	96.3	15.3	55.2
MiniBooNE	474.6	91.8	324.2	11.8	831.6	17.0	45.1
OneBig	122.3	22.3	432.5	3.9	253.0	3.2	11.9
Shuttle	35.0	17.5	8.4	0.9	1.5	0.9	2.7
Letters	48.2	22.2	5.5	0.8	2.5	0.9	1.2
RingCurve	6.4	5.7	0.3	0.2	2.4	0.2	0.2
Wave	6.8	5.8	0.3	0.2	2.5	0.1	0.3
Magic04	13.5	10.7	1.8	0.4	8.8	0.3	0.9
Mammography	4.7[a]	6.7[a]	0.3	0.3	0.5	0.1	0.2
Pendigits	12.5[b]	12.5[b]	2.3	0.4	1.6	0.3	0.6

[a] In each fold of a 10-fold cross-validation of Mammography, there are 10,065 instances in training set. DEMass-Bayes samples 10,000 from 10,065 instances to build each tree, whereas DEMass-Bayes′ uses the entire 10,065 instances. The tree-building time of DEMass-Bayes (with 10,000 instances) and DEMass-Bayes′ (with 10,065 instances) is almost the same. DEMass-Bayes is slower than DEMass-Bayes′ because of the sampling time

[b] In case of Pendigits, both DEMass-Bayes and DEMass-Bayes′ use the entire training set of 9893 instances in each fold to construct each tree as $\psi (=10{,}000) > n (=9{,}893)$

as a single block. The accuracies of ADOE and BayesNet were increased to 97.35 and 97.34 %, respectively, with unsupervised 10-bin equal-frequency discretisation [7], but they were still significantly worse than that of DEMass-Bayes. DEMass-Bayes, which estimates the multidimensional likelihood directly considering both the dimensions at once, models the distribution well. Hence, it produced significantly better accuracy than the other Bayesian classifiers in the Wave data set.

The biggest difference between DEMass-Bayes and DEMass-Bayes′ was observed in the CoverType data set. Even in this data set, DEMass-Bayes produced significantly better classification accuracy than the existing Bayesian classifiers except BayesNet. It produced lower accuracy than BayesNet because the sample size was not enough. More samples are required to grow the trees further to model the distribution well if the class distribution in the feature space is complex. A detailed discussion will be provided in Sec. 6.3.2.

Table 13 shows that DEMass-Bayes was generally slower than the other Bayesian classifiers in terms of runtime in smaller data sets. However, it should be noted that the existing Bayesian classifiers assume some kind of conditional independence and estimate the simplified surrogates of $p(\mathbf{x}|y)$ one dimension at a time. In contrast, the proposed method estimates $p(\mathbf{x}|y)$ in multidimensional space directly from the given training data without making any explicit assumptions. Nevertheless, the sub-sample version DEMass-Bayes was an order of magnitude faster than BayesNet and NB-KDE in some large data sets such as MiniBooNE and OneBig; DEMass-Bayes and BayesNet were in the same order of magnitude in CoverType. Similarly, DEMass-Bayes′, BayesNet and NB-KDE were in the same order of magnitude in the MiniBooNE and OneBig data sets. Note that the presented runtime results for AODE did not include the discretisation time that was done as a preprocessing step. The discretisation time was significantly large in large data sets. For example, it took 52 seconds in the largest data set, CoverType.

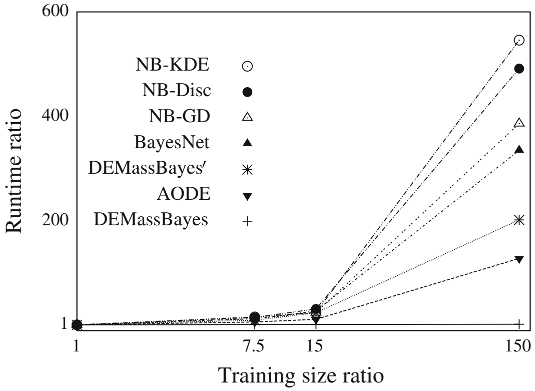


Fig. 6 Scale-up test for training time: DEMass-Bayes versus existing Bayesian Classifiers in the 48-dimensional RingCurve-Wave-TriGaussian data set. The base for data size ratio is 70,000 instances, and the base for runtime ratio is the runtime on 70,000 instances. The training size ratio is on a logarithmic scale of base 10

This runtime result does not provide a full picture about the time complexities of DEMass-Bayes′ and DEMass-Bayes. Therefore, we had conducted a scale-up test of the algorithms to present a more accurate idea about the time complexity in the following subsection.

6.3.1 Scale-up test

In order to examine how well the classifiers scale up to large data sets, we used the 48-dimensional RingCurve-Wave-TriGaussian data set, used in Sect. 6.1. Data size was increased from 70,000 to half-a-million, 1 million and 10 million.

Figure 6 shows the increase in training time of both variants of DEMass-Bayes and the existing Bayesian classifiers. Note that AODE had an unfair advantage over the other contenders in terms of runtime because it is the only algorithm which does not include the additional discretisation time in the preprocessing step which increased linearly with the training data size.

With the increase in data size by a factor of 7.5, 15 and 150, DEMass-Bayes increased its runtime to learn a classification model by a factor of 1.6, 1.7 and 2.1, respectively. The closest contender AODE increased its runtime by a factor of 6, 11 and 128, followed by BayesNet (12, 27, 335), NB-GD (12, 24, 386), NB-Disc (15, 31, 491) and NB-KDE (16, 24, 545). DEMass-Bayes′ increased its runtime by a factor of 9, 23 and 201, respectively.

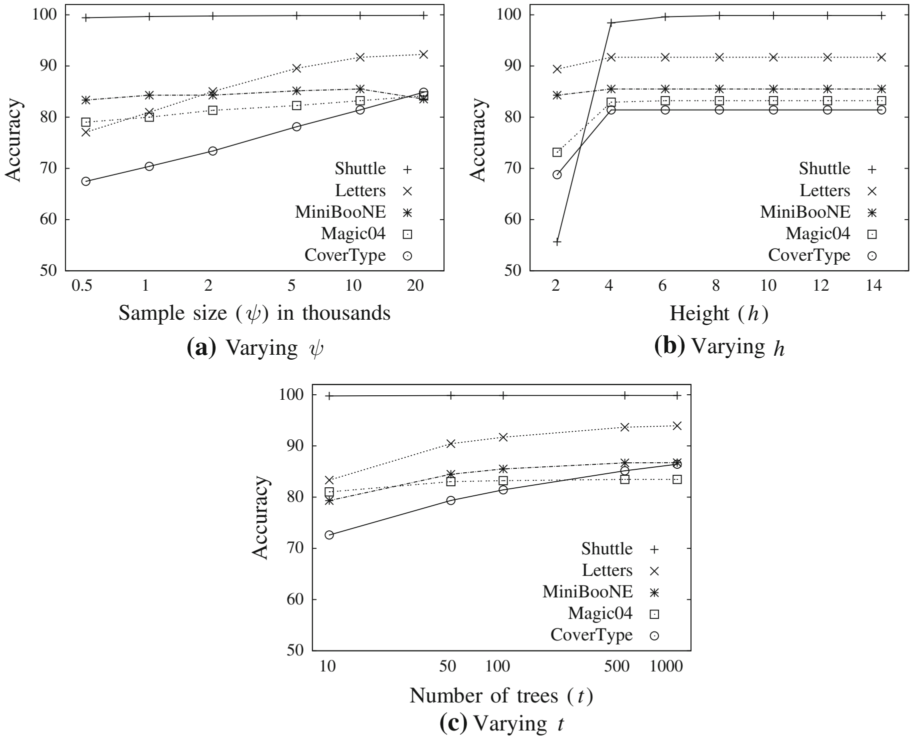
The training time of DEMass-Bayes was constant; some fluctuations in the runtime were due to the memory management issue of Java, especially when the process demands high memory. The training time complexity of DEMass-Bayes′ was slightly worse than that of AODE but better than that of the other contenders (BayesNet and NB).

In a nutshell, DEMass-Bayes has a better scale-up capability than the existing Bayesian classifiers in big data. The result is consistent with the time complexities presented in Table 5.

6.3.2 Sensitivity of parameters

In order to examine the effect of the parameters, i.e., sample size ψ, height h and number of trees t, on the classification accuracy of DEMass-Bayes, we ran three experiments on five real large data sets, namely CoverType, MiniBooNE, Shuttle, Letters and Magic04:

- Vary sample size ψ with a fixed number of trees ($t = 100$) and height ($h = 10$).
- Vary height h with a fixed sample size ($\psi = 10,000$) and number of trees ($t = 100$).
- Vary number of trees t with a fixed sample size ($\psi = 10,000$) and height ($h = 10$).



(a) Varying ψ

(b) Varying h

(c) Varying t

Fig. 7 Effect of parameters sample size (ψ), height (h) and number of trees (t) on the accuracy of DEMass-Bayes in Shuttle, Letters, MiniBooNE, Magic04 and CoverType. The horizontal axes of sample size (ψ) and the number of trees (t) are on logarithmic scales of base 2 and base 10, respectively, in **(a)** and **(c)**

Figure 7 shows the effect of the three parameters on the classification accuracy of DEMass-Bayes.

With the increase in sample size ψ as shown in Fig. 7a, the accuracy increased up to a certain point ($\psi = 10,000$) and then remained almost flat in all data sets, except CoverType. In CoverType, accuracy kept increasing because the two biggest classes have more than two hundred fifty thousand instances each and the continuous accuracy improvement is a result of improved accuracy for these two classes. These two big classes are highly overlapped. The two distributions can be better distinguished if the regions around the overlapped areas are small. If there are more samples to grow trees further, better estimation for $p(\mathbf{x}|y)$ can be achieved and the classification accuracy can be improved. The accuracy was increased up to 84.87 % with $\psi = 20{,}000$ and 88.53 % with $\psi = 50{,}000$. Figure 8 shows the improvement in accuracy in CoverType when the sample size was increased. DEMass-Bayes using 2,50,000 instances produced the same result as DEMass-Bayes′ using the entire training set of 581,012 instances.

When h was increased as shown in Fig. 7b, the accuracy was increased initially and then remained almost constant after $h = 6$. In Shuttle, there were small improvements until $h = 10$. If there are not enough samples to grow a tree further, tree building stops early. Hence, increasing h does not affect the performance if it is already set to a sufficiently high value.

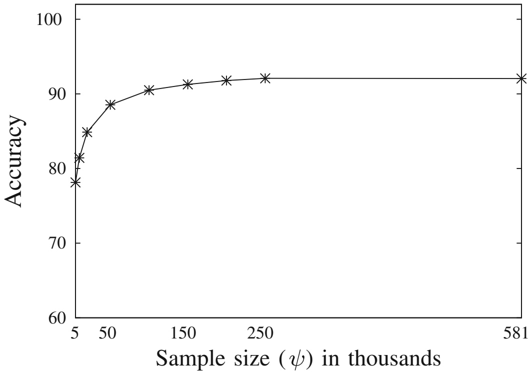


Fig. 8 Increase in accuracy of DEMass-Bayes with increase in sample size in the CoverType data set. Accuracies are measured with ψ = 5,000, 10,000, 20,000, 50,000, 100,000, 150,000, 200,000, 250,000 and 581,012 (n)

When the number of trees t was increased as shown in Fig. 7c, the accuracy increased initially and remained constant after reaching a certain point ($t = 100$) except in CoverType where accuracy kept increasing. Increasing the number of trees provides better approximation of the local density in the overlapped areas and improves the classification accuracy of the two largest classes.

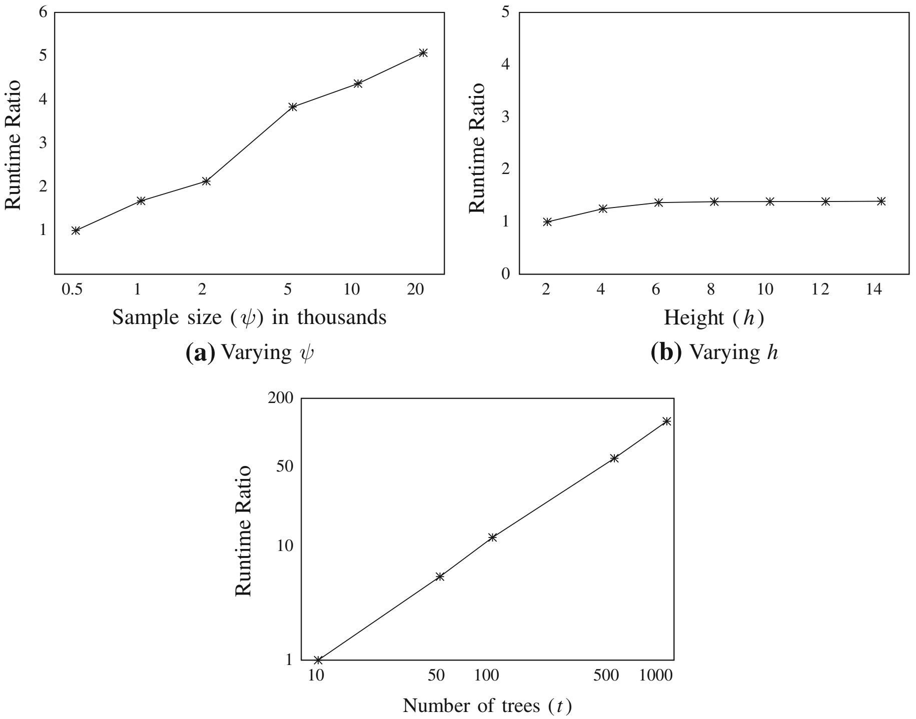
Figure 9 shows the effect of parameters on the average runtime over a 10-fold cross-validation in the biggest data set, CoverType. The runtime in the plots is presented as a runtime ratio to show the increase in runtime when parameters were increased. The bases for the runtime ratio while varying ψ, h and t are the total runtime (including training and testing) for $\psi = 500$, $h = 2$ and $t = 10$, respectively. The runtime increased linearly with t and sub-linearly with ψ. Since the number of dimensions was 10 and $\psi = 10{,}000$, the tree building stopped early when all the instances had been separated. Hence, as shown in Fig. 9b, after reaching a certain point ($h = 6$), further increase in h did not affect the runtime.

7 Discussion

What we have presented is the first density estimation method that utilises no distance measures. It potentially solves fundamental problems such as the curse of dimensionality in which the use of a distance measure plays a key part in creating the problem [4,18]. Although the current version of DEMass cannot deal with high-dimensional problems because of the grid-based implementation, a non-grid-based implementation that utilises no distance measure can potentially break the curse of dimensionality.

There are significant improvements of nearest neighbour search in recent times. For example, indexing schemes to speed up nearest neighbour search such as Cover Trees [5] and M-Trees [8] are claimed to have time complexity significantly better than $O(n^2)$. Indexing schemes such as Cover Trees and M-Trees rely on distance-based pruning methods in both the index tree construction and range query processes. Distance-based pruning methods cannot scale up to massive data, and they are known to be inefficient even for a moderate number of dimensions. Thus, it is unlikely that any of the recent indexing schemes can be used to speed up nearest neighbour search to the level that has been achieved already by DEMass-DBSCAN and DEMass-LOF, especially in big data.

An exception to the above indexing schemes is PINN (projection-indexed nearest neighbours) [37] which employs Random Projection [9,20] to project a high-dimensional space to a low-dimensional space in order to get an accurate approximation for k-NN distances for density calculation within LOF. This indexing scheme has reduced LOF's time com-



(a) Varying ψ

(b) Varying h

(c) Varying t

Fig. 9 Effect of parameters sample size (ψ), height (h) and number of trees (t) on the runtime of DEMass-Bayes in the CoverType data set. The horizontal axes of sample size (ψ) and the number of trees (t) are on logarithmic scales of base 2 and base 10, respectively, in (**a**) and (**c**). *The vertical axis* for runtime ratio when varying t is on a logarithmic scale of base 10 in (**c**)

plexity from $O(n^2)$ to sub-quadratic. It is interesting to investigate how PINN may help to enable DEMass-LOF to deal with high-dimensional problems, when DEMass-LOF does not involve distance calculation and has already achieved sub-linear time complexity without any indexing scheme.

Note that the purpose of trees used in DEMass differs from that used for Cover Trees or M-Trees. Trees in DEMass are used to estimate mass and density, the core computation process. In contrast, Cover Trees or M-Trees are indices used to speed up nearest neighbour search. The indices are required because the core computation, i.e., the requirement to calculate distance for every pair of instances, is slow. In other words, one uses trees directly in the core process, and the other uses trees to aid the core process where trees are not used in the actual computation of distance.

The cost of KDE estimation can be lowered, for example, by reducing the given data set D to some 'representative' subset, where each representative kernel is derived from a sub-sample using a maximum likelihood method such as an expectation–maximisation (EM) algorithm [10,16]. This reduces the KDE estimation time, but it comes with a cost of an expensive preprocessing step.

DENCLUE [19], a generic density-based algorithm, builds a density distribution from data and then uses a threshold to determine clusters—all connected points above the threshold form a cluster. DBSCAN is a special case of DENCLUE. DEMass-DENCLUE has exactly the same procedure as DEMass-DBSCAN, where $MinPts$ or the equivalent density threshold stated in Sec. 5.1 is employed as the threshold.

It is possible to use neighbours to compute LOF for DEMass-LOF. However, the runtime advantage over LOF will be significantly reduced because of the additional computations required to calculate the density of each neighbour, even though it does not need to find neighbours based on distance calculations.

Other k-NN-based anomaly detectors (e.g., [2,3]) have employed some search space pruning methods to reduce the cost of the nearest neighbour search to achieve near-linear time complexity. DEMass can be similarly applied in these algorithms, like what we have done to LOF, without distance calculation, and the resultant algorithms will have better time and space complexities.

Feature Bagging [23] has been proposed to improve the detection accuracy of LOF by building multiple models, where each model is constructed from a subset of randomly chosen features, and the final prediction is combined by averaging the anomaly scores from individual models. While the method has been shown to produce a small improvement over a single model LOF in terms of AUC, it requires a substantial increase in runtime (proportional to the number of models required). This adds to the high computational cost of LOF we have already discussed in Sect. 6.2.

It is possible to avoid density estimation. For example, uLSIF (unconstrained least-squares importance fitting) [17], OSVM (one-class support vector machine) [28] and SVDD (support vector data description) [32] are anomaly detectors which do not employ density estimators. uLSIF directly estimates the density ratio between the training set and test and uses the density ratio as the anomaly score. OSVM and SVDD find the smallest region that covers the majority of the normal instances and regard instances outside the region as anomalies. uLSIF is found to perform comparably with LOF and run significantly faster [17]. It is interesting to compare our density-based approach DEMass-LOF with these non-density-based approaches to determine their relative strengths and weaknesses.

DEMass-Bayes is the first Bayesian classifier with the constant training time complexity in the number of training instances. It is the ideal classifier for big data and data streams where there are potentially infinite data. It is interesting to note that DEMass-Bayes has the flexibility to trade off between accuracy and the computational cost as required. The time and space required can be reduced by setting lower values for the parameters t and ψ if a higher misclassification rate can be tolerated. These parameters can be set to higher values in order to achieve better classification accuracy at the expense of higher computational cost. The current implementation does not suit problems with small data sets because the estimation relies on sufficiently large sample in order to model accurately the local density distributions of classes in the data space.

We had also experimented a version of DEMass-Bayes which samples ψ instances per class, i.e., $|\mathcal{D}_{i,y}| = \psi$. With this formulation, the likelihood $p(\mathbf{x}|y)$ is estimated as follows:

$$p(\mathbf{x}|y) \equiv \bar{f}_{\mathbf{m}}(\mathbf{x}|y) = \frac{1}{t}\sum_{i=1}^{t}\frac{\mathbf{m}(T_{i,y}(\mathbf{x}))}{\psi v_{i,y}} \tag{10}$$

where $T_{i,y}(.)$ is constructed from $\mathcal{D}_{i,y}$, which is a subset of ψ samples from class y. This formulation forces uniform class distribution in building h:d-$Trees$. By sampling equal number of instances from every class, an instance in smaller class is sampled more frequently than the one in larger class. The 'forced' uniform class distribution has distorted the accurate estimation of local class density in overlapping regions. The formulation discussed in Sect.

5.3 employs approximately the original class distribution. It provides a better estimation of $p(\mathbf{x}|y)$ than Eq. (10) and runs significantly faster too.

DEMass sets a new benchmark of what density-based algorithms can achieve. In contrast to the density-based approaches, mass-based approaches [34,35] solve problems without the use of a density estimator. Mass-based approaches have been shown to perform better than the current density-based approaches in terms of time and space complexities. It is thus interesting to compare the new benchmark achieved by DEMass-density-based approaches with mass-based approaches.

The current implementation of DEMass has two limitations. First, it has step subdivisions controlled by a global parameter h. The limited possible steps may be too coarse for some applications, and the setting is not adaptive to local variations in density. Second, the grid-based implementation carries all the limitations associated with grid-based approaches, especially dealing high-dimensional problems. All these limitations can be overcome by using a non-grid method which is adaptive to the local data distribution. This non-grid-based implementation will eliminate one global parameter and potentially tackle high-dimensional problems more effectively.

8 Conclusions and future work

The new density estimation method we introduced has two unique features which cannot be found in existing density estimation methods. First, it is the first density estimator that utilises no distance measures. Second, it has average case sub-linear time complexity and constant space complexity in the number of instances. Existing density estimators must use a distance measure and have time and space complexities a lot worse than linear. The time and space complexities achieved set a new benchmark for density-based algorithms, of what previously thought impossible.

The asymptotic analysis reveals that the new density estimator has the same characteristic as KDE, i.e., both have a smoothing parameter used to trade-off between systematic error (bias) and random error (variance).

Making full use of the features in the new density estimator, we show that two current algorithms, in the unsupervised learning setting from two key areas of data mining, can be significantly simplified through set-based definitions rather than the current point-based definitions. This has directly contributed to their significantly improved time complexities. In the supervised learning setting, DEMass enables direct estimation of multidimensional likelihood $p(\mathbf{x}|y)$ for the first time, without any assumptions.

Our evaluation shows that the new density estimator not only successfully replaces existing density estimators in three density-based algorithms, DBSCAN, LOF and Bayesian classifiers, but (a) reduces the runtime of DBSCAN and LOF to become algorithms with sub-linear time complexity and (b) scales down existing Bayesian classifiers' best training time complexity from linear to constant. In addition, DEMass-DBSCAN, DEMass-LOF and DEMass-Bayes often achieve equivalent or better task-specific performances than DBSCAN, LOF and existing Bayesian classifiers, respectively.

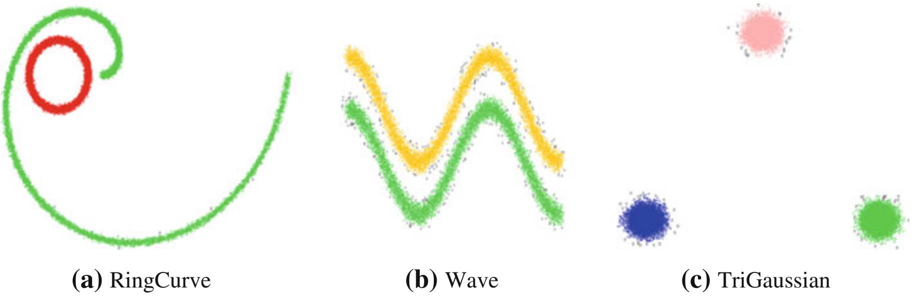
Our result implies that most, if not all, density-based algorithms can reap the immediate benefit of significantly lowering their time complexities by simply replacing the existing density estimators with the new one, with a potential further improvement in the task-specific performance.

Future work has three directions. First, we will apply the new density estimator in existing algorithms in more areas. We will ascertain whether there are areas in which the new density estimator cannot replace existing density estimators. Second, we will compare DEMass-density-based approaches with mass-based approaches to determine their relative strengths and weaknesses. Third, we will explore DEMass's ability to deal with high-dimensional problems.

Acknowledgments This work is partially supported by the Air Force Research Laboratory, under agreement# FA2386-10-1-4052. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Takashi Washio is partially supported by JSPS (Japan Society for the Promotion of Science) Grant-in-Aid for Scientific Research (B)# 22300054. Sunil Aryal is supported by Monash University Postgraduate Publications Award to complete the work on DEMass-Bayes. Xiao Yu Ge assisted in experiments using ELKI. Hiroshi Motoda, Zhouyu Fu and the anonymous reviewers had provided many helpful comments to improve this paper.

Appendix: Data characteristic of the RingCurve-Wave-TriGaussian data set

The characteristic of the RingCurve-Wave-TriGaussian data set, used in Sect. 5.1, is shown in Fig. 10. Each of the Ring-Curve, Wave and Triangular-Gaussian is a two-dimensional data set, and together, there is a total of seven clusters. Each cluster has 10,000 instances. When used in the scale-up experiment, the data size in each cluster was scaled by a factor of 0.1, 1, 75, 150 to 1,500.



(a) RingCurve **(b)** Wave **(c)** TriGaussian

Fig. 10 Scatter plot of the clusters in the RingCurve-Wave-TriGaussian data set

References

1. Achtert E, Kriegel H-P, Zimek A (2008) ELKI: a software system for evaluation of subspace clustering algorithms. In: Proceedings of the 20th international conference on scientific and statistical database management, pp 580–585
2. Angiulli F, Fassetti F (2009) DOLPHIN: an efficient algorithm for mining distance-based outliers in very large datasets. ACM Trans Knowl Discov Data 3(1):4:1–4:57
3. Bay SD, Schwabacher M (2003) Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp. 29–38
4. Beyer KS, Goldstein J, Ramakrishnan R, Shaft U (1999) When is "nearest neighbor" meaningful? In: Proceedings of the 7th international conference on database theory, pp 217–235
5. Beygelzimer A, Kakade S, Langford J (2006) Cover trees for nearest neighbor. In: Proceedings of the 23rd international conference on machine learning, pp 97–104
6. Breunig MM, Kriegel H-P, Ng RT, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of ACM SIGMOD international conference on management of data, pp 93–104

7. Catlett J (1991) On changing continuous attributes into ordered discrete attributes. In: Proceedings of the European working session on learning, pp 164–178
8. Ciaccia P, Patella M, Zezula P (1997) M-tree: an efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd international conference on very large data, bases, pp 426–435
9. Deegalla S, Bostrom H (2006) Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification. In: Proceedings of the 5th international conference on machine learning and applications, IEEE Computer Society, Washington, pp 245–250
10. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J Roy Stat Soc Ser B 39(1):1–38
11. Dougherty J, Kohavi R, Sahami M (1995) Supervised and unsupervised discretization of continuous features. In: Proceedings of the 12th international conference on machine learning, Morgan Kaufmann, pp 194–202
12. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of KDD, AAAI Press, pp 226–231
13. Fayyad UM, Irani KB (1995) Multi-interval discretization of continuous valued attributes for classification learning. In: Proceedings of 14th international joint conference on artificial intelligence, pp 1034–1040
14. Frank A, Asuncion A (2010) UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences. URL: http://archive.ics.uci.edu/ml
15. Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29:131–163
16. Hastie T, Tibshirani R, Friedman J (2001) Chapter 8.5 the EM algorithm. In The elements of statistical learning, pp 236–243
17. Hido S, Tsuboi Y, Kashima H, Sugiyama M, Kanamori T (2011) Statistical outlier detection using direct density ratio estimation. Knowl Inf Syst 26(2):309–336
18. Hinneburg A, Aggarwal CC, Keim DA (2000) What is the nearest neighbor in high dimensional spaces? In: Proceedings of the 26th international conference on very large data bases, pp 506–515
19. Hinneburg A, Keim DA (1998) An efficient approach to clustering in large multimedia databases with noise. In: Proceedings of KDD, AAAI Press, pp 58–65
20. Johnson WB, Lindenstrauss J (1984) Extensions of Lipschitz mapping into Hilbert space. In: Proceedings of conference in modern analysis and probability, contemporary mathematics, vol 26. American Mathematical Society, pp 189–206
21. Langley P, Iba W, Thompson K (1992) An analysis of Bayesian classifiers. In: Proceedings of the tenth national conference on artificial intelligence, pp 399–406
22. Langley P, John GH (1995) Estimating continuous distribution in Bayesian classifiers. In: Proceedings of eleventh conference on uncertainty in artificial intelligence
23. Lazarevic A, Kumar V (2005) Feature bagging for outlier detection. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery and data mining, ACM, pp 157–166
24. Liu FT, Ting KM, Zhou Z-H (2010) On detecting clustered anomalies using sciforest. In: Proceedings of ECML PKDD, pp 274–290
25. Liu FT, Ting KM, Zhou Z-H (2012) Isolation-based anomaly detection. ACM Trans Knowl Discov Data 6(1):3:1–3:39
26. Nanopoulos A, Theodoridis Y, Manolopoulos Y (2006) Indexed-based density biased sampling for clustering applications. IEEE Trans Data Knowl Eng 57(1):37–63
27. Rocke DM, Woodruff DL (1996) Identification of outliers in multivariate data. J Am Stat Assoc 91(435):1047–1061
28. Schölkopf B, Platt JC, Shawe-Taylor JC, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443–1471
29. Silverman BW (1986) Density estimation for statistics and data analysis. Chapmal & Hall, London
30. Tan P-N, Steinbach M, Kumar V (2006) Introduction to data mining. Addison-Wesley, Reading
31. Tan SC, Ting KM, Liu FT (2011) Fast anomaly detection for streaming data. In: Proceedings of IJCAI, pp 1151–1156
32. Tax DMJ, Duin RPW (2004) Support vector data description. Mach Learn 54(1):45–66
33. Ting KM, Washio T, Wells JR, Liu FT (2011) Density estimation based on mass. In: Proceedings of the 2011 IEEE 11th international conference on data mining, IEEE Computer Society, pp 715–724
34. Ting KM, Wells JR (2010) Multi-dimensional mass estimation and mass-based clustering. In: Proceedings of IEEE international conference on data mining, pp 511–520
35. Ting KM, Zhou G-T, Liu FT, Tan SC (2012) Mass estimation. Mach Learn, pp 1–34. doi:10.1007/s10994-012-5303-x
36. Vapnik VN (2000) The nature of statistical learning theory, 2nd edn. Springer, Berlin
37. Vries TD, Chawla S, Houle M (2012) Density-preserving projections for large-scale local anomaly detection. Knowl Inf Syst 32:25–52

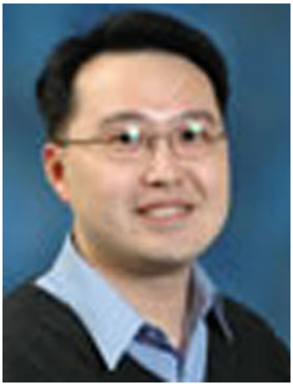
38. Webb GI, Boughton JR, Wang Z (2005) Aggregating one-dependence estimators. Mach Learn 58:5–24
39. Witten IH, Frank E, Hall MA (2011) Data mining: Practical machine learning tools and techniques, 3rd edn. Morgan Kaufmann, San Francisco
40. Yamanishi K, Takeuchi J-I, Williams G, Milne P (2000) On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. In: Proceedings of ACM SIGKDD international conference on knowledge discovery and data mining, pp 320–324

Author Biographies

Kai Ming Ting received his Ph.D. from the University of Sydney, Australia. He had worked at the University of Waikato and Deakin University before joining Monash University in 2001. He currently serves as the Associate Dean Research Training in Faculty of IT and an Associate Professor in Gippsland School of IT at Monash University. He is an associate editor for Journal of Data Mining and Knowledge Discovery. He had co-chaired the Pacific-Asia Conference on Knowledge Discovery and Data Mining and had served as a member of program committees for a number of conferences including ACM SIGKDD, IEEE ICDM and ICML. His research projects are supported by grants from Australian Research Council, US Air Force of Scientific Research (AFOSR/AOARD), Toyoto InfoTechnology Center and Australian Institute of Sport.

Takashi Washio received the Ph.D. degree in nuclear engineering from Tohoku University, Japan, in 1983, on the topic of process plant diagnosis based on qualitative reasoning. He is a professor in the Institute of Scientific and Industrial Research (ISIR), Osaka University. At ISIR, he works on the study of scientific discovery, graph mining and high-dimensional data mining. He received the best paper award from the Atomic Energy Society of Japan in 1996, the best paper award from the Japanese Society for Artificial Intelligence in 2001 and the Journal Award of Computer Aided Chemistry in 2002, Contribution Award from the Japanese Society for Artificial Intelligence in 2009. He is a member of the IEEE Computer Society.

Jonathan R. Wells worked as a Software Engineer in commercial environments for a number of years. He received his degrees at Monash University in Bachelor of Computing and Master of Information Technology (Research), and he is currently doing his PhD at the same university. His research interests are in the areas of data mining and cognitive science.



Fei Tony Liu received his Ph.D. in 2011 from Monash University. During his postgraduate studies, he was awarded with the Best Paper and Best Student Paper Awards in the Tenth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2006) and the Runner-Up Best Theoretical/Algorithms Paper Award in the IEEE International Conference on Data Mining (ICDM 2008). His research interests include ensemble learning, outlier detection and predictive classification.

Sunil Aryal received a BIT degree from Purbanchal University, Nepal, in 2005 and an MIT degree from University of Southern Queensland, Australia, in 2008. He worked as a Software Developer for a couple of years. He is currently a postgraduate research student at Monash University, Australia. His research interests include mass-based data mining, and mining large data and data streams.

MassBayes: A New Generative Classifier with Multi-dimensional Likelihood Estimation

Sunil Aryal and Kai Ming Ting

Gippsland School of Information Technology
Monash University, Australia
{sunil.aryal,kaiming.ting}@monash.edu

Abstract. Existing generative classifiers (e.g., BayesNet and AnDE) make independence assumptions and estimate one-dimensional likelihood. This paper presents a new generative classifier called *MassBayes* that estimates multi-dimensional likelihood without making any explicit assumptions. It aggregates the multi-dimensional likelihoods estimated from random subsets of the training data using varying size random feature subsets. Our empirical evaluations show that MassBayes yields better classification accuracy than the existing generative classifiers in large data sets. As it works with fixed-size subsets of training data, it has constant training time complexity and constant space complexity, and it can easily scale up to very large data sets.

Keywords: Generative classifier, Likelihood estimation, MassBayes.

1 Introduction

The learning task in classification is to learn a model from a labelled training set that maps each instance to one of the predefined classes. The model learned is then used to predict a class label for each unseen test instance. Each instance $\mathbf{x}$ is represented by a d-dimensional vector $\langle x_1, x_2, \cdots, x_d\rangle$ and given a class label $y \in \{y_1, y_2, \cdots, y_c\}$, where c is the total number of classes. The training set D is a collection of labelled instances $\{(\mathbf{x}^{(i)}, y^{(i)})\}$ $(i = 1, 2, \cdots, N)$.

The generative approach of classifier learning models the joint distribution $p(\mathbf{x}, y)$ and predicts the most probable class as:

$$\hat{y} = \arg\max_{y}\ p(\mathbf{x}, y) \tag{1}$$

Using the product rule, the joint probability can be factorised as:

$$p(\mathbf{x}, y) = p(y) \times p(\mathbf{x}|y) \tag{2}$$

Generative classifiers learn either the joint distribution $p(\mathbf{x}, y)$ or the likelihood $p(\mathbf{x}|y)$. However, estimating $p(\mathbf{x}, y)$ or $p(\mathbf{x}|y)$ directly from data using existing data modelling techniques is difficult. Density estimators such as Kernel Density Estimation [1], k-Nearest Neighbour [1] and Density Estimation Trees [2] are impractical in large data sets due to their high time and space complexities. The research has thus focused on learning one-dimensional likelihood to approximate $p(\mathbf{x}, y)$ in different ways.

J. Pei et al. (Eds.): PAKDD 2013, Part I, LNAI 7818, pp. 136–148, 2013.

Existing generative classifiers allow limited probabilistic dependencies among attributes and assume some kind of conditional independence. Different generative classifiers make different assumptions and allow different level of dependencies. They learn a network (or its simplification) of probabilistic relationship between the attributes and estimate the likelihood at each node given its parents from D (i.e., one-dimensional likelihood estimation). The joint distribution $p(\mathbf{x}, y)$ is estimated as the product of likelihood of each attribute given their parents in the network:

$$\hat{p}(\mathbf{x}, y) = p(x_1|\pi_1) \times p(x_2|\pi_2) \times \cdots \times p(x_d|\pi_d) \times p(y|\pi_y) \tag{3}$$

where π_i is $parent(x_i)$ and π_y is $parent(y)$.

Though these one-dimensional likelihood generative classifiers have been shown to perform well [3,4,5,6,7], we hypothesize that a multi-dimensional likelihood generative classifier will produce even better results.

In this paper, we propose an ensemble approach to estimate multi-dimensional likelihood without making any explicit assumption about attribute independence. The idea is to construct an ensemble of t multi-dimensional likelihood estimators using random sub-samples $\mathcal{D}_i \subset D$ $(i = 1, 2, \cdots, t)$. Each estimator estimates the multi-dimensional likelihood using a random subset of d attributes from $\mathcal{D}_i$. The average estimation from t estimators provides a good approximation of $p(\mathbf{x}|y)$. We call the resulting generative classifier *MassBayes*. It has constant space complexity and constant training time complexity because it employs a fixed-size training subset to build each of the t estimators.

The rest of the paper is structured as follows. Section 2 provides a brief overview of well-known generative classifiers. The proposed method is described in Section 3 followed by the implementation details in Section 4. The empirical evaluation results are presented in Section 5. Finally, we provide conclusions and directions for future research in Section 6.

2 Existing Generative Classifiers

Naive Bayes (NB) [3] is the simplest generative approach that estimates $p(\mathbf{x}, y)$ by assuming that the attributes are statistically independent given y:

$$\hat{p}(\mathbf{x}, y)_{NB} = p(y) \prod_{i=1}^{d} p(x_i|y) \tag{4}$$

Despite the strong independence assumption, it has been shown that NB produces impressive results in many application domains [3,4]. Its simplicity and clear probabilistic semantics have motivated researchers to explore different extensions of NB to improve its performance by relaxing the unrealistic assumption.

BayesNet [5] learns a network of probabilistic relationship among the attributes including the class attribute from the training data. Each node in the network is independent of its non-descendants given the state of its parents. At each node, the conditional probabilities with respect to its parents are learned from D. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y)_{BayesNet} = p(y|\pi_y) \prod_{i=1}^{d} p(x_i|\pi_i) \tag{5}$$

Learning an optimal network requires searching over a set of every possible network, which is exponential in d. It is intractable in high-dimensional problems [8]. NB is the simplest form of a Bayesian network, where each attribute is dependent on y only.

In another simplification of BayesNet, AnDE [7] relaxes the independence assumption by allowing dependency between y and a fixed number of privileged attributes or super-parents. The other attributes are assumed to be independent given the n super-parents and y. AnDE with $n = 0$, A0DE, is NB. AnDE avoids the expensive searching in learning probabilistic dependencies by constructing an ensemble of n-dependence estimators. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y)_{AnDE} = \sum_{s \in S^n} p(\mathbf{x}_s, y) \prod_{j \in \{1,2,\cdots,d\} \setminus s} p(x_j|\mathbf{x}_s, y) \tag{6}$$

where S^n is the collection of all subsets of size n of the set of d attributes $\{1, 2, \cdots, d\}$; and $\mathbf{x}_s$ is a n-dimensional vector of values of $\mathbf{x}$ defined by s.

It has been shown that A1DE and A2DE produce better predictive accuracy than the other state-of-the-art generative classifiers [6,7]. However, it only allows dependencies on a fixed number of attributes and y. Because of the high time complexity of $O\left(N\binom{d}{n+1}\right)$ [1] and space complexity of $O\left(c\binom{d}{n+1}v^{n+1}\right)$, where v is the average number of values for an attribute [7], only A2DE or A3DE is feasible even for a moderate number of dimensions. Furthermore, selecting an appropriate value of n for a particular data set requires a search.

AnDE and many other implementations of BayesNet require all the attributes to be discrete. The continuous-valued attributes must be discretised using a discretisation method before building a classifier.

3 MassBayes: A New Generative Classifier

Rather than aggregating an ensemble of n-dependence single-dimensional likelihood estimators, we propose to aggregate an ensemble of t multi-dimensional likelihood estimators where each likelihood is estimated using different random subsets of d attributes from data. The likelihood $p(\mathbf{x}|y)$ is estimated as:

$$\hat{p}(\mathbf{x}|y) = \frac{1}{t} \sum_{g \in G_t} p(\mathbf{x}_g|y) \tag{7}$$

where G_t is a collection of t subsets of varying sizes of d attributes; and $\mathbf{x}_g$ is a $|g|$-dimensional vector of values of $\mathbf{x}$ defined by g; and $1 \leq |g| \leq d$.

Each $p(\mathbf{x}_g|y)$ is estimated using a random subset of training instances $\mathcal{D} \subset D$, where $|\mathcal{D}| = \psi < N$.

$$\hat{p}(\mathbf{x}_g|y) = \frac{|\mathcal{D}_{y,\mathbf{x}_g}|}{|\mathcal{D}_y|} \tag{8}$$

[1] $\binom{d}{n}$ is a binomial coefficient of n out of d.

where $|\mathcal{D}_{y,\mathbf{x}_g}|$ is the number of instances having attribute values $\mathbf{x}_g$ belonging to class y in $\mathcal{D}$ and $|\mathcal{D}_y|$ is the number of instances belonging to class y in $\mathcal{D}$.

Rather than relying on a specific discretisation method in the preprocessing step, we propose to build a model directly from data, akin to an adaptive multi-dimensional histogram, to determine $\mathbf{x}_g$ which adapts to the local data distribution. The feature space partitioning we employed (to be discussed in Section 4) produces large regions in sparse area and small regions in the dense area of the data distribution.

Let $T(\cdot)$ be a function that divides the feature space into non-overlapping regions and $T(\mathbf{x})$ be the region where $\mathbf{x}$ falls. In a multi-dimensional space, each instance in $\mathcal{D}$ can be isolated by splitting only on few dimensions i.e., only a subset of d attributes ($g \subset \{1, 2, \cdots, d\}$) is used to define $T(\mathbf{x})$. Hence, $|\mathcal{D}_{y,\mathbf{x}_g}|$ is the number of instances belonging to class y in the region $T(\mathbf{x})$. Let $p(T(\mathbf{x})|y)$ be the probability of region $T(\mathbf{x})$ when only class y instances in $\mathcal{D}$ are considered.

$$p(T(\mathbf{x})|y) = \hat{p}(\mathbf{x}_g|y) = \frac{|\mathcal{D}_{y,\mathbf{x}_g}|}{|\mathcal{D}_y|} \tag{9}$$

The new generative classifier, called *MassBayes*, estimates the joint distribution as:

$$\hat{p}(\mathbf{x}, y)_{MassBayes} = p(y)\ \frac{1}{t} \sum_{g \in G_t} p(\mathbf{x}_g|y) = p(y)\ \frac{1}{t} \sum_{i=1}^{t} p(T_i(\mathbf{x})|y) \tag{10}$$

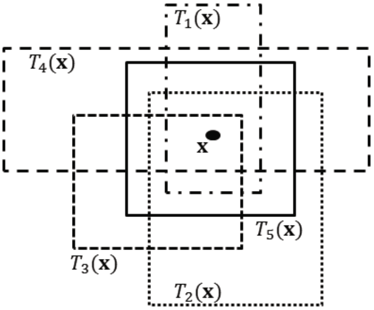


Fig. 1. Different regions from different $T_i(\cdot)$ ($i = 1, 2, \cdots, 5$) that cover $\mathbf{x}$

The average probability of t different regions $T_i(\mathbf{x})$ ($i = 1, 2, \cdots, t$), constructed using $\mathcal{D}_i \subset D$, provides a good estimate for $p(\mathbf{x}|y)$ as it estimates the multi-dimensional likelihood by considering the distribution in different local neighbourhood of $\mathbf{x}$ in the data space. An illustrative example is provided in Figure 1. Note that, the estimator employed in MassBayes is not a true density estimator as it does not integrate to 1.

MassBayes has the following characteristics in comparison with An*DE*:

1. In each estimator, AnDE estimates one-dimensional likelihood given a fixed number of super-parents and y, whereas MassBayes estimates multi-dimensional likelihood using varying number of dimensions.
2. In AnDE, the ensemble size is fixed to $\binom{d}{n}$. But, MassBayes allows the flexibility for users to set the ensemble size.

3. AnDE requires continuous-valued attributes to be discretised before building the model. The performance of AnDE is affected by the discretisation technique used. In contrast, MassBayes builds models directly from data. It can be viewed as a dynamic multi-dimensional discretisation where the information loss is minimised by averaging over multiple models.
4. Each model in MassBayes is built with training subset of size $\psi < N$ which gives rise to the constant training time. In contrast, each model in AnDE is trained using the entire training set.
5. AnDE is a deterministic algorithm whereas MassBayes is a randomised algorithm.
6. Like AnDE, MassBayes is a generative classifier without search.

4 Implementation

In order to partition the feature space to define the regions $T_i(\cdot)$, we use the implementation described by Ting and Wells (2010) using a binary tree called *h:d-tree* [9]. A parameter h defines the maximum level of sub-division. The maximum height of a tree is $h \times d$.

Let the data space that covers the instances in $\mathcal{D}$ be $\varDelta$. The data space $\varDelta$ is adjusted to become δ using a random perturbation conducted as follows. For each dimension j, a split point v_j is chosen randomly within the range $max_j(\varDelta) - min_j(\varDelta)$. Then, the new range δ_j along dimension j is defined as $[v_j - r, v_j + r]$, where $r = max(v_j - min_j(\varDelta), max_j(\varDelta) - v_j)$. The new range on all dimensions defines the adjusted work space for the tree building process.

A subset $\mathcal{D}$ is constructed from D by sampling ψ instances without replacement. The sampling process is restarted with D when all the instances are used. The random adjustment of the work space and random sub-sampling, as described earlier, ensure that no two trees are identical.

The dimension to split is selected from a randomised set of d dimensions in a round-robin manner at each level of a tree. A tree is constructed by splitting the work space into two equal-volume half spaces at each level. The process is then repeated recursively on each non-empty half-space. The tree building process stops when there is only one instance in a node or the maximum height is reached.

At the leaf node, the number of instances in the node belonging to each class is stored. Figure 2 shows a typical example of an implementation of $T(\cdot)$ as an *h:d-tree* for $h = 2$ and $d = 2$. The dotted lines enclosed the instances in $\mathcal{D}$ and the solid lines enclosed the adjusted work space which has ranges δ_1 and δ_2 on x_1 and x_2 dimensions. $R1, R2, R3, R4$ and $R5$ represent different regions in $T(\cdot)$ depending on the data distribution in $\mathcal{D}$. Region $R1$ is defined by splitting the work space in x_1 dimension only, $g = \{1\}$, whereas the other four regions use dimensions x_1 and x_2, i.e., $g = \{1, 2\}$.

In the original implementation by Ting and Wells (2010) for mass estimation, each tree is built to the maximum height of $h \times d$ resulting in equal-size regions regardless of the data distribution [9]. In our implementation, in order to adapt

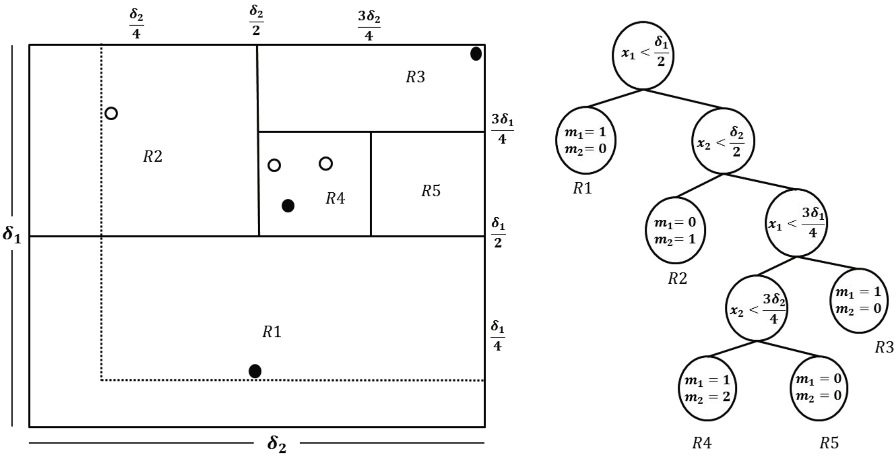


Fig. 2. An example of an h:d-*tree* for $h = 2$ and $d = 2$

to the data distribution, the tree building stops early once the instances are separated. We use the same algorithm as used by Ting and Wells (2010) to generate h:d-*trees* to represent $T_i(\cdot)$ in [9] with the required modification.

The procedures to generate t trees from a given data set D are provided in Algorithms 1 and 2.

The maximum height of each tree is hd, and ψ instances have to be assigned to either of the two child nodes at each level of a tree. Hence, the total training time complexity to construct t trees is $O(thd\psi)$. There are a maximum of ψ (as $\psi < 2^{hd}$ in general) leaf nodes in each tree. The total space complexity is $O(t(d + c)\psi)$.

The time and space complexities of two variants of NB (NB-KDE that estimates $p(x_i|y)$ through kernel density estimation [4]; and NB-Disc that estimates $p(x_i|y)$ through discretisation [10]), AnDE and MassBayes are presented in Table 1. Both training time complexity and space complexity of MassBayes are

Table 1. Time and space complexities of different generative classifiers

Classifiers	Training time	Testing time	Space
NB-KDE [4]	$O(Nd)$	$O(cmd)$	$O(cmd)$
NB-Disc [6]	$O(Nd)$	$O(cd)$	$O(cdv)$
AnDE [7]	$O\left(N\binom{d}{n+1}\right)$	$O\left(cd\binom{d}{n}\right)$	$O\left(c\binom{d}{n+1}v^{n+1}\right)$
MassBayes	$O(thd\psi)$	$O(thd)$	$O\left(t(d+c)\psi\right)$

N: total number of training instances, m: average number of training instances in a class, d: number of dimensions, c: number of classes, v: average number of discrete values of an attribute, n: number of super-parents, t: number of trees, h: level of divisions, and ψ: sample size.

independent of N. Note that the complexities for NB-Disc and AnDE do not include the additional discretisation needed in the preprocessing.

Algorithm 1. BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - number of times an attribute is employed in a path.
Output: F - a set of t *h:d–trees*

```
$H \leftarrow h \times d$ {Maximum height of a tree}
Initialize $F$
for $i = 1$ to $t$ do
    $\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
    $(min, max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
    $A \leftarrow$ {Randomised list of $d$ attributes.}
    $F \leftarrow F \cup$ SingleTree$(\mathcal{D}, min, max, 0, A)$
end for
return $F$
```

Algorithm 2. SingleTree$(\mathcal{D}, min, max, \ell, A)$

Inputs: $\mathcal{D}$ - input data, *min* & *max* - arrays of minimum and maximum values for each attribute that define a work space, A - a randomised list of d attributes, ℓ - current height level.
Output: an *h:d–tree*

```
Initialize $Node(\cdot)$
while ($\ell < H$ and $|\mathcal{D}| > 1$) do
    $q \leftarrow nextAttribute(A, \ell)$ {Retrieve an attribute from $A$ based on height level.}
    $mid_q \leftarrow (max_q + min_q)/2$
    $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < mid_p)$
    $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq mid_q)$
    if ($|\mathcal{D}_l| = 0$ ) or ($|\mathcal{D}_r| = 0$) then {Reduce range for single-branch node.}
        if ($|\mathcal{D}_l| > 0$ ) then $max_q \leftarrow mid_q$
        else $min_q \leftarrow mid_q$
        end if
        $\ell \leftarrow \ell + 1$
        continue at the start of while loop
    end if
    {Build two nodes: $Left$ and $Right$ as a result of a split into two half-spaces.}
    $temp \leftarrow max_q$; $max_q \leftarrow mid_q$
    $Left \leftarrow$ SingleTree$(\mathcal{D}_l, min, max, \ell + 1, A)$
    $max_q \leftarrow temp$; $min_q \leftarrow mid_q$
    $Right \leftarrow$ SingleTree$(\mathcal{D}_r, min, max, \ell + 1, A)$
    terminate while loop
end while
$classCount \leftarrow updateClassCount(\mathcal{D})$
return $Node(Left, Right, SplitAtt \leftarrow q, SplitValue \leftarrow mid_q, classCount)$
```

5 Empirical Evaluation

This section presents the results of the experiments conducted to evaluate the performance of MassBayes against seven well known contenders: two variants of NB (NB-KDE and NB-Disc), BayesNet, three variants of AnDE (A1DE, A2DE, A3DE) and decision tree J48 (i.e., the WEKA [11] version of C4.5 [12]).

MassBayes was implemented in Java using the WEKA platform [11] which also has implementations of NB, BayesNet, A1DE and J48. For A2DE and A3DE, we used the WEKA implementations provided by the authors of AnDE.

All the experiments were conducted using a 10-fold cross validation in a Linux machine with 2.27 GHz processor and 100 GB memory. The average accuracy (%) and the average runtime (seconds) over a 10-fold cross validation were reported. A two-standard-error significance test was conducted to check whether the difference in accuracies of two classifiers was significant. A win or loss was counted if the difference was significant; otherwise, it was a draw.

Ten data sets with $N > 10000$ were used. All the attributes in the data sets are numeric. The properties of the data sets are provided in Table 2. The RingCurve, Wave and OneBig data sets were three synthetic data sets and the rest were real-world data sets from UCI Machine Learning Repository [13]. RingCurve and Wave are subsets of the RingCurve-Wave-TriGaussian data set used in [9] and OneBig is the data set used in [14].

Table 2. Properties of the data sets used

Data sets	#N	#d	#c	Data sets	#N	#d	#c
CoverType	581012	10	7	RingCurve	20000	2	2
MiniBooNE	129596	50	2	Letters	20000	16	26
OneBig	68000	20	10	Magic04	19020	10	2
Shuttle	58000	8	7	Mamograph	11183	6	2
Wave	20000	2	2	Pendigits	10992	16	10

For AnDE, BayesNet and NB-Disc, data sets were discretised by a supervised discretisation technique based on minimum entropy [15] as suggested by the authors of AnDE before building the classification models.

Two variants of MassBayes were used: MassBayes with ($\psi = 5000$) and MassBayes$'$ ($\psi = N$). The other two parameters t and h were set as default to 100 and 10, respectively.

For BayesNet, the parameter *'maximum number of parents'* was set to 100 to examine whether a large number of parents produces better results; and the parameter *'initialise as Naive Bayes'* was set to *'false'* to initialise an empty network structure. The default values were used for the rest of the parameters. All the other classifiers were executed with the default parameter settings.

Table 3. Average classification accuracies (%) over a 10-fold cross validation

Data sets	Mass Bayes′	Mass Bayes	A3 DE	A2 DE	A1 DE	Bayes Net	NB-KDE	NB-Disc	J48
CoverType	94.00	78.21	88.16	80.81	72.89	87.79	66.72	66.61	92.39
MiniBooNE	92.68	91.11	N/A*	91.48	89.58	90.25	86.07	86.29	90.47
OneBig	100.00	100.00	N/A*	99.81	99.69	99.99	99.98	99.97	99.84
Shuttle	99.89	99.89	99.94	99.94	99.85	99.93	92.68	94.36	99.97
Letters	96.63	95.63	95.11	94.31	88.81	86.97	74.21	73.94	87.92
RingCurve	100.00	100.00	99.99	99.99	99.99	99.99	99.27	99.48	99.91
Wave	100.00	100.00	78.51	78.51	78.51	78.51	77.91	78.51	99.79
Magic04	85.72	85.53	85.08	84.57	83.00	83.46	76.13	78.27	85.01
Mamograph	98.69	98.71	98.51	98.37	98.42	98.54	97.86	97.62	98.57
Pendigits	99.45	99.28	98.80	98.82	97.84	96.81	88.64	87.9	96.56

* Did not complete because of integer overflow error.

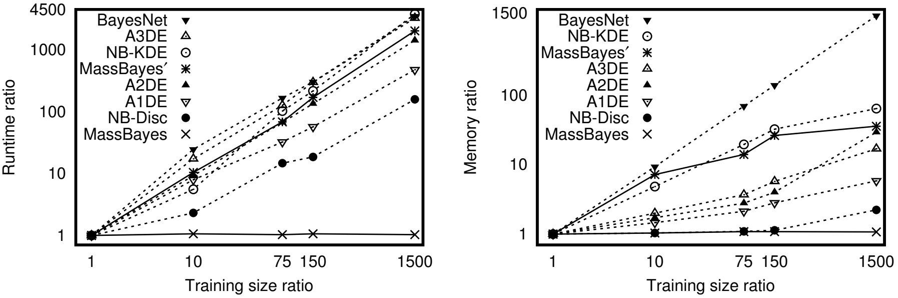
Table 4. Win:Loss:Draw counts of MassBayes over the other contenders in terms of classification accuracy based on the two-standard-error significance test

	A3DE	A2DE	A1DE	BayesNet	NB-KDE	NB-Disc	J48
MassBayes′	4:1:3	7:1:2	7:0:3	7:1:2	10:0:0	10:0:0	7:1:2
MassBayes	3:2:3	6:3:1	7:0:3	6:2:2	10:0:0	10:0:0	6:2:2

Table 5. Average runtime (seconds) over a 10-fold cross validation

Data sets	Mass Bayes′	Mass Bayes	A3 DE	A2 DE	A1 DE	Bayes Net	NB-KDE	NB-Disc	J48
CoverType	1075.8	45.7	45.6	13.9	4.9	387.9	96.3	3.2	3690.7
MiniBooNE	431.1	33.7	N/A	231.3	5.9	308.9	831.6	2.1	323.8
OneBig	113.9	10.5	N/A	11.6	3.9	432.5	253.0	0.8	15.1
Shuttle	48.5	8.0	1.8	0.7	0.5	6.8	1.5	0.4	4.2
Letters	18.9	5.5	11.5	2.6	0.8	4.9	2.5	0.4	7.3
RingCurve	4.4	2.3	0.2	0.2	0.2	0.3	2.4	0.2	0.4
Wave	4.9	2.1	0.2	0.2	0.2	0.2	2.5	0.1	0.6
Magic04	10.9	3.9	0.7	0.5	0.2	0.7	8.8	0.2	3.4
Mamograph	4.7	3.1	0.3	0.2	0.2	0.3	0.5	0.2	0.4
Pendigits	7.3	3.6	5.7	0.9	0.4	1.8	1.6	0.2	1.2

Table 3 shows the average classification accuracies of MassBayes′ and MassBayes in comparison to the other contenders. The results of the two-standard-error significance test in Table 4 show that both MassBayes′ and MassBayes produced better classification accuracy than the other contenders in most data sets.



(a) Runtime (b) Space

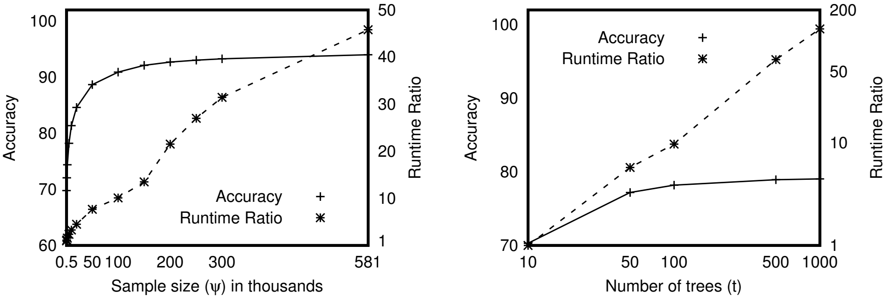
Fig. 3. Scale-up test: MassBayes versus existing generative classifiers. The base for training size ratio is 7000 instances and the bases for runtime ratio and memory ratio are the training time and memory required to save a classification model for 7000 instances. Axes are on logarithmic scales of base 10.

MassBayes produced slightly poorer results than A2DE, A3DE, BayesNet and J48 in CoverType. This was because the default sample size was not enough to yield a good estimate. The accuracy was increased up to 84.62% with $\psi = 20000$ and 88.66% with $\psi = 50000$. More samples are required to grow the trees further to model the distributions well if the class distributions in the feature space are complex. Figure 4(a) shows the improvement in accuracy of MassBayes in CoverType when the sample size was increased.

Table 5 presents the average runtime. In terms of runtime, MassBayes was an order of magnitude faster than A2DE in MiniBooNE; BayesNet in CoverType, MiniBooNE and OneBig; NB-KDE in MiniBooNE and OneBig; and J48 in CoverType and MiniBooNE. It was of the same order of magnitude as A3DE, A2DE, BayesNet, NB-KDE and J48 in many cases and an order of magnitude slower than NB-Disc and A1DE. MassBayes′ was an order of magnitude slower than the other contenders in many data sets. However, it was of the same order of magnitude as A3DE in Letters; A2DE in MiniBooNE; BayesNet and NB-KDE in MiniBooNE and OneBig; and J48 in CoverType and MiniBooNE.

Note that the reported runtime results for AnDE, BayesNet and NB-Disc did not include the discretisation time that must be done as a preprocessing step, which give the existing generative classifiers (except NB-KDE) an unfair advantage over MassBayes. The discretisation time can be substantially large in large data sets. For example, the discretisation took 52 seconds in the largest data set, CoverType. This discretisation time alone was more than the total runtime of MassBayes. Thus, MassBayes in effect runs faster than all existing generative classifiers on equal footing.

In order to examine the scalability of the classifiers in terms of training time and space requirements with the increase in training size N, we used the 48-dimensional (42 irrelevant attributes with constant values) RingCurve-Wave-Tri-Gaussian data set previously employed by Ting and Wells (2010) in [9]. The training data size was increased from 7000 to 70000, half-a-million, 1 million and



(a) Sample size (ψ)

(b) Trees (t)

Fig. 4. Effect of parameters ψ and t on the classification accuracy and runtime of MassBayes in the CoverType data set. The base for the runtime ratio while varying ψ and t is the total runtime (training and testing over a 10-fold cross validation) for $\psi = 500$ and $t = 10$, respectively. The horizontal axis of t and the vertical axis of runtime ratio in (b) are on logarithmic scales of base 10.

10 million by a factor of 1, 10, 75, 150 and 1500, respectively. Figure 3 shows the increase in classification model building time and memory space required to store the classification model for different generative classifiers. Note that the discretisation time was not included in the presented results. The discretisation time increases linearly with the increase in training data size. This additional time for discretisation will increase the training time of AnDE, BayesNet and NB-Disc. MassBayes had constant training time and constant space requirements.

In order to examine the sensitivity of the parameters ψ, t and h in classification accuracy and runtime of MassBayes, we conducted a set of experiments by varying one parameter and fixing the other two to the default values. The result of the experiment varying ψ and t in the largest data set (CoverType) is shown in Figure 4. The increase in runtime was plotted as a ratio to show the factor of runtime increased when the parameters were increased.

In general, accuracy increased up to a certain point and remained flat when each of the three parameters was increased. This indicates that the parameters of MassBayes are not too sensitive in terms of classification accuracy if they are set to sufficiently high values. The runtime increased linearly with t and sub-linearly with ψ. With fixed sample size ($\psi = 5000$), increase in h after a certain point did not affect the runtime because the tree building process stopped before reaching the maximum level h once the instances are separated.

6 Conclusions and Future Work

In this paper, we presented a new generative classifier called *MassBayes* that approximates $p(\mathbf{x}|y)$ by aggregating multi-dimensional likelihoods estimated using varying size subsets of features from random subsets of training data. In contrast, existing generative classifiers make assumptions about attribute independence and estimate single-dimensional likelihood only. Our empirical results show that MassBayes produced better classification accuracy than the existing generative classifiers in large data sets.

In terms of runtime, it scales better than the existing generative classifiers in large data sets as it builds models in an ensemble using fixed-size data subsets. The constant training time and space complexities make it an ideal classifier for large data sets and data streams.

Future work includes applying the proposed method in data sets with discrete and mixed attributes and investigating the effectiveness of MassBayes in the data stream context. In this paper, we have rigorously assessed MassBayes with the state-of-the-art Bayesian classifiers. In the near future, we will assess its performance against some well-known discriminative classifiers and their ensembles. The feature space partitioning can be implemented in various ways. It would be interesting to investigate a more intelligent way of feature space partitioning rather than dividing at mid-point of a randomly selected dimension.

Acknowledgement. This work is supported by the Air Force Research Laboratory, under agreement# FA2386-11-1-4112. Sunil Aryal is partially supported by Monash University Postgraduate Publications Award to write this paper. We would like to thank Geoff Webb and the anonymous reviewers for their helpful comments.

References

1. Silverman, B.W.: Density Estimation for Statistics and Data Analysis. Chapmal & Hall/CRC (1986)
2. Ram, P., Gray, A.G.: Density Estimation Trees. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 627–635. ACM, New York (2011)
3. Langley, P., Iba, W., Thompson, K.: An Analysis of Bayesian Classifiers. In: Proceedings of the Tenth National Conference on Artificial Intelligence, pp. 399–406 (1992)
4. Langley, P., John, G.H.: Estimating continuous distribution in Bayesian classifiers. In: Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence (1995)
5. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning 29, 131–163 (1997)
6. Webb, G.I., Boughton, J.R., Wang, Z.: Not So Naive Bayes: Aggregating one-dependence estimators. Machine Learning 58, 5–24 (2005)
7. Webb, G., Boughton, J., Zheng, F., Ting, K., Salem, H.: Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification. Machine Learning 86, 233–272 (2012)
8. Chickering, D.M.: Learning Bayesian Networks is NP-Complete. In: Fisher, D., Lenz, H.J. (eds.) Learning from Data: Artificial Intelligence and Statistics V, pp. 121–130. Springer, Heidelberg (1996)
9. Ting, K.M., Wells, J.R.: Multi-Dimensional Mass Estimation and Mass-Based Clustering. In: Proceedings of IEEE ICDM, pp. 511–520 (2010)

10. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and Unsupervised Discretization of Continuous Features. In: Proceedings of the 12th International Conference on Machine Learning, pp. 194–202. Morgan Kaufmann (1995)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1) (2009)
12. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo (1993)
13. Frank, A., Asuncion, A.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2010), `http://archive.ics.uci.edu/ml`
14. Nanopoulos, A., Theodoridis, Y., Manolopoulos, Y.: Indexed-based density biased sampling for clustering applications. IEEE Transaction on Data and Knowledge Engineering 57(1), 37–63 (2006)
15. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous valued attributes for classification learning. In: Proceedings of 14th International Joint Conference on Artificial Intelligence, pp. 1034–1040 (1995)

LiNearN: A New Approach to Nearest Neighbour Density Estimator

Jonathan R. Wells[a], Kai Ming Ting[a], Takashi Washio[b]

[a] *Gippsland School of Information Technology, Monash University Gippsland Campus*
Northways Road, Churchill, Victoria, 3842, Australia
[b] *The Institute of Scientific and Industrial Research, Osaka University*
8-1 Mihogaoka, Ibarakishi, Osaka, 5670047, Japan

Abstract

Despite their wide spread use, nearest neighbour density estimators have two fundamental limitations: $O(n^2)$ time complexity and $O(n)$ space complexity. Both limitations constrain nearest neighbour density estimators to small data sets only. Recent progress using indexing schemes has improved to near linear time complexity only.

We propose a new approach, called **LiNearN** for **Li**near time **Near**est **N**eighbour algorithm, that yields the first nearest neighbour density estimator having $O(n)$ time complexity and constant space complexity, as far as we know. This is achieved without using any indexing scheme because LiNearN uses a subsampling approach for which the subsample values are significantly less than the data size. Like existing density estimators, our asymptotic analysis reveals that the new density estimator has a parameter to trade off between bias and variance. We show that algorithms based on the new nearest neighbour density estimator can easily scaleup to data sets with millions of instances in anomaly detection and clustering tasks.

Keywords: k-nearest neighbour, density-based, anomaly detection, clustering

1. Introduction and Motivation

Existing methods have utilised nearest neighbour density estimators as the basis to solve all facets of pattern recognition problems from classification and regression to clustering and anomaly detection [5, 7, 11, 13, 15].

While existing nearest neighbour density estimators have been effective, the time complexity is still basically $O(n^2)$ because of the need to find the nearest neighbour for every instance in a given data set. This makes existing methods utilising nearest neighbour density estimator impractical for problems with large data sets. Recent research has substantially improved the k-nearest neighbour search by introducing various indexing schemes to speed up the search (e.g., Cover Trees [9], M-Trees [12] and R*-Tree [8]) to near linear time complexity.

The premise of the current research is that finding the nearest neighbour for every instance in the given data set is inevitable which leads to $O(n^2)$ time complexity. Since the aim is to do density estimation, we reject this premise and find a way to reduce the number of distance calculations required to achieve this aim.

We propose a new approach to nearest neighbour density estimator. Instead of focusing on speeding up the nearest neighbour search, the new approach first generates many local regions from subsamples and then produces the final result in an ensemble method. The speedup is achieved because the size of the subsamples required is significantly smaller than the given data set. This not only eliminates the need of using an indexing scheme but enables the new density estimator to run in orders of magnitude faster than existing nearest neighbour density estimators.

Email addresses: `jonathan.wells@monash.edu` (Jonathan R. Wells), `kaiming.ting@monash.edu` (Kai Ming Ting), `washio@ar.sanken.osaka-u.ac.jp` (Takashi Washio)

Preprint submitted to Pattern Recognition *August 30, 2013*

We make three contributions in this paper:

1. Introduce a new nearest neighbour density estimator that defines local neighbourhoods using nearest neighbours in each of the many subsamples by building a region centered at each instance. This differs from the existing nearest neighbour density estimators where the local neighbourhoods are defined based on either k nearest neighbours or a fixed radius.

2. Provide an asymptotic analysis and it reveals that the new density estimator has a parameter which trades off between bias and variance, as in existing density estimators such as k-nearest neighbour density estimators.

3. Demonstrate the advantages of the new approach over the existing nearest neighbour density estimators in two tasks: anomaly detection and clustering. The new approach reduces the time complexity from $O(n^2)$ to $O(n)$ and the space complexity complexity from $O(n)$ to constant. We call the new approach **LiNearN** for **Li**near time **Near**est **N**eighbour algorithm.

Since nearest neighbour density estimators are the core mechanism in many pattern recognition algorithms, we will begin the next section with a description of existing nearest neighbour density estimators. Section 3 introduces the new nearest neighbour density estimator and provides the asymptotic analysis. Section 4 describes how nearest neighbour density estimators, both existing and the new, are applied to anomaly detection and clustering tasks. Section 5 reports the empirical evaluation results. Discussion and the conclusions are provided in the last two sections.

2. Existing Nearest Neighbour Density Estimators

We describe three existing nearest neighbour density estimators below.

1. A k-nearest neighbour (k-NN) density estimator can be expressed as follows [11, 32].

$$f_{kNN}(x) = \frac{|N(x,k)|}{n \sum\limits_{x' \in N(x,k)} \|x - x'\|_p}$$

where $N(x,k)$ is the set of k-nearest neighbours to x; and $|S|$ denotes the cardinality of set S, and $\|x - x'\|_p$ denotes the distance measured by L^p-norm between x and x'. The search for nearest neighbours is conducted over D of size n, where D is the given data set.

2. A k^{th} nearest neighbour density estimator is defined as follows [25]:

$$f_{k^{th}NN}(x) = \frac{|N(x,k)|}{n\alpha(d,p)\|x - x_k\|_p^d}$$

where x_k is the k^{th} nearest neighbour to x and $\alpha(d,p)$ is the volume of an unit ball in $(\Re^d, L^p)$

3. An ϵ-neighbourhood density estimator is defined as follows:

$$f_\epsilon(x) = \frac{|N_\epsilon(x)|}{n\epsilon}$$

where $N_\epsilon(x) = \{q \in D \mid \|x - q\|_p \leq \epsilon\}$. Since the denominator $n\epsilon$ is the same for all x, it is usually omitted in the implementation (e.g., in DBSCAN [15]).

Each of the above determines a local neighbourhood based on a global parameter, i.e., k or ϵ; and the density is calculated based on one variable: distance of k-nearest neighbours in f_{kNN} or $f_{k^{th}NN}$ since the numerator is a constant k; and $N_\epsilon(\cdot)$ in f_ϵ since the denominator is a constant.

In addition, the nearest neighbour search is conducted over the entire data set, D, which is the main computational expense of the whole process; therefore, leading to a time complexity of $O(n^2)$ for n queries. Research has focused on reducing this cost by devising different indexing schemes.

Table 1: Key differences between existing nearest neighbour algorithms and LiNearN in terms of methodology and time complexity.

	Existing NN	LiNearN
Methodology	Single model Density for each $x \in D$ is derived from a single local region via NN searches (e.g., f_{kNN}, $f_{k^{th}NN}$ or f_ϵ). Indexing† is required to speed up NN search. Often rely on triangle inequality to prune the search space	Multiple models Density for each $x \in D$ is derived from many local regions (LR). NN search without indexing. 1. NN search in a subset of D (t times) to define LR. 2. NN search to make the final estimation for each $x \in D$.
Time complexity		
- LR building	Not Applicable	1. $\psi(\psi + \Psi)t$
- Index building	nil or $n \log n$ ‡	Not Applicable
- n Queries	n^2 or $n \log n$	2. ψnt

† An alternative to indexing is clustering based search [27] which often needs higher time cost than indexing.
‡ Without indexing, n queries in existing nearest neighbour algorithms have $O(n^2)$ time complexity; with indexing methods such as Cover Trees [9] and M-Trees [12], n queries have $O(n \log n)$ time complexity.

We suggest a new approach to compute density based on nearest neighbour with the following distinguishing features:

- Both the number of instances in the local neighbourhood and its volume are adaptive to the data distribution in the local region; neither is fixed by a global parameter, unlike $f_{kNN}(\cdot)$, $f_{k^{th}NN}(\cdot)$ and $f_\epsilon(\cdot)$.
- The nearest neighbour search is conducted over a data subset which is significantly smaller than the given data set.

We describe the new density estimator in the next section.

3. New nearest neighbour density estimator

We propose a new nearest neighbour density estimator, called **LiNearN** for **Li**near time **Near**est **N**eighbour algorithm. It estimates the density for a point x by averaging densities of multiple local regions covering x. Whilst the local regions could be implemented in different ways, we focus on deriving the local regions using nearest neighbours. Because these local regions can be defined by using a significantly smaller data set than the given data set, the computational expense for nearest neighbour search is reduced to such an extent that an indexing scheme becomes unnecessary.

We describe LiNearN in the following five subsections. After describing the key differences between the new and existing density estimators in the first subsection, LiNearN is formally defined in the second subsection with an illustration in the third subsection. The asymptotic error analysis is given in the fourth subsection followed by its implementation in the fifth subsection.

3.1. Key Differences

The key differences between LiNearN and existing density estimators based on nearest neighbour are shown in Table 1. Since all parameters, except n, are constant and both $\psi \ll n$ and $\Psi \ll n$ (see definitions in Section 3.2), the time complexity of LiNearN is $O(n)$, which is significantly smaller than $O(n^2)$ or $O(n \log n)$.

Unlike ϵ-neighbourhood density estimator which employs a global ϵ (where every local region has the same size), LiNearN adapts the size of each local region to the local data distribution — sparse regions have large local regions; whereas, dense regions have small local regions. While k-nearest neighbour density estimator can adapt to local data distributions in simple cases where there are only two densities by choosing an appropriate k, it is not possible in more complex cases where many different densities exist in the data.

Instead of relying on a single estimation using the entire data set, LiNearN computes the average of multiple estimations. Each of the estimations can be done with a significantly smaller data subset — this is where the significant speedup over existing nearest neighbour density estimators is achieved.

The typical nearest neighbour algorithm (e.g., LOF [11], ORCA [7] or DBSCAN [15]) requires to store all instances of a given data set for the distance calculation. In contrast, LiNearN requires to store $t\psi$ instances only, which is constant.

Though LiNearN requires nearest neighbour search, only a linear search is required because it involves a small sample size only, where $\psi \ll n$ and $\Psi \ll n$. Existing algorithms which employ nearest neighbour density estimators must rely on indexing to reduce the time complexity from $O(n^2)$ to $O(n \log n)$ [9, 12].

3.2. Definitions

Let D be a set of i.i.d. samples in a $(\Re^d, L^p)$ space, where the length of $x = (x_1, x_2, \cdots, x_d)^T \in \Re^d$ is measured by an L^p-norm:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_d|^p)^{\frac{1}{p}}.$$

where p is in $(0, \infty]$, and $\|x\|_\infty = max\{|x_1|, |x_2|, \cdots, |x_d|\}$. Furthermore, let $\mathcal{D} \subset D$ be selected using i.i.d. sampling without replacement.

Definition 1. *$\mathbb{H}_c \subset (\Re^d, L^p)$ is a local region having its center at $c \in \mathcal{D}$ and its radius $r_c = \frac{1}{2}\|c - x\|_p$, where x is the nearest neighbour of c in $\mathcal{D}$.* □

L^p-norm determines the shape of the local region $\mathbb{H}_c$. For example, $\mathbb{H}_c$ is a hypersphere if $p = 2$, and a hypercube if $p = \infty$. Note that a region $\mathbb{H}_c$ does not overlap with any other regions within a given set of $\mathcal{D}$.

Definition 2. *H is the set of $|\mathcal{D}|$ local regions, $\mathbb{H}_c$, constructed from $\mathcal{D}$: $H = \{\mathbb{H}_c | c \in \mathcal{D}\}$.* □

Definition 3. *Given H and $\mathfrak{D} \subset D$ selected using i.i.d. sampling without replacement, if $\exists \mathbb{H}_c \in H, x \in \mathbb{H}_c$, a distance-based density of x is defined as*

$$\rho(x|H, \mathfrak{D}) = \frac{|\mathfrak{D}(\mathbb{H}_{cnn(x)})|}{|\mathfrak{D}| r_{cnn(x)}}, \tag{1}$$

where $cnn(x) = \underset{c \in \mathcal{D} \;\; s.t. \;\; x \in \mathbb{H}_c \in H}{\arg\min} \|c - x\|_p$ and $\mathfrak{D}(\mathbb{H}_{cnn(x)}) = \{o \in \mathbb{H}_{cnn(x)} | o \in \mathfrak{D}\}$. □

$\rho(x|H, \mathfrak{D})$ is given by H and $\mathfrak{D}$ if $\mathbb{H}_c$ containing x exist in H, while H is defined by $\mathcal{D}$. $|\mathfrak{D}|$ is generally set to be larger than $|\mathcal{D}|$. Note that $\rho(x|H, \mathfrak{D}) = 0$ if $x \notin \mathbb{H}_c \in H$. This distance-based density is defined as a ratio of number of instances and distance, following the similar density definition called 'inverse distance' for k-NN density estimator [32].

Definition 4. *Average distance-based density of a point x is estimated from t local regions as follows:*

$$\bar{f}(x) = \frac{1}{t} \sum_{i=1}^{t} \rho(x|H^i, \mathfrak{D}^i), \tag{2}$$

where H^i is generated from $\mathcal{D}^i \subset D$. This summation excludes $\rho(x|H^i, \mathfrak{D}^i)$ such that $\nexists \mathbb{H}_c \in H^i, x \in \mathbb{H}_c$. $\mathfrak{D}(\mathbb{H}_{cnn(x)})$ of every $\mathbb{H}_{cnn(x)} \in H^i$ is estimated from $\mathfrak{D}^i \subset D$. $\forall i$, $|\mathcal{D}^i| = \psi$, and $|\mathfrak{D}^i| = \Psi$. □

When producing an estimate from multiple local regions, it is important to ensure that only one estimation from each H^i contributes to the final result. The use of $\mathbb{H}_{cnn(x)} \in H^i$ ensures that $\overline{f}(x)$ at any point x uses only one of the local regions in H^i.

We further introduce volume-based density which has a consistent definition with population probability density $\phi(x)$ of D.

Definition 5. *Given H and $\mathfrak{D} \subset D$ selected using i.i.d. sampling without replacement, if $\exists \mathbb{H}_c \in H, x \in \mathbb{H}_c$, a volume-based density of x is defined as*

$$\wp(x|H, \mathfrak{D}) = \frac{|\mathfrak{D}(\mathbb{H}_{cnn(x)})|}{|\mathfrak{D}||\mathbb{H}_{cnn(x)}|}. \tag{3}$$

where $cnn(x)$ and $\mathfrak{D}(\mathbb{H}_{cnn(x)})$ are defined in Definition 3. $|\mathbb{H}_{cnn(x)}|$ is the volume of $\mathbb{H}_{cnn(x)}$ given by $|\mathbb{H}_{cnn(x)}| = \alpha(d,p) r^d_{cnn(x)}$, where $\alpha(d,p)$ is the volume of an unit ball in $(\Re^d, L^p)$. □

For example, $\alpha(d,2) = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$ where $\Gamma(\cdot)$ is a gamma function, and $\alpha(d,\infty) = 2^d$. $\rho(x|H, \mathfrak{D})$ is related to $\wp(x|H, \mathfrak{D})$ as follows:

$$\rho(x|H, \mathfrak{D}) = \frac{|\mathbb{H}_{cnn(x)}|}{r_{cnn(x)}} \wp(x|H, \mathfrak{D}). \tag{4}$$

We define average volume-based density by $\wp(x|H, \mathfrak{D})$ as follows.

Definition 6. *Similar to Definition 4, average volume-based density of a point x is estimated from t local regions as follows:*

$$\overline{\zeta}(x) = \frac{1}{t} \sum_{i=1}^{t} \wp(x|H^i, \mathfrak{D}^i). \tag{5}$$

□

For the rest of this paper, we will refer to the shape for each region, $\mathbb{H}_c$, as a hypercube by using $p = \infty$, with the understanding that the shape for each region can be easily changed by changing the L^p-norm.

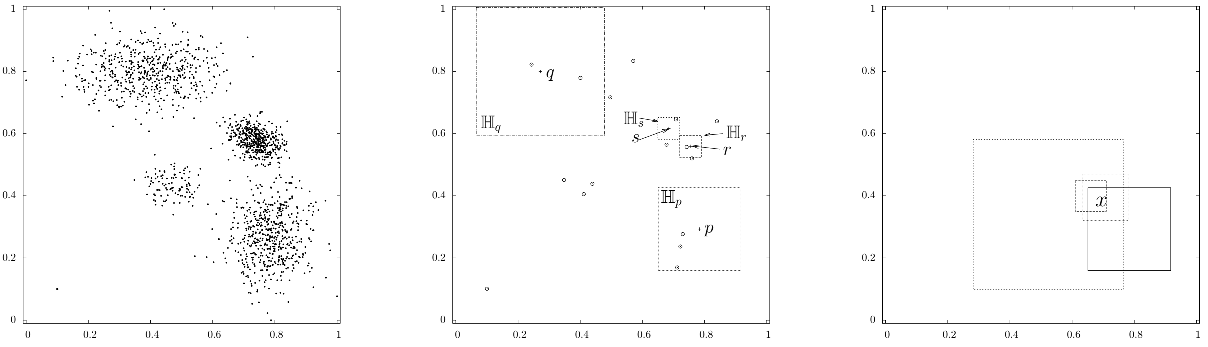
3.3. Illustration

An example of hypercubes, as a result of using L^∞-norm in a 2-dimensional data set, generated from $\mathcal{D} \subset D$ is shown in Figures 1(a) and 1(b). Figure 1(c) shows an example of estimating density for an instance x using multiple hypercubes from $\{H^i | i = 1, \ldots, t\}$ where $t = 4$.

The number of hypercubes in H, ψ, determines the smoothness of the estimation. It has the similar effect as the k parameter in k-nearest neighbour density estimator. Three example density distributions, as shown in Figures 2(a)-2(c), are generated by LiNearN using three different values of ψ. Low ψ produces a smooth distribution; whereas, high ψ produces a more spiky corresponding distribution. The density distributions as generated by f_{kNN} and f_ϵ are shown in Figures 2(d)-2(f) and Figures 2(g)-2(i), respectively.

Notice that there is an anomaly cluster in the bottom left corner of Figure 1(a) which consists of five instances densely packed into a small region. The density distributions produced by LiNearN in Figure 2 shows that the anomaly cluster has low density when a small ψ is used and the ‘true’ density only starts to emerge when a high ψ is used. This feature is especially useful for the purpose of anomaly detection and has the following impacts:

- Both scattered anomalies and clustered anomalies can be detected using a small value of ψ.
- The time cost increases linearly with ψ. A small ψ value means that LiNearN could detect anomalies with a small time cost.

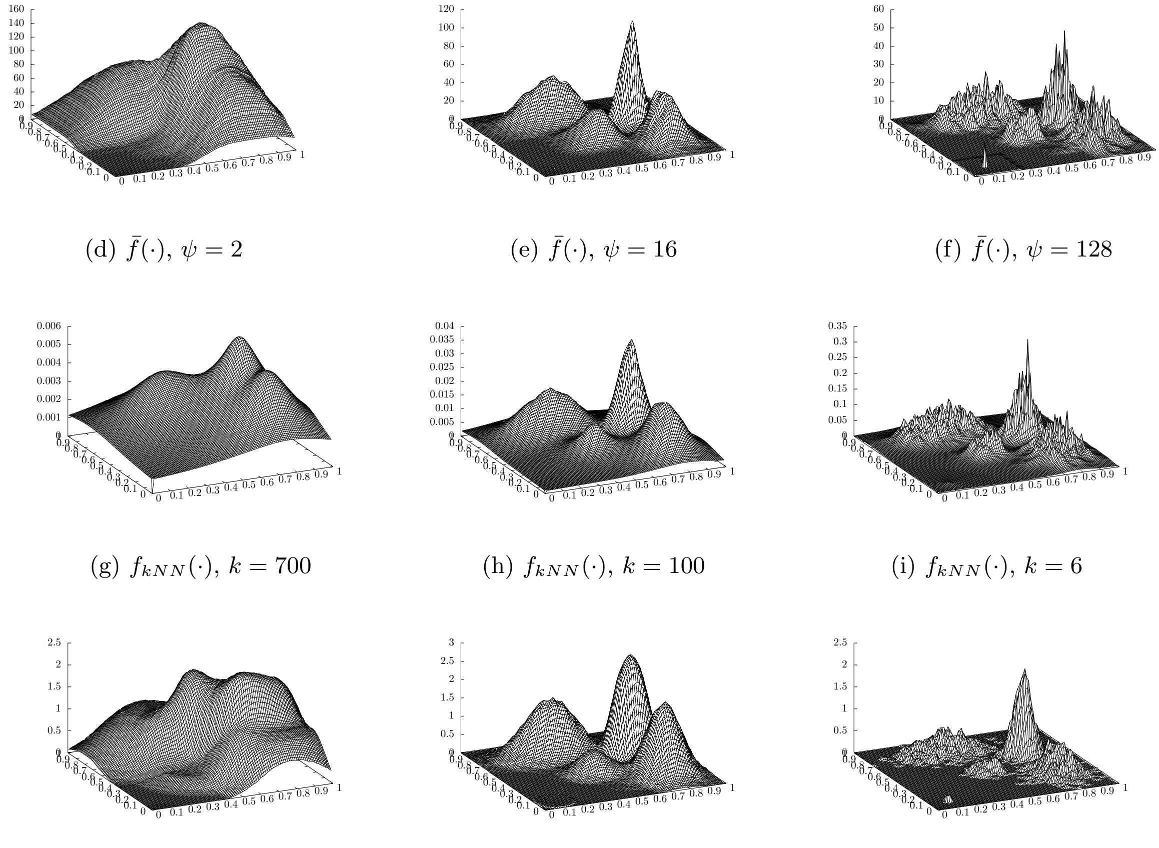


(a) Given data set, D. Modified from [21].

(b) Example hypercubes for $|\mathcal{D}|$=4 and $|\mathfrak{D}|$=16.

(c) Example density estimation from multiple hypercubes using $\bar{f}(x)$ in Equation (2).

Figure 1: An example of constructing hypercubes using $\mathcal{D} \subset D$ (see Algorithm 2 in Section 3.5) and assigning sample mass in each hypercube to estimate density using $\mathfrak{D} \subset D$. These two processes are shown in (b).



(d) $\bar{f}(\cdot)$, $\psi = 2$

(e) $\bar{f}(\cdot)$, $\psi = 16$

(f) $\bar{f}(\cdot)$, $\psi = 128$

(g) $f_{kNN}(\cdot)$, $k = 700$

(h) $f_{kNN}(\cdot)$, $k = 100$

(i) $f_{kNN}(\cdot)$, $k = 6$

(j) $f_{\epsilon}(\cdot)$, $\epsilon = 0.3$

(k) $f_{\epsilon}(\cdot)$, $\epsilon = 0.1$

(l) $f_{\epsilon}(\cdot)$, $\epsilon = 0.02$

Figure 2: Example density distributions for the data set shown in Figure 1(a). Figures (a)-(c) show the distributions generated from LiNearN having $\bar{f}(\cdot)$ uses three different values of ψ with $\Psi = 256$ and $t = 1000$. The corresponding density distributions for $f_{kNN}(\cdot)$ and $f_{\epsilon}(\cdot)$ are shown in (d)-(f) and (g)-(i), respectively.

In contrast, when k-nearest neighbour is used to detect clustered anomalies (e.g., in LOF [11] and ORCA [7]), in order to detect clustered anomalies, k needs to be set to a value larger than the number of instances in every anomaly cluster [24]. Otherwise, the k-nearest neighbour algorithms would fail to detect anomaly clusters having number of instances larger than k. This has two implications. First, the time cost increases as k increases. Second, a more detrimental effect to the runtime of k-nearest neighbour anomaly detectors is that k needs to be searched in a large range of values. This will add a significant time cost to their already expensive $O(n^2)$ time complexity.

We will demonstrate these advantages of LiNearN over existing nearest neighbour algorithms in Section 5.1.

3.4. Asymptotic Error Analysis

$\overline{\zeta}(x)$ can be thought of as a random variable because of its dependence on D, its random subsamples $\mathcal{D}^i$ and $\mathfrak{D}^i$ $(i = 1, \ldots, t)$. Accordingly, we analyse Mean Squared Error (MSE) of $\overline{\zeta}(x)$ from the population probability density $\phi(x)$. It is defined as

$$MSE(\overline{\zeta}(x)) = E\big[\{\overline{\zeta}(x) - \phi(x)\}^2\big]$$

where the expectation $E[\cdot]$ is taken over the distribution of $\overline{\zeta}(x)$. This is rewritten by introducing the expectation of $\overline{\zeta}(x)$, $E[\overline{\zeta}(x)]$, as follows [30].

$$\begin{aligned} MSE(\overline{\zeta}_k(x)) \quad &= \quad \big\{E[\overline{\zeta}_k(x)] - \phi(x)\big\}^2 \\ &\quad\; + E\big[\{\overline{\zeta}_k(x) - E[\overline{\zeta}_k(x)]\}^2\big]. \end{aligned} \tag{6}$$

The first term on the rhs is called 'square bias' and the second term 'variance.'

Preliminary to evaluating magnitudes of these terms, we present the moment expectations of the nearest distance distribution at a point $x \in \Re^d$ under a generic population distribution $\phi(x)$ in a $(\Re^d, L^p)$ space. Evans et al. [17] already assessed its associated problem. Although they did not explicitly state the applicability of its analysis to generic L^p distance measures, the applicability was indicated in a subsequent paper [16]. These papers focused on the moment expectations marginalized over a compact convex space having a unit volume. Here, we further assess the moment expectations at a point $x \in \Re^d$ based on their analyses.

Theorem 1. *Let C be a compact convex body in $(\Re^d, L^p)$. Assume that all instances in D and hence $\mathcal{D} \subset D$ are i.i.d. sampled from C according to their population probability density $\phi(x)$ satisfying three conditions: $\phi(x)$ is continuous, positive, and has bounded partial derivatives for all $x \in C$. Then for all $0 < \epsilon < 1/d$ and integer $m \geq 0$, the expectation of the m-th moment of r_c defined in Definition 1 is represented as follows for some $0 < \nu(d, p, x) \leq 1$.*

$$E[r_c^m] = \frac{c(d, m, p)}{(\psi + 1)^{m/d}} + O(\frac{1}{\psi^{m/d+\epsilon}}) \tag{7}$$

holds as $\psi \to \infty$, where $\psi = |\mathcal{D}|$, and $c(d, m, p) = \frac{\Gamma(m/d+1)}{\{\nu(d,p,c)\alpha(d,p)\phi(x)\}^{m/d}}$ is a constant not depending on ψ, and $\alpha(d, p)$ is the volume of a unit ball in $(\Re^d, L^p)$.

The proof is provided in Appendix A.

The compactness and convexity of C and the three assumptions of $\phi(x)$ on C virtually do not limit the applicability of this result, since C can be chosen to sufficiently cover the areas containing all data points in D, a continuous and smooth $\phi(x)$ can be assumed, and small positive values of $\phi(x)$ can be assumed even in the areas where the data points in D are not located in C.

Now, we obtain the following theorem evaluating asymptotic behaviours of the square bias and the variance in the MSE of $\overline{\zeta}(x)$ in terms of number of subsamples. In addition to the assumptions of Theorem 1, we further introduce an assumption of bounded second order partial derivatives of $\phi(x)$. These assumptions do not limit the applicability of the result to practical problems.

Theorem 2. *Under the assumptions of Theorem 1 and an assumption that $\phi(x)$ has bounded second order partial derivatives at each point $x \in C$, magnitudes of the square bias and the variance of $\overline{\zeta}(x)$ asymptotically behave as follows.*

$$\left\{E[\overline{\zeta}(x)] - \phi(x)\right\}^2 = O(\psi^{-2/d}), \ \textit{and} \tag{8}$$

$$\begin{aligned} & E\left[\{\overline{\zeta}(x) - E[\overline{\zeta}(x)]\}^2\right] \\ & = \begin{cases} O(t^{-1}\psi^{-1}\Psi^{-1}) & \textit{if } d = 1, \\ O(t^{-1}\psi^{1-2/d+\epsilon}\Psi^{-1}) \ \textit{for all } 0 < \epsilon < 1/d & \textit{if } d \geq 2. \end{cases} \end{aligned} \tag{9}$$

The proof is provided in Appendix B.

This result indicates that the square bias of $\overline{\zeta}(x)$ diminishes and the variance increases as the size of $\mathcal{D}$ increases, i.e., the size of the local region $\mathbb{H}_c$ decreases, except for the case of one-dimensional data. This property of the average volume-based density estimator for the size of $\mathcal{D}$ is similar to that of the conventional k-nearest neighbour estimator which also shows decrease of the bias and increase of the variance for the reduction of the k value. On the other hand, a significant advantage of our approach is the dependency of the variance to the inverse of t and the inverse of the size of $\mathfrak{D}$. By increasing these parameters, we can reduce the variance while maintaining the squared bias. The average volume-based density estimator has superior performance to the k-nearest neighbour estimator in theory while maintaining the lower computation cost.

3.5. Implementation

The procedural outline of LiNearN is given below:

① Select a subsample $\mathcal{D}$ of size ψ from D, where $\psi \ll |D|$.

② $\forall c \in \mathcal{D}$, identify its nearest neighbour; and construct a hypercube region centered at c with 'radius' r_c. A total of ψ non-overlapping hypercube regions is produced from $\mathcal{D}$. (Definitions 1 and 2)

③ Repeat steps ① and ② t times to produce $\{H^i | i = 1, \ldots, t\}$.

④ A subset $\mathfrak{D} \subset D$ is used to estimate the number of instances covered by each hypercube region. (Definition 3)

⑤ Estimate the density for each $x \in \Re^d$ by averaging the densities of t hypercube regions that cover x. (Definition 4)

Steps ① to ③ are implemented in Algorithm 1, and the actual construction of hypercube regions is implemented in Algorithm 2. Steps ④ and ⑤ are implemented in Algorithms 3 and 4, respectively.

```
Algorithm 1: LiNearN($D, t, \psi, \Psi$)
input  : $D$ - input data, $t$ - number of subsamples $\mathcal{D}$, $\psi$ - the size of subsample used for constructing
         a set of hypercube regions, $H^i$. $\Psi$ - the size of subsample used to estimate density in $H^i$
output: $\{H^i | i = 1, \ldots, t\}$ where each $H^i$ contains a total of $\psi$ non-overlapping hypercube regions
         with estimated densities.
for $i$ = 1 to $t$ do
    $\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement};
    $H^i \leftarrow$ BuildHyperCubes($\mathcal{D}$);
end
AssignSampleMass($\{H^i | i = 1, \ldots, t\}, D, \Psi$)
return $\{H^i | i = 1, \ldots, t\}$ with estimated densities;
```

Algorithm 2: BuildHyperCubes($\mathcal{D}$)

```
input  : $\mathcal{D}$ - subsample used to build hypercube regions.
output: $H$ - a set of $|\mathcal{D}|$ non-overlapping hypercube regions.
$H \leftarrow \emptyset$;
for $m$ = 1 to $|\mathcal{D}|$ do
    Initialise $\mathbb{H}$;
    $\mathbb{H}.center \leftarrow x_m$;
    $x_{NN} \leftarrow \underset{o \in \mathcal{D} \setminus \{\mathbb{H}.center\}}{\arg\min} \|\mathbb{H}.center - o\|_\infty$ {find the nearest neighbour};
    $\mathbb{H}.radius \leftarrow \frac{1}{2}\|\mathbb{H}.center - x_{NN}\|_\infty$;
    $\mathbb{H}.mass \leftarrow 0$;
    $H \leftarrow H \cup \{\mathbb{H}\}$;
end
return $H$;
```

Algorithm 3: AssignSampleMass($\mathbb{C}, D, \Psi$)

```
input  : $\mathbb{C}$ - $\{H^i | i = 1, \dots, t\}$, $D$ - input data, $\Psi = |\mathfrak{D}|$ - the size of subsample used for density
         estimation.
output: $\mathbb{C}$ - $\{H^i | i = 1, \dots, t\}$ with their mass values estimated from $\mathfrak{D}^i$, respectively.
for $i$ = 1 to $t$ do
    $\mathfrak{D}^i = \{x_1, \dots, x_\Psi\} \leftarrow sample(D, \Psi)$ {strictly without replacement};
    for $j$ = 1 to $\Psi$ do
        $\mathbb{H} \leftarrow search(H^i, x_j)$ {return $\mathbb{H} \subset H^i$ that covers $x_j$, i.e., $||\mathbb{H}.center - x_j||_\infty < \mathbb{H}.radius$};
        if $\mathbb{H}$ is not NULL then
            $\mathbb{H}.mass \leftarrow \mathbb{H}.mass + 1$;
        end
    end
end
return $\mathbb{C}$;
```

Algorithm 4: Density($x, \mathbb{C}$)

```
input  : $x$ - input point, $\mathbb{C}$ - $\{H^i | i = 1, \dots, t\}$.
output: $\rho/t$ - average density estimated for $x$.
$\rho \leftarrow 0$;
for $i$ = 1 to $t$ do
    $\mathbb{H} \leftarrow search(H^i, x)$ {return $\mathbb{H}$ that covers $x$};
    if $\mathbb{H}$ is not NULL then
        $\rho \leftarrow \rho + (\mathbb{H}.mass/\mathbb{H}.radius)$;
    end
end
return $\rho/t$;
```

Table 2: Principal steps in LOF and LiNearN for anomaly detection.

Steps	LOF	LiNearN
1	Compute density distribution: $f_{kNN}(x)$	Compute density distribution: $\bar{f}(x)$
2	Compute $LOF(x)$ using $\dfrac{\sum_{x' \in N(x,k)} \dfrac{f_{kNN}(x')}{\lvert N(x,k) \rvert}}{f_{kNN}(x)}$	[Step 2 is not required for LiNearN.]
3	Rank all instances based on their LOF values in descending order	Rank all instances based on their $\bar{f}(x)$ values in ascending order

4. Density estimators in anomaly detection and clustering tasks

4.1. Anomaly detection

The prevalent anomaly detection approaches (e.g., LOF and ORCA) rely on distance-induced density based on the following definition:

> **Density-based Anomalies** are instances in regions of low density or low relative density in the local neighbourhood.

Variants of the definition are used in literature. Note that though they do not use the term 'density' in their definitions, they are essentially estimating density based on the distance calculations where density is a ratio of the number of instances in a spherical region and the radius of the region. The first definition below has a fixed radius; whereas, the second and third definitions have a fixed number of instances k.

1. **($MinPts$, ϵ)-Distance Anomalies** are instances which have fewer than $MinPts$ instances within a distance ϵ [22].
2. k^{th} **NN Distance Anomalies** are the top-ranked instances whose distance to the k^{th} nearest neighbour is greatest [26].
3. **Average kNN Distance Anomalies** are the top-ranked instances whose average distance to the k nearest neighbours is greatest [4].
4. **Anomalies based on Local Outlier Factor** (LOF) have high LOF values, where LOF of x is a ratio of average density in the local neighbourhood of x and the density of x [11]. The density can be defined using f_{kNN} or other density estimators.

For a given x, LOF computes the average density of k nearest neighbours of x, where the region occupied by these neighbours is the local neighbourhood of x. LOF is computed as the ratio of the average density and the density of x. LOF ≈ 1 indicates that x has the similar density as instances in its local neighbourhood; LOF $\gg 1$ indicates that x has significantly lower density than that from its local neighbourhood, or its nearest neighbours are far away from x; thus x is more likely to be an anomaly.

As pointed by Breunig et al. [11] that the density computed by k-nearest neighbour density estimator, while suitable for detecting global anomalies, will fail to detect local anomalies. LOF is a 'correction' of the density calculated in order to detect both global and local anomalies.

Table 2 shows the principal steps in LOF and LiNearN used to detect anomalies. Not only $\bar{f}(x)$ executes faster than $f_{kNN}(x)$, LiNearN can directly use the density computed to rank instances to detect anomalies, without the additional 'correction' in step 2 required in LOF. This is because $\bar{f}(\cdot)$ is using the nearest neighbour to define local neighbourhood which provides the most localised estimation in a k-nearest neighbour approach. In contrast, $f_{kNN}(\cdot)$ often requires $k \gg 1$ in order to detect clustered anomalies that exist in a data set. The larger k is the less localised is the estimation. See the typical values of k required in the experiments reported in Table 6 in Section 5.1.

Like LiNearN, ORCA requires two steps only but the two steps are coupled to prune the search space in order to improve its time complexity. Unlike LiNearN, the density computed by ORCA does not allow it to detect local anomalies.

4.2. Clustering

In clustering, DBSCAN [15] employs the ϵ-neighbourhood density estimator (f_ϵ) defined in Section 2. DBSCAN requires nearest neighbour searches of the entire data set to find out how many instances are within an ϵ distance from an instance of interest. A data point, x, is identified as a core point when the number of points is equal to or greater than an user defined parameter, $MinPts$, within ϵ distance of x. After all core points have been identified, the clustering process starts with the first core point, marked as belonging to the first cluster. The next core point is selected from unmarked core points within the boundary of the cluster. As more core points are marked, the boundary of the cluster continues to expand. This expansion continues until there are no further unmarked core points found within the cluster boundary. This entire process is repeated for the next cluster until there are no more unmarked core points. Each border point, which is within the ϵ-neighbourhood of a core point but is not a core point, is then connected to the nearest cluster.

Conceptually, the new clustering algorithm is the same as DBSCAN with the following three differences. Firstly, we use the new density estimator $\bar{f}$ to define the density. Secondly, there are no border points. Finally, the clustering is done on the core regions not on the individual core points. We called the algorithm LiNearN-Cluster. Table 3 gives a comparison between the point-based definitions of DBSCAN and the region-based definitions of LiNearN-Cluster.

LiNearN-Cluster employs Algorithms 1 to 3 specified in Section 3.5 to estimate the density of each hypercube region. An additional procedure is required to link all connecting hypercubes, having intersections of non-empty sets, to form a cluster. The procedural outline of this linking process is given below:

① Identify all core regions, $\mathbb{H}$, which has $density \geq MinDensity$. (Algorithm 5)

② Remove all (noise) points in D that do not fall into any of the core regions (Algorithm 6)

③ Assign an id to each core region which has a point $x \in D$ (Algorithm 7)
   (a) Find one previously assigned core region (line 5-13)
   (b) If none found then increment the current id (line 14-17),
   and assign the current id to the unassigned core region (line 31)
   (c) If the core region is previously assigned then produce a link between the current id and previously assigned id (line 23-27)

④ Merge core regions which have connecting link ids to form a cluster (Algorithm 7 : line 36)

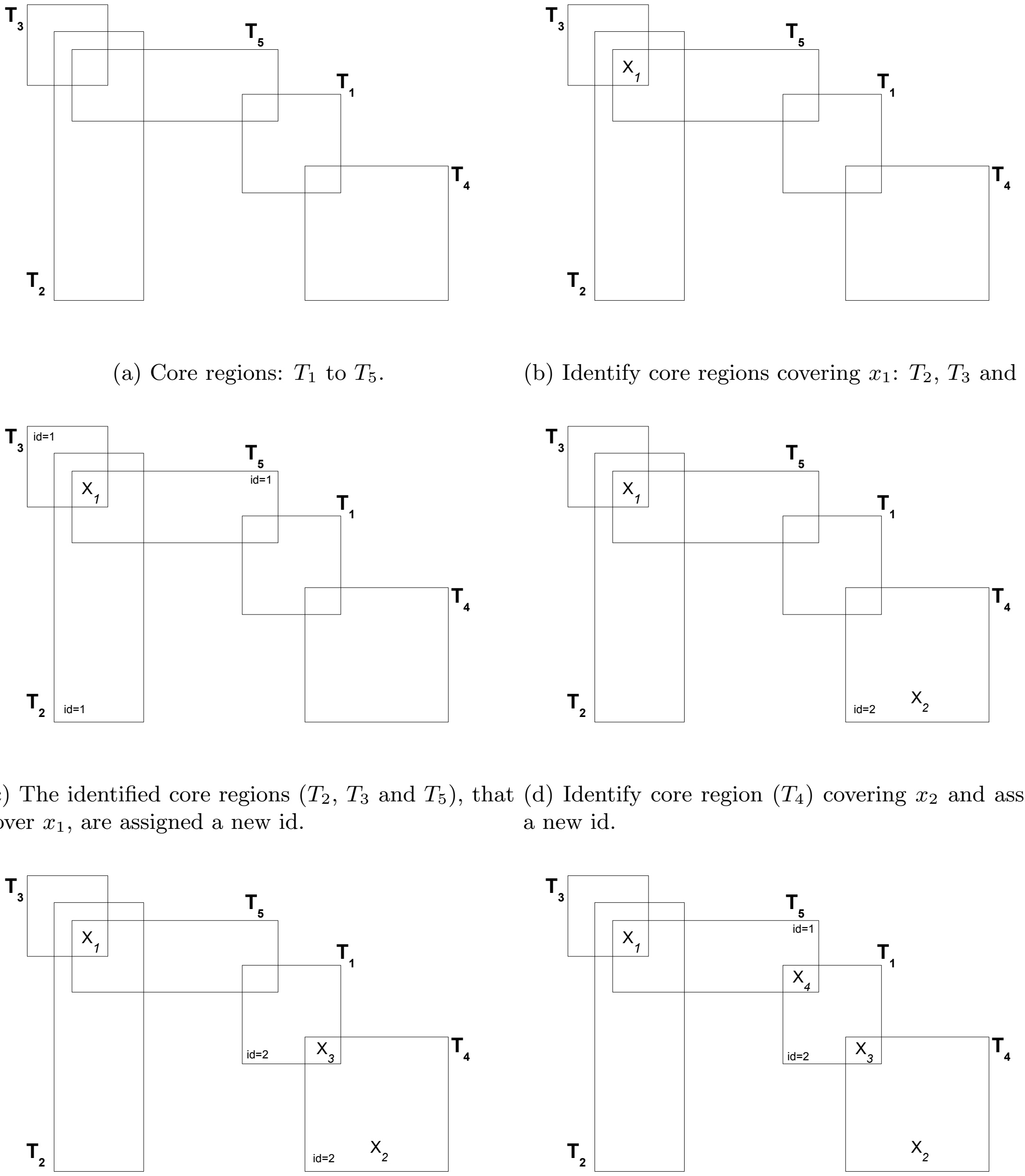
Algorithms 5 to 7 are shown in Appendix C.

Figure 3 illustrates an example linking process as outlined in the above steps for instances x_1, x_2, x_3 and x_4. Figure 3(a) shows a set of five core regions. Three core regions (T_2, T_3, T_5) covering x_1 are identified (Figure 3(b)) and are assigned the current id of 1 (Figure 3(c)). Figure 3(d) shows that x_2 is covered by a single core region, T_4, and it is assigned the next id of 2. Core regions T_1 and T_4 cover x_3 (Figure 3(e)). Since T_4 has been previously assigned an id of 2, T_1 will be assigned the same id. The final point x_4 falls into core regions T_1 and T_5 (Figure 3(f)). Now that T_1 and T_5 have previously assigned ids, they are linked as a pair.

After core regions, covering all instances in D, have been assigned id's, all linking pairs will be merged into a single cluster with the same id as the final step of the clustering process.

Table 3: Point-based definitions for DBSCAN versus region-based definitions for LiNearN-Cluster. The definitions for DBSCAN are extracted from Ester et al. [15] for comparison.

DBSCAN (Point-based)	LiNearN-Cluster (Region-based)
Definition P1: (ϵ-neighbourhood of a point) The **ϵ-neighbourhood** of a point p, denoted by $N_\epsilon(p)$, is defined by $N_\epsilon(p) = \{q \in D \mid dist(p,q) \leq \epsilon\}$.	*Definition S1:* $T(x)$ is a **core region** of point x wrt $MinDensity$ if $\rho(x \mid H, \mathfrak{D}) \geq MinDensity$.
Definition P2: (directly density-reachable) A point p is **directly density-reachable** from a point q wrt ϵ, $MinPts$ if 1. $p \in N_\epsilon(q)$ and 2. $\lvert N_\epsilon(q) \rvert \geq MinPts$ (core point condition).	*Definition S2:* $T_r(\cdot)$ is **density-connected** to $T_s(\cdot)$ wrt $MinDensity$ if there is a chain of regions $T_1(\cdot), \ldots, T_g(\cdot)$ where $r = 1$ and $s = g$ such that $T_\imath(\cdot) \cap T_{i+1}(\cdot) \neq \emptyset$ and $T_\imath(\cdot)$ is a core region for all $\imath$ wrt $MinDensity$.
Definition P3: (density-reachable) A point p is **density-reachable** from a point q wrt ϵ and $MinPts$ if there is a chain of points $p_1, \cdots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i.	*Definition S3:* An **arbitrary-shape cluster** C wrt $MinDensity$ is a non-empty subset of a database D satisfying the following condition: $\forall r, s; T_r(\cdot), T_s(\cdot) \subset C$: $T_r(\cdot)$ is density-connected to $T_s(\cdot)$ wrt $MinDensity$.
Definition P4: (density-connected) A point p is **density-connected** to a point q wrt ϵ and $MinPts$ if there is a point o such that both p and q are density-reachable from o wrt ϵ and $MinPts$.	*Definition S4:* Let $C_1, \ldots, C_k$ be the clusters of D wrt $MinDensity$. **Noise** is the set of points in D not belonging to any cluster $C_\jmath$, i.e., noise $= \{\mathbf{x} \in D \mid \forall \jmath : \mathbf{x} \notin C_\jmath\}$.
Definition P5: (cluster) Let D be a database of points. A **cluster** C wrt ϵ and $MinPts$ is a non-empty subset of D satisfying the following conditions: 1. $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt ϵ and $MinPts$, then $q \in C$. (Maximality). 2. $\forall p, q \in C$: p is density-connected to q wrt ϵ and $MinPts$ (Connectively).	
Definition P6: (noise) Let $C_1, \cdots, C_k$ be the clusters of the database D wrt parameters ϵ_i and $MinPts_i$, $i = 1, \cdots, k$. Then we define the **noise** as the set of points in the database D not belonging to any cluster C_i, i.e. noise $= \{p \in D \mid \forall i : p \notin C_i\}$.	



(a) Core regions: T_1 to T_5.

(b) Identify core regions covering x_1: T_2, T_3 and T_5.

(c) The identified core regions (T_2, T_3 and T_5), that cover x_1, are assigned a new id.

(d) Identify core region (T_4) covering x_2 and assign a new id.

(e) Identify core region (T_1) covering x_3 and assign an existing id.

(f) Identify core regions (T_1 and T_5) covering x_4 and create a link between the two core regions which already have assigned id's.

Figure 3: An example of the LiNearN-Cluster linking process.

Table 4: A comparison of time and space complexities.

	Time complexity	Space complexity
LiNearN	$O(nt\psi d)^{\ddagger}$	$O(t\psi d)$
DOLPHIN	$O(n^2 d)$	$O(nd)$
	$O(\frac{k}{p}nd)^{\dagger}$	$O(\frac{k}{p}d)^{\dagger}$
ORCA	$O(n \log n \cdot d)$	$O(nd)$
LOF	$O(n^2 d)$	$O(nd)$
DBSCAN	$O(n^2 d)$	$O(nd)$
LiNearN-Cluster	$O(nt\psi d)$	$O(t\psi d)$

‡ $O(t(\psi + \Psi)\psi d)$ is the time complexity required in 'training', i.e., defining the local regions using $\mathcal{D}$ and estimating density of the local regions using $\mathfrak{D}$, as described in steps ① to ④ in Section 3.5. $O(nt\psi d)$ is the time complexity in 'testing', i.e., estimating every $x \in D$, as described in step ⑤. Since $\psi + \Psi$ is constant and significantly less than n, it is omitted in the Big-O notation.

† Under a special condition: p is the probability of randomly picking a point from the data set which is a neighbour of the point under consideration using a search index; k is the number of nearest neighbours; d is the number of dimensions.

4.3. Time and Space Complexities for nearest neighbour-based anomaly detection and clustering algorithms

This section compares the time and space complexities of three state-of-the-art anomaly detection algorithms [3, 11] and the state-of-the-art clustering algorithm [15] with LiNearN. All of these algorithms uses density as the basis of their operations. Table 4 lists the complexities for the four algorithms, LiNearN and LiNearN-Cluster.

In nearest neighbour algorithms, the most computationally expensive part of the process is to do the nearest neighbour search which has $O(n^2)$ time complexity. These algorithms have already used or could use some indexing scheme to their time complexities.

LiNearN has a significant advantage over the three k-nearest-neighbour-based anomaly detection algorithms in terms of both time and space complexities. This is mainly due to the fact that LiNearN only needs a small subsample to identify local neighbourhoods, where $\psi \ll n$; ψ (also t) can be fixed in practice, regardless of the size of the given training set.

LiNearN-Cluster requires steps for the clustering process in addition to those steps in LiNearN. Even with these additional steps, the time and space complexity are still the same as the basic LiNearN.

Another distinguishing feature is that the space complexities of both LiNearN and LiNearN-Cluster are constant, independent of the data size. These properties make LiNearN and LiNearN-Cluster an ideal candidate to apply to domains with huge data size or infinite data such as data streams.

5. Empirical Evaluations

All evaluations were conducted in the unsupervised learning settings in both anomaly detection and clustering tasks.

The experiments were conducted as single thread jobs processed at 2.27 GHz in a Linux cluster with 40 GB memory unless a different memory requirement is specified. Both LiNearN and LiNearN-Cluster algorithms were written in JAVA in the WEKA platform [20], so as DBSCAN. LOF and DBSCAN were written in Java in the ELKI platform version 0.5 [1]. ORCA was written in C++ (www.stephenbay.net/orca/).

In anomaly detection, we compare LOF and ORCA with LiNearN using the same eleven data sets as used by Liu et al. [24]. Ring-Curve-Wave-TriGaussian, OneBig and Pendigits data sets used by Ting et al. [33] are employed to compare LiNearN-Cluster with DBSCAN. The additional Animals, Segment, WDBC, Iris and Yeast data sets are from UCI Machine Learning Repository [18]. The characteristics of the data sets are shown in Table 5. They were selected to have different data characteristics of data size, number of dimensions and clusters.

Table 5: Data sets. The first eleven data sets are for anomaly detection; the last eight data sets are for clustering.

Data	Size n	d	anomaly class
Http	567,497	3	attack (0.4%)
ForestCover (FC)	286,048	10	class 4 (0.9%) vs. class 2
Mulcross	262,144	4	2 clusters (1%)
Smtp	95,156	3	attack (0.03%)
Shuttle	49,097	9	classes 2, 3, 5, 6, 7 (7%)
Mammography	11,183	6	class 1 (2%)
Satellite	6,435	36	3 smallest classes (32%)
Pima	768	8	pos (35%)
Breastw	683	9	malignant (35%)
Arrhythmia	452	274	classes 03, 04, 05, 07, 08, 09, 14, 15 (15%)
Ionosphere	351	32	bad (36%)
Animals	200,000	72	4 clusters
OneBig	68,000	20	9 clusters and 10,000 noise instances
Pendigits	10,992	16	10 clusters
Segment	2,310	19	19 clusters
Yeast	1,484	8	10 clusters
WDBC	569	30	2 clusters
Iris	150	4	3 clusters
Ring-Curve-Wave-TriGaussian	7,000 to 10 million	48	7 clusters in 6 relevant attributes; 42 irrelevant attributes

In anomaly detection tasks, we conduct a parameter search for all three algorithms and report the best result. For LOF and ORCA, the best k is searched between 5 and 4000 (or up to the data size for small data sets), with steps from 5, 10, 20, 40, 60, 80, 150, 250, 300, 500, 1000, 2000, 3000 to 4000. For LiNearN, the best ψ is searched from 2, 4, 8, 16, 32, 64 to 128; and we also report the result of the default setting (i.e., $\psi = 2$). The other default settings are fixed: $t = 1000$ and $\Psi = 256$ for LiNearN; and $p = 2$ for both LOF and ORCA. The remaining parameter settings for ORCA are set to their default values except for the number of returned anomaly points which is set to twice the number of anomaly points in the selected data set but capped at 50% of the total data size. The parameter setting not only provides information which is usually not available in practice but also reduces the number of distance calculations for the top anomaly points only. This setting gives ORCA an unfair advantage to LiNearN which computes the density for every point in the data set.

We report the result in anomaly detection task in terms of CPU runtime and AUC (Area Under ROC Curve) based on the ranked result. The clustering result was reported in terms of CPU runtime, number of clusters identified, number of unassigned instances, and F-measure which was calculated based on assigned instances only. F-measure = 1 when all assigned instances are in the correct clusters, i.e., perfect clustering; and F-measure = 0 if all instances are assigned to wrong clusters.

We report the results in anomaly detection in Section 5.1, clustering in Section 5.2 and a summary is provided in Section 5.3.

5.1. Anomaly Detection

The ELKI platform has an option to use an indexing scheme when using LOF to speed up the nearest neighbour search process. We used the R*-Tree indexing [8] for all of the data sets except http. LOF was unable to process http within the physical memory of 40GB; therefore, we reported the results without using indexing for this data set. We used the default settings for indexing except in the arrhythmia data set. The default page size was changed from 4KB to 32KB because of the high dimensionality in the arrhythmia data

Table 6: AUC results for LiNearN, LOF and ORCA.

	AUC				Best Parameter		
	LiNearN		LOF	ORCA	LiNearN	LOF	ORCA
	$\psi = 2$	best ψ			ψ	k	k
http	1.00	1.00	1.00	1.00	2	500	3000
FC	0.81	0.95	0.94	0.88	8	1000	3000
mulcross	1.00	1.00	1.00	1.00	2	2000	3000
smtp	0.78	0.95	0.95	0.74	32	1000	40
shuttle	0.99	0.99	0.98	0.99	2	4000	4000
mammography	0.82	0.87	0.68	0.67	16	4000	80
satellite	0.69	0.72	0.79	0.78	4	1000	500
pima	0.72	0.74	0.72	0.73	4	250	150
breastw	0.98	0.98	0.96	1.00	2	300	300
arrhythmia	0.67	0.71	0.80	0.80	128	80	150
ionosphere	0.94	0.96	0.90	0.93	4	80	5

Table 7: Runtime results (in seconds) for LiNearN, LOF and ORCA. Separate training and testing results are shown for LiNearN.

	LiNearN				LOF	ORCA
	Training		Testing			
	$\psi = 2$	best ψ	$\psi = 2$	best ψ		
http	0.27	0.27	71.4	71.4	19,965	78,931
FC	0.25	0.29	57.6	444.5	2,918	94,336
mulcross	0.29	0.29	33.2	33.2	2,169	56,372
smtp	0.28	0.67	12.6	556.5	373	125
shuttle	0.34	0.34	7.8	7.8	656	16,137
mammography	0.29	0.34	1.1	8.9	127	4.0
satellite	0.30	0.27	1.3	3.4	23.60	56.1
pima	0.20	0.21	0.21	0.21	0.44	0.46
breastw	0.23	0.24	0.10	0.15	0.44	1.02
arrhythmia	0.96	13.03	1.27	28.60	1.18	0.45
ionosphere	0.24	0.23	0.11	0.13	0.26	0.03

set. It should be pointed out that LOF within the ELKI platform does a pre-computation of every pair-wise distance before commencing the algorithm as a speedup technique.

Table 6 shows the AUC results. It is interesting to note that LiNearN with the default setting $\psi = 2$ has competitive AUC results in seven out of the eleven data sets. With a parameter search, all three algorithms, LiNearN, LOF and ORCA, produce similar AUC results. However, both LOF and ORCA require a much wider range of parameter search than LiNearN in order to achieve good AUC results. In contrast, LiNearN achieves good AUC performance using small ψ values. Note that the smaller ψ is the faster LiNearN runs; this applies to LOF and ORCA too for the k parameter. Both the range of search and the actual value required put LiNearN in a more favourable position than LOF and ORCA.

In addition, LiNearN also runs significantly faster than LOF and ORCA, in the large data sets. In the largest data set, http, LiNearN is faster than LOF and ORCA by a factor of 279 and 1101, respectively. The actual runtime results are shown in Table 7.

5.1.1. Scaleup tests

We conducted two scaleup tests. We scaled up the data size in the first and the number of dimensions in the second.

The results of the first scaleup test using Mulcross are shown in Figure 4. The k parameter for LOF is set to 2,000 and the ψ parameter for LiNearN is set to 2. The left graph of Figure 4 shows the result when scaling up by data size. When the data size was increased by a factor of 2, 4, 10 and 40, LiNearN increased its runtime by a factor of 2.2, 4.5, 14 and 61, respectively. In contrast, the factors of runtime increases by LOF, without indexing, are 4.2, 17, 105, 1700 (the last one is a projected value); and the corresponding factors for LOF with indexing are 1.9, 5, 15 and 88. With ten million instances (at the last point of each line), LiNearN completed in just under 31 minutes; whereas, LOF with indexing took 1.7 hours. Without indexing, LOF is projected to take 100 days! These results are consistent with the time complexities stated in Table 4.

The right graph of Figure 4 shows the result of the same experiment but the number of dimensions was increased from 4 to 5. Note that the k value for LOF had to be reduced from 2000 to 10 in order to run within 200GB of physical memory; whereas, the memory usage is under 40G for all points in the left graph. The reason for the high memory usage is two fold: ELKI does the entire indexing in memory and before running the LOF algorithm, it pre-computes all the distance calculations for each pair-wise distance for a given k value. The contrast in the left and right graphs in Figure 4 show that LOF has significantly increased its runtime and runtime ratio by increasing the number of dimension from 4 to 5 only.

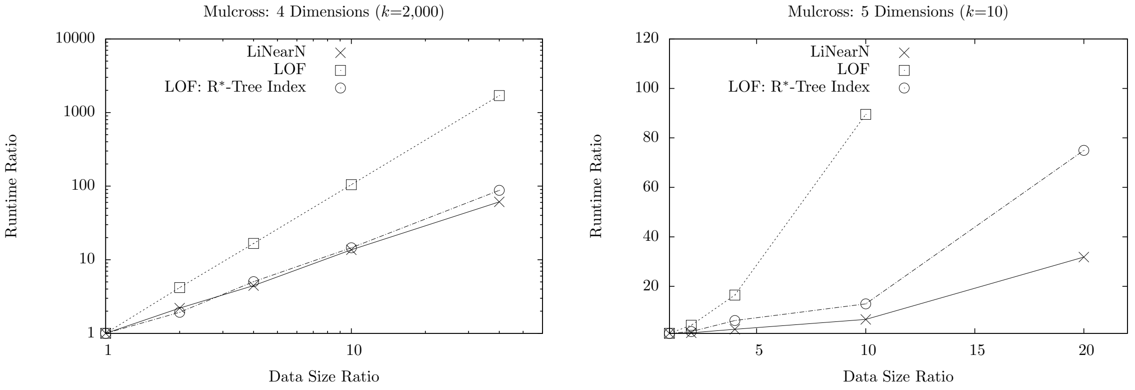


Figure 4: Scaleup test for LiNearN and LOF as data size increases from 250,000 to ten million. By using 250,000 as the base, these increases have the ratios of 1 to 40. The final point for LOF without indexing is a projected value in the left graph; and the final point for LOF without indexing, in the right graph, is not available because it took longer than 100 days.

Figure 5 shows the runtime result when scaling up by increasing the number of dimensions from 5 by factors of 2, 4, 10 and 20. For LiNearN, the increases in the runtime are 1.3, 1.3, 1.6 and 1.9, respectively; LOF without indexing, the increases are 1.6, 2.0, 2.5 and 4.0; however, with indexing, LOF's increases are 5.1, 15 and 25 for the first three points. The last point could not produced because the process ran out of memory. For the second last point which has 50 dimensions, LiNearN completed within 134 seconds and LOF without indexing completed in over 6 hours; however, LOF with indexing took nearly 18.3 hours to complete. This highlights that such an indexing scheme has significant overheads for problems with a moderate number of dimensions; and LOF with indexing will run slower than LOF without it; in addition to high memory demands.

This experiment shows that while an indexing scheme such as R*-Tree can speed up the nearest neighbour search significantly, it has high memory requirements. Also, this speedup only occurs in problems with few dimensions. With tens of dimensions[1], it is better not to use indexing in the Mulcross data set.

[1] These are not high dimensional problems with tens of thousands of dimensions. LiNearN and LOF are not designed to

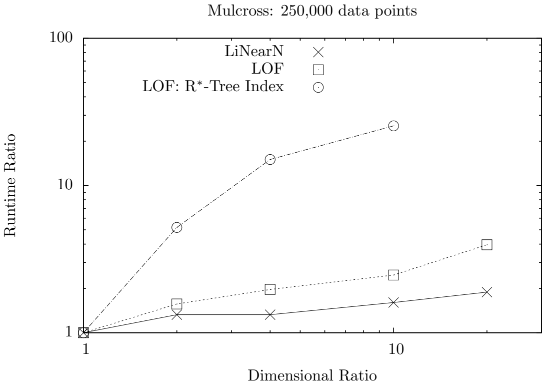


Figure 5: Scaleup test for LiNearN and LOF as the number of dimensions increases from 5 to 100. By using 5 as the base, these increases have ratios of 1 to 20. The final point for LOF when indexing is not available because the memory is not sufficient to run the experiment.

5.1.2. Sensitivity test

The aim of this subsection is to examine how sensitive the parameters are to the performance of LiNearN in terms of AUC and runtime. Because the results are similar, only the results of the two largest data sets are shown. Figure 6 shows AUC and runtime as Ψ was increased from 10 to 1000. It is interesting to note that only a small Ψ is required to achieve good AUC result, and a large Ψ value gives minor or no improvement in AUC.

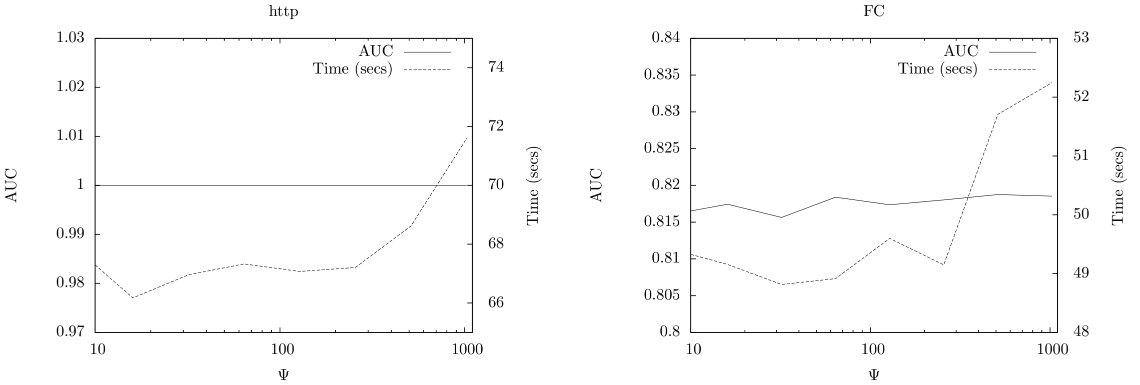


Figure 6: AUC and runtime as a result of varying Ψ of LiNearN. The default settings are used for the other parameters: $\psi = 2$ and $t = 1000$.

Figure 7 shows the AUC and runtime results as t was increased from 10 to 1000. It also shows that only a small t of 100 is required to achieve good AUC result.

Note that the runtime increase between $t = 100$ and $t = 1000$ was due to the memory swap between cache and main memory or disk, not because of algorithmic procedure as it is linear to t.

Figure 8 shows the results as ψ was increased from 2 to 128 by multiplying with 2 at each step increase. Among the three parameters, ψ has the highest influence on the AUC result because it controls the smoothness of the density distribution estimated, as shown in Figure 2. It is interesting to note that small ψ values achieve good AUC results, and this allows LiNearN to detect anomalies fast.

deal with high dimensional problems.

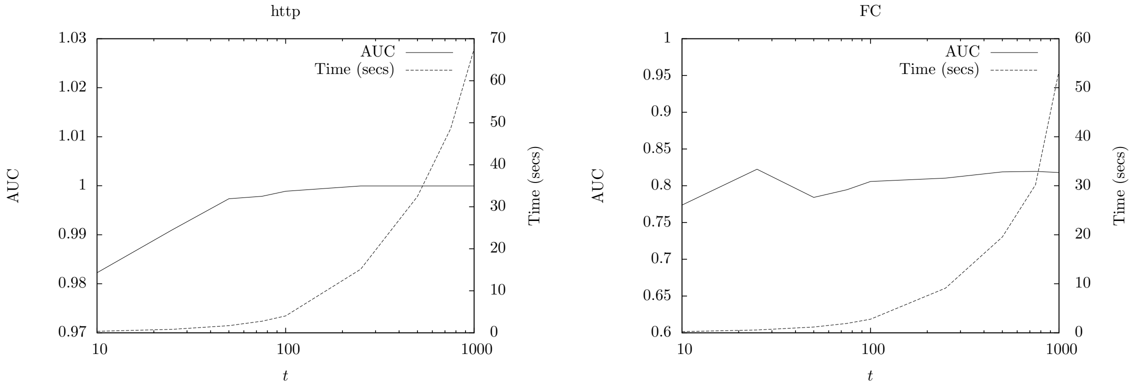


Figure 7: AUC and runtime as a result of varying t of LiNearN. The default settings are used for other parameters: $\psi = 2$ and $\Psi = 256$.

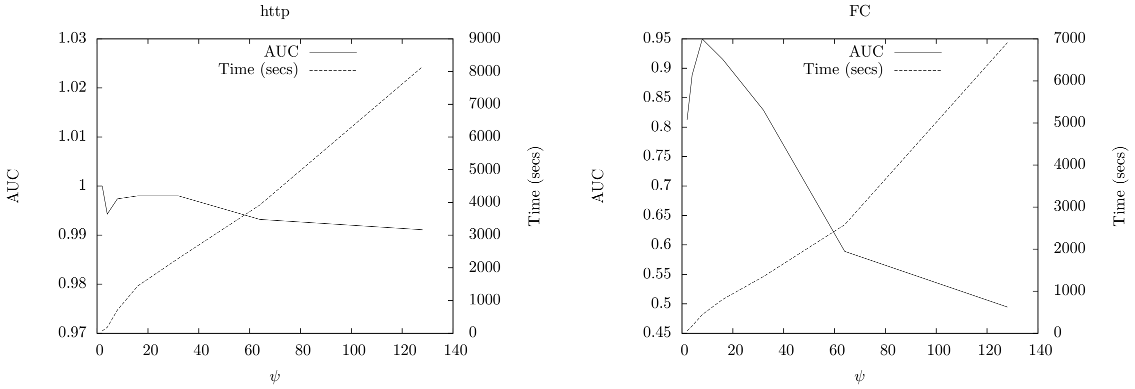


Figure 8: AUC and runtime as a result of varying ψ of LiNearN. The default settings are used for other parameters: $t = 1000$ and $\Psi = 256$.

5.1.3. Identifying Local Anomalies

With LiNearN, the probability of selecting an anomaly into a subsample is significantly smaller than normal points. Thus, only a small number of t subsamples will include anomalies.

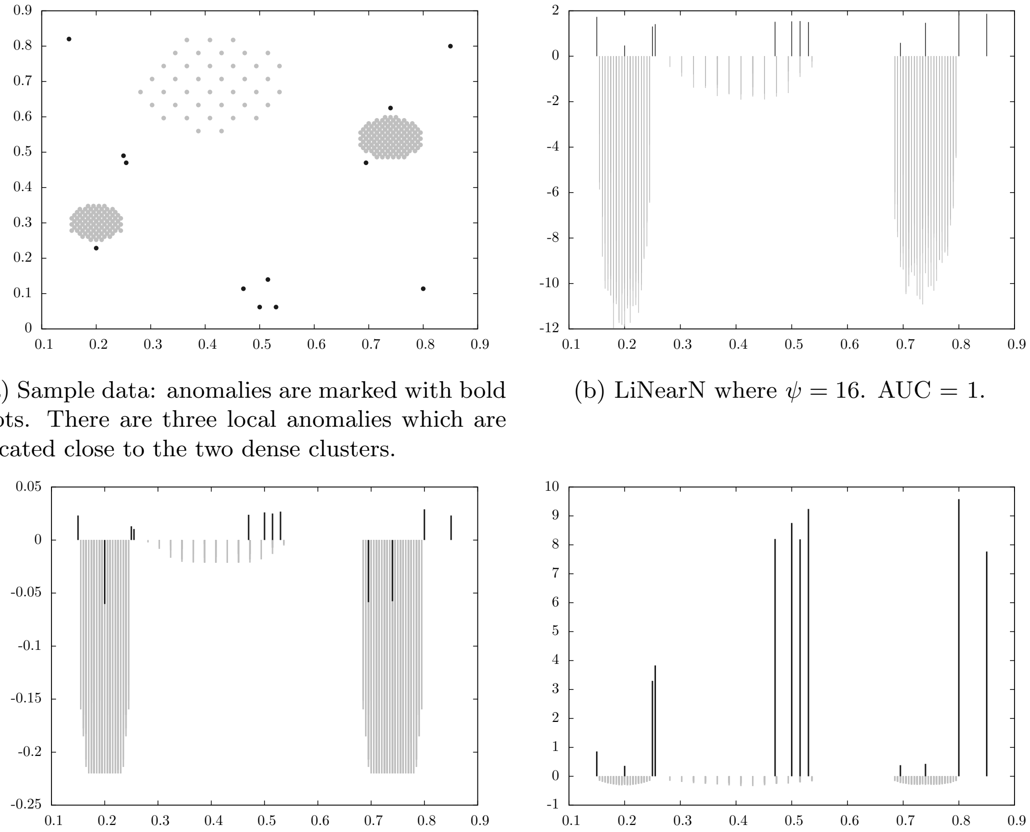
Let ϵ be the probability of selecting an anomaly into a subsample; and t is the number of subsamples or models generated from subsamples. Thus, there are $(1 - \epsilon) * t$ models built without anomalies and they will all have zero density estimation for anomalies. Only $\epsilon * t$ models built from subsamples containing anomalies may have low density estimations for anomalies[2]. As a result, the densities estimated by LiNearN for anomalies will be low.

On the other hand, all t models will be built from subsamples containing mostly normal points, if not all normal points. Thus, $\bar{f}(\cdot)$ will estimate the densities for normal points to be higher than those for anomalies.

Note that the above effect is the same, regardless the point is a local or global anomaly.

It is interesting to note that k-nearest neighbour density estimator is known to have problems in identifying local anomalies [11]. A similar example provided by Breunig et al. [11] is given in Figure 9, where the local anomalies are estimated to have the same density of the sparse cluster; thus, they would not be identified as an anomaly by k-nearest neighbour-based anomaly detector such as ORCA, as shown in Figure 9(c).

[2] These models could still produce zero density estimation for anomalies when they are significantly different from those appeared in the training subsamples.



(a) Sample data: anomalies are marked with bold dots. There are three local anomalies which are located close to the two dense clusters.

(b) LiNearN where $\psi = 16$. AUC = 1.

(c) kNN where $k = 16$. AUC = 0.96.

(d) LOF where $k = 16$. AUC = 1.

Figure 9: Examining the ability to detect local anomalies using LiNearN, kNN and LOF. The 'inverse' anomaly scores are shown for LiNearN and kNN. Low scores indicates normal points; whereas, the higher score indicates anomaly. To highlight the difference between anomalies and normal points, the scores shown are a result of subtracting the maximum score of the normal points such that all normal points will have (adjusted) scores below or equal to zero. Anomalies are shown as black lines and normal points as grey lines.

Both LiNearN and LOF can detect all anomalies as shown in Figure 9(b) and 9(d).

In contrast, based on nearest neighbour, LiNearN can easily detect local anomalies for the reason stated above. Instead of computing relative density, as in LOF [11] to 'correct' the deficiency of k-nearest neighbour procedure in identifying local anomalies, LiNearN employs sampling to reduce the anomalies' presence in the training samples, as the key step to identify local and global anomalies.

5.2. Clustering

Table 8 shows the clustering results for LiNearN-Cluster and DBSCAN. In the OneBig data set that has nine clusters with 10000 noise points, LiNearN-Cluster produced ten clusters which include a cluster consists of noise, and additional 3000 points not assigned to any clusters. DBSCAN also produced nine clusters with slightly more than ten thousand unassigned points. Both algorithms produced F-measures equal to one for all the nine clusters. In terms of runtime, both LiNearN-Cluster and DBSCAN took about the same time to complete this task.

In the pendigits data set, LiNearN-Cluster produced a better clustering result than DBSCAN in terms of the number of clusters and the number of unassigned instances. Because LiNearN-Cluster assigned about

Table 8: Clustering results for the OneBig and Pendigits data sets using LiNearN-Cluster ($\psi = 600$ for OneBig; $\psi = 512$ for Pendigits; $\psi = 43$ and $t = 30000$ for Animals. Other default settings are $\Psi = 256$ and $MinDensity = 6$) and DBSCAN ($\epsilon = 0.1$ for OneBig; $\epsilon = 0.2$ for Pendigits; $\epsilon = 0.7$ for Animals and $MinPts = 6$).

		OneBig			Pendigits			Animals	
		LiNearN-Cluster	DBSCAN		LiNearN-Cluster	DBSCAN		LiNearN-Cluster	DBSCAN
Runtime		8565	8406		821	181		38532	240187
#cluster	[9]	10	9	[10]	18	65	[4]	4	4
#unassigned		2920	10005		1495	6251		18597	3347
F-measure		1.00	1.00		0.68	0.75		1.00	1.00

5000 more instances to clusters, it has slightly lower F-measure than DBSCAN. In this relatively small data set, DBSCAN ran faster than LiNearN-Cluster.

The Animals is an interesting synthetic data set because of how it is constructed. It has approximately 16000 mini-clusters, each has either 12 or 13 points. These mini-clusters are grouped into 4 clusters with 2 being in close proximity of each other. With $\epsilon = 0.3$, DBSCAN produced the 16000 mini-clusters. In order for DBSCAN to detect the 4 clusters, ϵ needed to be set to 0.7. In this kind of data characteristics, LiNearN requires to have a high t in order to link the related mini-clusters into a cluster. Using $\psi = 43$ and $t = 30000$, LiNearN produced the correct 4 clusters but had approximately 1500 mini-clusters unassigned.

Table 9 shows the results using two smaller data sets, Iris and Yeast. In terms of unassigned points and the number of clusters for Iris, LiNearN-Cluster produced a better result; whereas, DBSCAN has almost half of the data points unassigned. Both LiNearN-Cluster and DBSCAN have similar F-Measure scores. For Yeast, DBSCAN has almost all of the data points unassigned; whereas, LiNearN-Cluster has about one-third unassigned. In terms of F-Measure, LiNearN-Cluster outperformed DBSCAN. LiNearN-Cluster has longer runtime than DBSCAN in these small data sets. Table 10 shows similar results in Segment and WDBC data sets.

None of the above results reveal the time complexities of the algorithms. Therefore, we conducted a scaleup test using the same data set as used by Ting et al. [33], i.e., Ring-Curve-Wave-TriGaussian. The data characteristic is shown in Appendix D.

We used DBSCAN with and without indexing using R*-Tree in the ELKI platform in this experiment. The scaleup result is shown in Figure 10. Using 7000 instances as the base, the data size were increased by a factor of 10, 75 and 150 to reach one million instances. With these data size ratio increases, LiNearN-Cluster's runtime ratio were increased by a factor of 12.5, 93.6 and 206, respectively. In contrast, without indexing, DBSCAN's runtime ratio were increased by a factor of 112, 6676 and 25754, respectively; with indexing, the ratio were 88, 6080 and 20980, respectively. At data size ratio = 150 with one million instances, DBSCAN without indexing completed the task in 98 hours; whereas, LiNearN-Cluster finished in 26 minutes. DBSCAN with indexing completed in 77 hours which gives an improvement of 21.5% over DBSCAN without indexing. This result shows the advantage of LiNearN-Cluster over DBSCAN, and using indexing does not improve DBSCAN's runtime significantly for problems with a moderate number of dimensions.

5.3. Summary

While the proposed approach also relies on distance calculations to find nearest neighbour, like all existing nearest neighbour algorithms, there are important differences. First, the proposed approach reduces the computationally expensive $O(n^2)$ nearest neighbour search process to a $O(n)$ search process which involves a small subset of size $\psi \ll n$. This is demonstrated in both anomaly detection and clustering tasks.

Second, as shown by the results in Table 6, existing k-nearest neighbour anomaly detectors require a significant amount of search to find an appropriate k in order to produce good results; this adds to its

Table 9: Clustering results for the Iris and Yeast data sets using LiNearN-Cluster ($\psi = 32$ for Iris; $\psi = 160$ for Yeast. Other default settings are $\Psi = 256$ and $MinDensity = 6$) and DBSCAN ($\epsilon = 0.1$ for Iris; $\epsilon = 0.07$ for Yeast; and $MinPts = 5$).

		Iris			Yeast	
		LiNearN-Cluster	DBSCAN		LiNearN-Cluster	DBSCAN
Runtime		0.51	0.1		21.3	1.8
#cluster	[3]	4	5	[10]	15	12
#unassigned		16	70		572	1197
F-measure		0.90	0.89		0.33	0.20

Table 10: Clustering results for the Segment and WDBC data sets using LiNearN-Cluster ($\psi = 400$ and $\Psi = 1024$ for Segment; $\psi = 30$ and $\Psi = 512$ for WDBC. The default setting is $MinDensity = 6$) and DBSCAN ($\epsilon = 0.1$ for Segment; $\epsilon = 0.3$ for WDBC; and $MinPts = 6$).

		Segment			WDBC	
		LiNearN-Cluster	DBSCAN		LiNearN-Cluster	DBSCAN
Runtime		50	7		1.5	1.3
#cluster	[19]	40	43	[2]	2	2
#unassigned		417	1282		319	331
F-measure		0.60	0.62		0.96	0.89

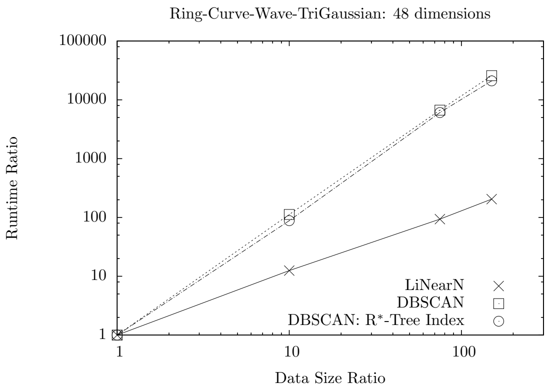


Figure 10: Scaleup test for LiNearN-Cluster and DBSCAN using the 48 dimensional Ring-Curve-Wave-TriGaussian data set. 7000 instances are used as the base for data size ratio. The data size is increased by a factor of 10, 75 and 150 which has one million instances.

already heavy computational cost. LiNearN's ability to use low ψ for anomaly detection has the following advantages over existing k-nearest neighbour anomaly detectors:

- Significantly lower runtime
- Require a search in a small range of ψ values only.
- Indexing schemes, normally required to speed up nearest neighbour search, become unnecessary.

In clustering, both LiNearN-Cluster and DBSCAN require a search of the key parameters, ψ and ϵ, respectively, in order to identify connecting local regions to achieve good results. Nevertheless, LiNearN-Cluster has more points assigned than DBSCAN in six out of the seven data sets, and with comparable results in terms of F-Measure and the number of clusters.

6. Discussion

The nature of nearest neighbour approach necessitates $O(n)$ distance calculations in order to find the nearest neighbours in a data set of size n. This paper shows that if the aim is to do density estimation, then $O(n^2)$ distance calculations are not required, even though nearest neighbour approach is adopted. The proposed new approach opens up a new opportunity for many tasks in which nearest neighbour algorithms have been applied. The aims of these tasks need to be carefully examined to determine whether $O(n^2)$ distance calculations are necessary. If they are not, then the proposed approach is a way to convert an $O(n^2)$ algorithm to an $O(n)$ algorithm. Only tasks in which $O(n^2)$ distance calculations are absolutely necessary that the current research focus in indexing is suitably applied.

Indeed, we show in anomaly detection tasks that, while the ranking requires density estimation, it does not require $O(n^2)$ distance calculations and a correction suggested by LOF to enable density to be used directly to detect local anomalies. In clustering tasks, the aim is to identify core regions and link all neighbouring core regions to form a cluster. We also show that this also does not require $O(n^2)$ distance calculations to achieve the aim.

It is interesting to identify the steps in the process where the speedup was achieved by LiNearN. In anomaly detection tasks, density must be estimated for every point in the given data set. The density for a point is estimated from the t hypercubes covering the point, rather than invoking $n - 1$ distance calculations. In clustering tasks, the speedup occurs in two steps. First, density needs to be estimated for local regions only in LiNearN-Cluster. In contrast, density must be computed for every single point in the given data set in DBSCAN. Second, because the number of local regions is significantly smaller than the number of points in the data set, the number of links required to form clusters become significantly smaller for LiNearN-Cluster than that for DBSCAN.

An ensemble of k-nearest neighbours does not usually work because k-nearest neighbour classifiers, in the classification context, are a stable learner like SVM and Naive Bayesian classifiers. Most of the ensemble approaches, e.g., Bagging [10] and Boosting [19], only work for unstable learners such as decision trees.

Although there are recent ensemble approaches (e.g., Feating [34] and $Local_{Model}$ [35]) that have been shown to work for stable learners such as k-nearest neighbour, the proposed approach has importance differences. First, Feating and $Local_{Model}$ are specifically designed for classification tasks only; whereas, LiNearN is a density estimator which has a wider application to different tasks. Second, LiNearN employs a subsample, which is significantly smaller than the given data set, to build each model in the ensemble. In contrast, Feating and $Local_{Model}$[3] build individual models using the entire data set. Third, both Feating and $Local_{Model}$ use a tree structure to define local regions; LiNearN uses nearest neighbours to define local regions. Fourth, in LinearN, the shape of the local regions can be easily changed by setting the p parameter in L^p-norm. The shape of the local regions is harder to change for both Feating and $Local_{Model}$, if not impossible.

[3] Note that $Local_{Model}$ is not an ensemble approach but constructs a global model which consists of many local models. Boosting can then be used to improve the predictive accuracy of individual $Local_{Model}$.

A feature bagging method [23] has been proposed to build multiple LOF models, each from a random feature subset, and then aggregate the LOF scores from all models to produce the final score. However, this method is reported to be marginally better than single LOF in five out of six data sets [23].

There are methods to reduce the number of instances in the given data in order to reduce the search cost (e.g., [2, 14, 29, 31, 36, 37]). These methods require to spend significant amount of time in the instance reduction process. In contrast, the instance selection process in LiNearN is a random sampling process which can be achieved very quickly.

In the context of supervised learning, Salzberg [28] proposed a k-nearest neighbour algorithm, called Nested Generalised Exemplars (NGE), which constructs hyper-rectangles to replace and reduce the number of correctly classified instances while storing incorrectly classified instances like all nearest neighbour algorithms. The purpose of NGE differs substantially from LiNearN. NGE is a classifier, LiNearN is an density estimator that can be used for various pattern recognition tasks, potentially include classification. The algorithmic differences are: NGE is a single model and stores both hyper-rectangles and instances; whereas, LiNearN is an ensemble approach which stores a small subset of instances to form hypercube regions.

DEMass [33] is a closely related work which is a grid-based method that has a global parameter, like existing nearest neighbour density estimators, to control the size of the grid. Like all grid-based methods, the grid in DEMass has a single size which does not adapt to different data distributions in local regions. Thus, LiNearN is expected to be more adaptive to data distribution in local regions than DEMass. On the other hand, LiNearN can be expected to run slower than DEMass because of the use of distance measure.

Like the Voronoi diagram [6], LiNearN divides the feature space using the nearest neighbour rule; but they have important differences. First, the union of all regions in the Voronoi diagram covers the entire feature space, but not in LiNearN. Second, the Voronoi diagram is a result of all instances in the given data set; LiNearN is a density estimator constructed from multiple models using subsets of the given data set.

A Voronoi region can be defined as:

$$V_c = \{x \in X| \; \|x - x_c\| \leq \|x - x_i\| \; \forall i \neq c \text{ and } x_i, x_c \in D\}.$$

In contrast, local region H_c in LiNearN can be defined as:

$$H_c = \{x \in X| \|x - x_c\| < r_c \text{ and } x_c \in \mathcal{D} \subset D\}.$$

If $\mathcal{D} = D$, then the bisector between the two nearest neighbours in $\mathcal{D}$ will be the same for V_c and H_c. Even in this case, the volume of H_c is solely determined by the nearest neighbour of x_c; whereas, the volume of V_c is determined by all nearest neighbours of x_c in all directions. In addition, the shape of H_c is solely due to L^p-norm; but the shape of V_c relies on x_c's nearest neighbours and L^p-norm.

It should be pointed out that in small data sets the proposed method runs slower than the existing nearest neighbour density estimator without indexing. This is also true for any indexing scheme which has overheads that outweigh its use in small data sets. Indexing schemes and our proposed method are designed to deal with large data sets which impose serious time and memory constraints. No such constraints exist in small data sets.

7. Conclusion and Future Work

By rejecting the premise that a nearest neighbour algorithm must find the nearest neighbour for every instance in the given data set, we propose a new approach to produce a nearest neighbour density estimator called LiNearN. It is the first nearest neighbour density estimator to have linear time complexity and constant space complexity, as far as we know. In contrast, existing nearest neighbour density estimators typically have $O(n^2)$ time complexity and $O(n)$ space complexity; and even with the aid of an indexing scheme, the time complexity can at best be reduced to near linear time only. LiNearN achieves linear time complexity without any indexing scheme.

Our asymptotic analysis reveals that LiNearN has a parameter which trades off between bias and variance, as in existing k-nearest neighbour density estimators.

We assess LiNearN in anomaly detection and clustering tasks and compare with three state-of-the-art nearest neighbour algorithms, ORCA, LOF and DBSCAN. LiNearN produces similar results compared with these algorithms in terms of task-specific performance measures, but it runs orders of magnitude faster than these algorithms in large data sets.

The advantages of the new nearest neighbour approach shown in these two tasks imply that it can potentially be adopted, in place of existing nearest neighbour algorithms, to solve other pattern recognition tasks. In each of these tasks, we shall first examine whether the aim can be achieved without $O(n^2)$ distance calculations. LiNearN may be able to reduce the time and space complexities of kernel density estimators. This is an interesting open question that deserves future investigation.

References

[1] Achtert, E., Hettab, A., Kriegel, H.-P., Schubert, E., Zimek, A., 2011. Spatial outlier detection: data, algorithms, visualizations. In: Proceedings of the 12th International Conference on Advances in Spatial and Temporal Databases(SSTD 11). Vol. 6849 of LNCS. Springer-Verlag, pp. 512–516.

[2] Agrawal, M., Gupta, N., Shreelekshmi, R., Narasimha Murty, M., 2005. Efficient pattern synthesis for nearest neighbour classifier. Pattern Recognition 38 (11), 2200–2203.

[3] Angiulli, F., Fassetti, F., 2009. DOLPHIN: An efficient algorithm for mining distance-based outliers in very large datasets. Transactions on Knowledge Discovery from Data 3 (1), 4:1–4:57.

[4] Angiulli, F., Pizzuti, C., 2002. Fast Outlier Detection in High Dimensional Spaces. In: Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 02). Vol. 2431 of LNCS. Springer-Verlag, pp. 15–26.

[5] Athitsos, V., Alon, J., Sclaroff, S., Kollios, G., 1 2008. BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 30 (1), 89–104.

[6] Aurenhammer, F., 1991. Voronoi diagrams — a survey of a fundamental geometric data structure. ACM Computing Surveys 23 (3), 345–405.

[7] Bay, S. D., Schwabacher, M., 2003. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (SIGKDD-03). ACM, pp. 29–38.

[8] Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B., May 1990. The R*-Tree: an efficient and robust access method for points and rectangles. SIGMOD Record (ACM Special Interest Group on Management of Data) 19 (2), 322–331.

[9] Beygelzimer, A., Kakade, S., Langford, J., 2006. Cover Trees for Nearest Neighbor. In: Cohen, W., Moore, A. (Eds.), Proceedings of the 23rd International Conference on Machine Learning (ICML 06). ACM, pp. 97–104.

[10] Breiman, L., 1996. Bagging predictors. Machine Learning 24 (2), 123–140.

[11] Breunig, M., Kriegel, H.-P., Ng, R., Sander, J., 2000. LOF: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of data (SIGMOD 00). ACM, pp. 93–104.

[12] Ciaccia, P., Patella, M., Zezula, P., 9 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 97). Morgan Kaufmann, pp. 426–435.

[13] Dasarathy, B. V., 1991. Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press.

[14] Dasarathy, B. V., Sánchez, J. S., Townsend, S., 2000. Nearest Neighbour Editing and Condensing Tools - Synergy Exploitation. Pattern Analysis & Applications 3, 19–30.

[15] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press, pp. 226–231.

[16] Evans, D., 2008. A law of large numbers for nearest neighbour statistics. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 464 (2100), 3175–3192.

[17] Evans, D., Jones, A., Schmidt, W., 2002. Asymptotic moments of near-neighbour distance distributions. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences 458 (2028), 2839–2849.

[18] Frank, A., Asuncion, A., 2010. UCI Machine Learning Repository.
URL `http://archive.ics.uci.edu/ml`

[19] Freund, Y., Schapire, R., 1996. Experiments with a New Boosting Algorithm. In: Proceedings of the 13th International Conference on Machine Learning (ICML 96). Morgan Kaufmann, pp. 148–156.

[20] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I., 2009. The WEKA Data Mining Software: An Update. SIGKDD Explorations 11 (1), 10–18.

[21] Handl, J., Knowles, J., 2004. Evolutionary multiobjective clustering. In: Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII). Vol. 3242 of LNCS. Springer, pp. 1081–1091.

[22] Knorr, E., Ng, R., 1999. Finding Intensional Knowledge of Distance-Based Outliers. In: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB 1999). Morgan Kaufmann, pp. 211–222.

[23] Lazarevic, A., Kumar, V., 2005. Feature bagging for outlier detection. In: Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (SIGKDD 05). ACM, pp. 157–166.

[24] Liu, F. T., Ting, K. M., Zhou, Z.-H., 2012. Isolation-Based Anomaly Detection. Transactions on Knowledge Discovery from Data 6 (1), 3:1–3:39.

[25] Loftsgaarden, D., Quesenberry, C., 1965. A Nonparametric Estimate of a Multivariate Density Function. The Annals of Mathematical Statistics 36 (3), 1049–1051.

[26] Ramaswamy, S., Rastogi, R., Shim, K., 2000. Efficient algorithms for mining outliers from large data sets. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of data (SIGMOD 00). ACM, pp. 427–438.

[27] Ramaswamy, S., Rose, K., 2011. Adaptive Cluster Distance Bounding for High-Dimensional Indexing. IEEE Transactions on Knowledge and Data Engineering 23 (6), 815–830.

[28] Salzberg, S., 1991. A Nearest Hyperrectangle Learning Method. Machine Learning 6 (3), 251–276.

[29] Sánchez, J. S., Barandela, R., Marqués, A., Alejo, R., Badenas, J., 2003. Analysis of new techniques to obtain quality training sets. Pattern Recognition Letters 24 (7), 1015–1022.

[30] Silverman, B., 1986. Density Estimation for Statistics and Data Analysis. Chapmal & Hall/CRC.

[31] Susheela Devi, V., Narasimha Murty, M., 2002. An incremental prototype set building technique. Pattern Recognition 35 (2), 505–513.

[32] Tan, P. N., Steinbach, M., Kumar, V., 2006. Introduction to Data Mining. Addison-Wesley.

[33] Ting, K. M., Washio, T., Wells, J. R., Liu, F. T., 2011. Density Estimation Based on Mass. In: Proceedings of the 11th IEEE International Conference on Data Mining (ICDM 11). IEEE Computer Society Press, pp. 715–724.

[34] Ting, K. M., Wells, J. R., Tan, J. S. C., Teng, S. W., Webb, G., 2011. Feature-subspace aggregating: ensembles for stable and unstable learners. Machine Learning 82 (3), 375–397.

[35] Ting, K. M., Zhu, L., Wells, J. R., 2013. Local Models–The Key to Boosting Stable Learners Successfully. Computational Intelligence 29 (2), 331–356.

[36] Triguero, I., García, S., Herrera, F., 2011. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. Pattern Recognition 44 (4), 901–916.

[37] Wu, Y., Ianakiev, K., Govindaraju, V., 2002. Improved k-nearest neighbor classification. Pattern Recognition 35 (10), 2311–2318.

Appendix A - Proof for Theorem 1

The ball of radius r centered at $c \in \mathcal{D}$ is denoted by $B_c(r)$. Let $\omega_c(r)$ denote the probability measure induced by $\phi(x)$ on the neighbourhoods of C,

$$\omega_c(r) = \int_{B_c(r)\cap C} \phi(x)dx.$$

$\omega_c(r) > 0$ on C since $B_c(r) \cap C$ is not empty and $\phi(x) > 0$ for all $x \in C$. Because of this fact and the convexity of C, $\omega_c(r)$ is strictly monotonic increasing for $0 \le r \le r_0$ for some $r_0 > 0$ and $\omega_c(r) = 1$ for $r_0 \le r$. Thus, the inverse function $r = h(\omega_c)$ exists, and the following holds according to the continuity of $\phi(x)$ on C and Eq. (5.15) with $k = 1$ in Evans et al. [17].

$$E[r_c^m] = (\psi - 1)\int_0^1 h(\omega_c)^m (1-\omega_c)^{\psi-2} d\omega_c. \quad (T1.1)$$

Within the interval of this integral, the integral over $[\omega_c(\delta), 1]$ where $\delta = 1/\psi^\epsilon$ is negligible for all $0 < \epsilon < 1/d$ by Lemma 5.3 with $k = 1$ in Evans et al. [17]. With this fact, the convexity of C, the continuity and the bounded partial derivatives of $\phi(x)$ on C, we obtain the following according to Eq. (5.22) in Evans et al. [17],

$$\omega_c(r) = (\phi(c) + O(\delta))|B_c(r) \cap C| \text{ as } \psi \to \infty, \quad (T1.2)$$

where $|B_c(r) \cap C|$ is the volume of $B_c(r) \cap C$.

In case that the shortest distance between c and the surface boundary of C is more than or equal to δ, $|B_c(r) \cap C| = |B_c(r)| = \alpha(d,p)r_c^d$. Thus, $\omega_c(r) = (\phi(c) + O(\delta))\alpha(d,p)r_c^d$ as $\psi \to \infty$ holds by Eq. (T1.2). In case that c is closer than δ from the surface boundary of C, there exists a constant $0 < \nu' < 1$ such that $\nu'\alpha(d,p)r_c^d < |B_c(r) \cap C|$ by Eq. (2.1), Proposition (2.1) and Eq. (2.3) in Evans et al. [17]. In concert with $|B_c(r) \cap C| < \alpha(d,p)r_c^d$, there exists $0 < \nu' < \nu(d,p,c) < 1$ and $|B_c(r) \cap C| = \nu(d,p,c)\alpha(d,p)r_c^d$ holds. By this fact and Eq. (T1.2), $\omega_c(r) = (\phi(c)+O(\delta))\nu(d,p,c)\alpha(d,p)r_c^d$ as $\psi \to \infty$ holds for some $0 < \nu(d,p,c) < 1$. Combining these two cases,

$$\omega_c(r) = (\phi(c) + O(\delta))\nu(d,p,c)\alpha(d,p)r_c^d \text{ as } \psi \to \infty \quad (T1.3)$$

for some $0 < \nu(d,p,c) \le 1$.

By following the manipulation from Eq. (5.24) to Eq. (5.33) in Evans et al. [17] with $k = 1$, our Eq. (T1.1) and (T1.3), we obtain

$$E[r_c^m] = \frac{\Gamma(m/d+1)}{\{\nu(d,p,c)\alpha(d,p)\phi(c)\}^{m/d}} \frac{1}{(\psi+1)^{m/d}}(1 + O(\delta)) \text{ as } \psi \to \infty.$$

Because $O(1/(\psi+1)^{m/d})O(\delta) = O(1/\psi^{m/d+\epsilon})$, Eq. (7) follows. □

Appendix B - Proof for Theorem 2

Let $\mathbb{C}_c$ be a hypercube such that $\mathbb{C}_c = \Pi_{j=1}^d [c_j - r_c, c_j + r_c]$ where c_j is the j-th element of $c \in C$. Note that $\mathbb{H}_c \subseteq \mathbb{C}_c$ for any $p > 0$ of L^p distance measure. Also, we use $Pr(\mathbb{H}_c)$ to denote $Pr(x \in \mathbb{H}_c | x \in \mathfrak{D})$ for brevity. In addition, according to the bounded first and second order partial derivatives of $\phi(x)$ at each point $x \in C$, we denote their bounds as

$$\left| \frac{\partial \phi(c)}{\partial c_j} |_{c \in \mathbb{H}_c} \right| \leq B_1(\phi, C) \text{ and } \left| \frac{\partial^2 \phi(c)}{\partial c_j^2} |_{c \in \mathbb{H}_c} \right| \leq B_2(\phi, C). \quad (T2.1)$$

Furthermore, the second order Taylor approximation of $\phi(x)$ around the center c of $\mathbb{H}_c$ is given as

$$\begin{aligned} \phi(x)|_{c \in \mathbb{H}_c} \quad &\approx \quad \phi(c)|_{c \in \mathbb{H}_c} + (x - c)^T \nabla \phi(c)|_{c \in \mathbb{H}_c} \quad (T2.2) \\ &\quad\quad + \frac{1}{2} \{(x - c)^T \nabla\}^2 \phi(c)|_{c \in \mathbb{H}_c}, \end{aligned}$$

where $\nabla = [\partial/\partial x_1, \ldots, \partial/\partial x_d]^T$.

From $\mathbb{H}_c \subseteq \mathbb{C}_c$, (T2.1), (T2.2) and the fact that the integral of an odd function around c over its symmetric region is zero, we obtain the following.

$$\begin{aligned} \frac{Pr(\mathbb{H}_c)}{|\mathbb{H}_c|} &= \frac{1}{|\mathbb{H}_c|} \int_{\mathbb{H}_c} \phi(y) dy \\ &\approx \phi(c)|_{c \in \mathbb{H}_c} + \frac{1}{|\mathbb{H}_c|} \int_{\mathbb{H}_c} \frac{1}{2} \{(y - c)^T \nabla\}^2 \phi(c)|_{c \in \mathbb{H}_c} dy \\ &= \phi(c)|_{c \in \mathbb{H}_c} + \frac{1}{\alpha(d,p) r_c^d} \int_{\mathbb{H}_c} \frac{1}{2} \sum_{j=1}^d (y_j - c_j)^2 \frac{\partial^2 \phi(c)}{\partial c_j^2} |_{c \in \mathbb{H}_c} dy \\ &\leq \phi(c)|_{c \in \mathbb{H}_c} + \frac{1}{\alpha(d,p) r_c^d} \int_{\mathbb{C}_c} \frac{1}{2} \sum_{j=1}^d (y_j - c_j)^2 \left| \frac{\partial^2 \phi(c)}{\partial c_j^2} |_{c \in \mathbb{H}_c} \right| dy \\ &\leq \phi(c)|_{c \in \mathbb{H}_c} + \frac{d 2^{d-1} B_2(\phi, C)}{3 \alpha(d,p)} r_c^2 \quad (T2.3) \end{aligned}$$

Because $\overline{\zeta}(x)$ is computed from two subsamples $\mathcal{D}$ and $\mathfrak{D}$ which are mutually independent, the expectation $E[\cdot]$ consists of two independent expectations over $\mathcal{D}$ and over $\mathfrak{D}$ denoted as $E\mathcal{D}[\cdot]$ and $E\mathfrak{D}[\cdot]$, respectively. $|\mathfrak{D}(\mathbb{H}_c)|$ under a given $\mathbb{H}_c$ follows a binomial distribution $B(\Psi, Pr(\mathbb{H}_c))$ over $\mathfrak{D}$ where its expected value is

$$E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}_c)|] = \Psi Pr(\mathbb{H}_c), \quad (T2.4)$$

and its variance

$$E\mathfrak{D}[\{|\mathfrak{D}(\mathbb{H}_c)| - E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}_c)|]\}^2] = \Psi Pr(\mathbb{H}_c)(1 - Pr(\mathbb{H}_c)). \quad (T2.5)$$

We denote $H^i = \{\mathbb{H}_c^i | c \in \mathcal{D}^i\}$ for the i-th subsampling in Eq. (5) and $r_{c,i}$ for the radius of $\mathbb{H}_c^i$. Then, from Eq. (3), (5), (T2.1), (T2.2), (T2.3) and (T2.4), the square bias of $\overline{\zeta}(x)$ is evaluated as follows. Note that $|x_j - c_j| \leq r_{cnn(x),i}$ holds for each dimension j since x is always in $\mathbb{H}_{cnn(x)}^i$ for each i. So, we apply inequality in the last line.

$$
\begin{aligned}
\left\{E[\overline{\zeta}(x)]-\phi(x)\right\}^2 &= \left\{\frac{1}{t}\sum_{i=1}^{t} E\mathcal{D}\left[\frac{E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})|]}{\Psi|\mathbb{H}^i_{cnn(x)}|}\right]-\phi(x)\right\}^2 \\
&\approx \left\{\frac{1}{t}\sum_{i=1}^{t}\left(E\mathcal{D}\left[\frac{Pr(\mathbb{H}^i_{cnn(x)})}{|\mathbb{H}^i_{cnn(x)}|}\right] - \phi(c)|_{c\in\mathbb{H}^i_{cnn(x)}} - (x-c)^T\nabla\phi(c)|_{c\in\mathbb{H}^i_{cnn(x)}}\right.\right. \\
&\quad \left.\left. -\frac{1}{2}\{(x-c)^T\nabla\}^2\phi(c)|_{c\in\mathbb{H}^i_{cnn(x)}}\right)\right\}^2 \\
&\leq \left\{\left(\frac{d2^{d-1}B_2(\phi,c)}{3\alpha(d,p)}E\mathcal{D}[r^2_{cnn(x),i}] + dB_1(\phi,C)E\mathcal{D}[r_{cnn(x),i}]\right.\right. \\
&\quad \left.\left. +\frac{d^2B_2(\phi,C)}{2}E\mathcal{D}[r^2_{cnn(x),i}]\right)\right\}^2 . \quad (T2.6)
\end{aligned}
$$

From Eq. (7) with $m=1$ or 2, the term $E\mathcal{D}[r_{cnn(x),i}]$ dominates in terms of ψ in Eq. (T2.6). Though this analysis uses the second order approximation of $\phi(x)$, the result using the higher order approximation is the same, since the first order term dominates the magnitude. This gives Eq. (8).

Next, from Eq. (3), (5), (T2.1), (T2.2), (T2.3) and (T2.5), the variance of $\overline{\zeta}(x)$ is evaluated as follows. Note that Eq. (7) indicates that the magnitudes of $1/|\mathbb{H}^i_{cnn(x)}|$ and $E\mathcal{D}[1/|\mathbb{H}^i_{cnn(x)}|]$ have the same order for ψ. So, we introduce an approximation to remove $E\mathcal{D}[\cdot]$ in the inner summation on the third line.

$$
\begin{aligned}
&E\left[\{\overline{\zeta}(x)-E[\overline{\zeta}(x)]\}^2\right] \\
&= E\mathcal{D}\left[E\mathfrak{D}\left[\left\{\frac{1}{t}\sum_{i=1}^{t}\frac{|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})|}{\Psi|\mathbb{H}^i_{cnn(x)}|} - \frac{1}{t}\sum_{i=1}^{t}E\mathcal{D}\left[\frac{E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})|]}{\Psi|\mathbb{H}^i_{cnn(x)}|}\right]\right\}^2\right]\right] \\
&\approx E\mathcal{D}\left[E\mathfrak{D}\left[\left\{\frac{1}{t}\sum_{i=1}^{t}\frac{|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})| - E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})|]}{\Psi|\mathbb{H}^i_{cnn(x)}|}\right\}^2\right]\right] \\
&= \frac{1}{t^2\Psi^2}\sum_{i=1}^{t}E\mathcal{D}\left[\frac{E\mathfrak{D}\left[\left\{|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})| - E\mathfrak{D}[|\mathfrak{D}(\mathbb{H}^i_{cnn(x)})|]\right\}^2\right]}{|\mathbb{H}^i_{cnn(x)}|^2}\right] \\
&= \frac{1}{t^2\Psi}\sum_{i=1}^{t}E\mathcal{D}\left[\frac{Pr(\mathbb{H}^i_{cnn(x)})}{|\mathbb{H}^i_{cnn(x)}|}\frac{1-Pr(\mathbb{H}^i_{cnn(x)})}{|\mathbb{H}^i_{cnn(x)}|}\right] \\
&\leq \frac{1}{t^2\Psi}\sum_{i=1}^{t}E\mathcal{D}\left[\frac{d2^{d-1}B_2(\phi,c)}{3\alpha(d,p)}r^2_{cnn(x),i}\left\{\frac{1}{\alpha(d,p)r^d_{cnn(x),i}} + \frac{d2^{d-1}B_2(\phi,c)}{3\alpha(d,p)}r^2_{cnn(x),i}\right\}\right] \\
&= \frac{1}{t\Psi}\left(\frac{d2^{d-1}B_2(\phi,c)}{3\alpha(d,p)^2}E\mathcal{D}[r^{-d+2}_{cnn(x),i}] + \frac{d^2 2^{2d-2}B_2(\phi,c)^2}{9\alpha(d,p)^2}E\mathcal{D}[r^4_{cnn(x),i}]\right). \quad (T2.7)
\end{aligned}
$$

From Eq. (7) with $m=4$ or $d-2$, the first term in the summation of Eq. (T2.7) always dominate the magnitude of this expression. If $d=1$, the variance is $O(\psi^{-1})$ from Eq. (7) with $m=1$. If $d\geq 2$, it is $O(\psi^{1-2/d+\epsilon})$ from Eq. (7) with $m=d-2$. Similarly to the analysis of the square bias, this uses the second order approximation of $\phi(x)$. However, the result using the higher order approximation is the same, since the second order term dominates the magnitude. Thus, Eq. (9) is derived. □

Appendix C - LiNearN-Cluster Algorithms

The procedures for LiNearN-Cluster are provided in the following three algorithms.

Algorithm 5: CoreRegions($\mathbb{C}$)

```
input  : $\mathbb{C}$ - $\{H^i | i = 1, \ldots, t\}$.
output: $\{H^i | i = 1, \ldots, s\}$ containing only core regions.
for $i$ = 1 to $t$ do
    foreach $\mathbb{H} \in H^i$ do
        $density \leftarrow \mathbb{H}.mass/\mathbb{H}.radius$;
        if $density < MinDensity$ then
            $H^i \leftarrow H^i \setminus \{\mathbb{H}\}$;
        end
    end
end
return $\{H^i | i = 1, \ldots, s\}$ containing only core regions;
```

Algorithm 6: FindPointsWithinCoreRegions($D, \mathbb{C}$)

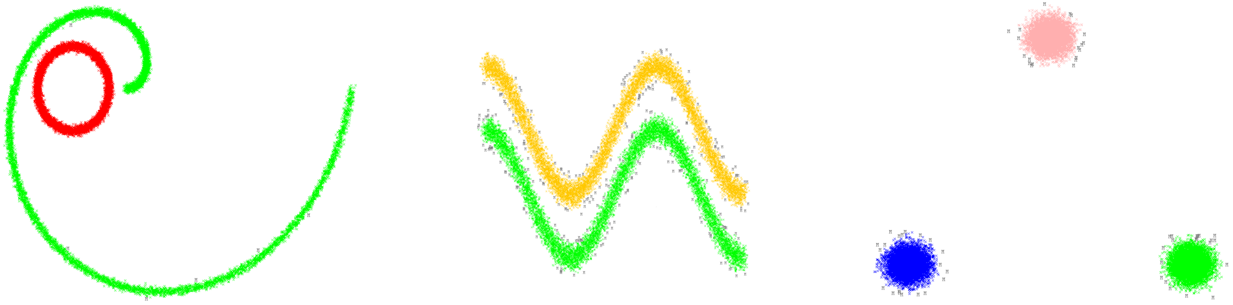
```
input  : $D$ - input data, $\mathbb{C}$ : $\{H^i | i = 1, \ldots, s\}$ containing only core regions.
output: $D$ containing only points found within core regions.
foreach $x \in D$ do
    $found \leftarrow false$;
    for $i$ = 1 to $s$ do
        $\mathbb{H} \leftarrow search(H^i, x)$ {return a $\mathbb{H}$ that covers $x$};
        if ($\mathbb{H}$ is not NULL) then
            $found \leftarrow true$;
            break $i$ loop {found a core region that $x$ falls into.};
        end
    end
    if not found then
        $D \leftarrow D \setminus \{x\}$;
    end
end
return $D$ containing only points found within core regions;
```

Algorithm 7: LiNearN-Clusters($D, \mathbb{C}$)

```
input  : $D$ - containing only points found within core regions, $\mathbb{C} : \{H^i | i = 1, \ldots, s\}$ - containing only
         core regions.
output: $\{H^i | i = 1, \ldots, s\}$ with id assigned to each core region.
1  $idNext \leftarrow 0$;
2  $\mathbb{P} \leftarrow \emptyset$;
3  foreach $x \in D$ do
4      $found \leftarrow false$;
5      {Search for the first assigned core region};
6      for $i$ = 1 to s do
7          $\mathbb{H} \leftarrow search(H^i, x)$ {return a $\mathbb{H}$ that covers $x$};
8          if ($\mathbb{H}$ is not NULL) and isAssigned($\mathbb{H}$) then
9              $idCurrent \leftarrow \mathbb{H}.id$;
10             $found \leftarrow true$;
11             break the $i$ loop {found a core region that has been assigned to an id};
12         end
13     end
14     if $notfound$ then
15         $idNext \leftarrow idNext + 1$;
16         $idCurrent \leftarrow idNext$;
17     end
18     {Set all unassigned regions that $x$ falls into to $idCurrent$};
19     for $i$ = 1 to s do
20         $\mathbb{H} \leftarrow search(H^i, x)$ {return a $\mathbb{H}$ that covers $x$};
21         if ($\mathbb{H}$ is not NULL) then
22             if isAssigned($\mathbb{H}$) then
23                 if $idCurrent \neq \mathbb{H}.id$ then
24                     {Current $\mathbb{H}$ is already assigned but has a different id; make a note and then merge
                       into one id in step 36};
25                     $P.idFirst \leftarrow \mathbb{H}.id$;
26                     $P.idSecound \leftarrow idCurrent$;
27                     $\mathbb{P} \leftarrow \mathbb{P} \bigcup \{P\}$;
28                 end
29             end
30             else
31                 $\mathbb{H}.id \leftarrow idCurrent$;
32             end
33         end
34     end
35 end
36 $merge(\mathbb{C}, \mathbb{P})$ {merge all ‘linked’ ids into one id};
37 return $\{H^i | i = 1, \ldots, s\}$ with an id assigned to each core region;
```

Appendix D - Data characteristic

The characteristic of the data set, Ring-Curve-Wave-TriGaussian, used in Section 5.2 is shown in Figure .11.



(a) Ring-Curve

(b) Wave

(c) Tri-Gaussian

Figure .11: Scatter plot of the clusters in the Ring-Curve-Wave-TriGaussian data set, as used in [33].

Mass-based Similarity Measure:
An Effective Alternative to Distance-based Similarity Measures

Submitted for Blind Review

***Abstract*—This paper introduces a unique similarity measure that does not compute distance. Instead, it is based on the cardinality of the smallest local region covering the two instances for which the similarity is measured. Theoretical analysis reveals that it is a generalisation of mass estimation. To show its utility, a new information retrieval system is created based on the new similarity measure. Empirical evaluations demonstrate that the new system has a significantly better retrieval performance than five state-of-the-art systems in image and music retrievals.**

***Keywords*-theory; data mining; unsupervised learning;**

I. Introduction and motivation

Data mining algorithms have traditionally relied on similarity measures to gauge the similarity between two instances, as the core operation to solve various data mining problems. For example, anomaly detection requires ranking of instances in a database according to their degrees of anomaly; an information retrieval task ranks instances in a database which are most similar to a query. These ranking tasks are traditionally accomplished by computing the similarity or distance between two instances as the key step to calculate the ranking. Because the ranking is with respect to the entire database, the similarity must be computed for all pairs of instances in the database for anomaly detection, and between the query and every instance in the database for information retrieval.

This paper is motivated by a recent content-based multimedia information retrieval (CBMIR) system called ReFeat [1]. ReFeat is unique in two aspects. First, it uses a similarity measure which is primarily based on data distribution in the local region. In contrast, commonly used distance measures are solely based on the positions of instances in the feature space. Second, at the heart of ReFeat is an anomaly detector which provides a ranking score $\ell(\mathbf{x})$ for an instance $\mathbf{x}$, independently of other instances. This is fundamentally different from most ranking measures that rely on a distance measure to compute the distance of an instance relative to another instance, $dist(\mathbf{x}, \mathbf{y})$ (e.g., ORCA [2], Qsim [3]).

The use of such a unique similar measure is the key reason why ReFeat has produced better retrieval performance than state-of-the-art CBMIR systems including manifold learning method MRBIR [4], Bayesian learning method BALAS [5], query-sensitive ranking methods InstRank [6] and Qsim [3].

Despite its unique approach and demonstrably excellent retrieval performance, the ReFeat paper [1] does not provide a satisfactory explanation as to why a unary score function could produce an appropriate ranking of database instances for a query which requires a binary function. More to the point: ReFeat does not guarantee that two 'similar' instances, having a similar ranking score $\ell(\cdot)$, are in the same local neigbourhood.

This paper investigates the source of the power of ReFeat. From a foundation in mass estimation [7], we derive a new mass-based similarity measure that enables a new CMBIR system to significantly improve the retrieval performance of ReFeat.

The contributions of this paper are:

1) Introducing a unique similarity measure, Massim, and establishing its theoretical foundation based on mass.
2) Creating a new CMBIR system called MassIR based on this similarity measure.
3) Empirically evaluating MassIR in comparison with ReFeat and systems which employ commonly used similarity measures, and showing its superiority in image and music information retrievals.

Massim has the following characteristics:

- Unlike the similarity measure used in ReFeat, Massim guarantees that two similar instances are in the same local neighbourhood.
- Unlike distance-based similarity measures, it does not compute distance and primarily based on data distribution in the local region.
- It is a generalisation of mass estimation. Under certain conditions, it reduces to mass estimation [7].

The rest of the paper is organised as follows. Sections II and III describe the related work and ReFeat, respectively. We introduce the intuition, the new similarity measures and the implementation in the next three sections. The new information retrieval system and the experimental results are described in Section VII. Discussion and the conclusions are provided in the last two sections.

II. Related Work

Similarity is a concept that is used extensively not only in data mining, but also in many other fields such as psychology [8] and ecology [9]. There are different types of similarity measures applied for different tasks. We refer the reader to [10] and [11] for a rich collection of similarity measures.

Table I: Unary ranking function for Similarity measure

	Ranking measure	Similarity measure
Function	$\ell(\mathbf{x})$	$Similarity(\mathbf{x},\mathbf{q})$
Model - Purpose:	Describe the data profile for each $\mathbf{x}$	Describe the similarity for any $\mathbf{x}$ and $\mathbf{q}$
- iTree:	$\mathbf{x}$ having long path length from an iTree is relevant to the data profile; and $\mathbf{x}$ having short path length is irrelevant	ReFeat employs t iTrees which map $\mathbb{R}^d$ to $\mathbb{R}^t$, where each iTree profiles one aspect of the database as a relevance feature in the new feature space. $similarity(\mathbf{x},\mathbf{q}) \equiv score(\mathbf{x}\vert\mathbf{q}) = \frac{1}{t}\sum_{\imath=1}^{t}(\frac{\ell_\imath(\mathbf{q})}{c}-1)\ell_\imath(\mathbf{x})$

Distance based similarity measures are often tested against four distance axioms to determine whether they are a metric. The suitability of similarity measures that adhere to all four axioms have been challenged by various researchers [12], [13]. Tversky [13] and Krumhans [14] have questioned the validity of using geometrical models as the measure of similarity. They introduced set theoretic and density-augmented geometrical alternatives, respectively.

Many nonmetric similarity measures [15] have been designed to explicitly violate the triangle inequality axiom. While this enhances similarity modeling (to model human perception), systems that use these nonmetric measures are inefficient because they cannot use many indexing schemes (that rely on the triangle inequality) to prune the search space. To improve efficiency, research has focused on ways to (i) enforce or approximate the triangle inequality while preserving the desired similarity orderings [9], [16], [17]; and (ii) condense the given data set into a smaller set of representative instances, and then employ a classification method to find the representative instance which is most similar to the query [18], [19].

Irrespective of whether metric or nonmetric, all existing similarity measures are binary functions.

iForest [20] employs a unary function to score each instance and was designed specifically for anomaly detection. ReFeat [1], mentioned in the introduction, adopted iForest to solve information retrieval problems. ReFeat has created a binary function to measure the similarity between a query $\mathbf{q}$ and a database instance $\mathbf{x}$. But, the function is not strictly a similarity measure because the relevance of $\mathbf{q}$ and $\mathbf{x}$ is measured against a reference model (i.e., iForest) independently and the model does not guarantee $\mathbf{q}$ and $\mathbf{x}$ to be in the same local neighbourhood even though both have similar scores.

In this paper, we establish a principled approach which ensures that two instances are similar if they are in the same local neighbourhood.

The new similarity measure, Massim, is fundamentally different from existing distance based similarity measures because it is based on mass [7] rather than distance. Mass estimation [7] was recently introduced as an alternative to density estimation to solve data mining problems. Mass models the centrality of a data cloud whereas density models the compactness. Mass estimation provides the theoretical foundation for iForest; and like the measure used in iForest, mass is a unary function. Here we use mass to define a binary function to measure similarity.

III. ReFeat: a unary function for similarity measurement

ReFeat [1] maps the original database with d features to a database with new t relevance features; and computes the similarity in the new space. The value of each relevance feature is derived from an iTree, i.e., $\ell_\imath(\mathbf{x})$ the path length of $\mathbf{x}$ traversing iTree $\imath$. The similarity score of a database instance $\mathbf{x}$ with respect to an query $\mathbf{q}$ is expressed as the weighted average of t relevance feature values, where the weight for feature $\imath$ is $w_\imath(\mathbf{q}) = \frac{\ell_\imath(\mathbf{q})}{c} - 1$, and c is a normalisation constant. The similarity function taking a query $\mathbf{q}$ and a database instance $\mathbf{x}$ produces a high (low) score if both $\mathbf{x}$ and $\mathbf{q}$ have long (short) path lengths on many relevance features. Table I provides a brief summary of how ReFeat employs a unary ranking measure to measure similarity between two instances.

ReFeat relies on iTrees to be unbalanced. Having distinct long and short path lengths in each iTree is a prerequisite to identify similar instances. Balanced iTrees always produce the same path length; thus they have no way to differentiate instances of different feature characteristics. Zhou et. al. [1] analyse empirically that iTrees are more likely to be unbalanced than balanced.

However, the analysis does not consider the fact that, even if both have long path lengths, there is no guarantee that $\mathbf{x}$ and $\mathbf{q}$ will fall into the same local neighbourhood in the feature space as $\ell(\mathbf{x})$ and $\ell(\mathbf{q})$ are assessed independently, albeit against the same reference model, i.e., iTree.

ReFeat works because it has employed a large number of iTrees (where $t = 1000$ in their experiments) such that similar instances are likely to have long path lengths in the same local neighbourhood of many iTrees; though some iTrees giving long path lengths may not have both instances in the same local neighbourhood.

In a nutshell, ReFeat derives its retrieval power from a ranking model that profiles the data distribution and it scores an instance's relevancy according to the profile. But its ranking score is based on a unary function which does not guarantee that similar instances are in the same local neighbourhood. We show in this paper that overcoming this weakness significantly improves the retrieval performance of ReFeat.

It is important to note that iTree is just one realisation of mass estimation [7], implemented using a tree structure

Table II: Mass-based similarity measure versus distance-based similarity measure

	Mass-based similarity measure	Distance-based similarity measure
Computation	$Mass(\mathbf{x},\mathbf{y})$ is primarily based on data distribution in the local region of the feature space.	$dist(\mathbf{x},\mathbf{y})$ is solely based on the positions of $\mathbf{x}$ and $\mathbf{y}$ in the feature space.
Definition	Mass base function $M(\mathbf{x},\mathbf{y})$ measures the cardinality of the *smallest* local region covering both $\mathbf{x}$ and $\mathbf{y}$.	$dist(\mathbf{x},\mathbf{y})$ measures the length of the *shortest* path from $\mathbf{x}$ to $\mathbf{y}$.
Inequality	$similarity(\mathbf{x},\mathbf{y}) > similarity(\mathbf{x},\mathbf{z}) \equiv$ $Mass(\mathbf{x},\mathbf{y}) < Mass(\mathbf{x},\mathbf{z})$	$similarity(\mathbf{x},\mathbf{y}) > similarity(\mathbf{x},\mathbf{z}) \equiv$ $dist(\mathbf{x},\mathbf{y}) < dist(\mathbf{x},\mathbf{z})$
Metric	The measure does not satisfy some distance axiom.	All distance axioms usually hold.

such that path length is a proxy to mass[1]. In general, a data distribution can be expressed as a mass distribution, instead of density distribution, and the mass distribution can be used as a basic data modelling tool to solve data mining problems (see [7], [21]). The data profile produced by iTrees has a one-to-one correspondence to mass: high (low) path length implies high (low) mass in the data distribution.

The new information retrieval system we present, `MassIR`, differs from `ReFeat` in that

- `MassIR` measures similarity using a binary function whereas `ReFeat` is based on a unary function.
- `MassIR` guarantees similar instances to be in the same local neighbourhood.
- The implementation of `Massim` uses balanced trees whereas `ReFeat` relies on unbalanced trees to work.
- `MassIR` and `ReFeat` behave differently (see section VII-B).

IV. Intuition

We will use the simplest form of mass, which is the cardinality of a region [7], to provide the intuition in this section.

Let r_i be a convex local region; and $|r_i|$ be the cardinality of region r_i; or in other words, the number of instances from the database that the region contains.

Mass inequality holds true as follows: $|r_j| < |r_i|$ for $r_j \subset r_i$ if regions $r_i,\ i \in \{1,\dots,m\}$ created by a model satisfy the conditions: $\forall i,j; i \neq j \quad r_i \neq \emptyset$ and $r_i \setminus r_j \neq \emptyset$.

With the assurance of the mass inequality, similarity of instances $\mathbf{x},\mathbf{y},\mathbf{z}$ can be inferred as shown in the following example: if $r_1 = \{\mathbf{x},\mathbf{y},\mathbf{z}\}$ and $r_2 = \{\mathbf{x},\mathbf{y}\}$ then $|R(\mathbf{x},\mathbf{y})| < |R(\mathbf{x},\mathbf{z})|$, where $R(\mathbf{a},\mathbf{b})$ is the smallest region covering both $\mathbf{a}$ and $\mathbf{b}$; r_1 and r_2 are the regions created by a model. This means that if $\mathbf{x}$ is in a local region of few instances with $\mathbf{y}$; and $\mathbf{x}$ is in a local region of many instances with $\mathbf{z}$, then $\mathbf{x}$ is more similar to $\mathbf{y}$ than $\mathbf{z}$.

We propose to use the cardinality of the smallest convex region covering both $\mathbf{a}$ and $\mathbf{b}$ as the base function to measure similarity between any two instances. Table II highlights the differences between this mass-based similarity measure and existing distance-based similarity measures.

Based on the simplest form of mass, more sophisticated forms of mass are derived in the next section.

V. Mass-based similarity measures

The theoretical basis of mass-based similarity, `Massim`, is given here. We introduce a new one-dimensional similarity measure in section V-A, provide its axioms in section V-B, and describe the multi-dimensional version in section V-C.

A. One-dimensional similarity measure

Definition 1: $R(x,y|E)$ is the smallest local region covering x and y in a given model E, which partitions the feature space into convex local regions, where $x,y \in \mathbb{R}$:

$R(x,y|E) = r^*$ such that $|r^*| = \min_{r \in E} |r|$ and $x,y \in r$.

E must be chosen to ensure that $R(x,y|E)$ is a unique region that covers both x and y. Let $E(s_i) = \{r_1,r_2,r_3\}$ be a model as a result of binary split s_i in the real line defined by $D = \{x_1,x_2,...,x_n\}$; let s_i be a binary split between x_i and x_{i+1} which divides the real line into two non-overlapping local regions, r_1 (where $x \leq s_i$) and r_2 (where $x \geq s_i$); and r_3 covers the entire real line.

Mass base function $M_i(x,y)$ denotes the mass of local region $R(x,y|E(s_i))$.

Definition 2: Mass-based similarity measure $Mass(x,y)$ is defined using D as a weighted power mean of a series of $M_i(x,y)$ weighted by $p(s_i)$ over $n-1$ splits as follows:

$$Mass(x,y) = (\sum_{i=1}^{n-1} M_i(x,y)^e p(s_i))^{\frac{1}{e}} \qquad (1)$$

where $p(s_i)$ is the probability of selecting the binary split s_i and $\sum_i p(s_i) = 1$; and

$$M_i(x,y) = \begin{cases} i & \text{if } R(x,y|E) = r_1 \\ n-i & \text{if } R(x,y|E) = r_2 \\ n & \text{if } R(x,y|E) = r_3 \end{cases}$$

For an example of five points, $x_1 < x_2 < ... < x_5$ and $e = 1$, $Mass(x_1,\cdot)$ for x_1 and each of the five points are given as follows:

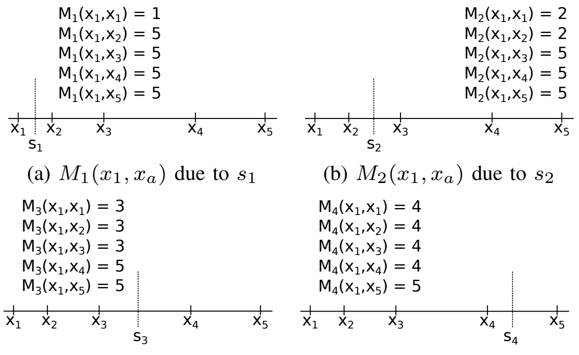
$$\begin{aligned} Mass(x_1,x_1) &= 1p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ Mass(x_1,x_2) &= 5p(s_1) + 2p(s_2) + 3p(s_3) + 4p(s_4) \\ Mass(x_1,x_3) &= 5p(s_1) + 5p(s_2) + 3p(s_3) + 4p(s_4) \\ Mass(x_1,x_4) &= 5p(s_1) + 5p(s_2) + 5p(s_3) + 4p(s_4) \\ Mass(x_1,x_5) &= 5p(s_1) + 5p(s_2) + 5p(s_3) + 5p(s_4) \end{aligned}$$

The components of the summation, $M_i(x_1,\cdot)$ due to each split s_i, are illustrated in Figure 1.

[1] The path length is computed as $L + f(m)$, where L is the path length traversed from the root to an external node, and $f(m)$ is a function which estimates the average path length of an unexpanded subtree for a training subset of size m.

Table III: Axioms used for Mass-based Similarity and distance-based similarity

	Mass-based Similarity	Distance-based Similarity	
Axiom 1	$Mass(x,y) \geq 1$	$dist(x,y) \geq 0$	(non-negativity)
Axiom 2	i.$\forall x,y \;\; Mass(x,x) \leq Mass(x,y)$ ii. $\exists x \neq y \;\; Mass(x,x) \neq Mass(y,y)$	$dist(x,y) = 0 \iff x = y$	(identity of indiscernibles)
Axiom 3	$Mass(x,y) = Mass(y,x)$	$dist(x,y) = dist(y,x)$	(symmetry)
Axiom 4	$Mass(x,z) < Mass(x,y) + Mass(y,z)$	$dist(x,z) \leq dist(x,y) + dist(y,z)$	(triangle inequality)



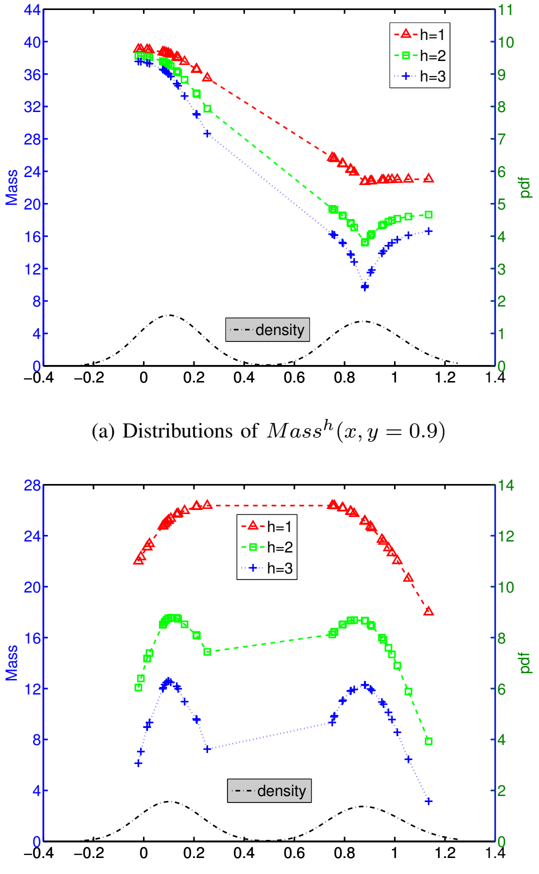
(a) $M_1(x_1, x_a)$ due to s_1

(b) $M_2(x_1, x_a)$ due to s_2

(c) $M_3(x_1, x_a)$ due to s_3

(d) $M_4(x_1, x_a)$ due to s_4

Figure 1: Example of mass base function $M_i(x_1, .)$ due to each of four binary splits s_1, s_2, s_3, s_4.



(a) Distributions of $Mass^h(x, y = 0.9)$

(b) Distributions of $Mass^h(x, x)$

Figure 2: Distributions of $Mass^h(x,y)$ and $Mass^h(x,x)$ in a data set having two Gaussian density distributions, where $\mu = \{0.1, 0.9\}$, $\sigma = 0.1$. Distributions for $h = 1, 2, 3$ are shown, where $e = -1$ is used.

Definition 3: Level-h $Mass^h(x,y)$ defined using D, where $1 < h < n$, is expressed as:

$$Mass^h(x,y) = (\sum_{i=1}^{n-1} Mass_i^{h-1}(x,y)^e p(s_i))^{\frac{1}{e}} \qquad (2)$$

where $Mass_i^{h-1}(x,y)$ is $Mass^{h-1}(x,y)$ defined using $D_i \subseteq D$, i.e., the data subset covered by $R(x,y|E(s_i))$.

Multi-modal data distribution requires level-h $Mass^h(x,y)$ that takes into account the nature of the data distribution more effectively.

Note that the distribution of $Mass^h(x,y)$ (for $e = 1$) reduces to mass distribution $mass^h(x)$ as defined by [7] when $y = x$:

$$mass^h(x) = \begin{cases} \sum_{i=1}^{n-1} m_i(x) p(s_i), & h = 1 \\ \sum_{i=1}^{n-1} mass_i^{h-1}(x) p(s_i), & h > 1 \end{cases}$$

where $m_i(x) = \begin{cases} i & \text{if } x \leq x_i \\ n - i & \text{if } x > x_i \end{cases}$

and $mass_i^{h-1}(x)$ is $mass^{h-1}(x)$ defined using $D_i \subset D$, i.e., the data subset separated by s_i that includes x.

B. Axioms for Massim

Function $Mass : \mathbb{R}^1 \times \mathbb{R}^1 \rightarrow \mathbb{R}^+$ satisfies the four axioms as shown in the second column of Table III.

A comparison with the axioms for distance-based similarity is also provided in Table III. The key difference is in axiom 2: (i) the minimum of $Mass(x,y)$ is $Mass(x,x)$; and (ii) $Mass(x,x)$ is not the same for all x. Figures 2(a) and 2(b) show that the distributions of $Mass(x,y)$ (for a given y) and $Mass(x,x)$, respectively.

The proofs for the axioms are omitted because of space limitation.

C. Multi-dimensional similarity measure

The one-dimensional `Massim` can be generalised to multi-dimensional by using a model **E** that partitions the feature space into local regions, instead of the binary splits.

We use the same approach, as proposed by [7], to eliminate the need to compute the probability of a binary split, $p(s_i)$; and this produces a randomised approximation.

The idea is to generate multiple random local regions that satisfy the mass inequality, and the mass similarity of any two instances is estimated by averaging the mass of all smallest local regions which cover both instances.

We show that random regions can be generated using axis-parallel splits called half-space splits. Each half-space split is performed on a randomly selected attribute in a multi-dimensional feature space. For an h-level split, a tree structure is formed in which each path has h half-space splits, and the tree has a total of 2^h non-overlapping local regions.

Let $\mathbf{E}(h)$ be a multi-dimensional model producing 2^h local regions as a result of an h-level split. Let $R(\mathbf{x},\mathbf{y}|\mathbf{E}(h))$ be the smallest region of $\mathbf{E}(h)$ covering both $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x},\mathbf{y} \in \mathbb{R}^d$; and $\mathfrak{M}_h(\mathbf{x},\mathbf{y})$ be the mass in $R(\mathbf{x},\mathbf{y}|\mathbf{E}(h))$.

In addition, multiple models are required which give rise to $\mathfrak{M}_{h,\imath}(\mathbf{x},\mathbf{y})$, the mass in $R(\mathbf{x},\mathbf{y}|\mathbf{E}_\imath(h))$.

In one-dimensional problems, Equations (1) and (2) can now be approximated as follows:

$$(\sum_{i=1}^{n-1} M_i(x,y)^e p(s_i))^{\frac{1}{e}} \approx (\frac{1}{t}\sum_{\imath=1}^{t} \mathfrak{M}_{1,\imath}(x,y)^e)^{\frac{1}{e}} \tag{3}$$

$$Mass^h(x,y) \approx (\frac{1}{t}\sum_{\imath=1}^{t} \mathfrak{M}_{h,\imath}(x,y)^e)^{\frac{1}{e}} \tag{4}$$

where $t > 1$ is the number of random regions to be used to define the mass similarity of x and y.

Since $\mathbf{E}(h)$ is defined in multi-dimensional space, the multi-dimensional mass similarity is the same as Equation (4) by simply replacing x and y with $\mathbf{x}$ and $\mathbf{y}$:

$$Mass^h(\mathbf{x},\mathbf{y}) \approx (\frac{1}{t}\sum_{\imath=1}^{t} \mathfrak{M}_{h,\imath}(\mathbf{x},\mathbf{y})^e)^{\frac{1}{e}} \tag{5}$$

An half-space tree implementing $\mathfrak{M}_{h,\imath}(\cdot,\cdot)$ that generates regions r, $\exists i,j\ r_j \subset r_i$, must satisfy the conditions specified in Section IV: $\forall i,j; i \neq j \quad r_i \neq \emptyset$ and $r_i \setminus r_j \neq \emptyset$ to ensure the mass inequality.

This is realised in a similarity tree (sTree), where a data subset $\mathcal{D} \subset D$ is partitioned into two equal-size subsets recursively until it cannot be subdivided. Each sTree is balanced having height $h = log_2(\psi)$ and ψ regions, where $\psi = |\mathcal{D}|$. D is then populated to the ψ regions to estimate the mass in each region. The detail of the implementation is provided in Section VI.

To simplify notation, we will drop h hereafter to denote $Mass^h(\cdot,\cdot)$ as $Mass(\cdot,\cdot)$, when the context is clear.

VI. Implementation

Mass-based similarity estimation is implemented in two stages. In the modelling stage, a Similarity Forest (sForest) is generated from D. The resultant sForest is the similarity model of the given dataset. Mass-based similarity between two instances is calculated in the estimation stage.

The sForest generation process is provided in Algorithm 1. A sForest consists of t Similarity Trees (sTrees).

Each sTree is built independently from $\mathcal{D}$, randomly selected without replacement from D, where $|\mathcal{D}| = \psi$. A sTree node can either be an external node or an internal node with exactly two child subtrees. An internal node consists of a randomly selected attribute a and a split value V, such that test $a \leq V$ divides the training set in this node into two equal-size subsets. The process is repeated in each subset recursively until $|\mathcal{D}| = 1$, as described in Algorithm 2. After creating a sTree, it is populated with D to determine

```
Algorithm 1: sForest($D$,$t$,$\psi$)
Input: $D$ - Database, $t$ - number of trees, $\psi$ - sub
    sampling size
Output: $sForest$
Initialize $sForest$
for $i = 1 \rightarrow t$ do
    $\mathcal{D} \leftarrow$ select $\psi$ instances from $D$ without re-
    placement.
    $T \leftarrow sTree(\mathcal{D})$
    $UpdateTreeMass(T, D)$
    $sForest \leftarrow sForest \cup T$
end for
return $sForest$
```

```
Algorithm 2: sTree($\mathcal{D}$)
Input: $\mathcal{D}$ - input data
Output: $sTree$
if $|\mathcal{D}|$ is 1 then
    return $exNode$
end if
Let $A$ be the complete list of attributes
$a \leftarrow$ Randomly selected attribute from $A$
$\mathcal{D}_{sorted} \leftarrow$ sort $\mathcal{D}$ on $a$
$V_l \leftarrow$ value of $a$ of $(\frac{|\mathcal{D}|}{2})^{th}$ item in $\mathcal{D}_{sorted}$
$V_r \leftarrow$ value of $a$ of $(\frac{|\mathcal{D}|}{2} + 1)^{th}$ item in $\mathcal{D}_{sorted}$
$V \leftarrow$ Randomly selected value between $V_l$ and $V_r$
$\mathcal{D}_l \leftarrow filter(\mathcal{D}, a \leq V)$
$\mathcal{D}_l \leftarrow filter(\mathcal{D}, a > V)$
return $innode\{ LeftChild \leftarrow sTree(\mathcal{D}_l)$,
    $RightChild \leftarrow sTree(\mathcal{D}_r)$, $SplitAttribute \leftarrow a$,
    $SplitValue \leftarrow V \}$
```

```
Algorithm 3: UpdateTreeMass($T$,$D$)
Input: $T$ - $sTree$ , $D$ - Data
Output: $T$(sTree), having all nodes updated with $\mathfrak{M}$
$T.mass \leftarrow |D|$
if $T$ is an internal node then
    $D_l \leftarrow filter(D, T.SplitAttribute$
            $\leq T.SplitValue)$
    $D_r \leftarrow filter(D, T.SplitAttribute$
            $> T.SplitValue)$
    $UpdateTreeMass(T.LeftChild, D_l)$
    $UpdateTreeMass(T.RightChild, D_r)$
end if
```

Table IV: Measures and feedback formulations used in ReFeat, MassIR and L^p-IR. Note that $\mathbf{x} =< x^{(1)}, x^{(2)}, \ldots, x^{(d)} >$.

	ReFeat	MassIR	L^p-IR
Score	$Score(\mathbf{x}\mid\mathbf{q}) = \frac{1}{t}\sum_{\imath=1}^{t}(w_\imath(\mathbf{q})\ \ell_\imath(\mathbf{x}))$	$Mass(\mathbf{x},\mathbf{q}) = (\frac{1}{t}\sum_{\imath=1}^{t}\mathfrak{M}_\imath(\mathbf{x},\mathbf{q})^e)^{\frac{1}{e}}$	$dist(\mathbf{x},\mathbf{q}) = (\sum_{\jmath=1}^{d}(x^{(\jmath)} - q^{(\jmath)})^p)^{\frac{1}{p}}$
Model	$w_\imath(\cdot)$ is a function of $\ell_\imath(\cdot)$; and $\ell_\imath(\cdot)$ is derived from iForest	$\mathfrak{M}_\imath(\cdot,\cdot)$ is derived from sForest	No trained models are required
Feedback $\mathcal{Q} = \mathcal{P} \cup \mathcal{N}$	$w_\imath(\mathcal{Q}) = \frac{1}{\lvert\mathcal{P}\rvert}\sum_{\mathbf{y}\in\mathcal{P}} w_\imath(\mathbf{y})$ $- \gamma\frac{1}{\lvert\mathcal{N}\rvert}\sum_{\mathbf{z}\in\mathcal{N}} w_\imath(\mathbf{z})$	$Mass(\mathbf{x},\mathcal{Q}) = [\frac{1}{\lvert\mathcal{P}\rvert}\sum_{\mathbf{y}\in\mathcal{P}} Mass(\mathbf{x},\mathbf{y})^e$ $- \gamma\frac{1}{\lvert\mathcal{N}\rvert}\sum_{\mathbf{z}\in\mathcal{N}} Mass(\mathbf{x},\mathbf{z})^e]^{\frac{1}{e}}$	$dist(\mathbf{x},\mathcal{Q}) = \frac{1}{\lvert\mathcal{P}\rvert}\sum_{\mathbf{y}\in\mathcal{P}} dist(\mathbf{x},\mathbf{y})$ $- \gamma\frac{1}{\lvert\mathcal{N}\rvert}\sum_{\mathbf{z}\in\mathcal{N}} dist(\mathbf{x},\mathbf{z})$
Axioms	Violate all four distance axioms	Violate distance axiom 2	$p \geq 1$: Satisfy all distance axioms $p < 1$: Triangle inequality violated

the mass in each node. It is done using Algorithm 3. To estimate $Mass(\mathbf{x},\mathbf{y})$, $\mathbf{x}$ and $\mathbf{y}$ are parsed through t sTrees as described in Algorithms 4 and 5.

Note that sTree is a balanced binary tree; and let $\mathcal{L}^i_{\mathbf{x}}$ be the index of the external node in which $\mathbf{x}$ falls in sTree i and $\mathcal{L}^i_{\mathbf{x}} \in [1, 2 \ldots \psi]$. Since $\mathcal{L}^i_{\mathbf{x}}$ is fixed for each $\mathbf{x} \in D$, it can be computed as preprocessing.

For any instance $\mathbf{y}$, $\mathfrak{M}_{h,\imath}(\mathbf{x},\mathbf{y})$ in Equation 5 is the mass of the lowest common ancestor node of $\mathcal{L}^i_{\mathbf{x}}$ and $\mathcal{L}^i_{\mathbf{y}}$ in sTree i. This implementation converts from the bulk of tree traversals of $(\mathbf{x},\mathbf{y})$ to table lookups based on indices ($\mathcal{L}^i_{\mathbf{x}}$, $\mathcal{L}^i_{\mathbf{y}}$). It reduces the time complexity for Algorithms 4 and 5 from $O(nt\log(\psi))$ to $O((n+\psi)t)$, where $n = |D|$.

```
Algorithm 4: Mass($\mathbf{x}$,$\mathbf{y}$,$F$,$e$)

Input: $F$ - sForest with $t$ number of sTrees, $e$ - Exponent, $\mathbf{x},\mathbf{y}$ - Instances
Output: $Mass^h(\mathbf{x},\mathbf{y})$ as in Equation 4
$Mass \leftarrow 0$
for $i = 1 \rightarrow t$ do
    $T \leftarrow i^{th}\ sTree$ in $F$
    $Mass \leftarrow Mass +$ {TreeMass(T,x,y)}$^e$
end for
$Mass \leftarrow (\frac{Mass}{t})^{\frac{1}{e}}$
return $Mass$
```

```
Algorithm 5: TreeMass($T$,$\mathbf{x}$,$\mathbf{y}$)

Input: $T$ - $sTree$, $\mathbf{x},\mathbf{y}$ - Instances
Output: mass of $\mathbf{y}$ with respect to $\mathbf{x}$ for $T$
if $T$ is an external node then
    return $T.mass$
end if
if $x^{(T.SplitAttribute)} \leq T.SplitValue$ And
        $y^{(T.SplitAttribute)} \leq T.SplitValue$ then
    return $TreeMass(T.LeftChild,\mathbf{x},\mathbf{y})$
else if $x^{(T.SplitAttribute)} > T.SplitValue$ And
        $y^{(T.SplitAttribute)} > T.SplitValue$ then
    return $TreeMass(T.RightChild,\mathbf{x},\mathbf{y})$
else
    return $T.mass$
end if
```

VII. MassIR

This section describes a new information retrieval system constructed from Massim, called MassIR. It is motivated to improve the retrieval performance of ReFeat.

Like in ReFeat, MassIR has two stages. In the off-line modelling stage, a sForest is built from D as described in section VI. In the on-line retrieval stage, mass-based similarity is used to rank the instances in D with respect to a query and its feedback instances.

For a given query $\mathbf{q}$, $Mass(\mathbf{x},\mathbf{q})$ is estimated for all $\mathbf{x}$ in D. The ranking with respect to $\mathbf{q}$ is conducted as follows: $\forall\ \mathbf{x}_i, \mathbf{x}_j \in D$, if $Mass(\mathbf{x}_i,\mathbf{q}) < Mass(\mathbf{x}_j,\mathbf{q})$ then $similarity(\mathbf{x}_i,\mathbf{q}) > similarity(\mathbf{x}_j,\mathbf{q})$.

The feedback mechanism in MassIR is given in the second column of Table IV, where $\mathcal{Q} = \mathcal{P} \cup \mathcal{N}$; and $\mathcal{P}$ denotes the set of positive feedbacks and query, and $\mathcal{N}$ denotes the set of negative feedbacks. The ranking of instances $\mathbf{x} \in D$ is based on $Mass(\mathbf{x},\mathcal{Q})$.

MassIR takes $O(t\psi\log(\psi))$ time to generate t sTrees. It takes $O(nt\log(\psi))$ to update the mass values in tree nodes. Altogether, the offline processing time is $O(nt\log(\psi))$. The time complexity for a query has already been discussed in Section VI which takes $O((n+\psi)t)$. Together with $|\mathcal{Q}|-1$ feedbacks, the total computation time takes $O((n+|\mathcal{Q}|\psi)t)$.

A. *Comparison with* ReFeat *and* L^p-IR

In addition to MassIR, we have created a new information retrieval system called L^p-IR which has a feedback mechanism similar to ReFeat but based on the commonly used distance function, L^p-norm. A comparison of MassIR, ReFeat and L^p-IR is provided in Table IV.

The score function of ReFeat is essentially based on the unary function $\ell(\cdot)$. MassIR and L^p-IR are based on Massim and L^p-norm, respectively, to compute the similarity of two instances. ReFeat and MassIR must build a model in order to assess the similarity between instances, but L^p-IR does not need a model.

The feedback mechanism has the same form in all three systems; but ReFeat modifies the weight function $w(\mathbf{q})$ only, independent of $\mathbf{x}$. MassIR and L^p-IR have exactly the same formulation when $e = 1$, except different similarity measures are used. ReFeat violates all four distance axioms; MassIR violates some; and L^p-IR satisfies all

Table V: Time Complexity Comparison

	Off-line	Query	Feedback
ReFeat	$O(nt\log(\psi))$	$O((n+\log(\psi))t)$	$O((n+\|\mathcal{Q}\|)t)$
MassIR	$O(nt\log(\psi))$	$O((n+\psi)t)$	$O((n+\|\mathcal{Q}\|\psi)t)$
L^p-IR	-	$O(nd)$	$O(n\|\mathcal{Q}\|d)$

axioms for $p \geq 1$, but violates the triangle inequality axiom for $p < 1$.

Table V contains the comparison of the time complexities of MassIR with ReFeat and L^p-IR. Except in very small databases, $n > |\mathcal{Q}|\psi$. Hence, query and feedback time complexities of MassIR and ReFeat become $O(nt)$. Similarly, the space complexities of MassIR and ReFeat are both $O(nt)$. The space complexity of L^p-IR is $O(n)$. In MassIR and ReFeat, t is a constant parameter. Therefore, space and time requirements of MassIR and ReFeat increase linearly with the number of instances in the database. However, this is not the case for L^p-IR as feedback efficiency is much affected by $|\mathcal{Q}|$ and the number of dimensions, d.

B. Experiments

The aims of the experiments are to (i) compare MassIR with state-of-the-art systems ReFeat [1], Qsim [3], InstRank [6], MRBIR [4] and BALAS [5]; and (ii) analyse the behaviours of MassIR, ReFeat and L^p-IR when key parameter settings are changed.

We employ two databases which were previously used in other studies: GTZAN music database [1], [22] and COREL image database [1], [23].

The GTZAN database contains 1000 songs. Each song belongs to one of 10 genres, namely classical, country, disco, hiphop, jazz, rock, blues, reggae, pop and metal. There are 100 songs in each genre. Each song is stored as a 22,050Hz, 16 bit mono-audio file, which is a 30-second excerpt [22]. Each instance has 230 attributes, extracted from the music files, as reported in [1]. The COREL database has 10,000 images. They belong to 100 classes. Each class has 100 images. Each instance has 67 attributes, comprised of 32 colour features, 24 texture features and 11 shape features. Feature vectors of the COREL database are elaborated in [23].

Experiments to evaluate the retrieval performances of competing methods were designed as follows. A query was randomly selected from the database. A retrieval method ranked the instances in the database with respect to the query. In each feedback round, two positive instances (having the same class as the query) and two negative instances (having classes other than the query class) were provided. The method then re-ranked each instance in the database with respect to the query and the feedbacks. Five different queries were randomly selected from each class in the database; and up to five feedback rounds were performed for each query, where instances from a feedback round were added to those from the previous rounds. Instances already used as query and feedbacks were not included in the database for retrieval. This experimental design is the same as used in [1].

The above was repeated 20 times, each using a different forest from MassIR or ReFeat. Accordingly, a total of 1,000 queries and 5,000 feedback rounds were tested in GTZAN; and a total of 10,000 queries and 50,000 feedback rounds were tested in COREL. The retrieval performance is measured in terms of Mean Average Precision (MAP). A two-standard error significance test is used to examine whether the performance difference in a comparison is significant.

MassIR and ReFeat were implemented using MATLAB. The experiments were conducted on Sun Grid Engine (SGE). 4GB and 8GB memory were allocated for information retrieval tasks in GTZAN and COREL, respectively.

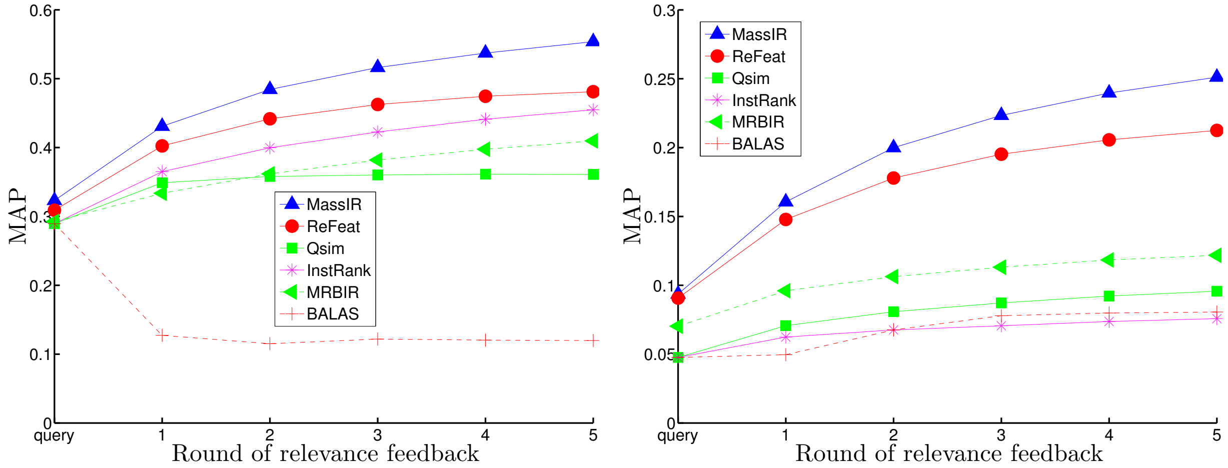
The results are presented in the following three paragraphs. Figure 3 shows the comparison of MassIR with ReFeat, Qsim, InstRank, MRBIR and BALAS. Results of Qsim, InstRank, MRBIR and BALAS were taken from [1]. The best parameter settings reported by [1] were used for ReFeat (i.e., $\psi = 4$ for GZTAN, $\psi = 8$ for COREL, and $\gamma = 0.25$). MassIR uses $\psi = 256$, $\gamma = 0$ and $e = -1$. The result shows that MassIR performs significantly better than all other existing methods. Also note that the performance gap increases as the number of feedback rounds increases, indicating that MassIR utilizes feedback instances more effectively than other methods.

Figure 4 shows the behaviour of MassIR and ReFeat when the sample size, ψ, changes. ReFeat performs best using small sample size. In contrast, the performance of MassIR monotonically increases as ψ increases.

Figure 5 shows the retrieval performance of MassIR and L^p-IR as γ varies. MassIR was tested for $e = -1$ and 1. L^p-IR was tested for $p =$ .25, 0.5, 0.75, 1 and 2. Recall that L^p-norm is a nonmetric violating the triangle inequality axiom when $p < 1$. The results show that MassIR with $e = -1$ was better than MassIR with $e = 1$ and L^p-IR for all γ. MassIR performed the best with γ = 0 in both databases. However, this is not the case for L^p-IR. Depending on the p value and the database, the performance peaked at either 0.5 or 0.75. In addition, MassIR is less sensitive with γ when compared to L^p-IR. It is interesting to note that MassIR performs the best at $\gamma = 0$, i.e., negative feedbacks have a negative impact, albeit small. In contrast, negative feedbacks have a significant positive impact on the retrieval performance of L^p-IR.

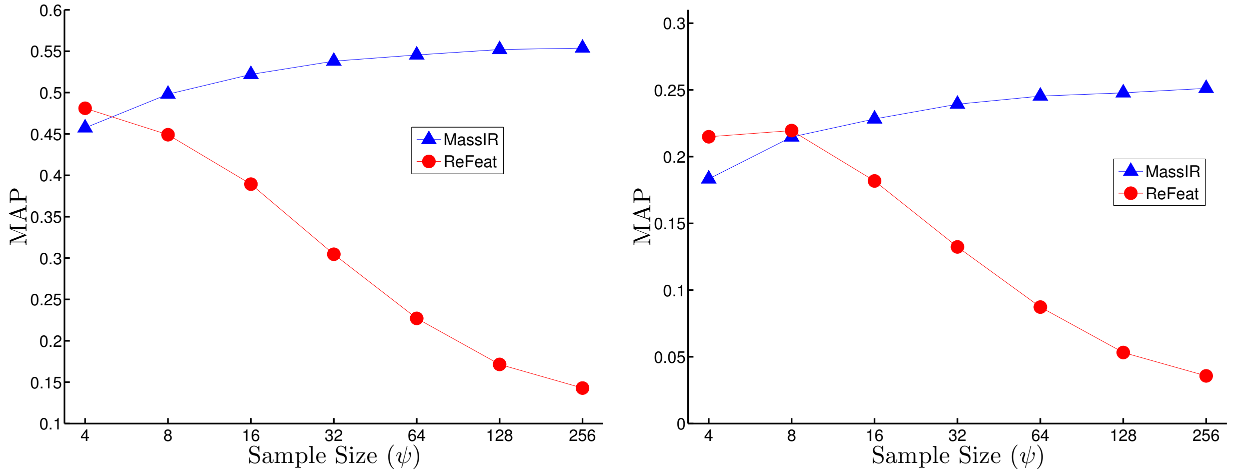
Note that L^p-IR is our creation of distance based approach. With the right setting, it performs better than Qsim, InstRank, MRBIR and BALAS (see their results showed in Figures 3 and 5); and L^p-IR has the same level of retrieval performance as ReFeat.

The execution time comparison is given in the Table VI. Compared head-to-head using $\psi = 8$, ReFeat runs 5.6 times faster than MassIR, i.e. they are in the same order of



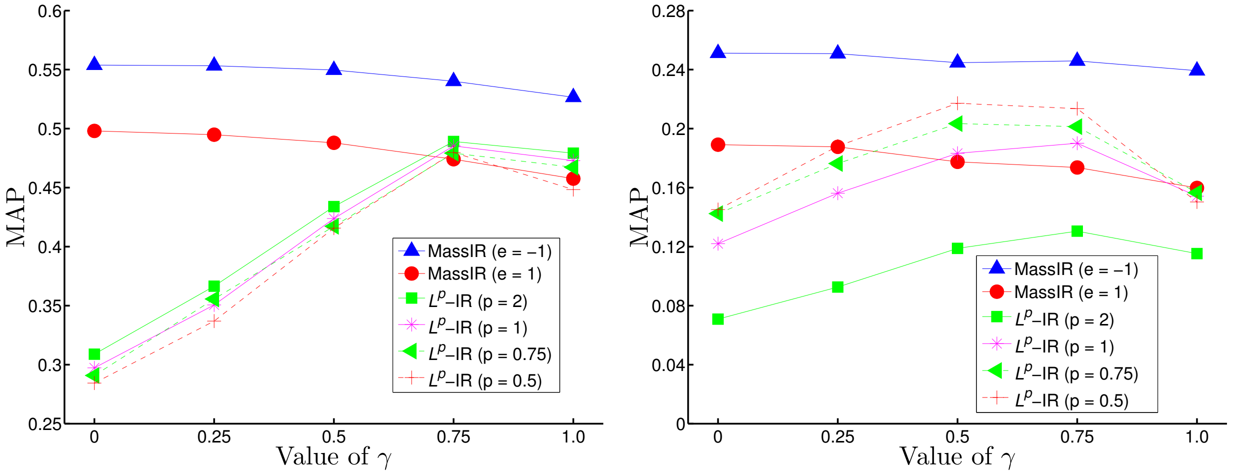
(a) GTZAN music database (b) COREL image database

Figure 3: Comparison of retrieval performance in terms of MAP



(a) GTZAN music database (b) COREL image database

Figure 4: Relative retrieval performance of MassIR and ReFeat as ψ increases (feedback round 5 performance).



(a) GTZAN music database (b) COREL image database

Figure 5: Relative retrieval performance of MassIR and L^p−IR for different values of γ (feedback round 5 performance). The results of L^p−IR ($p = 0.25$) were worse than the best results from other p values and they were omitted to increase readability.

Table VI: Online execution time per query/feedback in milliseconds for COREL for `MassIR` $\psi = 8$ (MIR8), `ReFeat` (RF), L^p−`IR` (LPIR), Qsim (QS), InstRank (IR),MRBIR (MR) and BALAS (BA)

Round	MIR8	RF	LPIR	QS	IR	MR	BA
query	194	35.8	5.5	24.7	24.7	612.9	N.A.
1	194	35.1	16.2	71.3	32.6	1172.4	262.8
2	195	35.4	26.8	146.3	33.4	1172.3	317.5
3	195	35.0	37.4	261.9	34.2	1172.3	373.0
4	196	34.8	48.0	417.9	34.9	1172.2	473.9
5	196	34.8	58.5	615.8	35.5	1172.1	506.0

time complexity $O(nt)$. This is consistent with the analysis of time complexity provided in section VII-A.

It is important to emphasise that `MassIR` behaves differently from `ReFeat` in two ways. First, the retrieval performance of `MassIR` improves as ψ increases; but this is not the case for `ReFeat`. Second, negative feedbacks have an impact on `ReFeat` but not on `MassIR`.

VIII. Discussion

Distance metric learning [24] aims to learn a metric, that obeys the four distance axioms, to achieve a better performance outcome than a standard metric such as Euclidean distance. There are two approaches. First, the supervised approach requires information about pairs of instances belonging to the same class as well as different classes in the training data. This information is used to compute the feature relevance that changes the neighbourhood of a test instance to improve the decision outcome. Second, the unsupervised approach or manifold learning aims to learn a low dimensional manifold where the distance between most instances is preserved; it is closely related to dimension reduction. In contrast, albeit an unsupervised approach, `Massim` is a nonmetric that neither computes distance nor requires dimension reduction.

Unlike many other nonmetric similarity measures (e.g., L^p-norm where $p < 1$), the new measure satisfies the triangle equality axiom. Thus, it can use existing indexing schemes such as M-Trees [25] to reduce the online time complexity of `MassIR` from $O(nt)$ to $O(log(n)t)$.

Similarities and differences between iForest and sForest are given as follows. Both ensure non-empty regions are constructed. iForest is likely to produce unbalanced trees, each using a ψ subset; and it is a unary function. Mapping $\mathbb{R}^d$ to $\mathbb{R}^t$ using iForest to produce t relevance features is an essential step in `ReFeat`.

In contrast, sForest produces balanced trees only, each using a ψ subset but requires the entire data set to determine the mass in each local region; and it is binary function. No feature space mapping is required in `MassIR`.

Also note that the resultant values of $\ell(\cdot)$ and $Mass(\cdot,\cdot)$ for any given test instances would increase if iForest were trained using a larger subsample or sForest were trained using a larger training set (even with the same subsample size). They can be easily normalised to be independent of training size, if this is required.

Our result shows that `MassIR` only requires positive feedbacks (i.e, $\gamma = 0$) to improve its retrieval performance. This reduces its time complexity from $O((n + |\mathcal{Q}|\psi)t)$ to $O((n + |\mathcal{P}|\psi)t)$.

The harmonic mean ($e = -1$) is less susceptible to large outliers than the arithmetic mean ($e = 1$). Therefore, we can expect `MassIR` to perform better when $e = -1$ than $e = 1$.

Treating $e = -1$ and $\gamma = 0$ as default, the only parameter that needs to be set for `MassIR` is the sample size, ψ (as t should be set to a high value such as 1000, as in the case of `ReFeat`). Since both the retrieval performance and the processing time increase with ψ, the parameter selection of `MassIR` is a trade-off between retrieval performance and runtime.

It is interesting to note that the retrieval performance of L^p−`IR` could be improved by using the harmonic mean rather than the arithmetic mean in the feedback formulations shown in Table IV (but not for `ReFeat`). However, this improvement is still worse than `MassIR`.

We have shown that `Massim` is significantly better than L^p-norm (used in InstRank, Qsim, BALAS and L^p−`IR`) in two data sets. However, a caveat is in order. One should not expect `Massim` to outperform L^p-norm in all cases. There are a number of factors which affect the performance, e.g., whether the algorithms make full use of the similarity measure, and whether the characteristics of the domain match the similarity measure.

There are many applications using tree structures. Before examining the superficial similarities, one must check the purposes first. sTree is created for similarity measurement. There are different trees created for indexing in order to speed up the nearest neighbour search, e.g., k-d tree. The superficial similarity between sTrees and k-d trees is: the latter is based on median split and the former is based on a randomly selected split close to median. There are many differences, e.g., (i) a k-d tree is created by cycling all dimensions one at a time to build each subsequent internal node in the tree, but sTree randomly selects an attribute to build an internal node; (ii) a k-d tree is constructed using the entire data set, but a sTree is built using a small subset. These similarities and differences are superficial as far as the purpose is concerned.

IX. CONCLUDING REMARKS

Mass-based similarity measures, `Massim`, represents a paradigm shift from measuring similarity in terms of distance to measuring similarity in terms of mass.

We establish the theoretical foundation of `Massim`, which is a generalisation of mass estimation. This generalisation shows that the new binary function for similarity measure $Mass(\cdot,\cdot)$ produces better retrieval performance than the commonly used distance-based similarity measures.

This paper shows the effect of overcoming the key weakness of ReFeat by ensuring that two similar instances are in the same local neighbourhood. This contributes directly to (i) the better retrieval performance of MassIR; and (ii) a more desirable behaviour of MassIR, where the retrieval performance of MassIR can be improved by increasing ψ. Our empirical results verify that MassIR has a better retrieval performance than ReFeat while having the same order of time and space complexities.

References

[1] G. T. Zhou, K. M. Ting, F. T. Liu, and Y. Yin, "Relevance feature mapping for content-based multimedia information retrieval," *Pattern Recognition*, vol. 45, no. 4, pp. 1707–1720, 2012.

[2] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, 2003, pp. 29–38.

[3] Z. H. Zhou and H. B. Dai, "Query-sensitive similarity measure for content-based image retrieval," in *Proceedings of the Sixth International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 1211–1215.

[4] J. He, M. Li, H. J. Zhang, H. Tong, and C. Zhang, "Manifold-ranking based image retrieval," in *Proceedings of the 12th Annual ACM International Conference on Multimedia*. New York, NY, USA, 2004, pp. 9–16.

[5] R. Zhang and Z. M. Zhang, "BALAS: Empirical bayesian learning in the relevance feedback for image retrieval," *Image and Vision Computing*, vol. 24, no. 3, pp. 211–223, 2006.

[6] G. Giacinto and F. Roli, "Instance-based relevance feedback for image retrieval," *Advances in Neural Information Processing Systems*, vol. 17, pp. 489–496, 2005.

[7] K. M. Ting, G. T. Zhou, F. T. Liu, and S. C. Tan, "Mass estimation," *Machine Learning*, vol. 90, pp. 127–160, 2013.

[8] F. G. Ashby and D. M. Ennis. Similarity measures. *Scholarpedia*, 2(12):4116, 2007.

[9] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios, "Nearest neighbor retrieval using distance-based hashing," in *IEEE 24th International Conference on Data Engineering*, 2008, pp. 327 –336.

[10] S. H. Cha, "Comprehensive survey on distance/similarity measures between probability density functions," *International Journal of Mathematical Models and Methods in Applied Sciences*, vol. 1, no. 4, pp. 300–307, 2007.

[11] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search - The Metric Space Approach*, ser. Advances in Database Systems. Springer, 2006, vol. 32.

[12] S. Santini and R. Jain, "Similarity measures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 871–883, 1999.

[13] A. Tversky, "Features of similarity." *Psychological review*, vol. 84, no. 4, pp. 327–352, 1977.

[14] C. L. Krumhansl, "Concerning the applicability of geometric models to similarity data: The interrelationship between similarity and spatial density." *Psychological Review*, vol. 85, no. 5, pp. 445–463, 1978.

[15] T. Skopal and B. Bustos, "On nonmetric similarity search problems in complex domains," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, pp. 34:1–34:50, 2011.

[16] L. Chen and X. Lian, "Efficient similarity search in nonmetric spaces with local constant embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 3, pp. 321–336, 2008.

[17] T. Skopal, "On fast non-metric similarity search by metric access methods," in *Proceedings of the 10th International Conference on Advances in Database Technology*. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 718–736.

[18] K. S. Goh, B. Li, and E. Chang, "Dyndex: a dynamic and non-metric space indexer," in *Proceedings of the Tenth ACM international Conference on Multimedia*. New York, NY, USA, 2002, pp. 466–475.

[19] D. Jacobs, D. Weinshall, and Y. Gdalyahu, "Classification with nonmetric distances: image retrieval and class representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 583–600, 2000.

[20] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation forest," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 413–422.

[21] K. M. Ting, G. T. Zhou, F. T. Liu, and J. S. C. Tan, "Mass estimation and its applications," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA, 2010, pp. 989–998.

[22] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[23] Z. H. Zhou, K. J. Chen, and H. B. Dai, "Enhancing relevance feedback in image retrieval using unlabeled data," *ACM Transactions on Information Systems*, vol. 24, no. 2, pp. 219–244, 2006.

[24] L. Yang and R. Jin, "Distance metric learning: A comprehensive survey," *Michigan State Universiy*, pp. 1–51, 2006.

[25] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 426–435.

Noname manuscript No.
(will be inserted by the editor)

An ensemble approach to estimate multi-dimensional likelihood in Bayesian classifier learning

Sunil Aryal · Kai Ming Ting

Abstract In Bayesian classifier learning, estimating the joint probability distribution $p(\mathbf{x}, y)$ or the likelihood $p(\mathbf{x}|y)$ directly from training data is considered to be difficult, especially in large multi-dimensional data sets. In order to circumvent this difficulty, existing Bayesian classifiers such as Naive Bayes, BayesNet and AηDE have focused on estimating simplified surrogates of $p(\mathbf{x}, y)$ from different forms of one-dimensional likelihoods.

Contrary to the perceived difficulty in multi-dimensional likelihood estimation, we present a simple ensemble approach to estimate multi-dimensional likelihood directly from data. The idea is to aggregate $p_i(\mathbf{x}|y)$ estimated from a random subsample of data $\mathcal{D}_i$ $(i = 1, 2, \cdots, t)$. This paper presents two ways to estimate multi-dimensional likelihoods using the proposed ensemble approach and introduces two new Bayesian classifiers called *ENNBayes* and *MassBayes* that estimate $p_i(\mathbf{x}|y)$ using a nearest neighbour density estimation and a probability estimation through feature space partitioning, respectively.

Unlike the existing Bayesian classifiers, ENNBayes and MassBayes have constant training time and space complexities and scale better than existing Bayesian classifiers in very large data sets. Our empirical evaluation shows that ENNBayes and MassBayes yield better predictive accuracy than the existing Bayesian classifiers on benchmark data sets.

Keywords Bayesian classifiers, Multi-dimensional likelihood estimation

1 Introduction

In classification task, the training data D is a collection of labelled instances $\{(\mathbf{x}^{(i)}, y^{(i)})\}$ $(i = 1, 2, \cdots, n)$, where $\mathbf{x}$ is a d-dimensional vector $\langle x_1, x_2, \cdots, x_d\rangle$ and y is a label from the predefined set of c classes.

S. Aryal
Monash University, Victoria, Australia
E-mail: sunil.aryal@monash.edu

K. M. Ting
Monash University, Victoria, Australia
E-mail: kaiming.ting@monash.edu

The Bayesian approach of classifier learning models the joint probability distribution $p(\mathbf{x}, y)$ and predicts the most probable class using Bayes rule as:

$$\hat{y} = \arg\max_{y} \; p(\mathbf{x}, y) \tag{1}$$

Using the product rule, the joint probability can be factorised as:

$$p(\mathbf{x}, y) = p(y) \times p(\mathbf{x}|y) \tag{2}$$

Bayesian classifiers learn either the joint probability distribution $p(\mathbf{x}, y)$ or the conditional probability distribution (likelihood) $p(\mathbf{x}|y)$. Estimating $p(\mathbf{x}, y)$ or $p(\mathbf{x}|y)$ directly from data is considered to be difficult because commonly used density estimators such as kernel density estimator (KDE), and k-nearest neighbour (kNN) density estimator are impractical in problems with large data sizes and moderate number of dimensions. This is due to their high space and time complexities (Silverman, 1986).

However, surrogates of $p(\mathbf{x}, y)$ can be estimated efficiently provided that some simplifying assumptions are made (e.g., attributes are independent given the class label, or data are assumed to have some known distribution such as Gaussian). Existing Bayesian classifiers make some kind of conditional independence assumption and estimate the simplified surrogate of $p(\mathbf{x}, y)$ as the product of one-dimensional likelihoods. Even though existing Bayesian classifiers such as Naive Bayes (Langley et al., 1992), BayesNet (Friedman et al., 1997) and AηDE (Webb et al., 2012) have been shown to perform well in many application domains, they can result in poor predictive accuracy in many other real-world problems as the estimate of $p(\mathbf{x}, y)$ is a biased one-dimensional approximation of the true distribution.

Instead of researching on different forms of one-dimensional likelihoods, we reexamine the premise that multi-dimensional likelihoods are difficult to estimate and find that there is a simple way to estimate multi-dimensional likelihoods directly from data using an ensemble approach.

In this paper, we make the following three contributions:

1. Presenting the first generic approach to estimate $p(\mathbf{x}|y)$ directly by aggregating estimations from an ensemble of t estimators where each estimates the multi-dimensional likelihood $p_i(\mathbf{x}|y)$ using a fixed-size random sub-sample $\mathcal{D}_i \subset D$ ($i = 1, 2, \cdots, t$). This is a generic approach because $p_i(\mathbf{x}|y)$ can be estimated using different data modelling methods.
2. Introducing two variants of the ensemble approach that estimate $p_i(\mathbf{x}|y)$ using a nearest neighbour density estimation and a probability estimation through feature space partitioning, and produce two new Bayesian classifiers called **ENNBayes** and **MassBayes**, respectively.
3. Verifying that the proposed classifiers produce better classification accuracy and scale better than the state-of-the-art Bayesian classifiers in large data sets.

The proposed approach has the following distinguishing characteristics over the existing Bayesian classifiers:

- Unlike existing Bayesian classifiers that estimate one-dimensional likelihoods, it estimates multi-dimensional likelihoods directly from data.

- As it employs fixed-size sub-samples of data, it has constant training time and constant space complexities. Thus, it can be easily applied in very large data sets.

The rest of the paper is structured as follows. Section 2 provides an overview of existing well-known Bayesian classifiers. Section 3 presents the ensemble approach to estimate $p(\mathbf{x}|y)$ directly from data and describes two new Bayesian classifiers based on the proposed approach - ENNBayes and MassBayes. The empirical evaluation results are presented in Section 4. Finally, we provide discussion and conclusions in the last two sections.

2 Related work

This section provides a survey of previous works related to this paper. Subsection 2.1 discusses some widely used methods to estimate likelihood in real domain ($\mathcal{R}^d$). Subsection 2.2 provides a survey of Bayesian classifier learning and describes some well-known Bayesian classifiers.

2.1 Likelihood estimation

The likelihood $p(\mathbf{x}|y)$ for continuous-valued attributes can be estimated either through density estimation or by estimating probability in the local region (LR) of $\mathbf{x}$.

The probability $p(\mathbf{x}|y)$ in a LR can be estimated as a ratio of the number of instances in the LR that belong to class y and the total number of instances in class y. In existing Bayesian classifiers, continuous-valued attributes are converted to discrete attributes through discretisation and the LR of $\mathbf{x}$ is then defined by the smallest discrete intervals that contains $\mathbf{x}$.

Discretisation divides the range of an attribute x into v discrete intervals and maps each $x \in \mathcal{R}$ to a corresponding interval, yielding a categorical attribute with v labels, i.e., $x^* \in \{a_1, a_2, \cdots, a_v\}$. Different methods of discretisation determine the intervals in different ways. In unsupervised discretisation (e.g., equal width or equal frequency discretisation (Catlett, 1991)), class label is not used while selecting the cut points. Supervised discretisation methods find the cut points based on some criterion that takes class information into consideration such as entropy of the intervals, error in the training data or some statistical measure. Fayyad and Irani (1995) proposed a supervised discretisation method that selects the cut points which minimise the class entropy. In classification, the supervised methods often produce better predictive accuracy than the unsupervised methods (Dougherty et al., 1995). Note that the combinations of values grow exponentially with d because there are v^d possible combinations. Many of those possible combinations may not appear in the observed data. If the unseen instance $\mathbf{x}$ has a combination of attribute values that did not appear in the observed data, it yields zero probability. Hence, the multi-dimensional likelihood estimation through discretisation may not provide a good estimate.

Density estimators approximate the unknown probability density function f from the observed data. The parametric estimators assume that the data are drawn from a known distribution and derive the function $\hat{f}$ by estimating parameters of the distribution such as mean and variance of a Gaussian distribution (Silverman, 1986). But, in reality, the underlying distribution does not always belong to a parametric distribution. Non-parametric approaches make less rigid assumptions and estimate the distribution from the observed data (Silverman, 1986). Kernel density estimator (KDE) and k-nearest neighbour density estimator (kNN) are two well-known non-parametric density estimators.

KDE estimates the density as the average of a kernel function $K(\cdot)$ centered on each observed instance as (Silverman, 1986):

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}^{(i)}}{h}\right) \tag{3}$$

where h is the smoothing parameter called bandwidth.

In order to estimate the density at point $\mathbf{x}$, kNN searches k nearest neighbours of $\mathbf{x}$ in the observed data. A kNN density estimation can be expressed as follows (Breunig et al., 2000; Tan et al., 2006):

$$\hat{f}(\mathbf{x}) = \frac{|N(\mathbf{x}, k)|}{n \displaystyle\sum_{\mathbf{x}' \in N(\mathbf{x},k)} distance(\mathbf{x}, \mathbf{x}')} \tag{4}$$

where $N(\mathbf{x}, k)$ is the set of k nearest neighbours of $\mathbf{x}$ and $|s|$ is the cardinality of set s.

Both KDE and kNN employ the expensive distance calculations over all n observed instances, which restricts them to small data sets. Apart from high time and space complexity, KDE and kNN require search to find optimal values for h and k. In Bayesian classification framework, the density distribution of each class has to be modelled separately. The same value for h or k may not be equally good for modelling density distribution of all the classes. Searching for an optimal value for each class will add further runtime cost to already expensive distance calculations. Hence, they are impractical in large multi-dimensional data sets.

DEMass (Ting et al., 2013) is the first efficient density estimator in large data set. It achieves significant improvement over KDE and kNN in terms of time and space complexities by aggregating density estimations from t different random sub-samples of data $\mathcal{D}_i$ $(i = 1, 2, \cdots, t)$. It is an ensemble approach where density in each sub-sample is estimated through multi-dimensional mass estimation (Ting and Wells, 2010) as follows:

$$\bar{f}_{\mathbf{m}}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \frac{|(T_i(\mathbf{x}))|}{|\mathcal{D}_i|\ v_i}. \tag{5}$$

where $T_i(\cdot)$ is a function which subdivides the feature space defined by $\mathcal{D}_i$ into non-overlapping regions through axis-parallel divisions; $|(T_i(\mathbf{x}))|$ is the number of instances in the smallest region of $T_i(\mathbf{x})$ in which $\mathbf{x}$ falls into; and v_i is the volume of region $T_i(\mathbf{x})$.

DEMass has sub-linear time complexity and constant space complexity (Ting et al., 2013). But, it is limited to low dimensional problems. Also, it is not adaptive to data distribution as it constructs equal volume regions in the data space.

2.2 Bayesian classifier learning

Naive Bayes (NB) (Duda and Hart, 1973; Langley et al., 1992) is the simplest Bayesian approach to solve classification problems. It assumes that the attributes $(x_1, x_2, \cdots, x_d)$ are statistically independent given the class label (y) (class conditional independence) and estimates $p(\mathbf{x}|y)$ as:

$$\hat{p}(\mathbf{x}|y) = \prod_{j=1}^{d} p(x_j|y) \tag{6}$$

Kononenko (1991) extended NB to detect dependencies between attributes and proposed a variant of NB called semi-naive Bayes. Langley and Sage (1994) proposed another variant of NB, called selective naive Bayes by removing the correlated attributes in the modelling process. In the traditional NB, the distribution of a continuous-valued attribute is assumed to be a Gaussian distribution and $p(x_i|y)$ is estimated through normal density estimation. Langley and John (1995) have shown the improvement in predictive accuracy of NB by replacing the parametric normal density estimator with a non-parametric kernel density estimator (KDE). A similar conclusion was drawn by Dougherty et al. (1995) through discretisation.

Despite the strong assumption of class conditional independence, NB produces impressive results in many application domains where the assumption does not always hold (Clark and Niblett, 1989; Kononenko, 1993; Domingos and Pazzani, 1997). Its simplicity and clear probabilistic semantics have motivated researchers to explore further to lessen its assumption of conditional independence.

Naive Bayes tree (NBTree) (Kohavi, 1996) combines the benefits of decision tree and NB by using NB at the leaves of the decision tree to make predictions. It has been shown that NBTree outperforms NB (Kohavi, 1996). Lazy Bayesian rules (LBR) (Zheng and Webb, 2000) builds a rule which best characterises each test instance during testing and builds local NB to classify. Another lazy Bayesian classifier is locally weighted naive Bayes (LWNB) (Frank et al., 2003) that searches for k nearest neighbours of each test instance and builds a NB based on k nearest neighbours weighted by their distance to the test instance. All these classifiers improve the prediction accuracy of NB, but they are expensive in terms of runtime and inapplicable in large data sets.

A Bayesian network (BayesNet) (Friedman et al., 1997) learns probabilistic relationships among the attributes and class from the training data, in the form of a directed acyclic graph (DAG). In a graph, each node is probabilistically independent of its non-descendants given the state of its parents. At each node, conditional probabilities with respect to its parents are learned from the training data. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y) = p(x_1|\pi_1) \times p(x_2|\pi_2) \times \cdots \times p(x_d|\pi_d) \times p(y|\pi_y) \tag{7}$$

where π_j is $parent(x_j)$ and π_y is $parent(y)$.

NB is the simplest form of Bayesian networks, where each attribute has y as its only parent.

Learning an optimal network of probabilistic dependencies is difficult and computationally intractable as it requires searching over every possible network (Chickering, 1996; Jiang et al., 2007). Thus, in practice, either some heuristics are used or some restrictions are imposed on the network structure.

Friedman et al. (1997) proposed a Bayesian network called tree augmented naive Bayes (TAN), where the probabilistic relationship is a tree-like structure. TAN allows each attribute to have one additional parent along with the class attribute. Another variant of TAN learning is super parent-TAN (SP-TAN) (Keogh and Pazzani, 1999) that searches for a common super parent which yields the minimum error in the training data. Grossman and Domingos (2004) proposed a heuristic to produce a Bayesian network classifier that learns a network structure (probabilistic relationship) by maximising conditional likelihood.

Aggregating η-dependence estimators (AηDE) (Webb et al., 2012) avoids the expensive search in learning probabilistic dependencies by constructing an ensemble of η-dependence estimators. It is the only ensemble approach that was designed specifically to estimate $p(\mathbf{x}, y)$. It began with one-dependence estimators (Webb et al., 2005) and then generalised to η-dependence estimators (Webb et al., 2012). Each estimator allows dependency between y and η privileged attributes or super-parents. The other attributes are assumed to be conditionally independent given the η super-parents and y. The joint probability $p(\mathbf{x}, y)$ is estimated as:

$$\hat{p}(\mathbf{x}, y) = \sum_{\mathbf{s} \in S_\eta} p(\mathbf{x_s}, y) \prod_{j \in \{1,2,\cdots,d\} \setminus \mathbf{s}} p(x_j | \mathbf{x_s}, y) \tag{8}$$

where S_η is the collection of all subsets of size η of the set of d attributes $\{1, 2, \cdots, d\}$; and $\mathbf{x_s}$ is a η-dimensional vector of values of $\mathbf{x}$ defined by $\mathbf{s}$.

NB is a member of AηDE family with $\eta = 0$, A0DE. A1DE and A2DE produce better predictive accuracy than the other state-of-the-art Bayesian classifiers (Webb et al., 2005, 2012). However, it only allows dependencies on a fixed number of attributes and y. Because of the high time complexity of $O\left(n\binom{d}{\eta+1}\right)$ [1] and space complexity of $O\left(c\binom{d}{\eta+1}v^{\eta+1}\right)$, where v is the average number of values for an attribute (Webb et al., 2012), only A2DE or A3DE is feasible even for a moderate number of dimensions. Furthermore, selecting an appropriate value of η for a particular data set requires a search.

All of the Bayesian classifiers surveyed so far estimate the joint distribution as the product of one-dimensional likelihoods. Both BayeNet and AηDE allow limited probabilistic dependencies among attributes. None of them consider the unrestricted inter-dependencies between the attributes, and they all have limitations in terms of time and space complexities in large data sets.

DEMassBayes (Ting et al., 2013) estimates the multi-dimensional likelihood directly without making any explicit assumptions through DEMass. It builds t models per class y from subsets of instances belonging to class y in each $\mathcal{D}_i$, i.e., $\mathcal{D}_{i,y} \subset \mathcal{D}_i$, and estimate $p(\mathbf{x}|y)$ as:

$$p(\mathbf{x}|y) = \frac{1}{t} \sum_{i=1}^{t} \frac{|(T_{i,y}(\mathbf{x}))|}{|\mathcal{D}_{i,y}| \; v_{i,y}} \tag{9}$$

[1] $\binom{d}{\eta}$ is a binomial coefficient of η out of d.

where $v_{i,y}$ is the volume of the region $T_{i,y}(\mathbf{x})$.

DEMassBayes had better predictive accuracy than A1DE and BayesNet in low dimensional problems. But, it had poor predictive accuracy in problems with a moderate number of dimensions. This is because volume $v_{i,y}$ reduces exponentially and becomes very small as the parameter controlling the number of regions increases; and the estimation of $p(\mathbf{x}|y)$ is adversely affected by regions having small volumes with few instances.

3 An ensemble approach for multi-dimensional likelihood estimation

Even though DEMassBayes has some limitations, it sets a benchmark to employ an ensemble approach to estimate the multi-dimensional likelihood from sub-samples of data. The approach used in DEMassBayes can be generalised as a generic ensemble approach to estimate $p(\mathbf{x}|y)$ as follows:

$$p(\mathbf{x}|y) = \frac{1}{t}\sum_{i=1}^{t} p_i(\mathbf{x}|y) \tag{10}$$

where $p_i(\mathbf{x}|y)$ is estimated from a fixed-size sub-sample $\mathcal{D}_i \subset D$ $(i = 1, 2, \cdots, t)$, $|\mathcal{D}_i| = \psi < n$.

From a small sub-sample $\mathcal{D}_i$, $p_i(\mathbf{x}|y)$ can be easily estimated by using existing data modelling techniques such as density estimation or other means.

DEMassBayes is one realisation of Equation 10 that estimates $p_i(\mathbf{x}|y)$ through mass-based density estimation. It estimates the density $f_i(\mathbf{x}|y)$ from each $\mathcal{D}_i$ as:

$$f_i(\mathbf{x}|y) = \frac{|(T_{i,y}(\mathbf{x}))|}{|\mathcal{D}_{i,y}|\ v_{i,y}} \tag{11}$$

From $f_i(\mathbf{x}|y)$, $p_i(\mathbf{x}|y)$ can be estimated as the probability that $\mathbf{x}$ lies in a small region ϵ.

$$p_i(\mathbf{x}|y) = \int_{\epsilon} f_i(\mathbf{x}|y)\ d\mathbf{x} \approx f_i(\mathbf{x}|y) \times \mathit{volume}(\epsilon) \tag{12}$$

Since $\mathit{volume}(\epsilon)$ is a constant, it can be ignored in classification decision and $p_i(\mathbf{x}|y)$ can be expressed as:

$$p_i(\mathbf{x}|y) \approx f_i(\mathbf{x}|y) \tag{13}$$

The ensemble approach estimates the multi-dimensional likelihood directly from data. It has the following characteristics:

- The average over t estimators provides a good approximation of $p(\mathbf{x}|y)$ as it considers distribution over different local neighbourhoods of $\mathbf{x}$.
- As it employs fixed-size data sub-samples, it has constant training time and constant space complexities.
- Existing density estimators, which have problem running in large data sets, can now be used as long as $\psi \ll n$.
- In large data sets where $t\psi < n$, the proposed method runs faster than estimating $p(\mathbf{x}|y)$ using the entire data set D.

- The time complexity can be further reduced by a factor of t by using parallel computing. Each estimator in the ensemble can be built independently in different computing node.

The proposed ensemble method differs from the existing ensemble methods in classifier learning in two ways. First, ensemble methods such as Bagging (Breiman, 1996), Boosting (Freund and Schapire, 1996), Random Forest (Breiman, 2001) and Feating (Ting et al., 2011) aggregate the posterior probability $p(y|\mathbf{x})$ from different models to make the final prediction, whereas the proposed method aggregates $p(\mathbf{x}|y)$ and uses the Bayes rule to make the final prediction. Second, each model in the existing ensemble methods is trained with a sample of size n, but each model in the proposed method can be built with a small sub-sample of training data $(\psi \ll n)$ and the ensemble still performs well.

Like the existing ensemble approach of AηDE, the proposed new ensemble approach avoids search in learning a Bayesian classifier. But, it has the following distinguishing characteristics in comparison with AηDE:

- AηDE estimates one-dimensional likelihoods given a fixed number of super-parents and y, whereas the proposed approach estimates multi-dimensional likelihoods directly.
- In AηDE, the ensemble size is fixed to $\binom{d}{\eta}$. But, the proposed approach has the flexibility for users to set the ensemble size.
- Each model in the proposed approach is built with training sub-sample of size $\psi < n$ which gives rise to the constant training time. In contrast, each model in AηDE is trained using the entire training set.
- AηDE is a deterministic algorithm whereas the proposed approach is a randomised algorithm.

Using the ensemble approach to estimate $p(\mathbf{x}|y)$, we introduce two new Bayesian classifiers using two new realisations of Equation 10: ENNBayes and MassBayes. ENNBayes uses a nearest neighbour density estimator to estimate $p_i(\mathbf{x}|y)$. MassBayes partitions the data space of $\mathcal{D}_i$ to create local regions and then estimates $p_i(x|y)$ from the numbers of instances in the local region (LR) and $\mathcal{D}_i$. ENNBayes is a lazy approach that defines LRs implicitly in terms of nearest neighbours when the test instance is presented. MassBayes is an eager learner that explicitly partitions the feature space enclosed by $\mathcal{D}_i$. The conceptual differences between ENNBayes, MassBayes and DEMass-Bayes are provided in Table 1. ENNBayes and MassBayes are described in the next two subsections.

Table 1: Three realisations of the proposed ensemble approach to estimate $p(\mathbf{x}|y)$: ENNBayes, MassBayes and DEMass-Bayes

Classifiers	Defining Local Regions	Estimating $p_i(\mathbf{x}|y)$	Model
ENNBayes	Implicit with nearest neighbours.	Nearest neighbour density estimation.	No model
MassBayes	Explicit with feature space partitioning.	Probability estimation in local regions.	One model for all classes.
DEMassBayes	Explicit with feature space partitioning.	Density estimation based on mass.	One model per class.

3.1 ENNBayes: An ensemble of nearest neighbour density estimators

We call the new Bayesian classifier that estimates the likelihood using an ensemble of nearest neighbour density estimators ***ENNBayes*** (**E**nsemble of **N**earest **N**eighbour **Bayes**ian classifier).

Using a random sub-sample of training instances $\mathcal{D}_i$, the conditional density $f_i(\mathbf{x}|y)$ can be estimated as:

$$f_i(\mathbf{x}|y) = \frac{|N(\mathbf{x}, k|\mathcal{D}_{i,y})|}{|\mathcal{D}_{i,y}| \sum\limits_{\mathbf{x}' \in N(\mathbf{x},k|\mathcal{D}_{i,y})} distance(\mathbf{x}, \mathbf{x}')} \tag{14}$$

where $\mathcal{D}_{i,y}$ is a set of instances belonging to class y in $\mathcal{D}_i$, $\bigcup_y \mathcal{D}_{i,y} = \mathcal{D}_i$; and $N(\mathbf{x}, k|\mathcal{D}_{i,y})$ is the set of k nearest neighbours of $\mathbf{x}$ in $\mathcal{D}_{i,y}$.

Since the nearest neighbour instance contributes significantly to $f_i(\mathbf{x}|y)$, the above equation can be simplified by considering the nearest neighbour only (i.e., $k = 1$).

In the case of skewed class distribution, instances from some classes may not present in $\mathcal{D}_i$. If there is no instance from any class in $\mathcal{D}_i$, $p_i(\mathbf{x}|y)$ for that class is zero. Hence, $p_i(\mathbf{x}|y)$ from $\mathcal{D}_i$ is estimated as:

$$p_i(\mathbf{x}|y) = \begin{cases} \frac{1}{|\mathcal{D}_{i,y}| \times distance(\mathbf{x}, NN(\mathbf{x}|\mathcal{D}_{i,y}))} & \text{if } |\mathcal{D}_{i,y}| > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

where $NN(\mathbf{x}|\mathcal{D}_{i,y})$ is the nearest neighbour of $\mathbf{x}$ in $\mathcal{D}_{i,y}$.

With $k = 1$, the final estimate considers different local neighbourhoods of the test instance (the nearest neighbour in each $\mathcal{D}_i$). An illustrative example of implicit local regions centered on the nearest neighbour of $\mathbf{x}$ in different sub-samples is shown in Figure 1. The shape of the regions depend on the distance measure (L^p-norm) used: hypersphere if $p = 2$ (Euclidean distance) and hypercube if $p = \infty$.

ENNBayes is a lazy learning approach. There is no model building process. In the training phase, each data sub-sample $\mathcal{D}_i \subset D$ ($i = 1, 2, \cdots, t$) is constructed

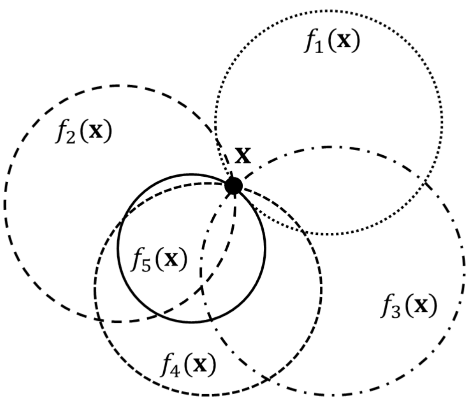


Fig. 1: Implicit regions to estimate $p_i(\mathbf{x})$ using Euclidean distance (L^2-norm).

by sampling ψ instances from D without replacement. The sampling process is restarted with D when all the instances in D are used. In the testing phase, the nearest neighbour of the test instance $\mathbf{x}$ in each class is searched in each sub-sample $\mathcal{D}_i$ and $p_i(\mathbf{x}|y)$ is estimated from Equation 15.

ENNBayes requires to store $t\psi$ instances. Hence, the space complexity is $O(t\psi d)$. In order to classify a test instance, the nearest neighbour in each class is searched in each of the t sub-samples. If φ is the average number of instances per class in each $\mathcal{D}_i$, the total testing time complexity is $O(ct\varphi d)$. Even though ENNBayes reduces the time complexity to constant in terms of n, it still requires distance calculation in order to find the nearest neighbour.

3.2 MassBayes: Multi-dimensional likelihood estimation through probability estimation in a local region defined by feature space partitioning

MassBayes estimates multi-dimensional likelihood $p_i(\mathbf{x}|y)$ through probability estimation in local region. Instead of converting continuous-valued attributes to discrete attributes through discretisation (as done by existing Bayesian classifiers, as a preprocessing step), MassBayes defines local regions through feature space partitioning in the model building process.

Let $T_i(\cdot)$ be a function that divides the data space of $\mathcal{D}_i$ into non-overlapping regions and $T_i(\mathbf{x})$ be the smallest local region where $\mathbf{x}$ falls into. The likelihood $p_i(\mathbf{x}|y)$ is defined as follows:

$$p_i(\mathbf{x}|y) = \frac{|T_i(\mathbf{x}, y)|}{|\mathcal{D}_{i,y}|} \tag{16}$$

where $|T_i(\mathbf{x}, y)|$ is the number of instances belonging to class y in $T_i(\mathbf{x})$ and $|\mathcal{D}_{i,y}|$ is the number of instances belonging to class y in $\mathcal{D}_i$.

The feature space partitioning is constructed using a binary tree structure called *h*:*d-tree*, as used by Ting and Wells (2010) in multi-dimensional mass estimation. The data space enclosing instances in $\mathcal{D}_i$ is sub-divided into two half-spaces by splitting at the mid-point on a dimension. The process is then repeated in each non-empty half-space with more than one instance in it on a different dimension.

In a multi-dimensional space, each instance in $\mathcal{D}_i$ can be isolated by splitting on a few dimensions i.e., only a subset of d attributes, $\mathbf{g} \subset \{1, 2, \cdots, d\}$ is used to define $T_i(\mathbf{x})$. Hence, $p_i(\mathbf{x}|y)$ is estimated as $p_i(\mathbf{x_g}|y)$ where $\mathbf{x_g}$ is a $|\mathbf{g}|$-dimensional vector of values of $\mathbf{x}$ defined by $\mathbf{g}$; and $1 \leq |\mathbf{g}| \leq d$.

$$p_i(\mathbf{x}|y) = p_i(\mathbf{x_g}|y) = \frac{|T_i(\mathbf{x_g}, y)|}{|\mathcal{D}_{i,y}|} \tag{17}$$

The regions $T_i(\mathbf{x})$ from different $\mathcal{D}_i$ are represented by different feature subsets. Hence, the multi-dimensional likelihood $p(\mathbf{x}|y)$ can be expressed as:

$$p(\mathbf{x}|y) = \frac{1}{t} \sum_{\mathbf{g} \in G_t} p(\mathbf{x_g}|y) \tag{18}$$

where G_t is a collection of t subsets of varying sizes of d attributes used to define $T_i(\mathbf{x})$ in each $\mathcal{D}_i$.

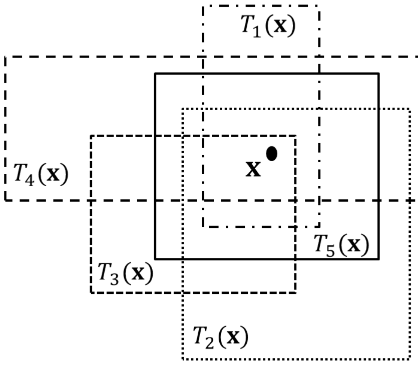


Fig. 2: Different regions from different $T_i(\cdot)$ $(i = 1, 2, \cdots, 5)$ that cover $\mathbf{x}$.

The average probability of t different regions $T_i(\mathbf{x})$ $(i = 1, 2, \cdots, t)$, constructed using different $\mathcal{D}_i$, provides a good estimate of $p(\mathbf{x}|y)$ as it considers the distribution in different local neighbourhoods of $\mathbf{x}$ in the data space. An illustrative example of different regions covering $\mathbf{x}$ is provided in Figure 2.

As our implementation is motivated from the multi-dimension mass estimation by Ting and Wells (2010), we call the resulting Bayesian classifier ***MassBayes***. MassBayes estimates $p(\mathbf{x}|y)$ by aggregating the multi-dimensional likelihoods estimated from random subsets of the training data using varying-size random feature subsets.

3.2.1 Implementation

We use the same algorithm as used by Ting and Wells (2010) to generate *h:d-trees* to represent $T_i(\cdot)$ with the following modification: the tree building process stops early once every instance is isolated. In the original implementation, each tree is built to the maximum height of $h \times d$ (where h is a parameter that defines the maximum level of sub-divisions) resulting in equal-size regions. Even in a moderate number of dimensions, many of the regions remain empty, and each non-empty region encloses a very small area around the observed data instances resulting in a poor estimate for unseen instances. The procedures to generate t trees from a given data set D are provided in Algorithms 1 and 2 in Appendix A.

Let the data space that envelops the instances in $\mathcal{D}$ be $\varDelta$. The data space $\varDelta$ is adjusted to become δ using a random perturbation conducted as follows. For each dimension j, a split point v_j is chosen randomly within the range $max_j(\varDelta) - min_j(\varDelta)$. Then, the new range δ_j along dimension j is defined as $[v_j - r, v_j + r]$, where $r = max(v_j - min_j(\varDelta), max_j(\varDelta) - v_j)$. The new range on all dimensions defines the adjusted work space for the tree building process. The random adjustment of the work space ensures that no two trees are identical even if they are constructed from the same subset of instances and covers a wider area than $\varDelta$.

The dimension to split is selected from a randomised set of d dimensions in a round-robin manner at each level of a tree. The maximum allowed height of a tree is $h \times d$. At the leaf node, the number of instances belonging to each class is stored.

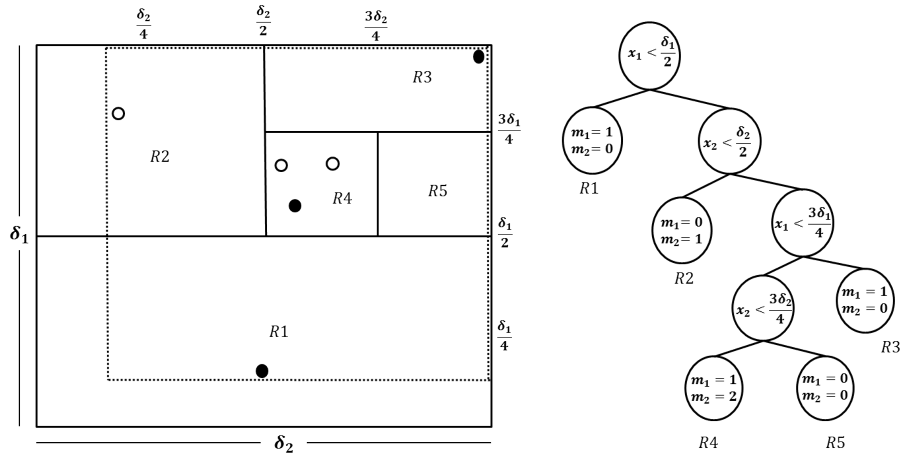


Fig. 3: An example of an $h\!:\!d$*–tree* for $h = 2$ and $d = 2$.

A typical example of an implementation of $T(\cdot)$ as an *h:d-tree* for $h = 2$ and $d = 2$ is shown in Figure 3. The dotted box envelopes the instances in $\mathcal{D}$ and the outer solid box represents the adjusted work space which has ranges δ_1 and δ_2 on x_1 and x_2 dimensions. $R1, R2, R3, R4$ and $R5$ represent different regions in $T(\cdot)$ depending on the data distribution in $\mathcal{D}$. $R1$ is defined by splitting the work space in x_1 dimension only (i.e., $\mathbf{g} = \{1\}$), whereas the other four regions use dimensions x_1 and x_2 (i.e., $\mathbf{g} = \{1, 2\}$).

In the worst case, an *h:d-tree* has height *hd*. At each level of a tree, a total of ψ instances are assigned to all nodes. Hence, the time complexity to construct a single tree is $O(hd\psi)$. Since t such trees are constructed, the training time complexity is $O(thd\psi)$. When classifying a test instance, each of the t trees is traversed from the root to a leaf node where the test instance falls into. The complexity of that searching is $O(thd)$. Each leaf node represents a region having a value range (with the minimum and the maximum value) in each of the d dimensions and the number of instances belonging to each of the c classes. There is a total of $min(2^{hd}, \psi)$ leaf nodes in each of the t trees. In general, $\psi < 2^{hd}$; thus, the total space complexity is $O(t(c + d)\psi)$. During the tree building process, an additional space of nd is required to store n training instances which can be discarded once the trees are constructed.

3.3 Proposed Bayesian classifiers versus existing Bayesian classifier

The decision rules of existing and the proposed Bayesian classifiers are provided in Table 2. Note that the existing Bayesian classifiers (NB, BayesNet and AηDE) estimate the conditional probability as the product of one-dimensional likelihoods assuming some kind of conditional independence. In contrast, DEMassBayes, MassBayes and ENNBayes estimate multi-dimensional likelihoods directly.

The time and space complexities of two variants of NB, AηDE, DEMassBayes, ENNBayes and MassBayes are presented in Table 3. Note that these complexi-

Table 2: Decision rules of different Bayesian classifiers.

Classifier	Decision Rule	Remarks
NB-KDE	$\arg\max_{y} \; p(y) \prod_{i=1}^{d} p(x_i\|y)$	$p(x_i\|y)$ is estimated with KDE (Gaussian kernel).
NB-Disc		$p(x_i\|y)$ is estimated through discretisation.
BayesNet	$\arg\max_{y} \; p(\pi_y, y) \prod_{i=1}^{d} p(x_i\|\pi_i, y)$	$\pi_i = parent(x_i)$, $\pi_y = parent(y)$; Probabilities are estimated through discretisation.
AηDE	$\arg\max_{y} \sum_{\mathbf{s}\in S\eta} p(\mathbf{x_s}, y) \prod_{j\in\{1,2,\cdots,d\}\backslash\mathbf{s}} p(x_j\|\mathbf{x_s}, y)$	$p(x_j\|\mathbf{x_s}, y)$ is estimated through discretisation.
DEMassBayes	$\arg\max_{y} \; p(y) \; \frac{1}{t} \sum_{i=1}^{t} f_i(\mathbf{x}\|y)$	$f_i(\mathbf{x}\|y)$ is estimated using Equation 11.
ENNBayes		$f_i(\mathbf{x}\|y)$ is estimated using Equation 15.
MassBayes	$\arg\max_{y} \; p(y) \; \frac{1}{t} \sum_{\mathbf{g}\in G_t} p(\mathbf{x_g}\|y)$	$p(\mathbf{x_g}\|y)$ is estimated using Equation 17.

Table 3: Time and space complexities of the existing (NB and AηDE) and the proposed (MassBayes and ENNBayes) Bayesian classifiers.

Classifiers	Training time	Testing time	Space
NB-KDE*	$O(nd)$	$O(cmd)$	$O(cmd)$
NB-Disc†	$O(nd)$	$O(cd)$	$O(cdv)$
AηDE‡	$O\left(n\binom{d}{\eta+1}\right)$	$O\left(cd\binom{d}{\eta}\right)$	$O\left(c\binom{d}{\eta+1}v^{\eta+1}\right)$
DEMassBayes*	$O(cthd\varphi)$	$O(cthd)$	$O(ctd\varphi)$
MassBayes	$O(thd\psi)$	$O(thd)$	$O\left(t\psi(d+c)\right)$
ENNBayes	-	$O(ct\varphi d)$	$O(t\psi d)$

* Langley and John (1995), † Webb et al. (2005), ‡ Webb et al. (2012), * Ting et al. (2013)
n: total number of training instances, m: average number of training instances in a class, d: number of dimensions, c: number of classes, v: average number of discrete values of an attribute, η: number of super-parents, t: number of trees, h: level of divisions ψ: sample size $|\mathcal{D}_i|$, and φ: average number of samples per class in $\mathcal{D}_i$.

ties do not include the additional discretisation needed in the preprocessing for NB-Disc and AηDE. Both training time complexity and space complexity of DEMassBayes, ENNBayes and MassBayes are independent of n.

Note that the average case training and testing time complexities of DEMassBayes and MassBayes are a lot better than the worst case complexities presented in Table 3 because the tree building process terminates early once every instance is isolated. With moderate values of d and h, the average height of the tree will be a lot less than the maximum height of hd. Hence, DEMassBayes and MassBayes run faster than ENNBayes.

4 Empirical evaluation

This section presents the results of the experiments conducted to evaluate the performance of ENNBayes and MassBayes against well known Bayesian classifiers. Since our focus is in the Bayesian approach of classifier learning, we chose the state-of-the-art Bayesian contenders: three variants of AηDE (A1DE, A2DE, A3DE), BayesNet, two variants of NB (NB-KDE and NB-Disc) and DEMassBayes. Webb et al. (2012) showed that A2DE is better or at least competitive to state-of-the-art ensemble approaches of Feating (Ting et al., 2011) and Random Forest (Breiman, 2001).

ENNBayes and MassBayes were implemented in Java using the WEKA platform (Hall et al., 2009) which also has implementations of NB, BayesNet and A1DE. For DEMassBayes, A2DE and A3DE, we used the WEKA implementations provided by the respective authors.

Fifteen data sets with $n > 10000$ were used. All the attributes in the data sets were numeric. The properties of the data sets are provided in Table 4. The RingCurve, Wave and OneBig data sets were three synthetic data sets and the rest were real-world data sets from UCI Machine Learning Repository (Frank and Asuncion, 2010). RingCurve and Wave are subsets of the RingCurve-Wave-TriGaussian data set used by Ting and Wells (2010) and OneBig is the data set used by Nanopoulos et al. (2006).

Table 4: Properties of data sets used.

Data sets	Data size	Dimensions	Classes
KDDCup99	5209460	32	40
CoverType	581012	10	7
YearPrediction	515345	90	2
Census	299285	7	2
SkinSegment	245057	3	2
Localisation	164860	3	11
MiniBooNE	129596	50	2
OneBig	68000	20	10
Shuttle	58000	8	7
Letters	20000	16	26
RingCurve	20000	2	2
Wave	20000	2	2
Magic04	19020	10	2
GasSensor	13790	128	6
Pendigits	10992	16	10

All the experiments were conducted in a Linux machine with 2.27 GHz processor and 20 GB memory. The performance measures were the classification accuracy and the CPU runtime. In large data sets ($n > 100000$), the data sets were divided into two folds, one for training and the other one for testing. The classification accuracy on the test set and the total runtime (training and testing) were reported. In small data sets ($n < 100000$), a 10-fold cross validation was used and average accuracy and average runtime (training and testing) were reported.

A two-standard-error significance test was conducted to check whether the difference in accuracies of two competing classifiers was significant. A win or loss was counted if the difference was significant; otherwise, it was a draw.

For AηDE, BayesNet and NB-Disc, data sets were discretised by a supervised discretisation technique based on minimum entropy (Fayyad and Irani, 1995) as suggested by the authors of AηDE before building the classification models. The reported runtime did not include the additional discretisation time that was done in preprocessing.

For BayesNet, the parameter *'maximum number of parents'* was set to 100 to examine whether a large number of parents produces better results; and the parameter *'initialise as Naive Bayes'* was set to *'false'* to initialise an empty network structure. The default values were used for the rest of the parameters.

In MassBayes and DEMassBayes, the parameters t, ψ and h were set as default to 100, 5000 and 10, respectively; whereas the parameters t and ψ in ENNBayes were set as default to 25 and 5000, respectively. Euclidean distance (L^2-norm) was used to find the nearest neighbour in ENNBayes.

We discuss the experimental results in detail in the following three subsections. The results in terms of classification accuracy and runtime are analysed in Section 4.1 and 4.2, respectively. We analyse the sensitivity of the parameters in ENNBayes and MassBayes in Section 4.3.

4.1 Classification accuracy comparison

A summary of the comparison is presented in Table 5 which shows the win:loss:draw counts of MassBayes and ENNBayes against the other contenders based on the two-standard-error significance test. Both ENNBayes and MassBayes perform significantly better than NB, A1DE and DEMass-Bayes in the complete set of 15 data sets. BayesNet, A2DE and A3DE cannot complete in all 15 data sets. Both ENNBayes and MassBayes still have more wins than losses in comparison with these classifiers. The complete result of ENNBayes, MassBayes and other contenders are provided in Tables 7 in Appendix B.

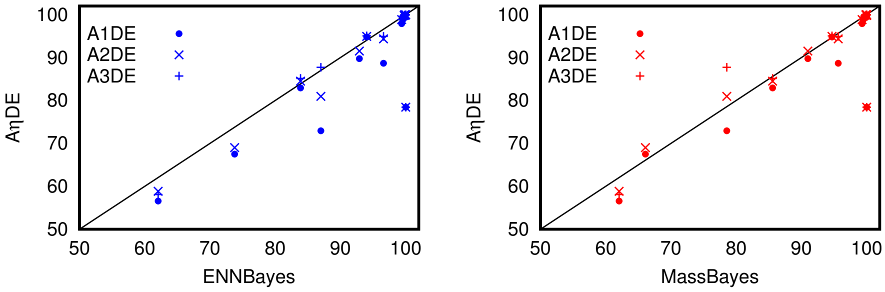
We discuss the classification results of ENNBayes and MassBayes against AηDE, BayesNet and NB, and DEMassBayes separately in the following three subsections.

Table 5: Win:Loss:Draw counts of MassBayes and ENNBayes against the other contenders in terms of accuracy based on the two-standard-error significance test. Note that A3DE and BayesNet did not complete in five data sets and A2DE did not complete in two data sets.

Contenders	MassBayes	ENNBayes
A3DE	5:3:2	5:4:1
A2DE	7:5:1	8:2:3
A1DE	10:1:4	12:1:2
BayesNet	6:3:1	6:2:2
NB-KDE	15:0:0	13:2:0
NB-Disc	15:0:0	14:1:0
DEMass-Bayes	10:0:5	7:1:7

4.1.1 ENNBayes and MassBayes versus AηDE

Since AηDE is the key contender, we analyse the result in more details in this subsection. The accuracies of ENNBayes and MassBayes in all the data sets were plotted against the accuracies of each of the three variants of AηDE in Figure 4. The coordinate values of each point in the plot are the accuracies of each pair of classifiers in a data set. If both the classifiers had produced the same accuracies in a data set, the point representing that data set lies on the diagonal. In both the plots, many points lies below the diagonal line and only a few points are above. This indicates that both the proposed methods are better than AηDE in many cases.



(a) ENNBayes versus AηDE

(b) MassBayes versus AηDE

Fig. 4: Scatter plot of accuracies of ENNBayes and MassBayes versus those of the three variants of AηDE (A1DE, A2DE and A3DE)

ENNBayes had 12 wins and 1 loss against A1DE; 8 wins and 2 losses against A2DE; and 5 wins and 4 losses against A3DE. Similarly, MassBayes had 10 wins and 1 loss against A1DE; 7 wins and 5 losses against A2DE; and 5 wins and 3 losses against A3DE.

It is interesting to note that A3DE did not complete in five out of the fifteen data sets used. In OneBig ($d = 20$), KDDCup ($d = 32$), MiniBooNE ($d = 50$), YearPrediction ($d = 92$) and GasSensor ($d = 128$), it did not complete because of the arithmetic overflow. The memory requirement in AηDE increases with d and c as it requires (η+2)-dimensional probability table to store the observed frequency for each combination of (η+1) attribute values and the class values (Webb et al., 2012). Table 6 shows the increase in memory with the increase in the number of dimensions and classes in three variants of AηDE. A3DE can only be used in low dimensional data sets ($d < 20$). It can not handle data sets even with a moderate number of dimensions. Similarly, A2DE did not complete in KDDCup ($d = 32$, $c = 40$) and GasSensor ($d = 128$, $c = 6$). A2DE can handle more dimensions than A3DE but still fails in data sets with a moderate number of dimensions and classes. A1DE, which can handle more attributes than A2DE, completed in all data sets.

Table 6: Memory required by three variants of AηDE in different data sets. v is the average number of discrete values per attribute.

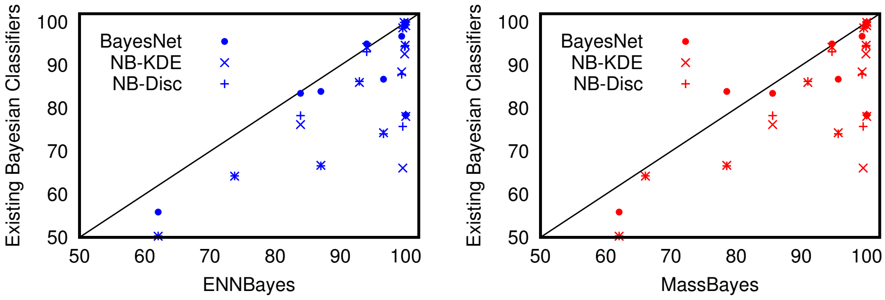
Data sets	#d	#c	Memory per class			Total memory		
			A1DE	A2DE	A3DE	A1DE	A2DE	A3DE
Census	7	2	$21v^2$	$35v^3$	$35v^4$	$42v^2$	$70v^3$	$70v^4$
Shuttle	8	7	$28v^2$	$56v^3$	$70v^4$	$196v^2$	$392v^3$	$490v^4$
CoverType	10	7	$45v^2$	$120v^3$	$210v^4$	$315v^2$	$840v^3$	$1470v^4$
Letters	16	26	$120v^2$	$560v^3$	$1820v^4$	$3120v^2$	$14560v^3$	$47320v^4$
OneBig	20	10	$190v^2$	$1140v^3$	$4845v^4$	$1900v^2$	$11400v^3$	$48450v^4$
KDDCup99	32	40	$496v^2$	$4960v^3$	$35960v^4$	$19840v^2$	$198400v^3$	$1438400v^4$
MiniBooNE	50	2	$1225v^2$	$19600v^3$	$230300v^4$	$2450v^2$	$39200v^3$	$460600v^4$
YearPrediction	90	2	$4005v^2$	$117480v^3$	$2555190v^4$	$1080v^2$	$234960v^3$	$5110380v^4$
GasSensor	128	6	$8128v^2$	$341376v^3$	$10668000v^4$	$48768v^2$	$2048256v^3$	$64008000v^4$

4.1.2 ENNBayes and MassBayes versus BayesNet and Naive Bayes

We focus on the comparison with the second key contenders BayesNet and Naive Bayes in this subsection.

In the scatter plots in Figure 5, most of the points are below the diagonal line. This shows that ENNBayes and MassBayes are better than BayesNet and Naive Bayes. ENNBayes had 6 wins and 2 losses against BayesNet; 13 wins and 2 losses against NB-KDE; and 14 wins and 1 loss against NB-Disc. Similarly, MassBayes had 6 wins and 3 losses against BayesNet; and all 15 wins over both the variants of NB (NB-KDE and NB-Disc).

Note that BayesNet did not complete in five data sets with a moderate number of dimensions - OneBig ($d = 20$), KDDCup99 ($d = 32$), MiniBooNE ($d = 50$), YearPrediction ($d = 90$) and GasSensor ($d = 128$) with out of memory error when a computing cluster node with 20GB was used. But, ENNBayes and MassBayes completed in all 15 data sets using the same machine.



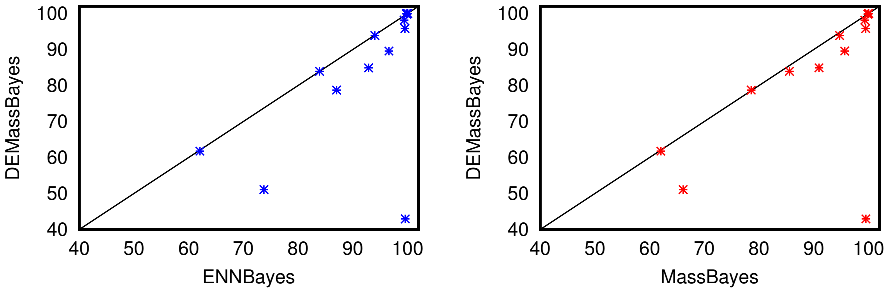
(a) ENNBayes versus BayesNet and NB (b) MassBayes versus BayesNet and NB

Fig. 5: Scatter plot of accuracies of ENNBayes and MassBayes versus those of BayesNet and the two variants of Naive Bayes (NB-KDE and NB-Disc).

4.1.3 MassBayes and ENNBayes versus DEMassBayes

This subsection focuses on comparison with DEMassBayes, a variant of the same generic ensemble approach.

As shown in the scatter plots in Figure 6, both ENNBayes and MassBayes produced better classification accuracies than DEMassBayes. MassBayes had 10 wins and no loss against DEMassBayes whereas ENNBayes had 7 wins and one loss. DEMassBayes produced competitive results to MassBayes and ENNBayes in low dimensional data sets such as RingCurve (d = 2), Wave (d = 2), Skin-Segment (d = 3), Localisation (d = 3), Census (d = 7) and Shuttle (d = 8). It had significantly worse results than both MassBayes and ENNBayes in data sets with a moderate number of dimensions such as KDDCup99 (d = 32), MiniBooNE (d = 50), YearPrediction (d = 90) and GasSensor (d = 128). When the number of dimensions increases, the volume of the regions decreases exponential which heavily influence the density estimation and degrades the performance of DEMassBayes.



(a) ENNBayes versus DEMassBayes (b) MassBayes versus DEMassBayes

Fig. 6: Scatter plot of accuracies of ENNBayes and MassBayes versus those of DEMassBayes.

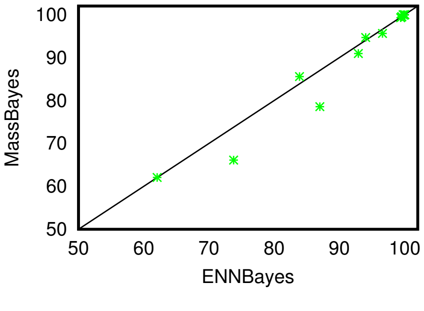


Fig. 7: Scatter plot of accuracies of ENNBayes versus MassBayes.

4.1.4 ENNBayes versus MassBayes

As shown in Figure 7, ENNBayes and MassBayes produced competitive results in many data sets. ENNBayes had 5 wins, 2 losses and 8 draws against MassBayes.

4.2 Runtime comparison

4.2.1 ENNBayes and MassBayes versus existing Bayesian classifiers

In terms of runtime, ENNBayes was generally one to three orders of magnitude slower than the existing Bayesian classifiers. But, the runtime of MassBayes was an order of magnitude faster than some contenders (such as A2DE and NB-KDE) in large data sets and it was competitive to many existing Bayesian classifiers in many data sets.

The runtime of the proposed and the existing Bayesian classifiers in the three largest data sets - KDDCup99, YearPrediction and CoverType was presented in Figure 8. In the largest data set KDDCup99, the runtime of MassBayes was of the same order of magnitude as DEMassBayes, A1DE and NB-KDE whereas it was an order of magnitude slower than NB-Disc. A2DE, A3DE and BayesNet did not complete in KDDCup99. In YearPrediction, MassBayes was an order of magnitude faster than A2DE and NB-KDE; and was of the same order of magnitude as DEMassBayes and A1DE. A3DE and BayesNet did not complete in the YearPrediction data set. In CoverType, the runtime of MassBayes was of the same order of magnitude as DEMassBayes, A3DE, BayesNet and NB-KDE, but it was an order of magnitude slower than A2DE, A1DE and NB-Disc. ENNBayes was one to three orders of magnitude slower than the existing Bayesian classifiers in all three data sets.

The complete runtime results of ENNBayes, MassBayes and other contenders are provided in Tables 8 in Appendix B.

Note that the reported runtime results for AηDE, BayesNet and NB-Disc did not include the discretisation time that must be done as a preprocessing step, which give AηDE, BayesNet and NB-Disc an unfair advantage over the proposed classifiers. The discretisation cost is significantly large in large and moderately high dimensional data sets. For examples, the discretisation took 1290 seconds in the KddCup99 data set, and 467 seconds in the YearPrediction data set. The discretisation time itself was of the same order of magnitude as the runtime of MassBayes. The runtime of the supervised discretisation (Fayyad and Irani, 1995) in all the data sets are provided in Table 9 in Appendix C.

The head-to-head comparison of runtime of ENNBayes, MassBayes and DEMassBayes against the existing Bayesian classifiers (AηDE, BayesNet and NB) is not entirely fair. The existing Bayesian classifiers assume some kind of conditional independence and estimate the simplified surrogates of $p(\mathbf{x}|y)$. In contrast, ENNBayes, MassBayes and DEMassBayes estimate $p(\mathbf{x}|y)$ directly from the given training data.

Among the three classifiers based on the proposed ensemble approach, ENNBayes was one to three orders of magnitude slower than MassBayes and DEMassBayes. MassBayes and DEMassBayes had runtime of the same order of magnitude. Among the three variants of AηDE, the runtime increases with the increase in η.

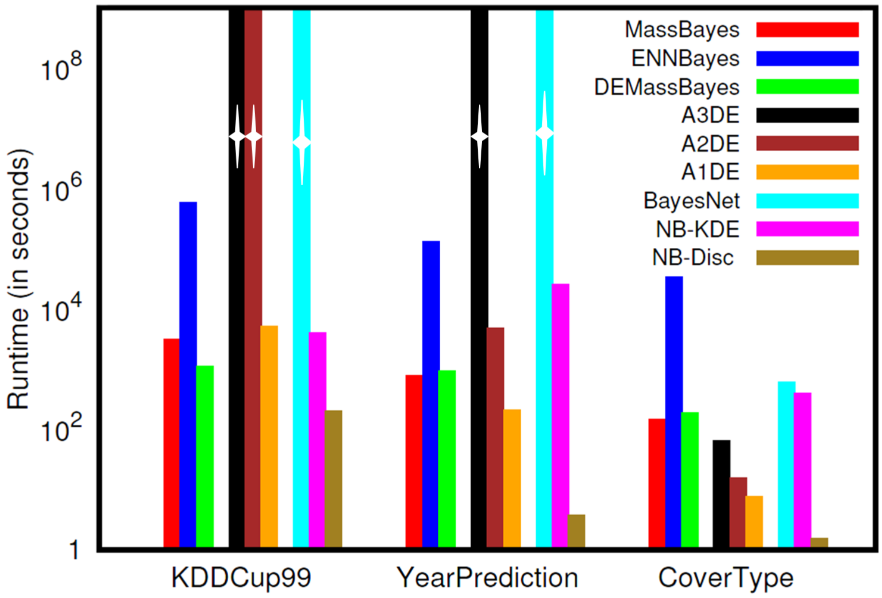


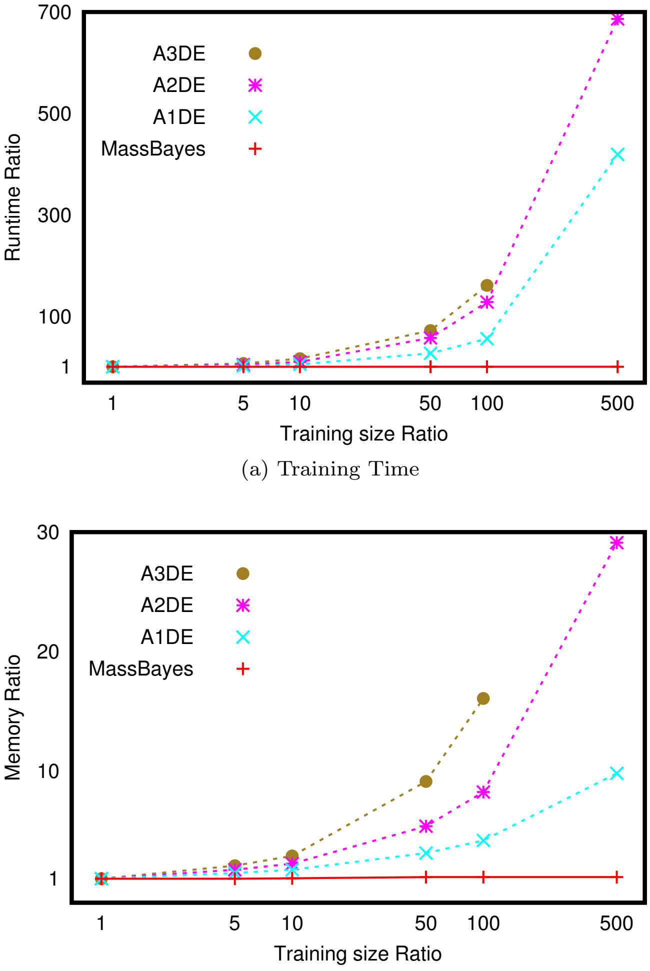
Fig. 8: Runtime of MassBayes, ENNBayes and the other contenders in the three largest data sets: KDDCup99, CoverType and YearPrediction. The vertical axis is on a logarithmic scale of base 10. For ease of reading, the classifiers are organised into groups of three: the first group has three classifiers (MassBayes, ENNBayes and DEMassBayes) based on the proposed ensemble approach; the second group has three variants of AηDE (A3DE, A2DE and A1DE); and the last group has BayesNet, NB-KDE and NB-Disc. Note that the discretisation time was not included in the runtime of AηDE, BayesNet and NB-Disc. Histograms with star which have the maximum height indicate that the classifiers did not complete the tasks.

The above runtime results do not provide a detailed picture about the scalability of the proposed classifiers. Hence, we examine the scalability of the proposed classifiers to present a more accurate idea about their time complexities in the following subsection.

4.2.2 Scaleup test

In order to examine how well the classifiers scaleup to large data sets, we used a subset of the KDDCup99 data set with the three largest classes ($d = 32, c = 3$)[1]. The training data size was increased from 10000 to 50000, 100000, half-a-million, million and five million with a factor of 1, 5, 10, 50, 100 and 500, respectively. The test set had 10000 instances. The increase in training time and space required to store the classification model of MassBayes and three variants of AηDE is presented

[1] The number of classes must be reduced for AηDE to run



(a) Training Time

(b) Memory requirement

Fig. 9: Increase in training time and memory requirement to learn a classification model with the increase in training size in a subset of KDDCup99 data set with three largest classes (i.e., $c = 3, d = 32$). The horizontal axes are on a logarithmic scale of base 10. A3DE did not complete when the training size was increased to five million.

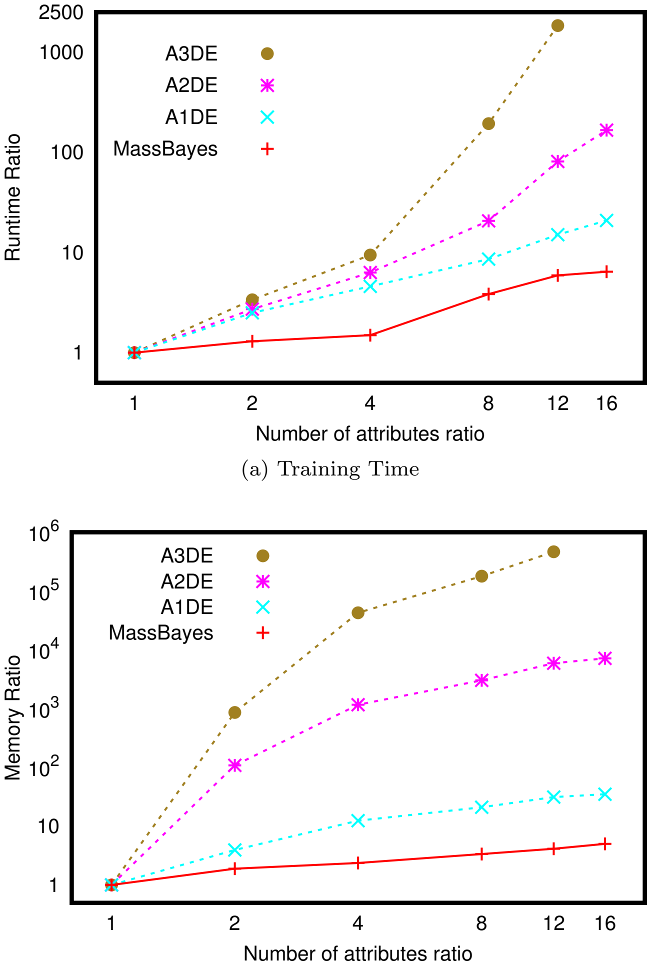
in Figure 9[2]. The increase in training time and space required is presented as a ratio with the training time and space required for 10000 training instances as the base. Note that AηDE had an unfair advantage over MassBayes in terms of runtime as the discretisation time was not included in the presented results.

With the increase in training size by a factor of 5, 10, 50, 100 and 500, the training time of A1DE increased by a factor of 3, 6, 28, 57 and 420 followed by

[2] The scaleup test was conducted against the key contender AηDE. ENNBayes was not included in the scaleup test as there is no training phase.

A2DE (5, 11, 58, 128 and 686) and A3DE (8, 18, 72 and 161). A3DE did not complete when the sample size was increased to five million instances by a factor of 500. The training time of MassBayes was constant, irrespectively of the training data size.

The memory requirement of A2DE was increased by a factor of 1.4, 1.7, 3, 4 and 10 followed by A2DE (1.7, 2.3, 5, 8 and 29) and A3DE (2.1, 3, 9 and 16). But, MassBayes had constant memory requirement.



(a) Training Time

(b) Memory requirement

Fig. 10: Increase in training time and memory requirement to learn a classification model with the increase in the number of dimensions in a subset of KDDCup99 data set with three largest classes (i.e., $c = 3, n = 5125369$). The horizontal and vertical axes in Figures (a) and (b) are on a logarithmic scale of base 2 and 10, respectively. A3DE did not complete when the number of dimensions was increased to 32.

When the training size was varied from 10000 instances to five million instances, all the classifiers produced similar classification accuracy. The classification accuracy of MassBayes varied from 99.96% to 99.99%, whereas that of A1DE and A2DE varied from 99.89% to 99.98% and 99.91% to 99.98%, respectively. This indicates that MassBayes scales better than the existing Bayesian classifiers in very large data sets both in terms of training time and memory requirement, and produce better classification accuracy.

In order to examine how well the classifiers scaleup to the increase in the number of attributes, we increased the number of attributes of the same subset of the KDDCup99 data set with the three largest classes ($n = 5125369, c = 3$) from 2 to 4, 8, 16, 24 and 32. These attributes were selected from the best attributes identified using Chi-squared attribute evaluation available in WEKA (Hall et al., 2009). The test set had 10000 instances. The increase in training time and space required to store the classification model of MassBayes and three variants of AηDE are presented in Figure 10. The bases for runtime ratio and memory ratio are the training time and memory required by the data set with 2 attributes.

With the increase in the number of attributes by a factor of 2, 4, 8, 12 and 16, MassBayes increased its training time by a factor of 1.3, 1.5, 3.8, 5.9 and 6.4, respectively. The closest contender A1DE increased its training time by a factor of 2.5, 4.6, 8.6, 15 and 21 followed by A2DE (2.7, 6.3, 21, 81 and 166) and A3DE (3.4, 9.4, 193 and 1830). A3DE did not complete when the number of attributes was increased to 32 by a factor of 16.

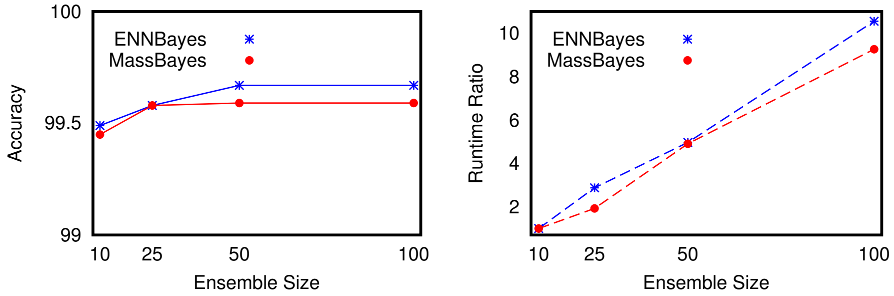
The memory requirement of MassBayes was increased by a factor of 1.9, 2.3, 3.4, 4.2 and 5, respectively. A1DE increased its memory requirement by a factor of 4, 13, 21, 31 and 35 followed by A2DE (109, 1161, 3052, 5952 and 7219) and A3DE (868, 42491, 179230 and 463939).

4.3 Role of parameters in ENNBayes and MassBayes

In this section, we present the results of a series of experiments conducted to investigate the sensitivity of parameters in ENNBayes and MassBayes. We conducted the following sets of experiments by varying one parameter and fixing the other parameters to the default values in the KDDCup99 data set.

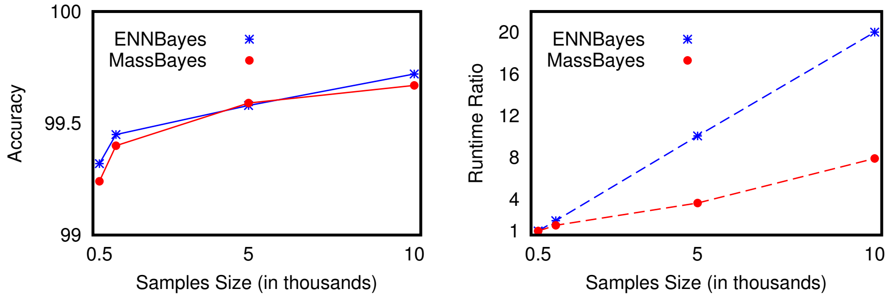
- ENNBayes
  1. Varying ensemble size (t) in the range of $[10, 25, 50, 100]$ and fixing $\psi = 5000$.
  2. Varying sample size (ψ) in the range of $[500, 1000, 5000, 10000]$ and fixing $t = 25$.
- MassBayes
  1. Varying ensemble size (t) in the range of $[10, 25, 50, 100]$ and fixing $\psi = 5000$ and $h = 10$.
  2. Varying sample size (ψ) in the range of $[500, 1000, 5000, 10000]$ and fixing $t = 100$ and $h = 10$.
  3. Varying height (h) in the range $[1, 5, 10, 15]$ and fixing $t = 100$ and $\psi = 5000$.

Figures 11 and 12 show the effect of parameters t and ψ in classification accuracy and runtime of ENNBayes and MassBayes and Figure 13 shows the effect of parameter h in classification accuracy and runtime of MassBayes.



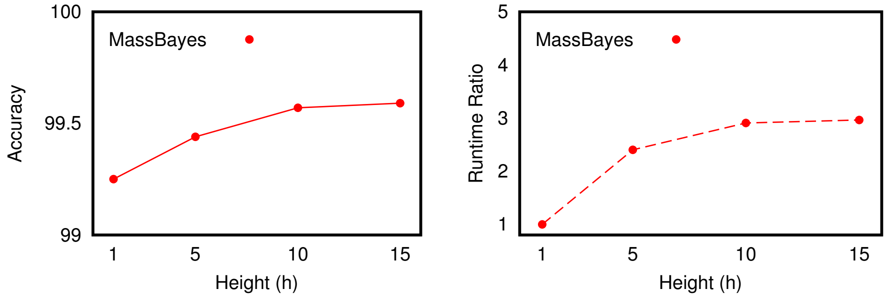
(a) Accuracy (b) Runtime Ratio

Fig. 11: The effect of ensemble size (t) on accuracy and runtime of ENNBayes and MassBayes.



(a) Accuracy (b) Runtime Ratio

Fig. 12: The effect of sample size (ψ) on accuracy and runtime of ENNBayes and MassBayes.



(a) Accuracy (b) Runtime Ratio

Fig. 13: The effect of height (h) on accuracy and runtime of MassBayes.

All the experiments were conducted using 10000 instances for testing and the rest for training. The increase in runtime was plotted as a ratio to show the fraction of runtime increased when the parameters were increased. The bases for the runtime ratio while varying t, ψ and h are the total runtime (including training and testing) for $t = 10$, $\psi = 500$ and $h = 1$, respectively.

When t was increased from 10 to 25, 50 and 100, the accuracies of both ENNBayes and MassBayes increased upto a certain point ($t = 50$) and then remained almost constant. Similarly, when the sample size was increased from 500 to 1000, 5000 and 10000, the accuracies of ENNBayes and MassBayes were increased rapidly initially upto $\psi = 5000$ and then increased gradually when ψ was increased to 10000. Similarly, as shown in Figure 13, the accuracy of MassBayes increased upto a certain point ($h = 10$) and then remained constant when h was increased.

The results show that the parameters are not too sensitive in terms of predictive accuracy if they are set to a sufficiently large value. But, setting the parameters to very large values will increase the runtime. The runtime varies linearly with t in both ENNBayes and MassBayes. With the increase in ψ, the runtime of ENNBayes and MassBayes varies linearly and sub-linearly, respectively. In MassBayes, if the sample size is enough, the increase in h will increase the runtime almost linearly. After reaching a certain height, the runtime remains constant as the tree building stops early once every instances is isolated.

5 Discussion

Between the two proposed classifiers, ENNBayes is computationally expensive due to the associated distance calculation. MassBayes is fast because the tree structure speed-up the search for local neighbourhood. In MassBayes, with moderate d and h, the trees are usually significantly shorter than the maximum height hd. The distinguishing characteristics between ENNBayes and MassBayes are summarised as follows:

1. ENNBayes estimates multi-dimensional likelihoods using all the features, whereas MassBayes estimates using feature subsets of different sizes.
2. ENNBayes is based on nearest neighbour and needs no training; MassBayes builds trees in the training stage.
3. ENNBayes estimates likelihoods using density estimation; MassBayes estimates likelihoods using probability estimation.

In ENNBayes, $f_i(\mathbf{x}|y)$ can also be estimated considering the volume in the feature space that includes the k nearest neighbours of $\mathbf{x}$ in $\mathcal{D}_{i,y}$, as Silverman (1986):

$$\hat{f}_i(\mathbf{x}|y) = \frac{|N(\mathbf{x}, k|\mathcal{D}_{i,y})|}{|\mathcal{D}_{i,y}| \times volume(N(\mathbf{x}, k|\mathcal{D}_{i,y}))} \tag{19}$$

This implementation does not provide a good estimate of $f_i(\mathbf{x}|y)$ because the estimate is heavily influenced by the volume which becomes very small, like in DEMassBayes, even in data sets with a moderate number of dimensions. In order to overcome this problem, we used the implementation based on distance as used by Breunig et al. (2000). We have used the Euclidean distance (L^p-*norm* with $p = 2$) in the experiment, but any value of p can be used.

It is interesting to note that we are not the first to employ nearest neighbour only to do density estimation. Naive Bayes Nearest Neighbour (NBNN) (Boiman et al., 2008) and local NBNN (McCann and Lowe, 2012) employ nearest neighbour only in a kernel estimator to estimate the one-dimensional likelihoods in the Naive Bayesian framework. In contrast, ENNBayes employs the nearest neighbour only in a k-nearest neighbour density estimator to estimate the multi-dimensional likelihoods in the more general Bayesian framework.

One obvious question with ENNBayes is: what if an ordinary kNN density estimator is used instead of an ensemble? In large data sets, it is impossible to run because of its high time complexity. For example, when kNN with $k = 1$ was used to estimate $p(\mathbf{x}|y)$ directly from D, it could not complete in the largest data set (KDDCup99) in 20 days and it is estimated to need 140 days to complete. But ENNBayes completed the task in less than seven days. With $\psi = 5000$ and $t = 25$, ENNBayes uses only 125000 training instances, 20 times less instances than the entire training size of 2.5 million instances and runs 20 times faster.

ENNBayes is the first lazy Bayesian classifier that estimates $p_i(\mathbf{x}|y)$ directly through the nearest neighbour density estimation. The existing lazy Bayesian classifiers, LBR (Zheng and Webb, 2000) and LWNB (Frank et al., 2003), estimate one-dimensional likelihoods and uses NB. LBR needs to convert continuous-valued attributes to categorical attributes and estimates $p(x_j|y)$ $(j = 1, 2, \cdots, d)$ through probability estimation in a local region defined by a conjunctive rule covering x. Similarly, LWNB estimates $p(x_j|y)$ through (i) a probability estimation in a local region covered by k nearest neighbours if x_j is a discrete attribute or (ii) through density estimation assuming Gaussian distribution if x_j is a continuous-valued attribute.

In ENNBayes, the nearest neighbour search can be improved by using indexing schemes such as Cover Trees (Beygelzimer et al., 2006), M-Trees (Ciaccia et al., 1997) and Projection-Indexed Nearest Neighbours (PINN) (Vries et al., 2012).

Due to the tree-based implementation, one may view MassBayes as an ensemble of decision trees like Bagging (Breiman, 1996), Random Forest (Breiman, 2001) and Random Trees (Liu, 2005). However, the trees are not decision trees because the tree building process uses neither class information nor any other evaluation criteria. The purpose of feature space partitioning in MassBayes is different from the one in decision trees. In MassBayes, the objective is to define local regions around the observed instances in order to estimate the likelihood $p(\mathbf{x}|y)$. In decision trees, the objective is to separate the classes as good as possible in order to estimate the class membership probability $p(y|\mathbf{x})$.

In terms of implementation, DEMassBayes and MassBayes are similar. The only different is MassBayes constructs t trees with ψ instances whereas DEMassBayes constructs t trees per class with a fewer number of instances (in average $\frac{\psi}{c}$).

The performance of both ENNBayes and MassBayes will be affected by the presence of irrelevant attributes. Irrelevant attributes make the distribution sparse in the feature space and affect the nearest neighbour search in ENNBayes and feature space partitioning in MassBayes. Irrelevant attributes affect other Bayesian classifiers in a similar way. The easiest solution is to conduct feature selection before building a classification model.

Density estimation tree (Ram and Gray, 2011), that estimates $f(\mathbf{x})$ using a decision tree, eliminates irrelevant attributes implicitly as they are never selected to split the data. But, it requires an expensive search to select an attribute and its cut-point to partition the data space, which makes it inapplicable to large data sets due to its high time and space complexities. Also, in high dimensional problems, the density estimation is heavily influenced by the volume of the leaf node as it becomes very small as in DEMassBayes.

LiNearN (Wells et al., 2012) is an ensemble approach to estimate $f(\mathbf{x})$ that overcomes the issues associated with DEMass. It defines local hypercube regions from a sub-sample of instances based on nearest neighbour defined by L^{∞}-*norm* and estimates mass in the regions using another sub-sample of instances. It is more adaptive than DEMass as it constructs regions with different volumes based on the data distribution. LiNearN produced good results in unsupervised learning tasks of clustering and anomaly detection. But, when it was used in Bayesian classifier to estimate $p(\mathbf{x}|y)$, the classification accuracy was not as impressive as ENNBayes and MassBayes. It is because many unseen test instances fall outside the hypercube regions resulting in zero mass and zero density.

The proposed ensemble approach to estimate the multi-dimensional likelihoods from sub-samples yields constant training time and space complexities. It is ideal for big data sets and data streams. It also allows user to trade-off between the prediction accuracy and the computational cost as required. The computational cost (time and space) can be reduced by setting lower values for the parameters t and ψ if a higher misclassification rate can be tolerated. Setting the parameters to higher values improves the predictive accuracy at the expense of increasing computational cost.

6 Conclusions

Existing Bayesian classifiers have been designed using one-dimensional likelihoods, based on the premise that multi-dimensional likelihoods are difficult to estimate directly from data. We take a fresh re-examination on this premise and find that there is a simple way to estimate multi-dimensional likelihoods directly from data using an ensemble approach.

This paper presents a generic ensemble approach of estimating multi-dimensional likelihood in Bayesian classification learning using random sub-samples of the training data. We show that:

1. With a small sub-sample of data, multi-dimensional likelihood can be estimated easily using existing data modelling techniques.
2. Aggregating estimations from multiple models provides a good estimate of $p(\mathbf{x}|y)$ as it considers different local neighbourhoods of $\mathbf{x}$.
3. The proposed ensemble approach reduces the training time and space complexities to constant with respect to the number of training instances.

Using the generic ensemble approach, we introduce two new Bayesian classifiers called *ENNBayes* and *MassBayes* using a nearest neighbour density estimator and a probability estimator, respectively, to estimate the multi-dimensional likelihoods. Our empirical evaluation shows that both ENNBayes and MassBayes produced better classification accuracy than the state-of-the-art Bayesian classifiers. Both the classifiers are efficient in terms of runtime and memory requirement, and they scale better than the existing Bayesian classifiers in very large data sets.

Between ENNBayes and MassBayes, ENNBayes has slightly better predictive accuracy in some data sets than MassBayes but it is one to three orders of magnitude slower than MassBayes due to the need to do nearest neighbour search.

Appendix

A Implementation of MassBayes

Algorithm 1 : BuildTrees(D, t, ψ, h)

Inputs: D - input data, t - number of trees, ψ - sub-sampling size, h - level of divisions.
Output: F - a set of t *h:d-trees*

```
$H \leftarrow h \times d$ {Maximum height of a tree}
Initialize $F$
for $i = 1$ to $t$ do
    $\mathcal{D} \leftarrow sample(D, \psi)$ {strictly without replacement}
    $(min,max) \leftarrow$ InitialiseWorkSpace($\mathcal{D}$)
    $A \leftarrow$ {Randomised list of $d$ attributes.}
    $F \leftarrow F \cup$ SingleTree($\mathcal{D}, min, max, 0, A$)
end for
return $F$
```

Algorithm 2 : SingleTree$(\mathcal{D}, min, max, \ell, A)$

Inputs: $\mathcal{D}$ - input data, *min* & *max* - arrays of minimum and maximum values for each attribute that define a work space, A - a randomised list of d attributes, ℓ - current height level.
Output: an *h:d-tree*

```
Initialize $Node(\cdot)$
while ($\ell < H$ and $|\mathcal{D}| > 1$) do
    $q \leftarrow nextAttribute(A, \ell)$ {Retrieve an attribute from $A$ based on height level.}
    $mid_q \leftarrow (max_q + min_q)/2$
    $\mathcal{D}_l \leftarrow filter(\mathcal{D}, q < mid_p)$
    $\mathcal{D}_r \leftarrow filter(\mathcal{D}, q \geq mid_q)$
    if ($|\mathcal{D}_l| = 0$ ) or ($|\mathcal{D}_r| = 0$) then {Reduce range for single-branch node.}
        if ($|\mathcal{D}_l| > 0$ ) then $max_q \leftarrow mid_q$
        else $min_q \leftarrow mid_q$
        end if
        $\ell \leftarrow \ell + 1$
        continue at the start of while loop
    end if
    {Build two nodes: $Left$ and $Right$ as a result of a split into two half-spaces.}
    $temp \leftarrow max_q$; $max_q \leftarrow mid_q$
    $Left \leftarrow$ SingleTree($\mathcal{D}_l, min, max, \ell + 1, A$)
    $max_q \leftarrow temp$; $min_q \leftarrow mid_q$
    $Right \leftarrow$ SingleTree($\mathcal{D}_r, min, max, \ell + 1, A$)
    terminate while loop
end while
$classCount \leftarrow updateClassCount(\mathcal{D})$
return $Node(Left, Right, SplitAtt \leftarrow q, SplitValue \leftarrow mid_q, classCount)$
```

B Detailed Results for experiments described in Sections 4.1 and 4.2.

Table 7: Classification accuracy (%) of different Bayesian classifiers.

Data sets	MassBayes	ENNBayes	DEMass-Bayes	A3DE	A2DE	A1DE	BayesNet	NB-KDE	NB-Disc
KDDCup99	99.53	99.55	42.96	N/A	N/A	99.52	N/A	98.73	98.62
CoverType	78.54	87.01	78.72	87.69	80.98	72.94	83.90	66.72	66.62
YearPrediction	66.10	73.77	51.11	N/A	69.04	67.48	N/A	64.22	64.25
Census	94.70	94.01	93.88	95.03	94.93	94.78	95.01	94.24	93.16
SkinSegment	99.93	99.92	99.92	99.85	99.85	99.37	99.47	94.64	94.62
Localisation	62.06	62.04	61.78	58.08	58.86	56.59	55.89	50.28	50.33
MiniBooNE	90.97	92.89	84.91	N/A	91.51	89.75	N/A	86.02	86.28
OneBig	100.00	99.77	99.98	N/A	99.80	99.69	N/A	99.98	99.97
Shuttle	99.89	99.85	99.84	99.94	99.95	99.86	99.93	92.66	94.37
Letters	95.65	96.59	89.59	94.95	94.38	88.67	86.78	74.31	74.08
RingCurve	100.00	100.00	100.00	99.99	99.99	99.99	99.99	99.25	99.43
Wave	100.00	100.00	99.99	78.43	78.43	78.43	78.43	78.09	78.43
Magic04	85.57	83.89	83.89	85.15	84.50	82.94	83.51	76.20	78.30
GasSensor	99.45	99.54	95.85	N/A	N/A	98.80	N/A	66.11	75.75
Pendigits	99.30	99.37	98.19	98.83	98.84	97.90	96.77	88.46	87.89

Table 8: Runtime (seconds). Note that discretisation time was not included in the runtime of AηDE, BayesNet and NB-Disc.

Data sets	MassBayes	ENNBayes	DEMass-Bayes	A3DE	A2DE	A1DE	BayesNet	NB-KDE	NB-Disc
KDDCup99	3093	572288	1098	N/A	N/A	5084	N/A	3935	197
CoverType	144	33549	181	64.2	15.5	7.4	590	389	1.5
YearPrediction	785	129580	929	N/A	4757	204	N/A	24603	3.7
Census	73.8	12667	68.5	2.3	1.1	0.52	3.1	7.7	0.34
SkinSegment	27.1	7865	22.1	0.44	0.54	0.22	2.0	6.1	0.27
Localisation	35.2	4569	66.4	0.77	0.61	0.53	2.5	847	0.52
MiniBooNE	128	19967	95.6	N/A	259	14.4	N/A	3048	0.65
OneBig	17.5	1142	20.6	N/A	11.0	1.6	N/A	231	0.15
Shuttle	10.6	506	6.7	1.4	0.29	0.12	7.1	1.2	0.07
Letters	9.3	240	9.1	12.1	1.6	0.52	24.0	2.0	0.07
RingCurve	3.5	78.6	2.8	0.02	0.02	0.01	0.04	2.0	0.02
Wave	3.4	91.7	2.4	0.03	0.02	0.13	0.03	2.0	0.02
Magic04	6.7	178	4.4	0.32	0.08	0.03	0.61	8.4	0.02
GasSensor	105	828	46.7	N/A	N/A	19.8	N/A	111	0.14
Pendigits	7.6	112	4.2	5.4	0.59	0.11	28.4	1.4	0.03

C Discretisation Time

Table 9: Total runtime for supervised discretisation (Fayyad and Irani, 1995).

Data sets	#n	#d	#c	Disc. time (s)
KDDCup99	5209460	32	40	1290
CoverType	581012	10	7	98
YearPrediction	515345	90	2	467
Census	299285	7	2	35
SkinSegment	245057	3	2	7
Localisation	164860	3	11	9
MiniBooNE	129596	50	2	100
OneBig	68000	20	10	15
Shuttle	58000	8	7	4
LetterRecognition	20000	16	26	3
RingCurve	20000	2	2	1
Wave	20000	2	2	1
Magic04	19020	10	2	3
GasSensor	13790	128	6	17
Pendigits	10992	16	10	2

References

Beygelzimer, A., Kakade, S. and Langford, J. (2006). Cover trees for nearest neighbor, *In Proceedings of the 23rd International Conference on Machine Learning*, pp. 97–104.

Boiman, O., Shechtman, E. and Irani, M. (2008). In defense of nearest-neighbor based image classification, *In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–8.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**(2): 123–140.

Breiman, L. (2001). Random forests, *Machine Learning* **45**: 5–32.

Breunig, M. M., Kriegel, H.-P., Ng, R. T. and Sander, J. (2000). LOF: Identifying Density-Based Local Outliers, *In Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 93–104.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes, *In Proceedings of the European Working Session on Learning, Porto, Portugal*, pp. 164–178.

Chickering, D. M. (1996). Learning Bayesian Networks is NP-Complete, *in* D. Fisher and H.-J. Lenz. (eds), *Learning from Data: Artificial Intelligence and Statistics V*, Springer-Verlag/ Heidelberg, pp. 121–130.

Ciaccia, P., Patella, M. and Zezula, P. (1997). M-tree: An Efficient Access Method for Similarity Search in Metric Spaces, *In Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 426–435.

Clark, P. and Niblett, T. (1989). The CN2 Induction Algorithm, *Machine Learning*, pp. 261–283.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss, *Machine Learning* **29**(2-3): 103–130.

Dougherty, J., Kohavi, R. and Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features, *In Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, pp. 194–202.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*, New York: Wiley.

Fayyad, U. M. and Irani, K. B. (1995). Multi-interval discretization of continuous valued attributes for classification learning, *In Proceedings of 14th International Joint Conference on Artificial Intelligence*, pp. 1034–1040.

Frank, A. and Asuncion, A. (2010). UCI Machine Learning Repository, http://archive.ics.uci.edu/ml. University of California, Irvine, School of Information and Computer Sciences.

Frank, E., Hall, M. and Pfahringer, B. (2003). Locally weighted naive bayes, *In Proceedings of the Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 249–256.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm, *In Proceedings of the Thirteenth International Conference on Machine Learning*, Morgan Kaufmann, San Francisco, pp. 148–156.

Friedman, N., Geiger, D. and Goldszmidt, M. (1997). Bayesian Network Classifiers, *Machine Learning* **29**: 131–163.

Grossman, D. and Domingos, P. (2004). Learning Bayesian network classifiers by maximizing conditional likelihood, *In Proceedings of the 21st International Conference on Machine Learning,*, ACM Press, pp. 361–368.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I. H. (2009). The weka data mining software: an update, *SIGKDD Explorations Newsletter* **11**(1): 10–18.

Jiang, L., Wang, D., Cai, Z. and Yan, X. (2007). Survey of improving naive bayes for classification, *in* R. Alhajj, H. Gao, X. Li, J. Li and O. Zaane (eds), *Advanced Data Mining and Applications*, Vol. 4632 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 134–145.

Keogh, E. and Pazzani, M. (1999). Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches, *In Proceedings of the seventh International Workshop on Artificial Intelligence and Statistics*, pp. 225–230.

Kohavi, R. (1996). Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid, *In Proceedings of the second International Conference on Knowledge Discovery and Data mining*, AAAI Press, pp. 202–207.

Kononenko, I. (1991). Semi-naive bayesian classifier, *In Proceedings of the European Working Session on Learning*, Springer-Verlag New York, pp. 206–219.

Kononenko, I. (1993). Inductive and bayesian learning in medical diagnosis, *Applied Artificial Intelligence* **7**: 317–337.

Langley, P., Iba, W. and Thompson, K. (1992). An Analysis of Bayesian Classifiers, *In Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 399–406.

Langley, P. and John, G. H. (1995). Estimating continuous distribution in Bayesian classifiers, *In Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345.

Langley, P. and Sage, S. (1994). Induction of selective bayesian classifiers, *In Proceedings of the 10th conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, pp. 399–406.

Liu, F. T. (2005). *The Utility of Randomness in Decision Tree Ensembles*, Master's thesis, Gippsland School of Information Technology, Faculty of Information Technology, Monash University, Australia.

McCann, S. and Lowe, D. G. (2012). Local Naive Bayes Nearest Neighbor for Image Classification, *In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3650–3656.

Nanopoulos, A., Theodoridis, Y. and Manolopoulos, Y. (2006). Indexed-based density biased sampling for clustering applications, *IEEE Transaction on Data and Knowledge Engineering* **57**(1): 37–63.

Ram, P. and Gray, A. G. (2011). Density estimation trees, *In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, ACM, New York, NY, USA, pp. 627–635.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*, Chapmal & Hall/CRC.

Tan, P.-N., Steinbach, M. and Kumar, V. (2006). *Introduction to Data Mining*, Addison-Wesley.

Ting, K. M. and Wells, J. R. (2010). Multi-Dimensional Mass Estimation and Mass-Based Clustering, *In Proceedings of IEEE International Conference on Data Mining*, pp. 511–520.

Ting, K., Washio, T., Wells, J., Liu, F. and Aryal, S. (2013). DEMass: a new density estimator for big data, *Knowledge and Information Systems* pp. 1–32. **URL:** *http://dx.doi.org/10.1007/s10115-013-0612-3*

Ting, K., Wells, J., Tan, S., Teng, S. and Webb, G. (2011). Feature-subspace aggregating: ensembles for stable andunstable learners, *Machine Learning* **82**: 375–397.

Vries, T. D., Chawla, S. and Houle, M. (2012). Density-preserving projections for large-scale local anomaly detection, *Knowledge and Information Systems* **32**: 25–52.

Webb, G., Boughton, J., Zheng, F., Ting, K. and Salem, H. (2012). Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly naive Bayesian classification, *Machine Learning* **86**: 233–272.

Webb, G. I., Boughton, J. R. and Wang, Z. (2005). Not So Naive Bayes: Aggregating one-dependence estimators, *Machine Learning* **58**: 5–24.

Wells, J. R., Ting, K. M. and Washio, T. (2012). A new approach to nearest neighbour algorithms, *Technical Report TR2012/9*, Gippsland School of Information Technology, Faculty of Information Technology, Monash University, Churchill, Victoria, Australia.

Zheng, Z. and Webb, G. I. (2000). Lazy Learning of Bayesian Rules, *Machine Learning* pp. 53–84.