Efficient Distribution Mining and Classification

Yasushi Sakurai
NTT Communication Science Labs
yasushi.sakurai@acm.org

Rosalynn Chong
University of British Columbia
rchong@interchange.ubc.ca

Lei Li
Carnegie Mellon University
leili@cs.cmu.edu

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

Abstract

We define and solve the problem of "distribution classification", and, in general, "distribution mining". Given n distributions (i.e., clouds) of multi-dimensional points, we want to classify them into k classes, to find patterns, rules and out-lier clouds. For example, consider the 2-d case of sales of items, where, for each item sold, we record the unit price and quantity; then, each customer is represented as a distribution/cloud of 2-d points (one for each item he bought). We want to group similar users together, e.g., for market segmentation, anomaly/fraud detection.

We propose *D-Mine* to achieve this goal. Our main contribution is Theorem 3.1, which shows how to use wavelets to speed up the cloud-similarity computations. Extensive experiments on both synthetic and real multi-dimensional data sets show that our method achieves up to *400 faster* wall-clock time over the naive implementation, with comparable (and occasionally better) classification quality.

1 Introduction

Data mining and pattern discovery in traditional and multimedia data has been attracting high interest. Here we focus on a new type of data, namely, a distribution, or, more informally, a "cloud" of multi-dimensional points. We shall use the terms "cloud" and "distribution" interchangeably.

Our goal is to operate on a collection of clouds, and find patterns, clusters, outliers. In distribution classification, we want to classify clouds that are similar to labeled data, in some sense. For example, Figure 1 shows 3 distributions corresponding to 3 motion-capture datasets. Specifically, they are the scatter-plots of the left- and right-foot position. Each motion is clearly a cloud of points, and we would like to classify these motions. Arguably, *Distributions* #1 and #2 are more similar to each other, and they should be both classified into the same group (in fact, they both correspond to "walking" motions). In contrast, *Distribution* #3 does not overlap with the other distributions at all, and thus it should probably go to another group (in fact, it belongs to a jumping motion).

There are many applications for distribution classification. Consider an e-commerce setting, where we wish to classify customers according to their purchasing habits. Suppose that, for every sale, we can obtain the time the customer spent browsing, the number of items bought, and their sales price. Thus, each customer is a cloud of 3-d points (one for each sale she did). The e-store would like to classify these clouds, to do market segmentation, rule discovery (*is it true that the highest volume customers spend more time on our web site?*) and spot anomalies (e.g., identity theft). Another example of distribution classification and mining is an SNS system such as a blog hosting service. Consider a large number of blog users where each user has a list of numerical attributes (e.g., total number of postings, average posting length, average number of outgoing links); suppose that we can record all these attributes, on a daily basis, for all blog users. For this collection of clouds, one would like to find patterns and groups, e.g., to do sociological and behavioral research.

All these questions require a good distance function between two clouds. The ideal distance function should (a) correspond to human intuition, declaring as 'similar', e.g., *Distributions* #1 and #2 of Figure 1 and (b) it should be fast to compute, ideally, much faster than the number of points in the clouds P and Q. Thus the base problem we focus on here is as follows:
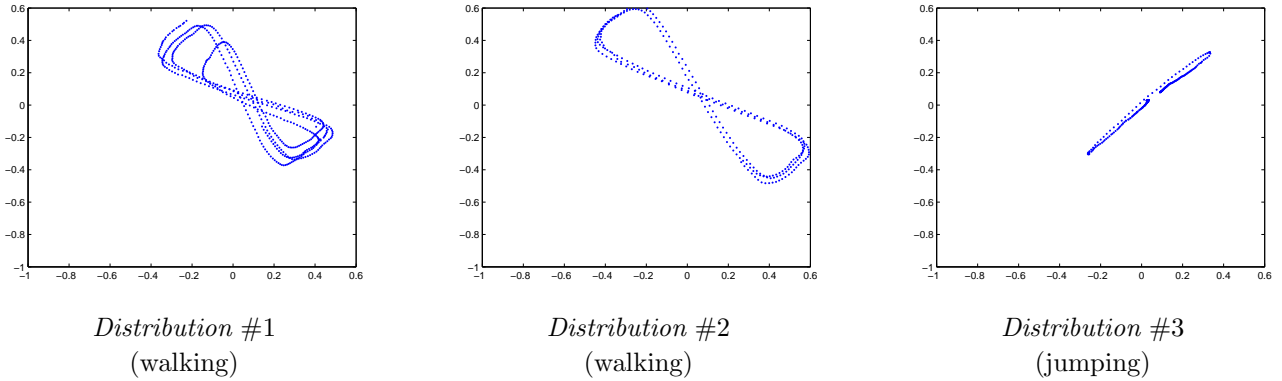
Figure 1: Three distributions are shown here, from motion-capture data: they all show the scatter-plot of the left foot position versus the right foot, for different motions. *Distributions* #1 and #2 "look" more similar, while *Distribution* #3 "looks" different. Our proposed distance function reflects such differences.

PROBLEM 1. (CLOUD DISTANCE FUNCTION) *Given 2 clouds P and Q of d-dimensional points, design a distance function between them, and develop an algorithm to compute it quickly*

Once we have a good (and fast) distance function, we can proceed to solve the following two problems:

PROBLEM 2. (CLOUD CLASSIFICATION) *Given n clouds of d-dimensional points, with a portion of them labeled and others unlabeled, classify the unlabeled clouds into the right group.*

And, more generally, we want to find patterns:

PROBLEM 3. (CLOUD MINING) *Given n clouds of d-dimensional points, summarize the collection of clouds, find outliers ("strange" clouds), anomalies and patterns.*

Our goal is to design fast algorithms for *distribution* mining and classification for large sets of multidimensional data. A related topic is distribution clustering. While we will not address the clustering in this paper, our techniques could be similarly applied to clustering settings. The contributions are the following:

(a) the *DualWavelet* method, that keeps track of two types of wavelet coefficients for each cloud, accelerating the calculation of the cloud-distance function. (b) Theorem 3.1, which proves the correctness of our method (c) The *GEM* method to find a good grid-cell size for a collection of clouds.

The rest of the paper is organized as follow: Section 2 discusses the related work. Section 3 describes the proposed method and identifies main tasks needed for *distribution* classification. We then explain the techniques and constraints we use in order for efficient KL-divergence calculations. Section 4 evaluates our method by conducting extensive experiments. Finally, Section 5 concludes the paper and introduces future work for our topic.

2 Related Work

We group the related work into four sections: clustering, similarity functions between distributions, wavelets.

Classification and Clustering A huge number of classification techniques have been proposed in various research fields, including Naive Bayes, decision tree [1, 17], support vector machine [2] and k-nearest neighbor [5]. We choose to use the nearest neighbor method because it has no additional training effort, while remains a nice asymptotic error bound [5]. A related mining technique is clustering, with techniques including CLARANS [15], k-means [7], BIRCH [23], CURE [9] and many many more. We did not directly address the distribution clustering problem, but with the techniques we proposed in this paper could also be adapted to clustering.

Comparing two distributions There are several statistical methods [16] to decide whether two distributions are the same (Chi-square, Kolmogorov-Smirnoff). However, they don't give a score; only a yes/no answer; and some of them can not be easily generalized for higher dimensions.

Functionals that return a score are motivated from image processing and image comparisons: The so-called *earth-moving distance* [18] between two images is the minimum energy (mass times distance) to transform one image into the other, where mass is the gray-scale value of each pixel. For two clouds of points P and Q

(= black-and-white images), there are several measures of their distance/similarity: one alternative is the distance of the closest pair (min-min distance); another is the Hausdorff distance (max-min - the maximum distance of a set P, to the nearest point in the set Q). another would be the average of all pairwise distances among P-Q pairs. Finally, tri-plots [21] can find patterns across two large, multidimensional sets of points, although they can not assign a distance score. The most suitable idea for our setting is the KL (Kullback-Leibler) divergence (see Equation (3.1)) which gives a notion of distance between two distributions. The KL divergence is commonly used in several fields, to measure the distance between two PDFs (probability density functions, as in, e.g., information theory [22], pattern recognition [19, 10]).

Probabilistic queries: A remotely related problem is the problem of probabilistic queries. Cheng et al. [3] classifies and evaluates probabilistic queries over uncertain data based on models of uncertainty. An indexing method for regular probabilistic queries is then proposed in [4]. In [20] Tao et al. presents an algorithm for indexing uncertain data for any type of probability density functions (pdf). Distributions for our work can be expressed by pdfs as well as histograms. However, the main difference between our study and [3] is that our work focuses on comparing differences between distributions, while Cheng et al.s' work focuses on locating areas in distributions that satisfy a given threshold.

Wavelets Wavelet analysis is a powerful method for compressing signals (time series, images, and higher dimensionalities) [6, 16]. Wavelets have been used for summarization of streams [8] and histograms [14].

Distribution classification and mining are problem that, to our knowledge, have not been addressed. The distance functions among clouds that we mentioned earlier either expect a continuous distribution (like a probability density function), and/or are too expensive to compute.

3 Proposed Method

In this section, we focus on the problem of distribution classification. This involves the following sub-problems, that we address in each of the upcoming sub-sections: (a) What is a good distance function to use, for clouds of points? (b) What is the optimal grid side, to bucketize our address space? (c) How to represent a cloud of points compactly, to accelerate the distance calculations? (d) What is the space and time complexity of our method?

Table 1 lists the major symbols and their definitions.

Table 1: Symbols and definitions

Symbol	Definition
n	number of clouds
k	number of classes
m	number of buckets in the grid
P, Q	two clouds of points ("distributions")
p_i, q_i	fractions of points at i-th grid-cell
$P(m)$	the histogram of P, for a grid with m buckets
d_{SKL}	symmetric KL divergence
$H_s(P)$	entropy of cloud P, bucketized with s segments per dimension
W_p	wavelet coefficients of P

3.1 Distance function for cloud sets The heart of the problem is to define a good distance function between two clouds of d-dimensional points:

Task: Given two "clouds" of points P and Q in d-dimensional space, we assess the similarity/distance between P and Q.

Given two clouds of points P and Q, there are several measures of their distance/similarity, as we mention in the literature survey section.

However, the above distances suffer from one or more of the following drawbacks: they are either too fragile (like the min-min distance); and/or they don't take all the available data points into account; and/or they are too expensive to compute.

Thus we propose to use information theory as the basis, and specifically the KL (Kullback-Leibler) divergence, as follows. Let's assume for a moment that the two clouds of points P and Q consist of samples from two (continuous) probability density functions $\mathcal{P}$ and $\mathcal{Q}$, respectively. If we knew $\mathcal{P}$ and $\mathcal{Q}$ we could apply the continuous version of the KL divergence; however, we don't. Thus we propose to bucketize all our clouds, using a grid with m grid-cells and use the discrete version of the KL divergence, defined as follows:

$$(3.1) \qquad d_{KL}(P,Q) \quad = \quad \sum_{i=1}^{m} p_i \cdot \log\left(\frac{p_i}{q_i}\right)$$

where p_i, q_i are the fractions of points of cloud P and Q respectively, that land into the i-th grid cell. That is, $\sum_{i=1}^{m} p_i = \sum_{i=1}^{m} q_i = 1$. Choosing a good grid-cell size is a subtle and difficult problem, that we address with our proposed *GEM* method in subsection 3.2.

The above definition is asymmetric, and thus we propose to use the symmetric KL divergence d_{SKL}:

$$
\begin{aligned}
(3.2) \quad d_{SKL}(P,Q) &= d_{KL}(P,Q) + d_{KL}(Q,P) \\
&= \sum_{i=1}^{m} (p_i - q_i) \cdot \log\left(\frac{p_i}{q_i}\right).
\end{aligned}
$$

There is a subtle point we need to address. The KL divergence expects non-zero values, however, histogram buckets corresponding to sparse area in multidimensional space may take the zero value. To avoid this, we introduce the Laplace estimator [13, 11]:

$$
(3.3) \qquad p_i' = \frac{p_i + 1}{|P| + m} \cdot |P|.
$$

where p_i is the original histogram value ($i = 1, \ldots, m$) and p_i' is the estimate of p_i. $|P|$ is the total number of points (i.e., $|P| = \sum_{i=1}^{m} p_i$).

Another solution is that we could simply treat empty cells as if they had "minimum occupancy" of ϵ. The value value for "minimum occupancy" should be $\epsilon << 1/|P|$, where $|P|$ is the number of points in the cloud P. We chose the former since it gives better classification accuracy, although our algorithms are completely independent of such choices.

3.2 *GEM*: Optimal grid-side selection

3.2.1 Motivation behind *GEM*
Until now, we assumed that the grid side was given. The question we address here is how to automatically estimate a good grid side, when we are given several clouds of points $P_1, \ldots, P_n$. Equivalently, we want to find the optimal number of segments s_{opt} that we should subdivide each dimension into.

We would like to have a criterion, which will operate on the collection of clouds, and dictate a good number of segments s per dimension. This criterion should have a spike (or dip) at the "correct" value of s. Intuitively, s should not be too small, because the grid will deteriorate to having one or two cells, and all clouds will look similar, even if they are not really similar. On the other extreme, s should not be too large, because the resulting grid will have a very fine granularity, with each grid-cell containing one or zero points, and thus any pair of clouds will have a high distance. We want the "sweet spot" for s.

Notice that straightforward approaches are not suitable, because they are monotonic. For example, the average entropy of a cloud clearly increases with the number of segments s. The same is true for any other variation based on entropy, like the entropy of the union of all the clouds. Our goal boils down to the following question: *what function of s reaches an extreme when we hit the 'optimal' number of segments s_{opt}?* It turns out that the (symmetric) KL divergence is also unsuitable since it increases monotonically with s. This leads us to our proposed "*GEM*" criterion. First we describe how to find a good value of s (number of grid-cells per axis) for a given pair of clouds (P, Q), and then we discuss how to extend it for a whole set of clouds.

3.2.2 *GEM* criterion
First, we propose a criterion to estimate the optimal number of s for a given *pair* of clouds, say, P and Q.

Our idea is to *normalize* the symmetric KL divergence, dividing it by the sum of entropies of the two clouds. The intuition is the following: for a given s, the KL divergence $d_{SKL(s)}(P,Q)$ expresses the cross-encoding cost, that is, the number of extra bits we suffer, when we encode the grid-cell id of points of cloud P, using the statistics from cloud Q, and vice versa. Similarly, $H_s(P)$ and $H_s(Q)$ give the (self) encoding cost, that is, the number of necessary bits, when we use the correct statistics for each cloud. Thus, we propose to normalize: a grid choice (with s grid-cells per dimension) would be good, if it can make the two clouds as dissimilar (= hard to cross-encode) as possible, *relative* to the intrinsic coding cost of each cloud.

Therefore, our proposed normalized KL divergence $C_s(P,Q)$, referred to as the *pairwise* GEM *criterion*, is given by

$$
(3.4) \qquad C_s(P,Q) = \frac{d_{SKL(s)}(P,Q)}{H_s(P) + H_s(Q)}
$$

where $H_s(P)$, $H_s(Q)$, $d_{SKL(s)}(P,Q)$ are the entropies and the KL divergence, for a grid with s segments per dimension. We want s_{opt}, the optimal value of s that maximizes the pairwise criteria as follows:

$$
(3.5) \qquad s_{opt}(P,Q) = arg\max_{s}(C_s(P,Q)).
$$

For example, Figure 2 shows the pairwise criteria of a sampled cloud pair of the *Gaussians* dataset (See Figure 4). We pick up one from the third class (*Corners*)
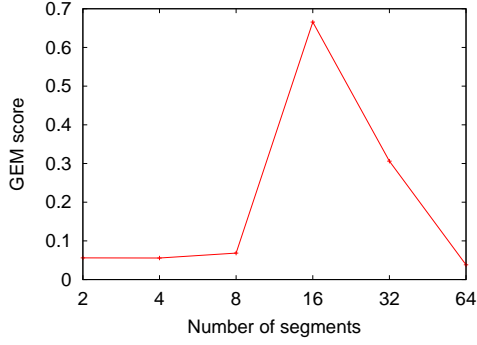
Figure 2: Example of pairwise *GEM* criterion.

and the other one from the fourth class (*Center & Corners*). As shown in Figure 4, it is hard to identify the difference between these clouds if we use a small number of segments. In Figure 2 $s = 16$ maximizes the pairwise criteria, which indicates that it is necessary to use $s = 16$ to identify the difference.

Once we obtain $s_{opt}()$ for every (sampled) pair the final step is to choose s_{opt} for the entire cloud set. We could choose the median value, the average value, or some other function. It turns out that the *maximum* gives the best classification accuracy. Thus, we propose

$$(3.6) \qquad s_{opt} = \max_{all \ P,Q \ pairs} s_{opt}(P,Q)$$

If there are too many clouds in our collection, we propose to use a random sample of (P, Q) pairs.

3.3 Cloud representation: *DualWavelet*

We described how to measure the distance of clouds, and how to estimate the optimal segment size of the histogram. The next question is how to store the cloud information, to minimize space consumption and response time. Recall that we plan to bucketize the d-dimensional address space into m grid cells, with s equal segments per dimension ($m = s^d$); and for each distribution P, we could keep the fraction of points p_i that fall in the i-th bucket.

Naive Solution: The naive solution is exactly to maintain an m-bucket histogram for each cloud, and to use such histograms to compute the necessary KL divergences, and eventually to run required mining algorithm (e.g., the nearest neighbor classification). We refer to it as *NaiveDense*.

However, this may require too much space, especially for higher dimensions. One solution would be to use the top c most populated grid cells, in the spirit of 'high end histograms'. The reason against it is that

we may ignore some sparse-populated bins, whose logarithm would be important for the KL divergence. We refer to it as *NaiveSparse*.

Instead, we propose to compress the histograms using *Haar* wavelets, and *then* keeping some appropriate coefficients, as we describe next. As we show later, this decision significantly improves both space as well as response time, with negligible effects on the mining results. The only tricky aspect is that if we just keep the top c wavelet coefficients, we might not get good accuracy for the KL divergence. This led us to the design of the *DualWavelet* method that we describe next. The main idea behind *DualWavelet* is to keep the top c wavelet coefficients for the histogram $P(m) = (p_1, \dots p_m)$, *as well as* the top c coefficients for the histogram of the *logarithms* $(\log p_1, \dots \log p_m)$. This is the reason we call the method *DualWavelet*. We elaborate next.

Let $\hat{P} = (\hat{p}_1, \dots, \hat{p}_m)$ be the histogram of the logarithms of $P = (p_1, \dots, p_m)$, i.e., $\hat{p}_i = \log p_i$. Let W_p and $\hat{W}_p$ be the wavelet coefficients of P and $\hat{P}$ respectively. We present our solution using W_p and $\hat{W}_p$.

Proposed Solution: We represent each distribution as a single vector; we compute W_p and $\hat{W}_p$ from P and $\hat{P}$ for each distribution, and then we compute the *GEM* criterion and the necessary KL divergences from the wavelet coefficients. Finally, we apply a classification algorithm (e.g., the nearest neighbor classification) to the wavelet coefficients.

The cornerstone of our method is Theorem 3.1, which effectively states that we can compute the symmetric KL divergence, using the appropriate wavelet coefficients.

THEOREM 3.1. *Let $W_p = (wp_1, \dots, wp_m)$ and $\hat{W}_p = (\hat{wp}_1, \dots, \hat{wp}_m)$ be the wavelet coefficients of P and $\hat{P}$, respectively. Then we have*

$$(3.7) \qquad d_{SKL}(P,Q) = \frac{1}{2} \sum_{i=1}^{m} f_{pq}(i)$$

$$f_{pq}(i) = (wp_i - \hat{wq}_i)^2 + (wq_i - \hat{wp}_i)^2 \\ -(wp_i - \hat{wp}_i)^2 - (wq_i - \hat{wq}_i)^2.$$

Proof: From the definition,

$$d_{SKL}(P,Q) = \sum_{i=1}^{m} (p_i - q_i) \cdot \log\left(\frac{p_i}{q_i}\right).$$

Then we have

$$d_{SKL}(P,Q) = \sum_{i=1}^{m}(p_i - q_i) \cdot (\log p_i - \log q_i)$$
$$= \frac{1}{2}\sum_{i=1}^{m} f_{pq}(i).$$

In light of Parseval's theorem, this completes the proof. □

The KL divergence can be obtained from Equation (3.7) using wavelets calculated from histogram data. The number of buckets of a histogram (i.e., m) could be large, especially for high-dimensional spaces, while the most of buckets may be empty. The justification of using wavelets is that very few of the wavelet coefficients of real data sets are often significant and a majority are small, thus, the error is limited to a very small value. Upon calculating the wavelets from the original histogram, we select a small number of wavelet coefficients (say c coefficients) that have the largest energy from the original wavelet array. This indicates that these coefficients will yield the lowest error among all wavelets.

For Equation (3.7), we compute the KL divergence with c coefficients, that is, we compute $f_{pq}(i)$ if we select either wp_i or wq_i ($\hat{wp}_i$ or $\hat{wq}_i$), otherwise, we can simply ignore these coefficients since they are very close to zero. Consider that the sequence describing the positions of the top c coefficients of W_p and W_q is denoted as $\mathcal{I}_{pq}$. Then, we obtain the approximate KL divergence of P and Q:

$$(3.8) \qquad d_{SKL}(P,Q) \approx \frac{1}{2}\sum_{i \in \mathcal{I}_{pq}} f_{pq}(i).$$

The positions of the best c coefficients of W_p could be different from those of W_q. We assume $wp_i = \hat{wp}_i = 0$ if wp_i and $\hat{wp}_i$ are not selected.

In order to estimate the optimal number of segments for histograms, we need to compute the entropy from each distribution (see Equation (3.4)).

LEMMA 3.1. *Let W_p and $\hat{W}_p$ be the wavelets of P and $\hat{P}$, respectively. Then we have*

$$(3.9) \qquad H(P) = \frac{1}{2}\sum_{i=1}^{m} g_p(i)$$
$$g_p(i) = (wp_i - \hat{wp}_i)^2 - wp_i^2 - \hat{wp}_i^2.$$

Algorithm *D-Mine*
   // *estimate the optimal grid side, using* GEM
   for each # of segments s **do**
      for each distribution P **do**
         compute the *DualWavelet* coefficients
                      W_p and $\hat{W}_p$ of s;
      endfor
      estimate the *GEM* criterion C_s from the set of
                 the *DualWavelet* coefficients;
      if C_s is the maximum value **then**
         $s_{opt} := s$;
   endfor
   // *keep the* DualWavelet *coefficients for the optimal*
   //   *grid side*
   use any data mining method (classification, clustering,
      outlier detection, etc) and apply it to the set of
      the *DualWavelet* coefficients of s_{opt};

Figure 3: *D-Mine*: algorithm for distribution mining.

Proof: From the definition,

$$H(P) = -\sum_{i=1}^{m} p_i \log p_i.$$

From Parseval's theorem, we have

$$H(P) = \frac{1}{2}\sum_{i=1}^{m} g_p(i),$$

which completes the proof. □

Again, since we want to compute the entropy with the top c wavelet coefficients, Equation (3.9) becomes

$$(3.10) \qquad H(P) \approx \frac{1}{2}\sum_{i \in \mathcal{I}_p} g_p(i).$$

Equation (3.4) requires the KL divergence and entropies of P_i and P_j to estimate the optimal number of segments. The KL divergence and entropies can be efficiently computed from their wavelets by using Equations (3.8)(3.10).

Figure 3 shows the overall algorithm for distribution mining.

3.4 Complexity In this section we examine the time and space complexity of our solution and compare it to the complexity of the naive solution. Among the possible data mining tasks, we focus on classification, to illustrate the effectiveness of our method. We chose the k-nearest neighbor classification with majority voting.

We made this chose because of the simplicity and the good performance, as we show next. Recall that n is the number of input distributions, m is the number of grid-cells that we impose on the address space, and c is the number of wavelet coefficients that our method keeps. For the classification, t is the number of distributions (=clouds) in the training data set.

3.4.1 Time Complexity

Naive method (*NaiveDense*) . We have the following Lemmas:

LEMMA 3.2. *The naive method requires $O(mn^2)$ time to compute the criterion for distribution classification.*

Proof: For d-dimensional histogram, the number of buckets is $m = s^d$. Computing the criterion for s requires $O(m)$ time for every distribution pair. For n distributions, it would take $O(mn^2)$ time. □

The calculation of the criterion requires $O(mn^2)$ time if we take account of all n distributions. We can optionally use random sampling of the distributions to save the computation cost.

With respect to the time for nearest neighbor searching, we resort to sequential scanning over the training dataset. The reason that we do *not* use a multi-dimensional index (like kd-tree, or R-tree), is because the KL divergence does not even obey the triangle inequality. Thus, we have:

LEMMA 3.3. *The naive method requires $O(nmt)$ time for the nearest neighbor classification.*

Proof: The calculation of the nearest neighbor classification for one distribution requires $O(mt)$ time because it tries to handle the values of m histogram buckets for t distributions. This is repeated for n number of distributions, producing a total cost of $O(nmt)$. □

Proposed Solution *DualWavelet* . We have the following Lemmas:

LEMMA 3.4. *The proposed distribution mining method requires $O(mn)$ time to compute all the wavelet coefficients.*

Proof: Wavelets can be computed in $O(m)$ time. For n distributions, it would take $O(mn)$ time. □

LEMMA 3.5. *The proposed method requires $O(n^2)$ time to estimate the optimal number of segments.*

Proof: Let c be the number of wavelet coefficients. Computing each criterion requires $O(cn^2)$ time. Since c is a small constant value, the time complexity to to estimate the optimal number of segments can be simplified to $O(n^2)$ □

LEMMA 3.6. *The proposed method requires $O(nt)$ time to perform the nearest neighbor classification.*

Proof: The calculation of the nearest neighbor classification requires $O(cnt)$ time. We handle c wavelets for t distributions in the training data set. This is repeated for n number of input distributions. Again, since c is a small constant value, the time complexity to classify distributions can be simplified to $O(nt)$. □

3.4.2 Space Complexity
We have the following Lemmas:

LEMMA 3.7. (SPACE OF *NaiveDense*) *The naive method requires at least $O(mn)$ space.*

Proof: The naive method requires storage of m-binned histograms of n distributions, hence the complexity $O(mn)$. □

LEMMA 3.8. (SPACE OF PROPOSED METHOD) *The proposed method requires at least $O(m + n)$ space.*

Proof: Our method initially allocates memory to store histogram of m buckets. It then calculates the wavelet coefficients and discards the histogram information. Then it reduces the number of wavelet coefficients to $O(c)$ and allocates $O(cn)$ memory for computing the criterion and the classification. However, c is normally a very small constant, which is negligible. We sum up all the allocated memory and we get a space complexity of $O(m + n)$. □

4 Experiments

In this section, we do experiments to answer the following questions:

- GEM*'s accuracy*: For the optimal grid side, does the *GEM* criterion lead to better mining results?

- *Quality*: how good is our cloud-distance function, and its *DualWavelet* approximation? We compare *DualWavelet* with two variants of the naive method: (1) *NaiveDense*, which uses all (m) histogram buckets to compute the distance (2) *NaiveSparse*, which uses only selected buckets since m could
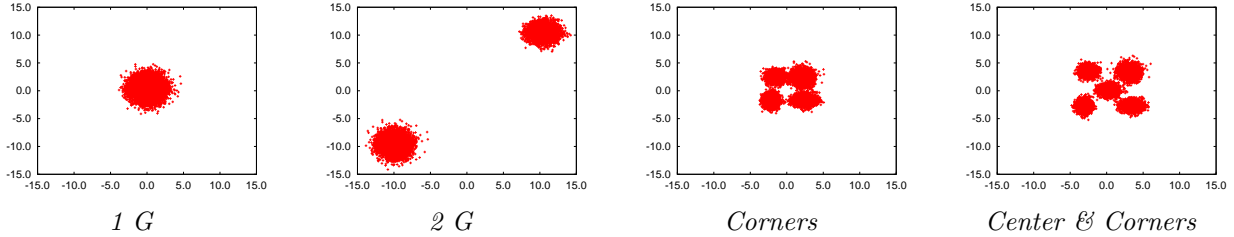
Figure 4: Sample clouds, each representing one of 4 different classes in the *Gaussians* dataset.

be too large while the most of buckets may be empty. We selected buckets that have the largest values from the original histogram. As a measure of accuracy, we use the classification error rate. For each input dataset, we know the ground truth (that is, the number of classes we have and the class labels for each input cloud) and we run the nearest neighbor classification on the dataset, with optimal s (according to *GEM*), and report the classification accuracy.

- *Speed*: how much faster is our method than the naive ones, *NaiveDense* and *NaiveSparse*? How does it scale with the database size n in terms of the computation time?

We run the experiments on a Pentium 4 CPU of 2.53GHz, and with 2GB of memory. For the number of wavelet coefficients c, we used enough, so that to retain 95% of the energy.

4.1 Data Sets We performed experiments on the following real and synthetic data sets.

- *Gaussians*: the data set consists of n=4,000 distributions, each with 10,000 points. Figure 4 shows some members of this set. We used 3-dimensional data for this data set. Each distribution is a mixture of Gaussians (1, 2, 2^d or ($2^d + 1$) Gaussians). There were thus 4 classes; for each class, the means were the same but the covariance matrices varied. The training data set contains 10 distributions for each class.

- *FinTime*: This is the financial time-series benchmark from [12]. We used historical stock market data for n=400 securities. The dataset consists of 3 dimensional data simulating the stock market with parameters: starting price, increasing (decreasing) rate and volume of transactions. Each distribution

of a stock contains 10,000 values. We chose this setting because it resembles real data such as product sales. There were 4 classes, and the training data set contains 10 for each class.

- *MoCap*: this dataset is from subject number 16, from the CMU Motion Capture Database (`mocap.cs.cmu.edu`). It contains n=58 real running, jumping and walking motions, that is, there are 3 classes. We used 9 motions for training and the rest of them for classification. In our framework, a motion is represented as a cloud of hundreds of frames, with each frame being an d-dimensional point; each dimension corresponds to the x, y, or z position of a body joint . Figure 7 shows sample scatter-plots of such motion data for coordinates of the left-foot and of the right-foot) (with only the z coordinate, for easier visualization) In our experiments we reduce the dimensionality from d=93 to d=2 by using SVD (Singular Value Decomposition), which gives better classification accuracy.

4.2 Accuracy of *GEM* We evaluated the accuracy of distribution classification when we varies the number of segments (i.e., bucket size) of histograms. For the classification accuracy, we used the classification error and the "confusion matrix". For example Table 2 shows the confusion matrix for the *MoCap* dataset, where the rows correspond to actual classes, and the columns correspond to recovered classes. The ideal matrix should be diagonal, in which case the classification error is minimal.

Figure 5 shows the classification error and the histogram of *GEM*, for the *Gaussians* and *FinTime* datasets. The bottom row of the figure shows the histogram of $s_{opt}(P, Q)$, the value of s that gives the highest pairwise *GEM*score (See Section 3.2). We estimated s_{opt} from 10,000 randomly sampled pairs.
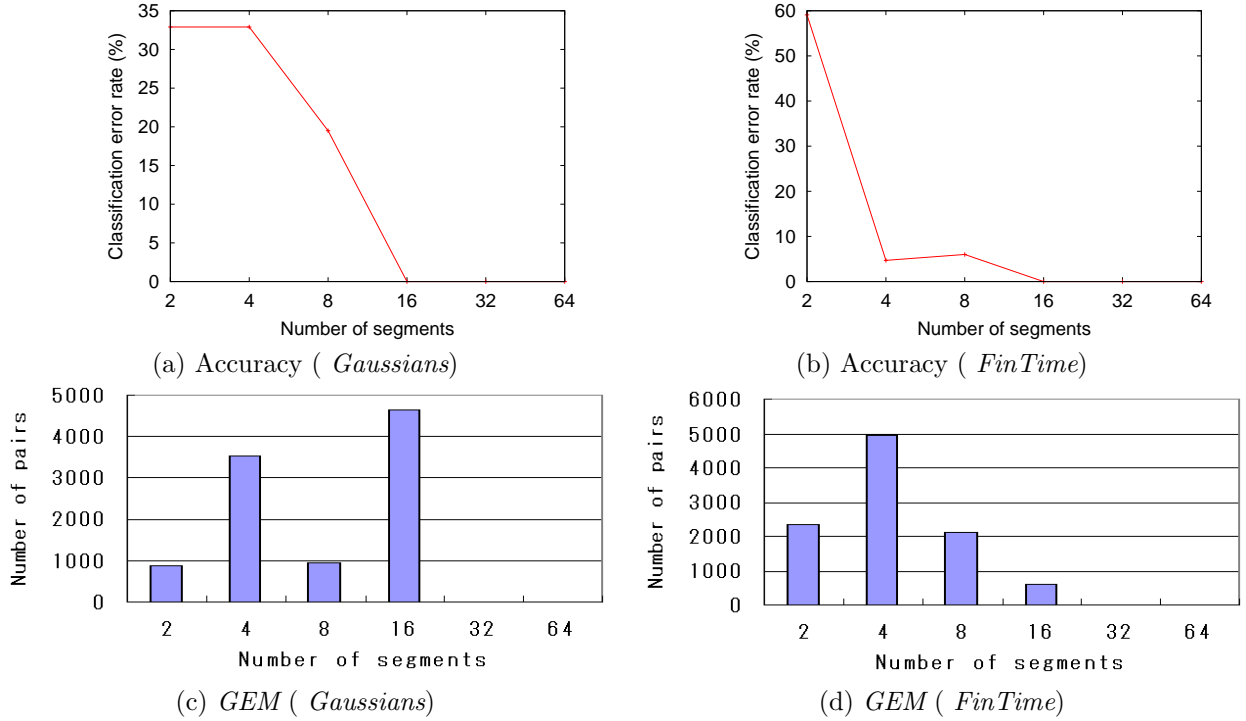
Figure 5: Justification for our *GEM* criterion - *Gaussians* and *FinTime* datasets. (a) and (b): Classification accuracy (error rate) versus number of segments s. (c) and (d): The histogram of $s_{opt}(P, Q)$ (the value of s that gives the highest pairwise *GEM*score). Notice that the maximum value of s_{opt} is 16 for both datasets, which gives zero classification error.

Figure 5 indicates that our algorithm can obtain a good classification (i.e., it can even achieve no error) when we choose $s = 16$ and higher as the number of segments. The maximum value of $s_{opt}(P, Q)$ (i.e., s_{opt}) is $s = 16$. Thus, *GEM* perfectly spots $s = 16$ as the optimal number of segments for both data sets, which means that the user is freed from having to set the s value.

As mentioned in Section 3.3, the cost for estimating the optimal number of segments can be reduced by *DualWavelet*. We will show later that *DualWavelet* drastically reduces the classification cost. We can also receive the same benefit when we compute the *GEM* score.

4.3 Quality of proposed methods We answer two questions here: (a) how good is the approximation that *DualWavelet* uses, and (b) how good are the classification results, of the full workflow (*GEM*, *DualWavelet*, and classification)

For the first question, Figure 6 shows the scatter plots of the computation cost versus the approximation quality. The x-axis shows the computation cost for KL divergences, and the y-axis shows their relative approximation error rate. We compare *DualWavelet* and *NaiveSparse* in the figure.

The figure implies a trade-off between quality and cost, but the results of *DualWavelet* are close to the lower left for both data sets, which means that the proposed method gives excellent performance in terms of quality and cost. In fact, it shows no error for classification quality for *Gaussians* and *FinTime* while it achieves a dramatic reduction in computation time.

We also study the behavior of *DualWavelet* for classification quality using the real motion data *MoCap*. Figure 7 shows 9 distributions, 3 from each class after all distributions are classified using our algorithm. For simplicity of representation, only data from two dimensions are displayed in the figure. We can see that our implementation classifies the distributions accordingly. The overall quality is represented by using confusion matrix as shown in Table 2. While this is not per-
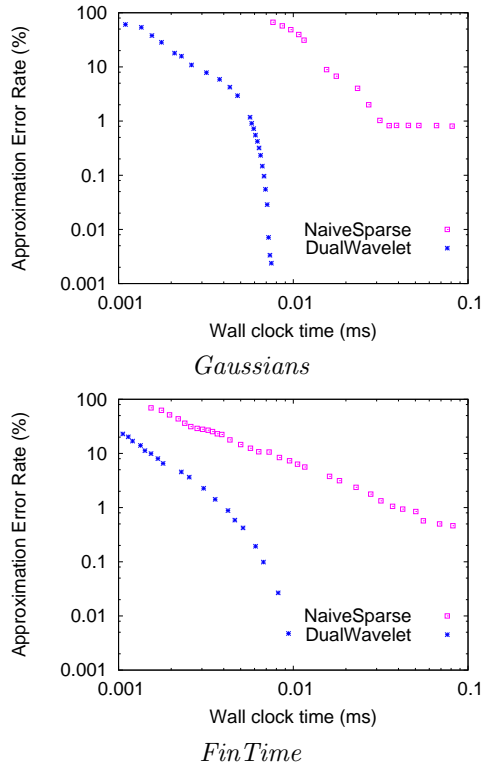
Gaussians

FinTime

Figure 6: Approximation quality: relative approximation error vs. wall clock time. Our method (in blue 'x's) gives significantly lower error, for the same computation time.

fect for this data set, *DualWavelet* can classify the data properly. The classification error of *DualWavelet* and *NaiveDense* is 4.1%. Note that the error rate of *NaiveSparse* is 6.1%. *DualWavelet* provides better classification quality while it is much faster than the naive method. For *Gaussians* and *FinTime* data sets, *DualWavelet* perfectly find the best classification. The resulting confusion matrix is a diagonal matrix, which is omitted due to the space limitation.

4.4 Speed and Scalability Figure 8 compares *DualWavelet* with the naive method in terms of computation cost. We vary the number of distributions we want to classify. Note that y-axis uses logarithmic scale. We conducted this experiment with histogram of $s = 16$ as *GEM* suggested.

The proposed method is significantly faster than the two variants of the naive method while it shows no error for classification quality for both data sets. *DualWavelet* achieves up to 400 times faster than *NaiveDense* and 6

Table 2: Confusion matrix for classification the *MoCap* data set. Notice that each detected class (column) strongly corresponds to a single actual class (row)

recovered correct	jumping	walking	running
jumping	3	0	0
walking	0	22	1
running	0	1	19

times faster than *NaiveSparse* in this experiment.

5 Conclusions and Future Work

Here we addressed the problem of *distribution* mining, and proposed *D-Mine*, a fast and effective method to solve it. Our main contributions are the following:

- We propose to use wavelets on both the histograms, as well as their logarithms. Theorem 3.1 shows that the appropriate wavelet coefficients can help us quickly estimate the KL divergence.

- We also proposed *GEM*, an information-theoretic method for choosing the optimal number of segments s.

- Based on Theorem 3.1 and on *GEM*, we tried nearest neighbor classification on real and realistic data. Our experiments show that our method works as expected, with better classification accuracy than the naive implementation, while being up to *400 times faster*.

We illustrated our Theorem and our method on classification. However, our Theorem would benefit *any* data mining algorithm that needs to operate on *distributions*, exactly because it can help estimate the popular KL divergence, by simply using wavelets.

A promising future direction is to extend our method for streams. As new points (e.g., sales, or stock trades) are added to each cloud, the wavelets can easily handle these additions incrementally, and, thanks to Theorem 3.1, we can thus still estimate quickly the new KL divergences.

References

[1] W. Buntine. Learning classification trees. In D. J. Hand, editor, *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall, London, 1993.
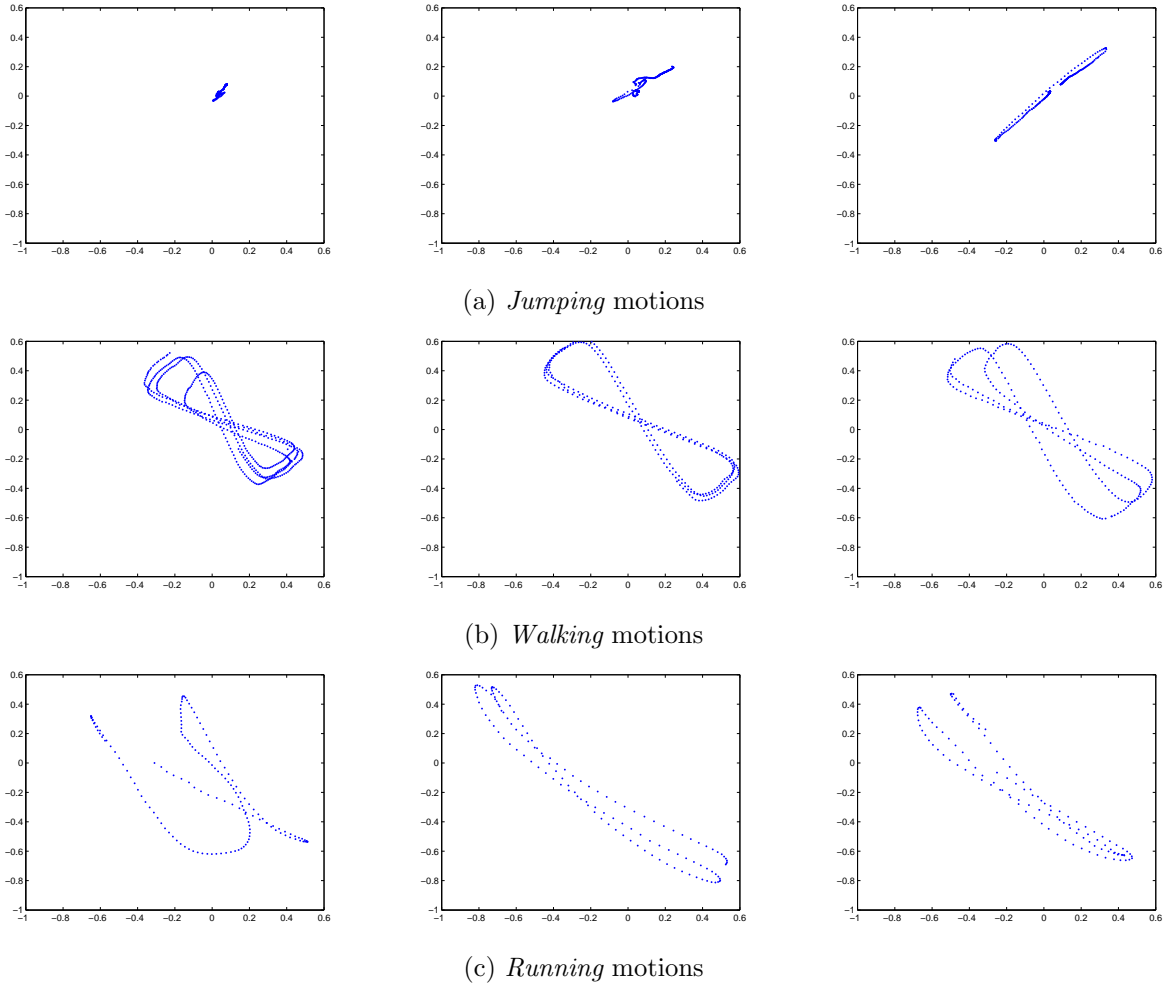
(a) *Jumping* motions

(b) *Walking* motions

(c) *Running* motions

Figure 7: Scatter plot of sample motions (z coordinates for left and right foot)

[2] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[3] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of ACM SIGMOD*, pages 551–562, San Diego, California, June 2003.

[4] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of VLDB*, pages 876–887, Toronto, Canada, August/September 2004.

[5] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[6] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Inc., 1996.

[7] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[8] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of VLDB*, pages 79–88, Rome, Italy, Sept. 2001.

[9] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of ACM SIGMOD*, pages 73–84, Seattle, Washington, June 1998.

[10] X. Huang, S. Z. Li, and Y. Wang. Jensen-shannon boosting learning for object recognition. In *Proceedings of IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 144–149, 2005.
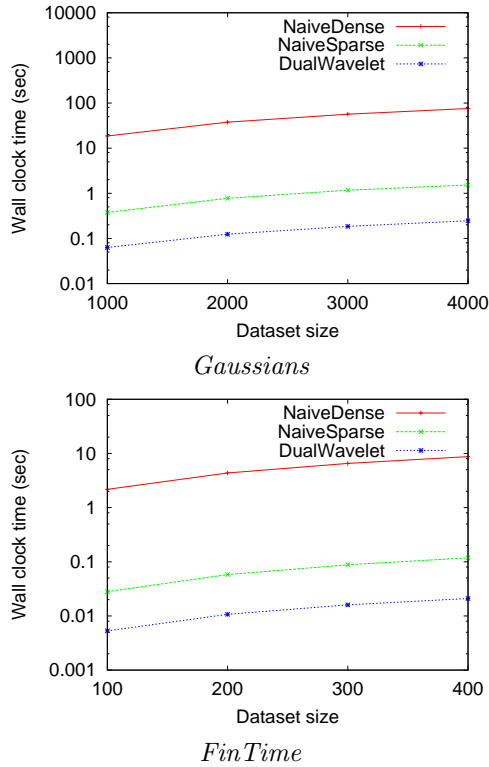
[11] Y. Ishikawa, Y. Machida, and H. Kitagawa. A dy-

Gaussians

FinTime

Figure 8: Scalability: wall clock time vs. database size n (= number of distributions). *DualWavelet* can be up to 400 times faster than *NaiveDense* and 6 times faster than *NaiveSparse*.

namic mobility histogram construction method based on markov chains. In *Proceedings of Int. Conf. on Statistical and Scientific Database Management (SSDBM)*, pages 359–368, 2006.

[12] K. J. Jacob and D. Shasha. Fintime — a financial time series benchmark. http://cs.nyu.edu/cs/faculty/shasha/fintime.html, March 2000.

[13] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[14] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 448–459, New York, NY, USA, 1998. ACM Press.

[15] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of VLDB Conf.*, pages 144–155, Santiago, Chile, Sept. 12-15 1994.

[16] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[17] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[18] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, 2000.

[19] Z. Sun. Adaptation for multiple cue integration. In *Proceedings of IEEE Computer Society International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 440–445, 2003.

[20] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proceedings of VLDB*, pages 922–933, Trondheim, Norway, August/September 2005.

[21] A. Traina, C. Traina, S. Papadimitriou, and C. Faloutsos. Tri-plots: Scalable tools for multidimensional data mining. *KDD*, Aug. 2001.

[22] J.-P. Vert. Adaptive context trees and text clustering. *IEEE Transactions on Information Theory*, 47(5):1884–1901, 2001.

[23] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD*, pages 103–114, Montreal, Canada, June 1996.