A Random Method for Quantifying Changing Distributions in Data Streams

Haixun Wang[1] and Jian Pei[2]

[1] IBM T. J. Watson Research Center, USA
haixun@us.ibm.com
[2] Simon Fraser University, Canada
jpei@cs.sfu.ca

Abstract. In applications such as fraud and intrusion detection, it is of great interest to measure the evolving trends in the data. We consider the problem of quantifying changes between two datasets with class labels. Traditionally, changes are often measured by first estimating the probability distributions of the given data, and then computing the distance, for instance, the K-L divergence, between the estimated distributions. However, this approach is computationally infeasible for large, high dimensional datasets. The problem becomes more challenging in the streaming data environment, as the high speed makes it difficult for the learning process to keep up with the concept drifts in the data. To tackle this problem, we propose a method to quantify concept drifts using a universal model that incurs minimal learning cost. In addition, our model also provides the ability of performing classification.

1 Introduction

In this paper, we study *the distance between two data distributions* instead of two vectors or two sequences. Assume tuples in a training set D are drawn from an unknown distribution $F(\mathbf{x}, t)$. Each tuple is of the form $(\mathbf{x}, t)$, where $\mathbf{x}$ is a vector and t is the class label of $\mathbf{x}$. The task of supervised learning or classification is to learn the unknown relationship between $\mathbf{x}$ and t, that is, to find a model $f^*(\mathbf{x})$, such that the averaged difference between $f^*(\mathbf{x})$ and t is minimum.

We assume there are concept drifts in the unknown data distribution $F(\mathbf{x}, t)$. How do we quantify the concept drift by defining and computing the distance between the original dataset D and a new data set D', which is drawn from the changed unknown distribution? Furthermore, how quantified changes can be used to tune the model $f^*(\mathbf{x})$ we learned before so that it maintains high accuracy on the changed data?

In the field of information theory, relative entropy, or the Kullback Leibler (K-L) divergence, has been suggested as an appropriate measure for comparing data distributions [5]. However, such methods are not computationally feasible for large, high dimensional datasets, or data coming from continuous streams. In the field of data mining, several works have studied how to *detect* changes of data distributions over streams and sequences [1, 10]. However, more often than not, change detection only serves to trigger a costly learning process, and the change itself is not used to mend the current prediction model directly. Recently, several works [8, 13] have studied how to update the current model $f^*(\mathbf{x})$ in response to the concept drifts in data streams, for instance, by assimilating new instances in D' and forgetting old instances in D. These can be very costly undertakings since they do not handle changes directly on the probability distribution level, but rely on a lot of learning and re-learning.

We aim at devising an efficient method to measure distribution changes in high-dimensional, labeled datasets. We assign a *signature* to each dataset, and compare distribution changes by comparing the signatures. Furthermore, the signature should also enable us to make predictions.

2 A Model-based Naive Approach

In this section, we introduce a naive but computationally feasible method for measuring distances between two datasets. We analyze the prediction error of this naive approach through bias/variance decomposition, and we study its impact on the distance measure. In the next section, we introduce a general approach based on the lessons learned here.

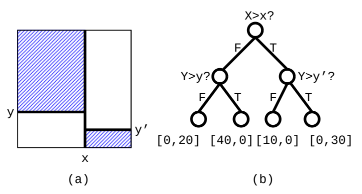

Fig. 1. Model-based description

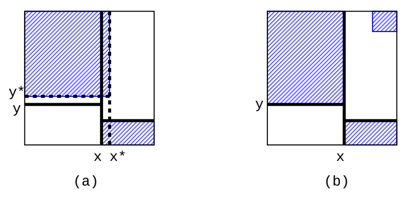

Fig. 2. Distribution Changes

2.1 Measuring Distribution Changes using a Classification Model

Assume we are given a dataset D which consists of a set of tuples $(\mathbf{x}, t)$, where $\mathbf{x}$ is a vector and t is the class label of $\mathbf{x}$. We learn a decision tree classifier T_D from D. The decision tree classifier T_D can be regarded as a summarization of the class distribution of dataset D. More specifically, let $n_1, n_2, ..., n_k$ be the leaf nodes of T_D. Each leaf node n_i is associated with a class distribution (number of objects belonging to each class). Together, $(n_1, n_2, ..., n_k)$ forms a special histogram of frequency counts.

For instance, in Figure 1(a) we show a two dimensional dataset where the shaded areas in the top-left and bottom-right corner are populated with objects of one class, and the rest of the area is populated with objects of the other class. In the rest of the paper, we assume the number of objects in an area is proportional to the size of the area.

From the dataset, we learn a decision tree classifier, which partitions the two dimensional space into 4 areas, each represented by a leaf node as shown in Figure 1(b). Each leaf node is associated with the number of objects of each class in that area. For instance, the second leaf node to the left represents the top-left area, where we assume [40,0] are the number of objects of the two classes in that area. All together, we can use the class distribution of the objects in the leaf nodes to describe the dataset. We call it the *signature* of the data:

$$([0, 20], [40, 0], [10, 0], [0, 30]) \tag{1}$$

Assume now there is some distribution change in the underlying dataset. In one case, the boundary of the shaded area moved from x to x^* horizontally and from y to y^* vertically, as shown in Figure 2(a).

We want to quantify the change using the model we learned from the original dataset. Here, we use the decision tree to classify the changed data set, and use the classification error to quantify the change. To a certain extent, the classification error represents the magnitude of the change, but certainly not the change itself. Because, for instance, datasets in Figure 2(a) and 2(b) will have the same classification error (compared with the original data set in Figure 1(a), they have the same amount of shaded area "out of the place"), but they have very different data distributions. Apparently, the error-based distance measure cannot be used to replace or tune the predictions made by the original decision tree for the changed data.

To ensure that the measure can represent, to a certain extent, the distribution of the change so that it can be used to help make predictions without learning a new model from the changed dataset, we simply 'throw' the objects in Figure 2(a) into the decision tree learned from the original dataset. The class distribution in the leaf nodes is now the signature of the changed dataset:

$$([0, 20], [38, 2], [10, 0], [2, 28]) \tag{2}$$

Now, the dataset in Figure 2(b) results in a different signature: $([0, 20], [40, 0], [10, 0], [4, 26])$, which means signatures are better than prediction errors in representing distributions.

Although we didn't learn a decision tree from the new datasets, the signature, which combines the original decision tree structure and the new class distributions in the leaf nodes, give us some ability to make predictions. Take the dataset in Figure 2(a) and its signature Eq (2) for example. If a test object is classified into the 2nd leaf node to the left, the prediction that the object belong to the positive class will be the probability output $\frac{n_1}{n_1+n_2} = \frac{38}{38+2}$, where n_1 and n_2 are the number of positive and negative nodes in that leaf node respectively.

The signatures also enable us to quantify the differences between the two datasets. If we treat the signature as a vector, we can use any L_p metric to compute their distance. For example, the distance function Eq (3) between two signatures a and b is based on the Manhattan distance:

$$Dist_s(a, b) = \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{c} \left| \frac{n_{a,j,k}}{N_a} - \frac{n_{b,j,k}}{N_b} \right| \tag{3}$$

where n is the number of leaf nodes, c is the number of different classes, $n_{a,j,k}$ is the number of nodes in the j-th leaf node that are of class label k, and N_a is the total number of objects in dataset a. For any two signatures a and b, we have $0 \leq Dist_s(a, b) \leq 1$.

This naive approach gives us the following benefits. First, it is computationally efficient to compare the differences of two data distributions. Second, the data descriptors can be used to make predictions. However, this naive method is also flawed.

2.2 Error Analysis

In the naive method, the model used to describe other datasets is partially learned from a dataset which may have a very different data distribution. This can result in significant prediction error and create problems for the distance measure. In this section, we first reveal such problems, then we use bias-variance decomposition to study their cause.

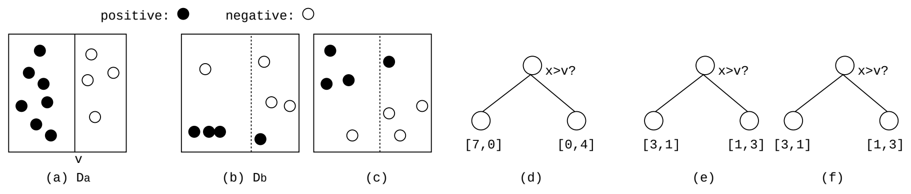

Fig. 3. A Greedy Learner

From D_a in Figure 3(a), we learn a decision tree, and we show the tree hierarchy in Figure 3(d). We then populate the leaf nodes of the decision tree with objects in other datasets. Figure 3(b) and 3(c) represent two very different data distributions. However, because of the tree structure learned from D_a, a same signature, $([3,1],[1,3])$, will be assigned to both datasets. Thus, the distance between the two very different distributions is 0. The signature is thus inaccurate because of the possible large variance introduced by training datasets such as D_a.

For similar reasons, using signatures assigned by the naive method for prediction is also flawed. We populate the leaf nodes of the decision tree learned from D_a in Figure 3(a) with objects in dataset D_b in Figure 3(b). This results in a signature of $([3,1],[1,3])$. Such a signature apparently has large prediction error – even when it is applied on D_b itself, the error can be as large as 25% under zero-one loss.

Clearly, this is due to the fact that D_a's data distribution is very different from D_b's. Decision trees are built in a divide-and-conquer, greedy manner, and in this case, there is no need to make a split on the Y axis for training set D_a, although such a split will result in the largest information gain as far as training set D_b is concerned. The difference of the two data distributions, combined with the greedy nature of the decision tree construction process, results in a large prediction error.

We observe samples $(\mathbf{x}, t)$ drawn independently from some unknown distribution. We want to learn the unknown relationship between $\mathbf{x}$ and t. That is, we want to find a function, $f^*(\mathbf{x})$, that minimizes a certain loss function $L(t, f^*(x))$, where L can be zero-one loss, square loss, absolute loss, etc.

We use the notation $f^*(\mathbf{x}|D)$ to indicate that the prediction model we learn depends on the training dataset D. We decompose the expected prediction error (EPE) into three terms: noise (σ^2), bias, and variance:

$$EPE(\mathbf{x}) = \sigma^2 + Bias(f^*(\mathbf{x}|D))^2 + Var(f^*(\mathbf{x}|D))$$

Let $E_D(f^*(\mathbf{x}|D))$ be the predicted value for sample $\mathbf{x}$ averaged over all the training datasets. The variance can be expressed by:

$$E_D(E_D f^*(\mathbf{x}|D) - f^*(\mathbf{x}|D))^2$$

The variance term measures how sensitive the predicted value at $\mathbf{x}$ is to random fluctuations in the training dataset. Traditionally, a model is learned from a training dataset D drawn from the data distribution we try to learn. In our case, we have two training sets, D_a and D_b. From D_a we learn the structure of the histogram (or equivalently the hierarchy of a decision tree), and from D_b we learn the data distributions within the structure or within the hierarchy. The variance can thus be expressed by:

$$E_{D_a,D_b}(E_{D_a,D_b} f^*(\mathbf{x}|D_a, D_b) - f^*(\mathbf{x}|D_a, D_b))^2$$

Since D_a might be drawn from a data distribution different from the distribution of D_b, which is the distribution we want to learn, by including both D_a and D_b in the condition, the variance is increased because of the added fluctuations.

3 A Universal Model

As discussed in the previous section, the majority of variance and bias is introduced due to training set D_a, from which we learn the structure of a histogram, or a hierarchy of a decision tree. Furthermore, it constitutes the major part of the learning cost. When the change of data distributions between D_a and D_b is non-trivial, the benefits of learning the tree structure from D_a becomes insignificant, since there is no guarantee that such a tree structure will fit the training dataset of D_b well. In this case, it becomes obvious that using an arbitrary tree structure not only serves the same purpose but at the same time eliminates the cost of learning such a structure.

Our goal is to find such an 'arbitrary' structure. It must be general and universal so that it can fit 'any' dataset D_b well, thus we can avoid the bias and variance component in the prediction error such as those introduced by one particular dataset D_a.

3.1 Distance by Random Signatures

A decision tree assigns a signature to a dataset. A signature can be regarded as a special histogram. Each bin, which corresponds to a leaf node in the decision tree, is 'cut out' or defined by the splitting conditions on the path from the root node to that leaf node. The learning procedure determines those conditions as well as their applying order through the computation of information gain.

Take the training set D_a in Figure 4 as an example. It is a two dimensional dataset with two class labels. From D_a, we learn a decision tree, which partitions the two dimensional space into a set of 'bins', each of which is in fact a leaf node in the decision tree. The signature is created by an *entropy-based partition*, since a decision tree is often constructed through the computation of information gain. Note that this learning procedure has super-linear complexity.

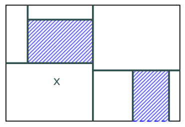

Fig. 4. A Decision Tree Histogram

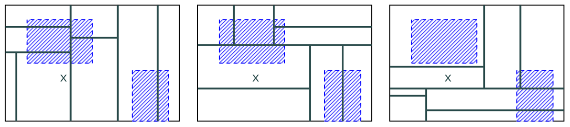

Fig. 5. Random Forest Histograms

We propose to create signatures by randomly partitioning the multi-dimensional space into a set of bins. Figure 5 is such an example. The positions and the order of the splits are totally random, and instead of creating one histogram, we create multiple histograms, each of which is independently and randomly partitioned. In the following, we study two different ways of random partitioning.

Random Forest We use the following procedure to create a random decision tree for a training dataset D.

1. $partition(D)$: randomly pick an unused attribute to partition D into $D_1, \cdots, D_n$;
2. for each partition D_i ($1 \leq i \leq n$), recursively invoke $partition(D_i)$ till the k-th recursive level.

We repeat this process N times to create a forest of N random trees [3]. Each tree defines a signature, and the random forest consists of N signatures for the dataset.

Random Histograms We use the following procedure to create a random histogram for a training dataset D.

1. Randomly pick k attributes, $a_1, \cdots, a_k$, as well as one value for each attribute, such that $\{a_1 = v_1, \cdots, a_k = v_k\}$ defines a bin in the histogram.
2. Repeat the above step M times so that we have a histogram of M bins.

We repeat the above process N times to create N random histograms.

Each of the above methods creates N random structures. Given a dataset D_x, we populate the random structures with objects in D_x, which results in N signatures $S_{x,1}, \cdots, S_{x,N}$ for D_x. We use the same random structures for all datasets. Clearly, for any two datasets, D_a and D_b, signatures $S_{a,i}$ and $S_{b,i}$ have the same number of bins and each bin defines the same subspace in the multi-dimensional data space. We then define the distance between two datasets D_a and D_b as: $Dist(D_a, D_b) = \frac{1}{N}\sum_{i=1}^{N} Dist_s(S_{a,i}, S_{b,i})$, where $Dist_s$ is the distance between two signatures defined in Eq (3), and we have $0 \leq Dist(D_a, D_b) \leq 1$.

The difference between this method and the naive method is that in this method, i) the structure of a signature does not rely on one dataset (which is known as D_a in the naive method), and ii) instead of having one signature, it uses multiple signatures. As will be discussed in detail in the following sections, the multiple random signatures is capable of 'fitting' any dataset, which means the distance metric and the prediction model will have high accuracy.

3.2 Classification by Random Signatures

A signature is composed of a set of histograms, each of which can be expressed by a vector $[n_1, \cdots, n_c]$, where n_i is the number of objects that belong to class i.

The signature is used for prediction: an testing object that falls into a bin with class histogram $[n_1, \cdots, n_c]$ is classified to be of class i if $i = \arg\max_i \frac{n_i}{\sum n_i}$. However, a random signature is often a "weak" classifier.

The weakness of a single random signature can often be averted as our random methods create N signatures for a training dataset. The final prediction is a voted combination of all signatures. In other words, each signature is a classifier, and the N signatures form a classifier ensemble.

Combining an ensemble of classifiers is an established research area [2, 6, 12]. Particularly, for random forests, the prediction accuracy is shown to be no less than that of normal decision trees. Although each random signature is possibly a very "weak" classifier, it has been shown that if each classifier in the ensemble is independent in the production of its error, the expected error of the ensemble can be reduced to zero as the number of the classifiers goes to infinity [7].

3.3 Signatures' Structural Diversity

Whether the signature-based distance metric and prediction model are meaningful depends on whether the random signatures can "fit" any dataset. The strength of an ensemble comes from its diversity [9]. In this section, we discuss how to guarantee signatures' structural diversity.

In an ensemble, a classifier is valuable if it disagrees on some inputs with the other classifiers. Building a diverse ensemble in which each hypothesis is as different as possible is important to an ensemble method. Normally, diversity is measured by prediction disagreements among ensemble classifiers. In our case, random structures are created without a training dataset, which means we can only measure diversity by directly studying the differences of their internal structures. In a signature, each bin corresponds to a set of attribute values. We use the number of different attribute combinations as a measure of diversity. Let A be the number of attributes of the datasets. For simplicity, in our discussion we assume each attribute has v unique values.

- In a *random forest*, each tree of height k has v^k leaf nodes. The path from the root node to any leaf node has $k-1$ edges. Thus, the diversity of attribute combinations in one random tree is at most $\min(v^{k-1}, \binom{k-1}{A})$. In the worst case, all leaf nodes (bins) share one attribute combination. Furthermore, attribute combinations may be correlated.
- For *random histograms*, each bin is defined independently by k attribute values. To compare with the above methods, we create v^k bins. The diversity can be as high as $\min(v^k, \binom{k}{A})$. In the worst case, all bins share one attribute combination. This occurs when all attributes are used ($k = A$), or each random selection returns the same set of attributes.

In summary, random histograms provide the most diverse set of attribute combinations with low correlation.

Our second question is how many bins should we keep in each random structure? We answer this question for random histograms. For random histograms, the number of attribute combinations is at most $\min(v^k, \binom{k}{A})$. Note that $\binom{k}{A}$ reaches maximum when $k = A/2$. Thus, when $v^{A/2} > \binom{A/2}{A}$, we shall use $k = A/2$ attributes for random histograms; otherwise, we shall use k attributes where k satisfies $v^k >= \binom{k}{A}$ and $v^{k-1} < \binom{k-1}{A}$.

4 Conclusion

The ability to quantify the similarity between two datasets is important to many applications, especially data stream applications that deal with time-changing data distributions. Statistical methods, such as K-L divergence and Kriging, are usually not computationally feasible for large, high speed datasets. In this paper, we propose a new approach based on the theory of random forests and classifier ensemble. To measure the difference between two data distributions, our approach measures the difference between the models derived from the datasets. To do this, we must use models that can truthfully represent the dataset, and models that can be trained efficiently. The models we propose for this purpose is the random histograms. The random histograms assign datasets signatures, which serve for two purposes: i) to measure distance between datasets by directly comparing signatures; and ii) to perform classification.

References

1. Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *SIGMOD*, 2003.
2. Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.
3. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
4. Sergey Brin. Near neighbor search in large metric spaces. In *VLDB*, Switzerland, 1995.
5. Thomas M. Cover. *Elements of Information Theory*. Wiley-Interscience, 1991.
6. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.
7. L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.
8. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *SIGKDD*, pages 97–106, San Francisco, CA, 2001. ACM Press.
9. A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 231–238. MIT Press, 1995.
10. Junshui Ma and Simon Perkins. Online novelty detection on temporal sequences. In *SIGKDD*, 2003.
11. M. A. Oliver and R. Webster. Kriging: a method of interpolation for geographical information systems. *International Journal Geographic Information Systems*, 4(3), 1990.
12. Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–403, 1996.
13. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.