The Setwise Stream Classification Problem

Charu C. Aggarwal
IBM T. J. Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY, USA
charu@us.ibm.com

ABSTRACT

In many applications, classification labels may not be associated with a single instance of records, but may be associated with a *data set* of records. The class behavior may not be possible to infer effectively from a single record, but may be only be inferred by an aggregate set of records. Therefore, in this problem, the class label is associated with a *set of instances both in the training and test data*. Therefore, the problem may be understood to be that of classifying a *set of data sets*. Typically, the classification behavior may only be inferred from the overall patterns of data distribution, and *very little information is embedded in any given record for classification purposes*. We refer to this problem as the *setwise classification problem*.

The problem can be extremely challenging in scenarios where the data is received in the form of a stream, and the records within any particular data set may not necessarily be received contiguously. In this paper, we present a first approach for real time and streaming classification of such data. We present experimental results illustrating the effectiveness of the approach.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

Keywords

Data Streams; Data Classification

1. INTRODUCTION

In many applications, classification labels are not associated with individual records, but with *groups of records* in the underlying data. Thus, each group of records is treated as an indivisible entity, along with an associated class label. In most cases, the classification behavior can only be inferred from the *overall distribution pattern* of the records in this entity, and a given record typically provides very little information about classification behavior. This kind of problem can arise in the context of a wide variety of applications:

- In a sensor application, a single event of significance may cause a large number of data records, and the nature of the event can only be classified on the basis of the *aggregate* patterns of these different records. Any single record is unlikely to provide any useful understanding of the underlying causality.
- A large supermarket chain may have thousands of stores, each of which is associated with buying behavior of different customers. It may be very difficult to make any prediction of the geographical information of a particular buying pattern from a single transaction. It is much easier to make predictions from *aggregates*.
- In an environmental application, many large scale predictions can often be made only on the basis of multiple samples of the data from different geographical regions. In such cases, the overall pattern of the data is more relevant than the value on any specific record.

This problem is referred to as the *setwise classification problem*. We note that each set may contain tremendous variations in the feature values of the given records, and in many applications, it is by analyzing the specific *patterns of these variations*, that the setwise entities can be assigned a particular class. In many cases, individual records cannot be meaningfully assigned to classes, because many different classes may contain very similar data points. In order to illustrate this point, we have shown the distribution of the data records in a two such entities belonging to different classes in Figure 1, which are marked by 'x' and 'o' respectively. At first sight, it would seem that there is no difference

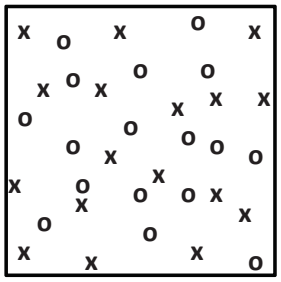


Figure 1: Setwise Classification Scenario

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623751 .

in the distributions of the two classes, and the two entities have identical distributions. The subtle differences in distribution can be understood only upon closer examination. In particular, the setwise entity belonging to the class marked 'x' has marginally greater concentrations at the peripheries of the data ranges, whereas the setwise entity belonging to the class marked 'o' has marginally greater concentration at the central regions. The average behaviors of the records in the two entities are also not very different. In such cases, it may not be meaningful to classify any *single* data point, because of the fact the differences in distributions are very subtle over different regions of the data. The example in Figure 1 is a very simple case, in which we have shown the relative behavior of only two such entities. In practice, there may be thousands of such entities with subtle variations in their data distributions and shapes. For example, another entity belonging to class 'x' may not necessarily be concentrated at the peripheries, but may have a slightly different shape which is characteristic of that class. Typically, there may be a small number of characteristic properties in the data distributions *for a particular class*, but different entities may use a different subset of these characteristic properties. When the entire data is viewed as a distribution of individual records across different classes, it may be virtually impossible to distinguish the instances at the lowest level. The challenge is to learn these subtle differences and similarities at the *entity-distribution level* efficiently in the stream scenario. Since the data points for a particular set entity are not received sequentially, this further increases the challenge from the perspective of stream processing [1, 6]. In this paper, we will design an efficient classification method which captures the subtle entity-specific characteristics in the form of classification signatures. These classification signatures essentially create a summarized model of the typical patterns in the underlying class behavior of different entities. We will show how to use this model for effective classification.

This paper is organized as follows. We will present related work in the remainder of this section. In section 2, we design efficient methods for classification of multi-set streams. The experimental results are presented in section 3. The conclusions and summary are present in section 4.

1.1 Related Work

The problem of classification has been widely studied in the machine learning and data mining literature [9, 12, 21, 23]. Surveys on data classification may be found in [7]. Recently, the problem has also been extended to the stream scenario. The earliest work in the area focussed on the extension of decision trees for stream classification [6, 11, 14, 15, 16, 22]. A popular ensemble classifier was proposed in [24], and the use of data selection for stream classification for explored in [13]. A method for on-demand classification of data streams was proposed in [2]. Recently, stream classification has also been adapted to the rare-class detection problem [8, 17, 18, 19]. A survey of stream classification may be found in [3]. All these methods are proposed for the model construction and classification of test instances.

A setwise method for the clustering problem has recently been proposed in [4]. Other methods have also been proposed in which test instances are classified on the basis of the a-priori knowledge that all *test* instances belong to the same class [20], or for labels to be attached to bags of training instances [10]. However, we note that these problems still attach labels to individual instances at a *fundamental* level both during training and testing. The specification of labels with bags of instances is simply a result of either unavailability of labels with individual training instances, or additional meta-information provided with test instances. Individual instances can be meaningfully classified to labels in these cases. In the case of [10], the attaching of labels to bags of training instances is simply an approximation because of unavailability of labels about individual training instances instances. In the case of [20], the additional knowledge that a set of *test* instances belong to the same class simply enhances the accuracy of classification of individual instances, which could otherwise be performed without this information. Our model is fundamentally different, because labels are attached to setwise entities both during training and testing, and it is not even meaningful to attach labels to individual instances of data records. Rather, the classification behavior of an entity can only be defined on the basis of the *distribution* of the records inside it. This is a much more challenging scenario for the classification process, and especially so in the stream scenario.

2. SETWISE STREAM CLASSIFICATION

We will first introduce the notations and definitions which are relevant to our work. We assume that the different setwise entities in the training stream are denoted by $\mathcal{D}_1 \ldots \mathcal{D}_N$. Each entity $\mathcal{D}_i$ is associated with the class label l_i. We assume that there are a total of k classes, and the class label is drawn from $\{1 \ldots k\}$. The dimensionality of each data set is d, and each data set has the same set of features. It is assumed that the ith data set $\mathcal{D}_i$ has n_i records. The jth record of the ith training entity is denoted by $\overline{X_j(i)}$. Since the dimensionality of the data is d, it is evident that the record $\overline{X_j(i)}$ is a vector containing d components. In addition, we have a set of n test entities, which are denoted by $\overline{\mathcal{T}_1 \ldots \mathcal{T}_n}$. The jth record of the ith test entity is denoted by $\overline{Z_j(i)}$.

Definition 1 (Setwise Classification). *Given a set of entities $\mathcal{D}_1 \ldots \mathcal{D}_N$, with associated labels $l_1 \ldots l_N$, construct a training model $\mathcal{M}$, which allows us to classify the different set entities $\mathcal{T}_1 \ldots \mathcal{T}_n$.*

The aforementioned problem definition is actually a simplification from the stream scenario. In the stream scenario, the individual records of an entity may be received in any particular order. Therefore, the individual records are tagged with the identifier of their entity, and also the corresponding entity label.

We assume that the data is received in the form of a stream $< \overline{Y_1}, entityid_1, label_1 > \ldots < \overline{Y_r}, entityid_r, label_r > \ldots$. The tuple $\overline{Y_r}$ represents a d-dimensional data point which could be *either* from the training or the test data, the notation $entityid_r$ represents the id of the entity, and $label_r$ represents the label of the underlying data. The value of $label_r$ is drawn from $\{1 \ldots k\}$, if the record $\overline{Y_r}$ is drawn from an entity belonging to the training data. Otherwise, the record $\overline{Y_r}$ is drawn from an entity belonging to the test data, and the value of $label_r$ is -1. Thus, the records are not only out of order, *but the records from the training and test stream may also be mixed with one another.* Furthermore, we note that the class label for a test entity can be predicted as soon as a sufficient number of records are received in order to be

able to predict its underlying distribution. As more records are received, the predicted label for a test entity will change (and typically become more accurate), because its underlying data distribution can also be estimated more accurately. The entity identifier is an integer drawn from $[1, N]$, if the entity is drawn from the training data. Otherwise, the entity identifier is an integer drawn from $[1, n]$. Therefore, assuming that $\overline{Y_r}$ is the jth data point of its entity, the value of $\overline{Y_r}$ is the same as either $\overline{X_j(entityid_r)}$ or $\overline{Z_j(entityid_r)}$ depending upon whether the rth record is a training record or test record respectively.

2.1 Setwise Model Maintenance

In this section, we will describe the process of setwise model maintenance of the different entities. The core idea is to create a set of supervised entity profiles which concisely characterize the distribution of the different entities. One possibility is to use a density estimation approach [25], which can model the relationships between the different classes and the corresponding data distributions. However, such an approach is unlikely to computationally efficient in the stream scenario, because of the need to track the densities over many different portions of the space; the number of such portions typically grows exponentially with the dimensionality of the space. Therefore, we will design a method which computes supervised class fingerprints in order to construct an effective classification model.

In order to concisely represent the distribution behavior of the different entities, we will use the concept of *fingerprints* in order to concisely summarize them. The idea of a fingerprint is to maintain the distribution patterns of the different entities in the data stream. These distribution patterns can be further leveraged to create concise representations of the profiles of different classes in the underlying data in *online fashion*. Clearly, such a model needs to be updated continuously. As more data points arrive in the stream, existing entity distribution patterns may evolve because of concept drift, and other entities with entirely new distribution patterns are received.

We note that the data points in the underlying data stream may belong to either labeled or unlabeled entities, depending upon whether they need to be used for model *building* (training data) or model-based *prediction* (test data). Clearly, this can be the case in many real applications, in which the model construction and prediction needs to be performed simultaneously. A given data point is used for model building, only if a label is associated with it, which is drawn from $\{1 \ldots k\}$. Otherwise, if the label value is -1, the entity is a test instance, and we update its current classification status with the new data record, which is added to the distribution of the test instance. This goal is achieved by modeling the entity distributions in relation to landmarks or anchor points that are generated from the underlying data. The entity-specific statistical information, which are also referred to as fingerprints, are used to dynamically maintain the sets of class profiles. These represent the common entity distributions which are relevant to the different classes. A given class may contain *multiple class profiles*, since there may be distributional differences in the profiles belonging to that class. For a given test entity, we determine its class label by using the relationship of the existing class labels to the *current distribution of* that test entity. If a given test entity is classified multiple times during stream progression, its reported class label may change, as more and more data points are received for that entity, and its underlying distribution changes. The changed distribution may sometimes more closely match the profile for another class. Such situations are quite natural, because the test entity can be known more accurately over time, as new data points arrive.

The supervised model maintenance process continuously maintains the entity-specific profiles, and the different class profiles which represent the key entity distribution characteristics. At the same time, we maintain the class-specific profiles of the different fingerprints. Both these statistics need to be maintained simultaneously in the stream scenario. Correspondingly, we also need two input parameters that correspond to the granularity of the fingerprint maintenance and that of class profile maintenance. The granularity of the fingerprints is regulated by the parameter q, which represents the number of anchor points used for fingerprint construction. An additional input to the algorithm is the parameter p, which represents the total number of class profiles tracked in the data. The value of p is typically much larger than k, which is the total number of classes in the data. This ensures that the variations in the distributions of the entities belonging to a particular class can be captured by one of these class profiles. We will denote the q different anchor points by $\overline{W_1} \ldots \overline{W_q}$. The simultaneous maintenance of the class profiles together with the fingerprints can be significantly challenging, because the former depends upon the latter, and the latter can change significantly, as new data points arrive and the distributions of the different entities evolve over time. The concept of *fingerprints* [4] is formally defined as follows;

DEFINITION 2 (FINGERPRINTS). *Let the entity containing the data points $\overline{Y_1} \ldots \overline{Y_r}$ be partitioned into the q clusters $\mathcal{C}_1 \ldots \mathcal{C}_q$, with anchors $\overline{W_1} \ldots \overline{W_q}$ respectively, where each data point is assigned to its closest anchor. The corresponding (relative) cluster frequencies are denoted by $f_1 \ldots f_q$, where $\sum_{i=1}^{q} f_i = 1$. Then, the fingerprint of the set of data points $\overline{Y_1} \ldots \overline{Y_r}$, defined with respect to these anchors is denoted by the q-dimensional tuple $[f_1 \ldots f_q]$.*

We denote the fingerprint with respect to anchors $\overline{W_1} \ldots \overline{W_q}$ and set of data points $\mathcal{D}$ by $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{D})$. It is important to notice that the fingerprints provide a way to represent the distribution of the data points in a given entity. As we will see later, this turns out to be very useful from the perspective of classification.

In order to dynamically maintain the classification profiles, a total of q anchors $\overline{W_1} \ldots \overline{W_q}$ are used during classification. These anchors are fixed throughout the algorithm execution, and are generated in an initial step with the use of a k-means approach. Let $\mathcal{E}_i(t)$ be the subset of the training entity $\mathcal{D}_i$, at the time of the arrival of the tth point in the data stream. The fingerprints for all the entities encountered so far, are actively maintained, and are denoted by $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{E}_1(t)) \ldots \mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{E}_N(t))$. Depending upon the number of such entities which have been encountered so far, these fingerprints can either be maintained in main memory, or they can be maintained on disk. If the training and test entities are arriving simultaneously, then the fingerprints of the test entities are maintained dynamically. As the test entity becomes successively more refined with the arrival of new data points, it is successively reclassified with its new representation. Presumably, the new

fingerprint is a more accurate representation of the class structure and behavior. Let $\mathcal{H}_i(t)$ be the subset of the test entity $\mathcal{T}_i$, at the time of arrival of the tth point in the data stream. We corresponding fingerprint is denoted by $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{H}_i(t))$.

These finger prints are used in order to maintain the different *class distribution profiles*. Multiple class distribution profiles may be associated with a single class. This reflects the fact that many entity distributions may be relevant to a particular class. A class distribution profile is constructed from a set of fingerprints, *all of which belong to the same class*. Therefore, we define a class distribution profile as follows:

DEFINITION 3 (CLASS DISTRIBUTION PROFILES). *The class profiles for a set $S = \{\mathcal{L}_1, \ldots \mathcal{L}_s\}$ of q-dimensional fingerprints, which* **belong to the same class label** *l is defined as the following $(q+2)$-tuple containing the following components:*

- *We maintain q values containing the sum of each fingerprint component and denote the vector by $\overline{AG(\mathcal{S})}$.*
- *We maintain the number of fingerprints in the cluster, which is denoted by $n(\mathcal{S}) = |\mathcal{S}| = s$.*
- *We maintain the class label l of each fingerprint.*

We will next describe the overall process of fingerprint and class profile maintenance. As mentioned earlier, the fingerprints are actively maintained by the algorithm. Since the fingerprints are generated with the use of anchor points, it is important to have an efficient method for generation of the these anchors in the initial phase of the algorithm. Ideally, the anchor points should be dense regions of the data, around which the pattern of data points can be analyzed. Therefore, we use a sample of the data points, denoted by *InitSample*, and we apply a k-means clustering algorithm on these data points in order to generate these initial anchors. The initial anchors are the centroids of the clusters determined on the initial sample of the data points. We denote these anchors by $\overline{W_1} \ldots \overline{W_q}$.

In the event that the tth incoming point $\overline{Y_t}$ is from a training entity set (and belongs to entity set $\mathcal{D}_j$), we denote the subset of entity set $\mathcal{D}_j$ to which the point belongs by $\mathcal{U}_j(t)$. In the event that the tth incoming point is from a test entity (and belongs to entity set $\mathcal{T}_j$), we denote the subset of entity set $\mathcal{T}_j$ to which the point belongs by $\mathcal{V}_j(t)$. This data point is then assigned to its closest anchor point in the data. In order to compute the closest anchor point, we compute the euclidian distance between the data point $\overline{Y_t}$ and the anchors $\overline{W_1} \ldots \overline{W_q}$. Once the assignment has been performed, the fingerprints are correspondingly updated. The update process of a fingerprint needs to account for the addition of the new data point. The index of the selected anchor is denoted by m. The fingerprints for the data subsets before and after modification are denoted by $[f_1 \ldots f_q]$ and $[f'_1 \ldots f'_q]$ respectively. Then, if the tth incoming data point is a training point, the new fingerprint after the addition of a data point is denoted as follows:

$$f'_i = \begin{cases} \frac{f_i \cdot |U_j(t)|}{|U_j(t)|+1} & i \neq m \\ \frac{f_i \cdot |U_j(t)|+1}{|U_j(t)|+1} & i = m \end{cases} \quad (1)$$

Algorithm *Set-Classify*(*Stream:* $\mathcal{Y}$, *Profiles:* p, *Anchors:* q);
begin
  Generate q anchors $\overline{W_1} \ldots \overline{W_q}$ on first *InitSample* points;
  Divide p class profiles into k different classes in proportion to initial class presence, and denote by $p_1 \ldots p_k$;
  Cluster class-specific segment i with parameter p_i;
  $t = 1$;
  for each $< \overline{Y_s}, j > \in \mathcal{Y}$ **do begin**
    if $j \geq 0$ (training data point) **do begin**
      Determine closest anchor $\overline{W_r}$ to data point $\overline{Y_s}$;
      Update the fingerprint $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{E}_j(t))$ with data point $\overline{Y_s}$ by adding to rth bucket;
      if fingerprint has $< min\_stat$ points **then** add to tentative set $\mathcal{R}$;
      else begin
        Determine closest profile with same class to $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{E}_j(t))$;
        Re-assign entity to closest class-profile if necessary;
        Update class profile statistics;
      end;
    end;
    else begin { Data point from test entity }
      Determine closest anchor $\overline{W_r}$ to data point $\overline{Y_s}$;
      Update the fingerprint $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{H}_j(t))$ with data point $\overline{Y_s}$ by adding a data point to rth bucket and updating relative frequencies;
      Determine closest profile to fingerprint $\mathcal{F}(\overline{W_1} \ldots \overline{W_q}, \mathcal{H}_j(t))$;
      report label of closest class profile;
    end
  $t = t + 1$;
  end
end

Figure 2: Setwise stream classification

In the event that the tth incoming data point is a test point, the new fingerprint after addition of the test point is as follows:

$$f'_i = \begin{cases} \frac{f_i \cdot |V_j(t)|}{|V_j(t)|+1} & i \neq m \\ \frac{f_i \cdot |V_j(t)|+1}{|V_j(t)|+1} & i = m \end{cases} \quad (2)$$

The denominator always increases by one point since the denominator represents the number of data points in the entity. However, the numerator increases by one for the case of the mth centroid, since only the fingerprint bucket for that bin has been updated.

At the same time, we maintain the class profiles for the different data points. Each class profile consists of a set of closely related fingerprints, all of which belong to the same class. A single class may of course contain multiple profiles, since there may be different entity distributions belonging to the same class. Only the entities which belong to the training data are used to update the class profiles. We assume that the p different class profiles which are consistently maintained are denoted by $\mathcal{P}_1 \ldots \mathcal{P}_p$. Each class profile $\mathcal{P}_i$ contains a number of different features which are tracked with it, according to Definition 3. The entities that belong to the test data are not assigned to any of the profiles. Rather, the fingerprints of these entities are continuously constructed over the course of the data stream and are used in order to perform the successively more accurate classifi-

cations as it gradually becomes possible to characterize the entity more and more accurately over time with the use of its fingerprint.

The average fingerprint characterization of the entities in a class profile $\mathcal{P}_i$ can be obtained by dividing each of the bins in $AG(\mathcal{P}_i)$ by $n(\mathcal{S}_i)$. This averaged profile can be very useful for computing the similarity between a fingerprint and an averaged class profile. For each incoming data point, we update its underlying entity fingerprint, and assign it to its closest class profile. The closeness of an entity fingerprint to the class profile is computed with the use of the cosine distance between the fingerprint and the averaged class profile. Because an entity may contain many points, the fingerprint will be repeatedly updated, after it has already been assigned to a particular class profile. Correspondingly, the class-profile assignment may also vary with the arrival of additional data points.

In order to meaningfully assign a fingerprint to a class profile, the fingerprint needs to have a sufficient number of data points, so that it can be characterized in a statistically robust way. Therefore, we maintain a *tentative set* of fingerprints $\mathcal{R}$, which correspond to those entities for which a sufficient number of data points have not been received so far. We impose the condition that any fingerprint needs to have a minimum of *min_stat* data points in order to meaningfully assigned to a class profile. Otherwise, it is assigned to the tentative set $\mathcal{R}$, where it "waits" for more individual data points to be included in it. Therefore, a fingerprint will always be assigned to a tentative set at the initial stage of the algorithm. The first time that a fingerprint contains more than *min_stat* data points, it will be assigned to one of the p class profiles denoted by $\mathcal{P}_1 \ldots \mathcal{P}_p$.

The other main issue is that each of the p different class profiles belongs to a specific label from the k different classes. In order to define the number of representatives of each class among the class profiles, we use the initial sample of *InitSample* points. Let $r_1 \ldots r_k$ be the fraction of entities which belong to the k different classes in this initial sample. Then, the p different class profiles are divided up among the k different classes in proportion to $r_1 \ldots r_k$. Specifically, the j-th class is assigned $\lfloor p \cdot r_j + 0.5 \rfloor$ different profiles. Furthermore, each class is assigned at least one profile, in the event that $\lfloor p \cdot r_j + 0.5 \rfloor$ evaluates to 0. Note that the profile distribution among the different classes may not necessarily sum to p. In the event that the number of profiles (assigned to the different classes) is more than p, we remove one profile allocation from each of the classes with the largest number of assigned profiles in decreasing order, until the total number is p. In the event that the number of profiles is less than p, we add one profile allocation to each of the classes with the least number of assigned profiles in increasing order, until the total number is p. We assume that the number of profiles allocated to the class i is denoted by p_i. Therefore, we have $\sum_{i=1}^{k} p_i = p$.

As an initial step in the algorithm, we use the *InitSample* (training) points in the data stream in order to create an initial profile set of classes. This is achieved by applying a k-means clustering algorithm to the entities (with at least *min_stat* points) from the initial sample of points. For each class i, a different k-means clustering is applied with the use of p_i centroids. The set of entities in each cluster are used to create a different class profile. Thus, a total of p profiles are available, which are distributed among the different classes. The entities which do not contain at least *min_stat* points are assigned to the tentative set $\mathcal{R}$.

Subsequently, the online profile maintenance and classification processes are executed simultaneously. The training and test data points may be mixed with one another. The training data points are used for profile maintenance, whereas the test data points are used for online classification. For each incoming (training) data point, we first update its entity fingerprint by adding the data point to the appropriate anchor bin within the fingerprint. At this point, if the fingerprint does not contain at least *min_stat* points, we allow it to stay in the tentative set $\mathcal{R}$. Otherwise, we determine the closest class profile **with the same class** among the current set of profiles, and update that class profile with the modified fingerprint. This process of modification requires us to remove the fingerprint from the profile to which it was assigned earlier, and assign it to a new profile. This requires us to subtract out the fingerprint from the old profile and add it to the new profile. Because the information stored in a profile is additive in nature, it is relatively easy to subtract out the information in one profile and add it to another. In some cases, the addition of a point to an entity may cause it to exceed the threshold of *min_stat*. In such cases, the setwise entity is moved out of the tentative set and added to the closest entity. In this case, the profile only needs to be updated in terms of adding an entity to the profile, but it does not need to be subtracted from a profile.

The fingerprints for the test entities are also maintained simultaneously with the training entities. For each incoming (test) data point, the fingerprint of its test entity is updated. These test entities are maintained in a separate test profile set $\mathcal{Q}$. Each time a fingerprint in $\mathcal{Q}$ is updated, it is assumed that the fingerprint is now presumably more "accurate" because of the addition of more data points to it. Therefore, this refined fingerprint is classified with the use of the training profiles already stored. The classification process determines the closest training profile to the test fingerprint with the use of the cosine metric. The label of this training profile is reported as the relevant label at that instant. This approach essentially re-classifies a test entity every time new information arrives in the stream in order to characterize it more accurately. This seems like a logical way to perform the classification in a scenario in which the entire entity is never available at any given time. The basic algorithm for classification is illustrated in Figure 2.

One observation about the process is that the change to each training entity fingerprint of $\mathcal{U}(t)$ is relatively small when the absolute number of elements in these sets (denoted by $|\mathcal{U}(t)|$) is large. Therefore, it is not necessary to update the class profile for a fingerprint, each time it is updated. Rather, we only update the fingerprint of the entity, but do not use it to update the class profile at all in most iterations. In fact, other than updating the fingerprint of the entity, we do not make any operation (such as cosine distance computations) on the class profiles at all. In order to decide the periods at which fingerprints have changed "sufficiently" in order to allow updates, we maintain an additional quantity for each training entity $\mathcal{U}(t)$ denoted by $\#^{\mathcal{U}(t)}$ at any given instant t. This quantity denotes the number of updates to the entity $\mathcal{U}(t)$ since the last time it was used to update a class profile. The quantity is updated each time the fingerprint is updated, and is reset to 0 every time the fingerprint is used to update a class profile. As long as the

entity $\mathcal{U}(t)$ has a significantly larger number of points than the number of updates to it, we know that the fingerprint has not changed significantly enough to worry about issues with training. When the value of $\#^{\mathcal{U}(t)}/|\mathcal{U}(t)|$ is less than a pre-defined fraction (say 1%), we do not perform the updates. The updates are performed, only when this threshold is reached. An immediate observation is that the frequency of class profile updates will reduce with progression of the data stream, as the number of points in the fingerprint become larger and larger. Since a specific *percentage* of points in the fingerprint need to be added in order for the class profiles to be updated, it is evident that the frequency of profile updates reduces, as the *absolute* number of points in each fingerprint increases over the progression of the stream. In reality, the updates to the class profiles tend to be relatively infrequent in steady state, and continuously reduces over time. This results in an increasing efficiency of the algorithm with stream progression.

3. EXPERIMENTAL RESULTS

In this section, we present results on two real and one synthetic data sets. For the real data sets, the class labels were either available with the data set, or could be derived from one of the features of the data sets. For the synthetic data sets, each class was generated from a mixture model gaussian distribution.

3.1 Data Sets

We used two real data sets and one synthetic data set in order to test our approach. The classes in these data sets were highly mixed with one another, and could not be distinguished from one another on the basis of individual data points. These data sets are referred to as *Hub64 Data Stream*, *E-Cover Data Streams*, and *Gauss50Mix10D1000 Data Stream*. We describe these data sets in detail below:

- **Hub64 Data Stream:** The data stream was generated from the *Hub64 data set* and contains the voice signal of 64 different public personalities such as Wolf Blitzer, Bill Clinton, Al Gore, Candy Crowley etc. As we will see later, the identity of the speakers will be useful in order to judge the effectiveness of the clustering process. We derived an 8-dimensional GPCC format from the compressed domain. Each record in this format contained a micro-second sample of the underlying speech with the use of different features such as pitch, amplitude etc. The samples from two different speakers could be very similar, when this sample corresponds to a moment of silence or other sounds which are not easily distinguishable across different speakers. In such cases, the speakers can only be identified from *multiple sets* of records rather than individual records which did not contain meaningful information. In order to create the multi-set entities, we segmented each speaker data into 10 different multi-set entities, each of which was given a different multi-set identifier. This created a total of 640 entities spread out over a stream of 397,041 data points. We randomly mixed the data points for the different speakers, and created a stream of data points that were tagged with the corresponding multi-set identifiers. A total of 80% of the entities were treated as training data, whereas the remaining was treated as test data. The test and training data were randomly mixed with one another, which made the problem much more challenging.

- **E-Cover Data Stream:** This was a derivative data stream from the *forest cover data set*, which was designed to make it more challenging for the multi-set clustering problem. The original version of the problem was labeled with cover-type. We used the first attribute (elevation), in order to create the different multi-set entities. This is because the elevation attribute is very noisily related to the different attributes, and the classification can only be inferred from the overall distribution of the multi-set entities with the same elevation. Let μ and σ be the mean and standard deviations of the different elevations. The elevations were divided into 10 different categories, based on 9 different discretization points corresponding to $\mu - 1.3 \cdot \sigma$, $\mu - 0.7 \cdot \sigma$, $\mu - 0.3 \cdot \sigma$, $\mu - 0.1 \cdot \sigma$, μ, $\mu + 0.1 \cdot \sigma$, $\mu + 0.3 \cdot \sigma$, $\mu + 0.7 \cdot \sigma$, and $\mu + 1.3 \cdot \sigma$ respectively. We divided each of the 10 different categories into 150 different multi-set entities, which resulted in 1500 entities being spread over a stream of 581,012 records. The order of the records was randomized in the data set in order to create the stream. Each multi-data entity was tagged with the corresponding multi-set identifier. We removed the elevation attribute from the data set for clustering purposes, though the discretized value was retained as meta-information for evaluation purposes. Each of the quantitative attributes was normalized by its standard deviation. As before, used 80% of the entities are training entities and 20% of the entities as test entities.

- **Gauss50Mix10D1000 Data Stream:** The third data stream was a 10-dimensional synthetic data stream which was generated with the use of gaussian mixture models. We generated 50 different classes, each of which was a mixture model of 10 different clusters. Furthermore, the clusters for different classes overlapped with one another. Since each class was a mixture model containing many clusters The center of each cluster was chosen from a uniform distribution in $[0.1, 0.9]$, and the radius of each cluster along each dimension was chosen from a uniform distribution in $[1, 1.5]$. Note that the radius of each cluster is greater than the range from which the centers are drawn. Furthermore, since each multi-set distribution is drawn from a mixture model of 10 different clusters with randomly generated centers, it follows that the data sets from the different distributions will be highly overlapping with one another in terms of clustering behavior. An example of a 2-dimensional cross-section of a sample of 4 of the distributions is illustrated in Figure 1. It is evident that the different distributions are highly mixed with one another and show little clustering in terms of individual data points. The number of points in each of the $50 * 10 = 500$ different constituent clusters of the mixture model was generated from the Zipf distribution $1/(i + 100)^{\theta}$. The parameter of the Zipf distribution was chosen to 0.1. A total of 10^6 data points were distributed across the different clusters. Each of the dimensions was normalized so as to equalize the standard deviation across all attributes. For each of the 50 mixture model distributions, we tagged

the corresponding data points with one of 20 different multi-set entities. Thus, the 10^6 different data points were distributed across 1000 different multi-set entities. We set 80% of the entities as training entities and the remaining as test entities. As in previous cases, the data points from the different entities were randomly mixed with one another in the form of a data stream.

3.2 Baseline Approach

We created multi-variate density distributions in order to characterize the different entities, and used these density distributions in order to construct the training models. The density distributions are constructed as sampled grid points in the data space. The number of such grid points are exponentially related to the number of dimensions. For each dimension, we had 3 possible grid points at $\mu_i - \sigma_i$, μ_i, and $\mu_i + \sigma_i$, where μ_i and σ_i are the mean and standard deviation of dimension i. Therefore, for a d-dimensional data set, we have 3^d possible grid points. Since this number can be very large, it is necessary to sample grid-points in order to create the density profiles in a reasonably efficient way. We used these density profiles were used as a surrogate for the fingerprints in conjunction with the approach in Figure 2 as the baseline technique. In order to create a comparable computational efficiency with the multi-set clustering technique, we constructed density estimates on as many grid points as the number of anchors which were used by our multi-set clustering technique. We used kernel density estimation on the incoming stream in order to create the density profiles at the grid points.

3.3 Effectiveness Results

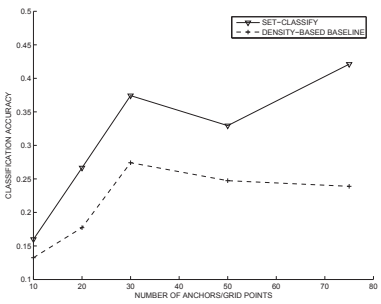
All results were tested on an Lenovo X201 Thinkpad running Microsoft Windows XP, with a 2.4GHz CPU and 2.9 GB of main memory. The code was implemented with Microsoft Visual C++. In all cases, unless otherwise mentioned, the default number of anchors (or grid-points) used was 50, the default number of classification profiles was 80, and default initial number of sample points was 10,000. We first tested the classification accuracy with increasing number of anchors. The results for the *Hub64 data stream*, *E-cover data stream* and the *Gauss50Mix10D1000* data stream are illustrated in Figures 3(a), (b), and (c) respectively. The number of anchors (or grid-points for the baseline scheme) is illustrated on the X-axis, whereas the classification accuracy is illustrated on the Y-axis. The number of classification profiles was set to its default value of 80. In most cases, an increase in the number of anchors increased the classification accuracy, though this was not always the case. This is because a certain amount of noise is associated with the anchor construction process, and a large number of anchors may not always imply a higher classification accuracy. In all cases, the accuracy of the *Set-Classify* scheme was *significantly* higher than the density-based baseline. For example, in the case of the *Hub64* data stream, the classification accuracy of the *Set-Classify* scheme was greater than 40% with the use of 75 anchors and 80 classification profiles, whereas the classification accuracy of the density-based baseline was about 25%. This difference was even larger for the other two data sets. For example, for the case of the *E-Cover* data set, the density-based scheme performed disastrously with a classification accuracy of only about 20% at these same parameter values. On the other hand, the *Set-Classify* scheme had an accuracy which was greater than 80%. The different between the two schemes was also very significant in the case of the synthetic data set.

We also tested the classification accuracy of the schemes with increasing number of classification profiles. The results for the *Hub64 data stream*, *E-cover data stream* and the *Gauss50Mix10D1000* data stream are illustrated in Figures 3(d), (e), and (f) respectively. The number of classification profiles are illustrated on the X-axis, whereas the classification accuracy is illustrated on the Y-axis. The number of anchors was set to the default value of 50. As in the previous case, the *Set-Classify* scheme was significantly more accurate than the baseline methods. The other observation was that the trend with increasing classification profiles was much less clear as compared to the trend with increasing number of anchors. This is because an increase in the number of profiles increased the representation granularity (which improves accuracy), but also reduced the number of points in each profile (which reduces its robustness and therefore the overall accuracy).

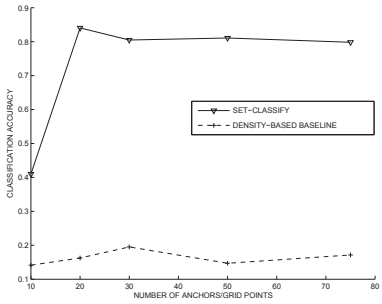
Finally, we also tested the classification accuracy with progression of the data stream. The results for the *Hub64 data stream*, *E-cover data stream* and the *Gauss50Mix10D1000* data stream are illustrated in Figures 3(g), (h), and (i) respectively. The progression of the data stream (in terms of the number of points) is illustrated on the X-axis, whereas the classification accuracy (on the last block of points) is illustrated on the Y-axis. The number of anchors and classification profiles was set to 50 and 80 respectively. We note that the progression of the stream on the X-axis is illustrated in terms of the number of *test data points*, which are mixed randomly with the training data points. It is clear that in each case, the classification accuracy increases rapidly with stream progression, but stabilizes over time, as the distribution of classification profiles reaches steady state. As in the previous cases, the classification accuracy of the *Set-Classify* method is significantly greater than the density-based baseline for all the three data sets.

3.4 Efficiency Results

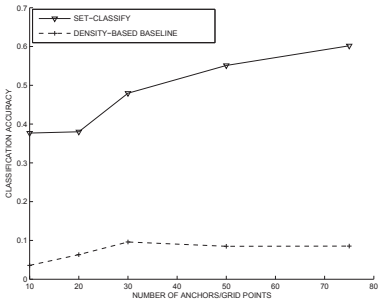
We also tested the efficiency of the scheme in terms of the number of data points processed per second. The processing rates for training and testing wee tabulated separately by separating out the running times for the algorithm loops which correspond to the training and testing data. The initialization times were charged to the training rates. We tested the processing rates with increasing number of anchors. The results for the *Hub64 data stream*, *E-cover data stream* and the *Gauss50Mix10D1000* data stream are illustrated in Figures 4(a), (b), and (c) respectively. The number of anchors are illustrated on the X-axis, whereas the processing rate (in terms of the number of data points processed per second) are illustrated on the Y-axis. The number of classification profiles was set to its default value of 80. The processing rates for all schemes reduces with an increasing number of anchors. This is because an increase in the number of anchors increases the amount of time it requires to update each fingerprint. It is evident that both the training and testing rates for the *Set-Classify* scheme are significantly greater than that for the baseline scheme. This is because the fingerprint generation process is much more efficient than the process of density estimation. Furthermore, the training rates tend to be significantly higher than the



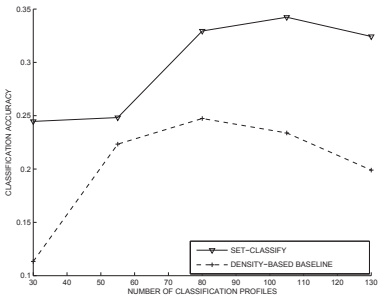
(a) Classification Acc. vs. Number of Anchors (or Grid Points) (*Hub64* Data Set)



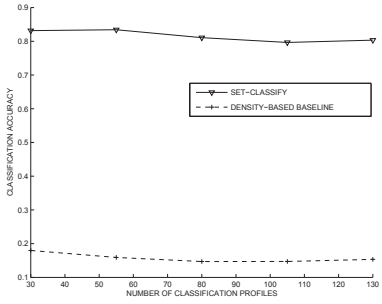
(b) Classification Acc. vs. Number of Anchors (or Grid Points) (*E-Cover* Data Set)



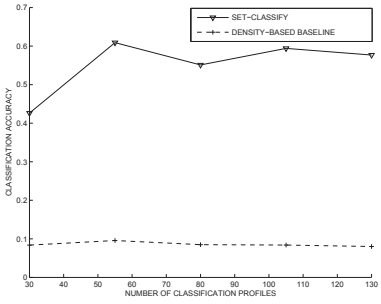
(c) Classification Acc. vs. Number of Anchors (or Grid Points) (*Gauss50Mix10D1000* Data)



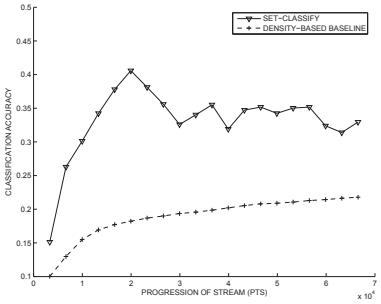
(d) Classification Acc. vs. Number of Clusters (*Hub64* Data Set)



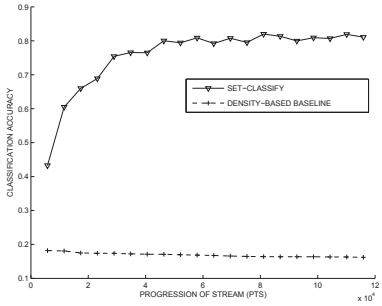
(e) Classification Acc. vs. Number of Clusters (*E-Cover* Data Set)



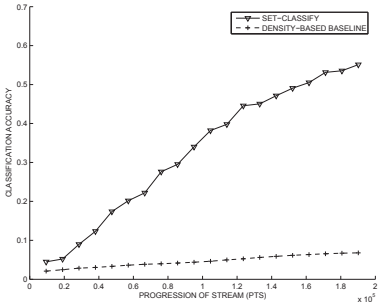
(f) Classification Acc. vs. Number of Clusters (*Gauss50Mix10D1000* Data)



(g) Classification Accuracy with Stream Progression (*Hub64* Data Set)

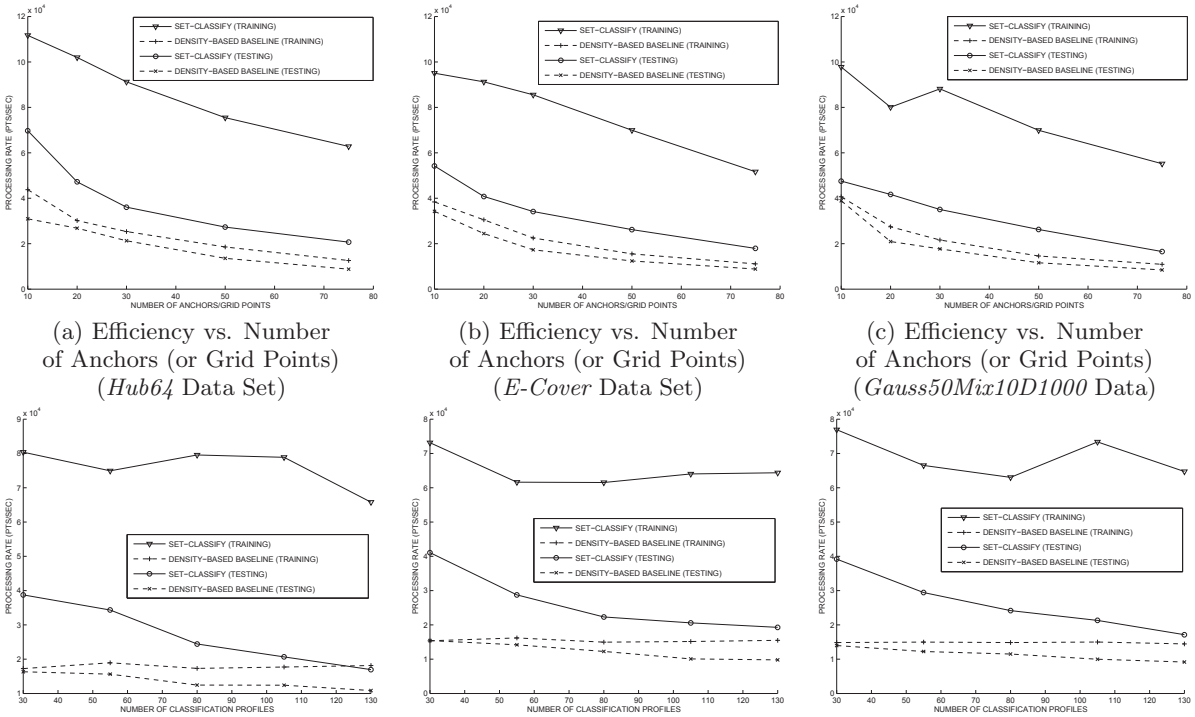


(h) Classification Accuracy with Stream Progression (*E-Cover* Data Set)



(i) Classification Accuracy with Stream Progression (*Gauss50Mix10D1000* Data)

Figure 3: Effectiveness Results



(a) Efficiency vs. Number of Anchors (or Grid Points) (*Hub64* Data Set)

(b) Efficiency vs. Number of Anchors (or Grid Points) (*E-Cover* Data Set)

(c) Efficiency vs. Number of Anchors (or Grid Points) (*Gauss50Mix10D1000* Data)

(d) Efficiency vs. Number of Clusters (*Hub64* Data Set)

(e) Efficiency vs. Number of Clusters (*E-Cover* Data Set)

(f) Efficiency vs. Number of Clusters (*Gauss50Mix10D1000* Data)

Figure 4: Efficiency Results

testing rates. The reason for this was that the fingerprint of a training entity only needed to be compared with the class profiles which belonged *to the same class*, whereas the fingerprint of a testing entity need to be compared with the class profiles of *every class*. This ensured that the training rates were always much higher than the testing rates. Furthermore, the absolute training and testing rates were of the order of several thousand data points per second. This ensures that very large volumes of streams could be processed with the use of such an approach.

We also tested the processing rates with increasing number of classification profiles. The results for the *Hub64 data stream*, *E-cover data stream* and the *Gauss50Mix10D1000* data stream are illustrated in Figures 3(d), (e), and (f) respectively. The number of classification profiles are illustrated on the X-axis, whereas the processing rates is illustrated on the Y-axis. The number of anchors was set to the default value of 50. One observation was that while the processing rates reduces with increasing number of classification profiles, they were not quite as sensitive to this parameter, as they were to the number of anchors. The reason for this is that most of the time is spent in constructing the fingerprints, and this phase is essentially independent of the number of class profiles. Therefore, the processing rates are only mildly sensitive to the number of classification profiles. As in the previous case, the processing rates of the *Set-Classify* scheme are significantly greater than that of the density-based classification scheme. Furthermore, these processing rates are typically of the order of several thousands data points per second. This ensures that our approach is an effective and efficient method for setwise classification of data streams.

4. CONCLUSIONS AND SUMMARY

Setwise stream classification scenarios are increasingly common, where one needs to classify an entire set of data records, rather than an individual data point. An additional complicating factor is that the stream points from different multi-set entities may be mixed with one another. This can make the classification process much more challenging. This paper designs a technique for online generation of classification profiles, which are used for the purposes of classification. We presented experimental results illustrating its effectiveness and efficiency.

5. REFERENCES

[1] C. Aggarwal, J. Han, J. Wang, and P. Yu. A Framework for Clustering Evolving Data Streams, *VLDB Conference*, pp. 81–92, 2003.

[2] C. Aggarwal, J. Han, J. Wang, P. Yu. On Demand Classification of Data Streams, *ACM KDD Conference*, pp. 503–508, 2004.

[3] C. Aggarwal. A Survey of Stream Classification Algorithms, *Data Classification: Algorithms and Applications*, CRC Press, 2014.

[4] C. Aggarwal. The Multi-Set Stream Clustering Problem, *SIAM Conference on Data Mining*, pp. 59–69, 2012.

[5] C. Aggarwal. On Segment-based Stream Modeling and its Applications, *SIAM Conference on Data Mining*, pp. 721–732, 2009.

[6] C. Aggarwal. *Data Streams: Models and Algorithms*, Springer, New York, 2007.

[7] C. Aggarwal. *Data Classification: Algorithms and Applications*, CRC Press, Boca Raton, FL, 2014.

[8] T. Al-Khateeb, M. Masud, L. Khan, C. Aggarwal, J. Han, and B. Thuraisingham. Stream Classification with Recurring and Novel Class Detection Using Class-Based Ensemble. *IEEE ICDM Conference*, pp. 31–40, 2012.

[9] T. M. Cover, and P. E. Hart. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 13(1), pp. 21–27, 1967.

[10] T. Dietterich, R. Lathrop, and T. Lozano-Perez. Solving the Multiple-Instance Problem with Axis-Parallel Rectangles. *Artificial Intelligence*, 89, pp. 31–71, 1997.

[11] P. Domingos, and G. Hulten. Mining High-Speed Data Streams. *ACM KDD Conference*, pp. 71–80, 2000.

[12] R. Duda, P. Hart, and D. Stork. *Pattern Classification*, Wiley-Interscience, 2000.

[13] W. Fan. Systematic Data Selection to Mine Concept Drifting Data Streams, *ACM KDD Conference*, pp. 128–137, 2004.

[14] J. Gama, R. Rocha, and P. Medas. Accurate Decision Trees for Mining High-Speed Data Streams, *ACM KDD Conference*, pp. 523–528, 2003.

[15] G. Hulten, L. Spencer, and P. Domingos. Mining Time Changing Data Streams. *ACM KDD Conference*, pp. 97–106, 2001.

[16] R. Jin, and G. Agrawal. Efficient Decision Tree Construction on Streaming Data, *ACM KDD Conference*, pp. 571–576, 2003.

[17] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham. Addressing Concept-Evolution in Concept-Drifting Data Streams. *IEEE ICDM Conference*, pp. 929–934, 2010.

[18] M. Masud, T. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham. Detecting Recurring and Novel Classes in Concept-Drifting Data Streams. *IEEE ICDM Conference*, pp. 1176–1181, 2011.

[19] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, A. Srivastava, and N. Oza. Classification and Adaptive Novel Class Detection of Feature-Evolving Data Streams, *IEEE Transactions on Knowledge and Data Engineering*, 25(7), pp. 1484–1497, 2013.

[20] X. Ning, and G. Karypis. The Set Classification Problem and Solution Methods. *SIAM Conference on Data Mining*, pp. 847–858, 2009.

[21] J. R. Quinlan. C4.5: Programs for Machine Learning, *Morgan Kaufmann*, 1993.

[22] S. Ruping. Incremental Learning with Support Vector Machines. *IEEE ICDM Conference*, pp. 641–642, 2001.

[23] V. Vapnik. *The Nature of Statistical Learning Theory*, Springer, New York, 1995.

[24] H. Wang, W. Fan, P. Yu, J. Han. Mining Concept-Drifting Data Streams using Ensemble Classifiers. *ACM KDD Conference*, pp. 226–235, 2003.

[25] T. Zhang, R. Ramakrishnan, M. Livny. Fast Density Estimation Using CF-Kernel for Very Large Databases. *ACM KDD Conference*, pp. 312–316, 1999.