2011 11th IEEE International Conference on Data Mining

Heuristic Updatable Weighted Random Subspaces for Non-stationary Environments

T. Ryan Hoens
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
thoens@cse.nd.edu

Nitesh V. Chawla
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, Indiana 46556
nchawla@cse.nd.edu

Robi Polikar
Electrical and Computer Engineering
Rowan University
Glassboro, NJ 08028
polikar@rowan.edu

***Abstract*—Learning in non-stationary environments is an increasingly important problem in a wide variety of real-world applications. In non-stationary environments data arrives incrementally, however the underlying generating function may change over time. While there is a variety of research into such environments, the research mainly consists of detecting concept drift (and then relearning the model), or developing classifiers which adapt to drift incrementally.**

We introduce Heuristic Updatable Weighted Random Subspaces (HUWRS), a new technique based on the Random Subspace Method that detects drift in individual features via the use of Hellinger distance, a distributional divergence metric. Through the use of subspaces, HUWRS allows for a more fine-grained approach to dealing with concept drift which is robust to feature drift even without class labels. We then compare our approach to two state of the art algorithms, concluding that for a wide range of datasets and window sizes HUWRS outperforms the other methods.

***Keywords*-Concept Drift; Hellinger Distance; Random Subspaces; Non-stationary learning;**

I. Introduction

One of the current topics in data mining research is learning from non-stationary data streams. In such learning scenarios, data arrive incrementally (or in batches), and, over the course of time, the underlying data generation function may change. Learning in such environments requires the classifier be able to respond to such changes, while ensuring that it still retains all (relevant) past knowledge. These two competing forces bring rise to what is known as the stability-plasticity dilemma [11]. This issue is further complicated if data cannot be stored due to space constraints.

Since in non-stationary environments the underlying generating function changes — or *drifts* — over time, such environments are said to exhibit *concept drift*. Typically when referencing concept drift, the drift is further specified as being *real* (i.e., the class conditional changes) or *virtual* (i.e., the prior probabilities change) [33]. While concept drift can be "real" or "virtual", it can also occur gradually, instantaneously (sometimes called *concept change*), or in a cyclic manner (sometimes called a *recurring context*).

In traditional incremental learning environments, instances arrive sequentially, i.e., an unlabeled instance first arrives for testing, and then, once a prediction is given, a label is made available to update the classifier. Alternatively, data may arrive in batches, i.e., a non-empty set of instances first arrive unlabeled for testing. Once predictions have been given for all instances in the set, the labels for each of the instances are made available in order to update the classifier.

For the sake of completeness, we consider both types of drift (i.e., "real" and "virtual") occurring gradually, as sudden concept change, or as a recurring concept. Due to the nature of our algorithm, however, we consider the case where instances arrive in batches instead of on-line. We discuss our future plans on mitigating this shortcoming in Section IX.

Contributions: The current state of the art techniques for learning in non-stationary environments consider concept drift to have occurred if drift is detected in any of the features. We claim that this is an oversimplification, as a different features may drift at different rates and times, and the information available in non-drifting features can be leveraged into making strong predictions. Therefore, we propose a method of learning in subspaces of the feature space called Heuristic Updatable Weighted Random Subspaces (HUWRS) (Section III), where each base classifier is weighted based on the amount of drift experienced in its subset of features. We then compare HUWRS to a variety of state of the art models (Section VI). We also specifically demonstrate how HUWRS exploits the nature of learning in subspaces to provide an accurate classifier even without the aid of labels (Section VII). We conclude with a discussion and future work (Section IX).

II. Motivation

The prevailing unstated assumption in the current state of the art in non-stationary learning is that a drift in a single feature results in having to relearn the entire model. In spite of this, when generating synthetic datasets for concept drift it is common practice to provide an option that defines what percentage of the features experience concept drift, while the remainder continue to be drawn from their original distribution. Such differing assumptions indicate that there is a disconnect between the state of the art learning algorithms,

1550-4786/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDM.2011.75

and the stream generation algorithms. This disconnect can be costly, as the amount of information (in terms of the models learned and data) discarded can be vast. This is especially true if only one out of tens or hundreds of features are drifting, in which case building a model on the remaining features would be a more practical solution than relearning the model once the drift in the drifting feature is too great.

In order to avoid this specious assumption, we recommend a different approach to learning in non-stationary data streams. Instead of building models on all features, and thus requiring retraining when any type of concept drift is detected, we recommend building multiple models on subsets of the features. This technique, which has been demonstrated to be effective in traditional data mining tasks [16], has received little attention in concept drift research.

With this goal in mind, we require a method for detecting individual drifting features. Feature drift is commonly detected by using the distribution of each feature with respect to each of the classes. Storing so much data per classifier is often unsatisfactory, however, as it is preferable to build models that rely only on the dataset for the current time step to update, rather than previous ones as well. Learning under these assumptions is called *incremental learning*.

In order to overcome this limitation, we first note that we can, instead of storing the feature value for each instance, bin each of the features by class. Binning is a common machine learning technique whereby continuous features are discretized into "bins" — the count of each bin represents the number of instances which have a feature value in the range of the bin[1]. Through the use of binning we can reduce the space required from being proportional to the number of instances, down to a fixed constant due to the number of bins. As the datasets continue to grow in size, this will result in substantial memory savings.

The primary task is now to use the binned feature values to determine if — and to what extent — concept drift has occurred. One attractive option, explored previously as a method of detecting dataset drift [6], [27], is to use Hellinger distance [15], as it is insensitive to the number of examples in a sample. Specifically, in order to detect concept drift we can, considering each class individually, measure the distance between the feature in the training dataset as well as the feature's current distribution. This is powerful, as it allows us to compare the amount of feature drift encountered even if the two samples are not (near) equisized (unlike Euclidean distance). Furthermore, the use of Hellinger distance also allows us to assign a weight to a feature based on how close it is to the reference distribution (i.e., how similar the newly seen instances are to the instances the classifier was trained on).

One common challenge when dealing with concept drift is updating the classifier when labels are unavailable. That is, most methods are unable to cope with feature drift if labels are unavailable, since the class label is required in order to detect drift. However, this overlooks the fact that while "real" drift cannot be detected without labels (as it is, by definition, a change in the class conditionals), "virtual" drift can still be detected. As Hellinger distance is a measure of distributional divergence, in addition to comparing each feature with respect to the class, we can also ignore the class when performing the computation. In this way we can obtain a weight between distributions of only unlabeled instances.

[1]Specifically, we consider equal-width binning

III. Method

In this section we begin by describing the Random Subspace Method (RSM) [16] in the traditional data mining context. We then extend this idea to the context of batched learning in the presence of concept drift with the use of Hellinger distance.

A. The Random Subspace Method

In the data mining community, an ensemble is a collection of *base learners* (also known as base classifiers) each built on (in some sense) separate training datasets, which then classify new instances by voting the individual base learners predictions [7]. Two of the most widely used ensemble methods are bagging [4] and AdaBoost [9].

Another widely used ensemble method is the Random Subspace Method (RSM) [16]. In the RSM (described in Algorithm 1) while multiple base learners are trained on the same dataset, each base learner actively uses only a (random) subset of the available features. Because each base classifier learns on incomplete data, each individual base classifier is (typically) less effective than a single classifier trained on all of the data. Since the RSM combines multiple classifiers of this type, each with a random bias based on the features it sees, RSM often prove more effective than learning the base classifier on all of the features.

In the context of drifting features, learning on only a subset of the features is an advantage, as it means that if not all features drift simultaneously then not all base learners must be retrained. Similarly, this also enables each of the base learners in the ensemble to be weighted by the subset of features it was built on. That is, if a base learner was built on a feature which exhibits concept drift, we can reduce the weight of that classifier relative to the other classifiers.

B. Heuristic Updatable Weighted Random Subspaces for Non-stationary Learning

In the previous section we saw how the RSM is applied in the typical batch learning context, and motivated its adaptation to non-stationary environments. In this section we further explore the requirements for adaptation into non-stationary environments. As a result, we must first be able to detect feature drift. Once we can detect this drift, we can

Algorithm 1 $Random\_Subspace\_Method$

Require: Training set X, number of features to consider p, and number of classifiers to train $n > 0$.
Ensure: CLASSIFIER is the model trained on training set X, consisting of n classifiers.
for $i = 1$ to n **do**
  Select F, a random subset of the features such that $|F| = p$.
  Let $Y \leftarrow X$.
  for all a such that a is a feature of Y **do**
    if $a \notin F$ **then**
      Remove feature a from Y
    end if
  end for
  Train CLASSIFIER$_i$ on dataset Y.
end for

weight each classifier appropriately in order to improve the overall classification accuracy of the ensemble.

In order to detect the drift of a feature, we introduce Hellinger distance [15]. We then specify how to incorporate it into the RSM in order to combat concept drift.

1) Hellinger Distance: Fundamentally, Hellinger distance is a measure of distributional divergence [20], [32]. In order to define Hellinger distance formally, let (P,B,ν) be a measure space [13], where P is the set of all probability measures on B that are absolutely continuous with respect to ν. Consider two probability measures $P_1, P_2 \in P$. The Bhattacharyya coefficient between P_1 and P_2 is defined as:

$$p(P_1, P_2) = \int_\Omega \sqrt{\frac{dP_1}{d\nu} \cdot \frac{dP_2}{d\nu}} d\nu. \tag{1}$$

The Hellinger distance is derived using the Bhattacharyya coefficient as:

$$\begin{aligned} h_H(P_1, P_2) &= 2\left[1 - \int_\Omega \sqrt{\frac{dP_1}{d\nu} \cdot \frac{dP_2}{d\nu}} d\nu\right] \\ &= \sqrt{\int_\Omega \left(\sqrt{\frac{dP_1}{d\nu}} - \sqrt{\frac{dP_2}{d\nu}}\right)^2 d\nu}. \end{aligned} \tag{2}$$

While this equation gives us the ability to compare two continuous distributions, when comparing two features we only have access to a sample of discrete instances. With this in mind, the discrete version of Hellinger distance is defined as:

$$d_H(D_1, D_2) = \sqrt{\sum_{i \in V} \left(\sqrt{P(D_1|X=i)} - \sqrt{P(D_2|X=i)}\right)^2}, \tag{3}$$

where D_1 and D_2 are the distributions of a feature at two different time s2.eps, and $X = i$ is the set of instance with value i for the current feature. If the feature is continuous, we use equal width binning on the feature to turn it into a discrete feature. Note that D_1 and D_2 can either be all of the instances seen so far, or merely the instances available for a particular class. We discus the implications of this in future sections.

While Hellinger distance is applicable to detecting drift in a feature, it is ill-suited for use as a weight. Since a low Hellinger distance means a high agreement in the two distributions, a low Hellinger distance should correspond to a high weight. Thus we obtain a weight from Hellinger distance[2] as:

$$hw(D_1, D_2) = \frac{\sqrt{2} - d_H(D_1, D_2)}{\sqrt{2}}, \tag{4}$$

resulting in a normalized [0,1] weight.

2) Combining Hellinger Distance and the RSM: By combining the RSM with Hellinger weights, we now define Heuristic Updatable Weighted Random Subspaces (HUWRS) (Algorithms 2 and 3). There are two main facets to the algorithm. First, Algorithm 2 updates the classifier when labeled instances become available. Second, Algorithm 3 updates the classifier as new testing (i.e., unlabeled) instances become available.

Algorithm 2 $Train\_HUWRS\_Method$

Require: Training sets X, number of features p, retraining threshold t, number of bins b, and ensemble size $n > 0$.
Ensure: CLASSIFIER is the model trained on training set X consisting of n classifiers, FEATURES$_i$ is a vector consisting of the features used to train CLASSIFIER$_i$, CLASS_WEIGHT is a length n vector containing the weights given to each base classifier.
CLASSIFIER, FEATURES $\leftarrow$ $Random\_Subspace\_Method(X_0)$
for $c = 1$ to n **do**
  BINS$_c \leftarrow BIN(X_0, \text{FEATURES}_c, b)$
end for
CLASS_WEIGHT $\leftarrow \overrightarrow{1}$
while New time step i is available **do**
  CLASSLESS_WEIGHT $\leftarrow \overrightarrow{0}$
  for $c = 1$ to n **do**
    $temp \leftarrow BIN(X_i, \text{FEATURES}_c, b)$
    CLASS_WEIGHT$_c \leftarrow Class_HW(temp, \text{BINS}_c)$
    CLASSLESS_WEIGHT$_c \leftarrow 0$
    if CLASS_WEIGHT$_c < t$ **then**
      Train CLASSIFIER$_i$ on dataset X_i using features in FEATURES$_i$.
      CLASS_WEIGHT$_c \leftarrow 1$
      BINS$_c \leftarrow BIN(X_i, \text{FEATURES}_c, b)$
    end if
  end for
end while

[2]Note that $\sqrt{2}$ is the maximum Hellinger distance between two distributions.

Algorithm 3 $Test\_HUWRS\_Method$

Require: CLASSIFIER as learned using $Train\_Updatable\_Random\_Subspace\_Method$, testing instance x, set of previously seen testing instances X, intra-batch update frequency u, and $Test$ returns a probability vector associated with testing x on a classifier.
Ensure: $prob_i$ contains the probability that x is class i, and CLASSLESS_WEIGHTupdated if $|X| \geq u$.

```
prob = 0⃗
num_classes ← the number of classes in the dataset.
X ← X ⋃ {x}
for c = 1 to n do
  if |X| ≥ u then
    tmp_bins ← BIN(X, FEATURES_c, b)
    CLASSLESS_WEIGHT_c ← Classless_HW(tmp_bins, BINS_c)
  end if
  w ← CLASS_WEIGHT_c + CLASSLESS_WEIGHT_c
  prob ← prob + (w · Test(CLASSIFIER_c, x)/n)
end for
```

When labeled instances are not available, we are unable to use the class labels to determine whether or not concept drift has occurred. That is, we cannot detect if "real" concept drift has occurred, but can only detect "virtual" drift. In order to detect virtual drift, we compute the Hellinger distance between the binned feature values for each base learner[3] and the current dataset, ignoring the class values.

Specifically, let D_1 and D_2 be two separate sets of probability distributions over a set of features. That is, let $D_{1,f}$ denote the probability distribution for dataset D_1 and feature f. We can convert Hellinger distance into a Hellinger weight as:

$$Classless\_HW(D_1, D_2) = \frac{1}{n}\sum_{a=1}^{n}\frac{\sqrt{2} - d_H(D_{1,f}, D_{2,f})}{\sqrt{2}} \tag{5}$$

where n is the number of features to consider. When classes are not available, the Hellinger weight is computed as the average of the Hellinger weights of each of the features.

The value of this approach is that we can attempt to characterize the drift between batches of instances even when labels are not available. Hence, we can update the weight periodically, and dynamically react to virtual drift as more instances become available. We perform this check at periodic intervals (e.g., every $X\%$ of a batch, we consider only the last $X\%$ instances to update the intra batch weight).

The ability to update each classifier's weight is important because in special cases (e.g., loan data), the length of time between seeing a batch of instances to test on and then their subsequent labels may be very long (e.g., years). By updating the classifier using only unlabeled data, we can exploit a frequently ignored portion of the data in order to improve classification accuracy over the course of time.

Similar to other techniques, we can also update weights when labels are available (enabling us to detect "real" concept drift in the process). To do so, we consider not just the distribution of the feature over the entire dataset, but the distribution of the feature for each class over the dataset. That is, for each class we compute the Hellinger distance between the two distributions and average them. Given these weights, we then choose the minimum value as the weight of the classifier. If this minimum weight is below a threshold, we relearn the classifier on the new data.

Specifically, let D_1 and D_2 be two separate sets of probability distributions over a set of features and classes. That is, let $D_{1,f,i}$ denote the probability distribution for dataset D_1, feature f, and class i. We convert Hellinger distance into a Hellinger weight as:

$$Class\_HW(D_1, D_2) = \min_{f \in ftrs}\frac{1}{c}\sum_{i=1}^{c}\frac{\sqrt{2} - hd(D_{1,f,i}, D_{2,f,i})}{\sqrt{2}} \tag{6}$$

where $ftrs$ is the set of features to consider, and c is the number of classes.

With these two methods ($Classless\_HW$ and $Class\_HW$) of detecting concept drift (and consequently weighting each classifier), we must determine how to combine them into a single, unified, weight. To do so, we simply add the two values, i.e., if $class\_weight$ denotes the weight of the classifier on the previous batch, and $classless\_weight$ denote the weight of the classifier on the current batch, then, when classifying an instance, we weight the classifier's output by $weight = class\_weight + classless\_weight$.

IV. Memory Usage

One of the important factors when developing an algorithm that handles concept drift in data streams is to ensure that the algorithm does not use a (potentially) infinite amount of memory. We now demonstrate that the memory usage of HUWRS is finite and bounded.

Consider how HUWRS is defined: the ensemble requires a number b which represents how many bins are to be used to detect concept drift, an ensemble size n, and a classifier type. Assume that the base classifier is guaranteed to use no more than x bytes of memory. For each base classifier, we must maintain b integers for each class (i.e., the number of instances in each of the bins for each class). Letting the number of classes be c, that is $4 \cdot b \cdot c$ (or 8 for 64-bit integers, however the factor of 2 does not affect the argument) bytes of storage required per classifier.

In addition to the space used by each classifier, we also must account for the amount of space consumed by the saved

[3]Note that even if a pair of classifiers share a feature, the bins may be different since one may have been retrained on a different time period due to a drift in a second, unshared, feature.

test instances. Since we can control the number of instances stored by the ensemble, we assume that we allow, at most, y bytes of space to be consumed by these instances.

Given this, the overall memory utilization of the ensemble is, at most, $y+n\cdot(4\cdot b\cdot c+x)$ bytes. Since this number is finite and bounded (by definition of each of the terms), we see that HUWRS can be easily modified to perform inside most memory footprints via appropriate choices of parameters.

V. Experimental Design

We begin with a description of the implementation details of the methods used, defining relevant parameters for each of the algorithms. We then describe each of the datasets used.

A. Implementation Details

We implemented HUWRS in MOA [2], a stream mining suite based off of Weka [12]. For our comparative analysis, we chose two state of the art algorithms: Adaptive Hoeffding Option Trees (AHOT) [3], and Dynamic Weighted Majority (DWM) [26] (further information about these methods can be found in Section VIII). As HUWRS is an ensemble applicable to any traditional base classifier, we chose C4.4 [30] decision trees, i.e., unpruned, uncollapsed C4.5 decision trees [31] with Laplace smoothing applied at the leaves. As outlined by Kolter and Maloof [26], the base learner used used for DWM was standard Naïve Bayes.

In addition to a base learner, HUWRS requires several other parameters. As recommended for many ensembles, we set the ensemble size for HUWRS to $n = 100$. When considering the threshold and the number of bins, due to experiments that tested the effectiveness of Hellinger distance in the context of drift detection, we set $t = 0.7$, and $b = 30$. The application of weights across all datasets in this manner sets a general guideline, and allow us to test the method without overfitting to any particular dataset. In addition to these parameters, we also define a frequency for updating the weights of the classifier based on test instances. Due to the varying sizes of the training datasets, we chose to update the classless weights every 10% of the window size, or 30 instances, whichever is larger. We can hence ensure a minimum number of instances to consider, while still allowing for multiple weight updates.

Finally, while HUWRS relies on using a subset of features to train each base learner, a comprehensive search through different subspace size may lead to dramatic overfitting and overstated performance. Therefore, instead of choosing a fixed subspace size for each classifier, every time a classifier is initially trained we choose a random subset size between 10% and 90% of the number of features. Thus we can test not only the effectiveness of the algorithm, but also its robustness to a variety of subspace sizes.

As HUWRS is a batch learner, we applied an interleaved batch chunk test-then-train on binary balanced datasets. That is, we partitioned the dataset into multiple equisized chunks, testing each instance in a chunk without labels first. Once all instances in a chunk were tested, we computed the accuracy of the classifier over the chunk, introduced labels for all of the instances, finally presenting them to the classifier for updating. The accuracy presented for each algorithm is computed as the average chunk performance of the classifier on the dataset.

We chose to perform the experiments over multiple different sizes of chunks in order to test HUWRS's robustness to the different chunk sizes. Each chunk size was chosen such that the dataset would yield at least 10 chunks at the chosen size. See Section IX for a more detailed explanation of our assumptions.

B. Datasets

To ensure a fair comparison, we tested the algorithms on multiple real world and synthetic datasets, the details of which are available in Table I. The choice to use a combination of real and synthetic datasets is motivated by the fact that while concept drift is assumed in real datasets, it is difficult to identify. For this reason, synthetic datasets are often considered when learning in the presence of concept drift, as the drift type and properties can be precisely controlled. We now describe the datasets in detail, referring to the original publication for more information.

One limitation of testing with real datasets is that only one proper ordering exists, i.e., since the dataset comes from a real source, the instances must appear to a classifier in a single order. Therefore the results for all real datasets are presented as the average of the accuracies on each chunk. For the synthetic dataset, however, we were able to run five iterations, and obtain the average overall performance for each set of parameters.

elec2: One widely used dataset in the concept drift community is the electricity dataset originally due to Harries [14]. The dataset consists of records collected from the Australian New South Wales Electricity Market, where prices vary based on the demand present in the market. Data points are collected in 30 minute intervals, where the class label indicates a change in price over the last 24 hours, thereby providing a level of smoothing.

email-data: The email dataset, created by Katakis, Tsoumakas, and Vlahavas [23], consists of a stream of emails sequentially presented to a user who, in turn, determines whether the emails are spam (not desirable) or ham (desirable) according to personal preference. The dataset is composed of a stream of emails covering several topics, namely: science/medicine, science/space, and recreation/sports/baseball. The stream is then partition into five, 300 instance chunks, where, in the odd number chunks science/medicine is considered as the positive class while the other two topics are considered the negative class. Conversely in the even number chunks, science/medicine is considered as the negative

class, while the other two topics are the positive class. Thus this dataset is an example of concept change, rather than concept drift.

spam-data: The spam dataset, also created by Katakis, Tsoumakas, and Vlahavas [21], consists of spam/ham emails sent to a user collected from Spam Assassin (http://spamassassin.apache.org/). According to the creators, the spam messages present in the dataset exhibit gradual, rather than instantaneous, concept drift.

usenet1: The usenet1 dataset, created by Katakis, Tsoumakas, and Vlahavas [22], was generated from three newsgroups, namely: science/medicine, science/space, and recreation/sports/baseball. The dataset is broken into chunks of 300 instances, where the chunks are defined as in email-data.

usenet2: In the usenet2 dataset, created by Katakis, Tsoumakas, and Vlahavas [22], the same newsgroups were used as in usenet1. The positive class, however, was chosen as only one topic at a time. Therefore, the positive class was, in order: science/medicine, science/space, recreation/sports/baseball, science/space, science/medicine.

RBF: The Random RBF (Radial Basis Function) generator is a synthetic dataset generator available in the MOA data mining suite. In this generator, a fixed number of random centroids are generated, where each centroid has a random location, a standard deviation, a class label, and a weight. Each new instance is generated by choosing a centroid at random (based on weights). Drift is introduced by selecting a subset of the features (in our experiments we consider drift in 4 of the 20 features), and introducing random Gaussian noise, where the center of the Gaussian drifts at the rate controlled by a parameter. In order to capture both slow and fast moving drift, we chose values of: 0.5, 1, 2, 5, and 10.

Dataset	# Ftrs	# Insts	WS
elec2	6	45,312	100, 300, 500, 700, 1000
email-data	914	1,500	30, 50, 70, 100, 150
spam-data	500	9,324	30, 50, 70, 100, 200, 500
usenet1	100	1,500	30, 50, 70, 100, 150
usenet2	100	1,500	30, 50, 70, 100, 150
RBF-{0.5,1,2,5,10}	20	1,000,000	5000, 10000, 50000

Table I
STATISTICS FOR THE DATASETS USED. # FTRS IS THE NUMBER OF FEATURES, # INSTS IS THE NUMBER OF INSTANCES, WS DENOTES THE VARIOUS WINDOW SIZES USED FOR THE DATASET. WHILE RBF IS ONLY LISTED ONCE, IT WAS BUILT WITH MULTIPLE DIFFERENT PARAMETERS FOR DRIFT SPEED AS ENUMERATED.

VI. RESULTS

We begin by discussion the synthetic dataset (Figure 1, as it illustrates the effectiveness of our algorithm in a known concept drift situation. We then discuss our results on the real datasets (Figure 2), showing how we can effectively function in such spaces as well.

A. Results on the Synthetic Datasets

For the synthetic dataset (Figure 1), HUWRS consistently outperforms the other two by up to approximately 20% in accuracy. This is a strong result, as it validates empirically what HUWRS aimed to overcome. That is, this experiment shows that HUWRS can quickly and easily overcome drift in a subset of the features in order to generate a highly accurate ensemble of classifiers.

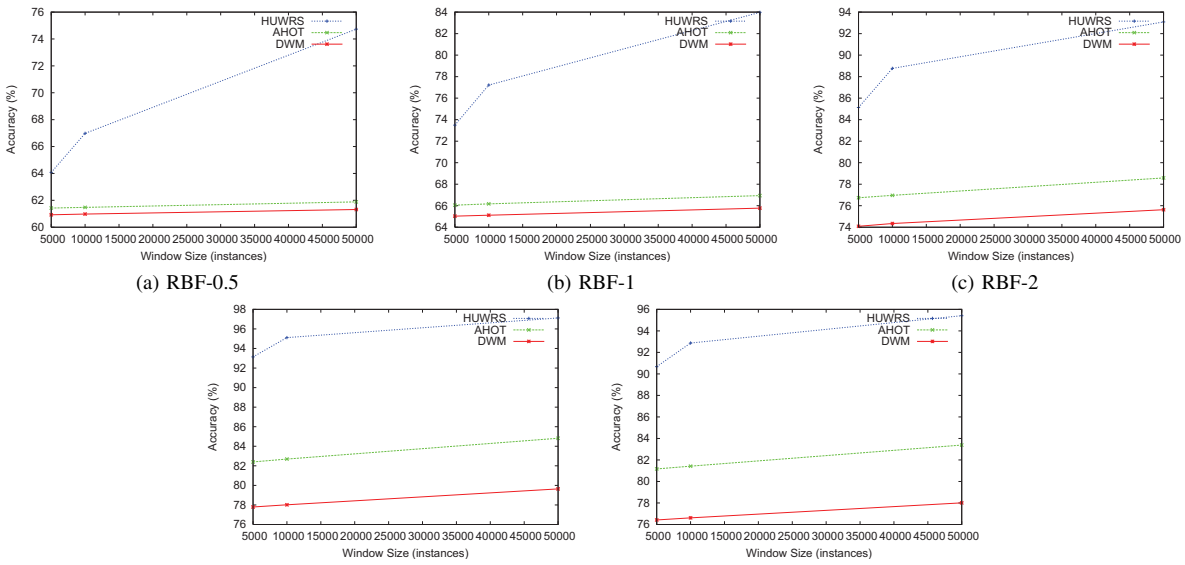
One interesting observation in these results is that HUWRS' method of drift detection seems much more suited to faster moving concept drift (Figure 1e) rather than drift that occurs more gradually (Figure 1a). This results from the fact that while in fast drift HUWRS is able to quickly (and efficiently) update the weights of each base learner before seeing labels, slow drift is missed. The weights of the base learners are therefore not properly updated, thereby negatively impacting performance.

Another observation which can be made is that HUWRS seems to preform universally better as the window size increases. This is due to the fact that HUWRS, upon detecting drift, retrains any "bad" base classifiers (i.e., those using the drifting attributes) and resets its weight to be unit. This, effectively, over weights these "bad" classifiers as they still contain drifting attributes. When larger window sizes are used, however, the intra-batch weighting mechanism is able to act more effectively over the incoming instances, and thus able to keep the weight of the "bad" classifiers down, resulting in a more effective ensemble.

B. Results on the Real-World Datasets

When considering the results on the real datasets (Figure 2), we see two major trends. First, for all datasets except usenet2, HUWRS outperforms the others in the vast majority of cases. This performance increase, however, is not limited to a single window size. Instead there exist a large number of (contiguous) cases for which HUWRS outperforms the others. Secondly, for the majority of datasets, the accuracy of the classifiers tends to decrease as the window size is increased. We begin by addressing the performance of the classifiers on usenet2.

While for the vast majority of cases we see that HUWRS outperforms the others, usenet2 shows a more convoluted picture. In order to understand the performance of the algorithms on usenet2, we consider how it was created. As described in Section V-B, usenet2 was created by collecting data for three concepts, and every 300 instances selecting a different concept to represent the positive class. Since the concept to represent the positive class reoccurs, DWM is well equipped for the task (assuming the window size selected is not too large). That is, since DWM (potentially)



(a) RBF-0.5 (b) RBF-1 (c) RBF-2

(d) RBF-5 (e) RBF-10

Figure 1. Accuracy results for the three classifiers used (Heuristic Updatable Weighted Random Subspaces, Adaptive Hoeffding Option Tree, and Dynamic Weighted Majority) on the synthetic datasets

learns a classifier at each time step, and weights each according to its performance on the current concept, it can (potentially) reuse previously learned base classifiers when the concept reoccurs. It is therefore not surprising to see DWM perform well on this task with reoccurring concepts. It is worth noting, however, that HUWRS does provide the highest overall accuracy. This shows that even for datasets which DWM is well suited, HUWRS can still provide exceedingly high performance.

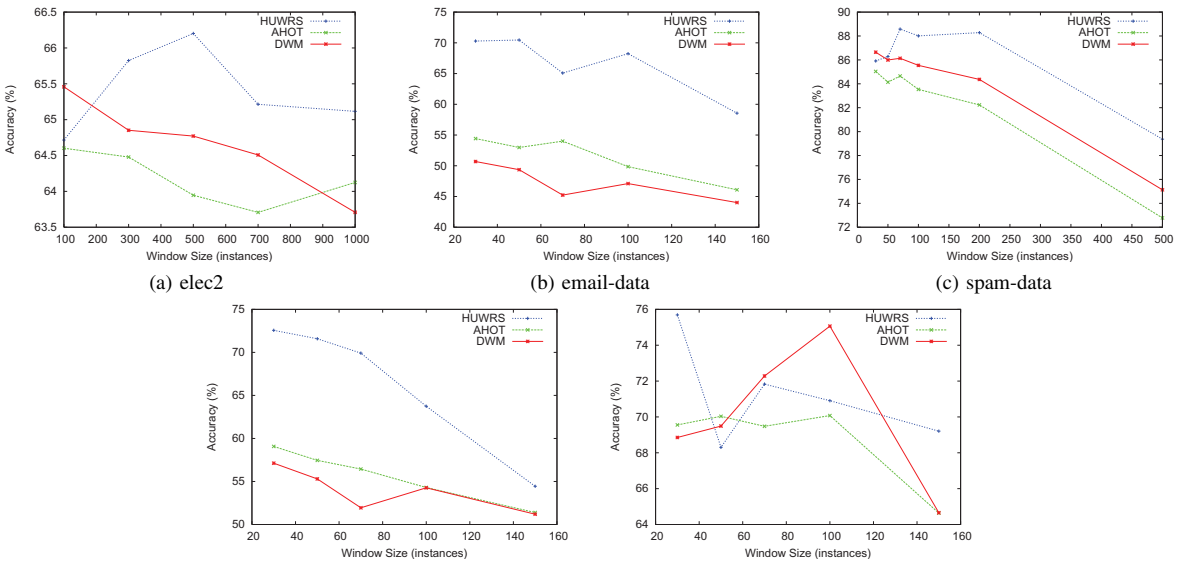
The second main trend observed is a decrease in accuracy as the window size increases. This affect is not surprising, as in the email (Figure 2b), usenet1 (Figure 2d), and usenet2 (Figure 2e) datasets in particular, there is a sharp concept change after 300 instances. Thus when the window size increases (especially to 150 instances), the methods perform very poorly on the first 150 instances after the concept change (as there are no class labels available to notice the change), explaining the observed drop in performance on each of the datasets. For the spam dataset, such a drop in performance may be due to the fact that, over such (relatively) large windows, the methods are not able to capture the drift fast enough, and thereby observe vast drops in performance.

VII. Analysis of the Intra Batch Weighting Technique

Given the results presented in Section VI-A, one obvious question is how effective the intra batch weighting scheme is at detecting and mitigating concept drift. In order to test this, we performed a modification of the experiments of the previous section on the RBF dataset. Instead of each classifier receiving instances as a series of chunks (i.e., chunk test-then-train), each classifier instead only received the labels for the first chunk, with the remaining instance's classes withheld for the entirety of the experiment. In this way we tested how effective the ensemble is at dealing with drift without the presence of labels, particularly relative to the other methods.

The results of this experiment are shown in Figure 3. Given the results, there are two main observations to be made. First, AHOT and DWM perform significantly worse than when they are given labels, with decreases of approximately 10-20% in accuracy. This result is not surprising, as neither of these methods is able to detect concept drift without labels, and thus are ill suited to task of learning without labels. The second observation stems from the fact that not only does HUWRS maintain a higher performance (in this case significantly higher than the others), its performance actually *increases* over that of its performance when given labels.

The increase in performance observed in this context in HUWRS is due mainly to the retrain feature (i.e., when a classifier's weight drops below a certain performance threshold it is retrained). Since in the RBF case drift is continuously occurring in a portion of the features, every time a classifier using one of the drifting features is retrained, it is mistakenly given a high weight under the assumption that the next time step will look similar to the current one. This increased weight results in the predictions made by these classifiers being given a weight which is too high, as the features are still drifting. Since the base classifiers are quickly no longer relevant due to the concept drift, this



(a) elec2 (b) email-data (c) spam-data

(d) usenet1 (e) usenet2

Figure 2. Accuracy results for the three classifiers used (Updtable Weighted Random Subspaces, Adaptive Hoeffding Option Tree, and Dynamic Weighted Majority) on the real datasets

results in decreased performance.

Finally, given that the performance of HUWRS actually increases (unlike the other two methods) when labels are omitted, we claim that our intra batch weighting scheme offers a benefit to classification performance.

VIII. Related work

In order to learn in the presence of concept drift, algorithm designers must deal with two main problems. The first problem is detecting concept drift present in the stream. Once concept drift has been detected, one must then determine how to best adapt to make the most appropriate predictions on the new data.

In the next section we present a variety of state of the art learning algorithms for concept drift (e.g., AHOT and DWM). Subsequently, we introduce a variety of popular concept drift detection techniques.

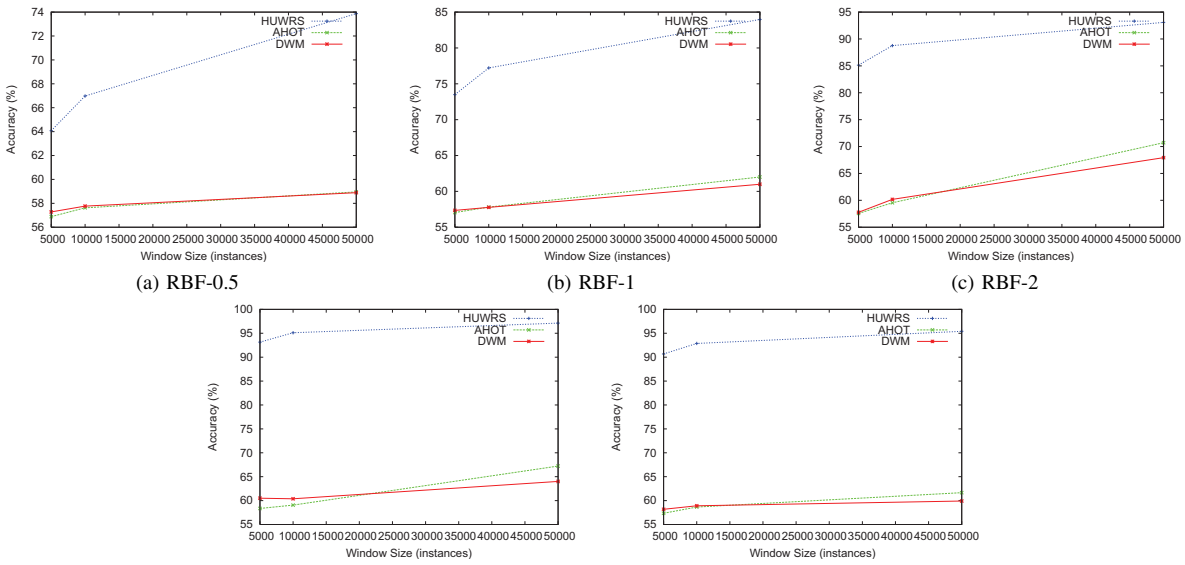
A. State of the Art Techniques

One base learner which has been heavily studied in the context of concept drift is the decision tree; of which the most common traditional variant is C4.5 [31]. The original extension of the decision tree learning algorithm, called VFDT, was proposed by Domingos and Hulten [8]. In VFDT, Hoeffding bounds [17], [28] are used to grow decision trees in streaming data. The authors show that in the case of streaming data, applying Hoeffding bounds to a subset of the data can, with high confidence, choose the same split feature as a method using all of the data. This observation allows for trees to be grown online that are nearly equivalent to those built offline. Since its inception, VFDT has been the basis for numerous extensions and improvements [19], [18], [1].

As an extension to standard decision trees, Buntine [5] introduced option trees, which were further explored by Kohavi and Kunz [25]. In standard decision trees, there is only one possible path from the root to a leaf node, where predictions are made. In option trees, however, a new type of node — known as an option node — is added to the tree, which splits the path along multiple split nodes. Pfahringer, Holmes, and Kirby combined the concept of Hoeffding trees and option trees to create Hoeffding Option Trees [29]. They combine these two methods by beginning with a standard Hoeffding tree and, as data arrives, if a new split is found to be better than the current split at a point in the tree, an option node is added and both splits are kept. Further extending Hoeffding Option Trees are Bifet et. al. [3]. In their extension, called Adaptive Hoeffding Option Trees, each leaf is provided with an exponential weighted moving average estimator, where the decay is fixed at 0.2. The weight of each leaf is then proportional to the square of the inverse of the error.

While Hoeffding Trees build a single decision tree (or in the case of Hoeffding Option Trees, a single option tree), Dynamic Weighted Majority (DWM) [26] instead creates an ensemble of classifiers. In order to test an instance using DWM, every new instance is classified by the ensemble by using a weighted vote of each of its base classifiers.

In order to train DWM, given a new training instance, the ensemble begins by attempting to classify it. Each base classifier which misclassifies the instance has its weight reduced by a multiplicative constant β. If the entire ensemble



(a) RBF-0.5 (b) RBF-1 (c) RBF-2

(d) RBF-5 (e) RBF-10

Figure 3. Accuracy results for the three classifiers used (Heuristic Updatable Weighted Random Subspaces, Adaptive Hoeffding Option Tree, and Dynamic Weighted Majority) on the synthetic datasets when labels are only available for the first chunk.

misclassifies the instance, the ensemble adds a new classifier with a weight of 1 after rescaling the weights of each existing classifier to 1 (in order to prevent new classifiers from dominating the voting process). The ensemble then removes each classifier with a weight lower than some threshold θ, and then provides the instance to each of the base classifiers for updating.

B. Drift Detection Techniques

One popular technique for drift detection is due to Klinkenberg and Joachims [24] who proposed a method based on Support Vector Machines (SVMs). Specifically, they proposed the use of $\xi\alpha$-estimator to compute a bound on the error rate of the SVM. Specifically, assuming t batches, they use the $\xi\alpha$-estimator to compute t error bounds. The first error bound corresponds to learning the classifier on just the newest batch, the second bound corresponds to learning on the newest 2 batches, etc. They then choose the window size which has the minimum estimated error.

Gama et. al. [10] proposed a method based on the error rate over time of the learning algorithm. To accomplish this, they assume that each new instance represents a random Bernoulli trial. Based on this, they then compute the probability of observing a "false" for instance i (p_i), and the standard deviation ($s_i = \sqrt{p_i(1-p_i)/i}$). They then argue that a significant increase in the error rate denotes a concept drift. With this in mind, they state that their algorithm issues a *warning* if $p_i + s_i \geq p_{min} + 2s_{min}$, and drift is detected if $p_i + s_i \geq p_{min} + 3s_{min}$.

IX. Discussion and Future Work

We introduced Heuristic Updatable Weighted Random Subspaces (HUWRS) as a method for dealing with concept drift in data streams. In this method an ensemble of classifiers is built, such that each classifier is learned on a different subspace of the features. When (sufficient) drift is detected in a feature, only those classifiers learned on that feature are retrained, thereby enabling the classifier to perform less retraining. Additionally, we demonstrate how the algorithm is able to cope even in scenarios where labels are rare, updating the weights of each classifier to reflect the current, perceived, drift in each of the features.

In order to prove the effectiveness of HUWRS, we compared it to two other state of the art methods, namely Adaptive Hoeffding Option Trees, and Dynamic Weighted Majority. For a wide variety of datasets and window sizes, we found that HUWRS outperformed these other methods. Additionally, we tested the case where each of the methods was only given one chunk with labels. In this non-standard scenario, HUWRS was able to demonstrate exceedingly high performance in the face of concept drift.

One of the obvious shortcomings of HUWRS when compared to the other methods is the requirement that instances arrive in batches. This shortcoming is due first to our choice of a non-incremental base learner (C4.4), and secondly to our choice of drift detection technique (i.e., a simple, non-sliding windowed technique). In future work we seek to remedy this failing, making HUWRS more applicable to a wider variety of problem. Due to the wide range of window sizes for which HUWRS outperformed the other state of the art methods, we believe this is an attainable goal.

Given all of these considerations, in scenarios where data arrives in batches and is subject to either gradual concept drift or sudden concept change, we recommend using Heuristic Updatable Weighted Random Subspaces.

ACKNOWLEDGEMENTS

Work is supported in part by the NSF Grant ECCS-0926170, NSF Grant ECCS-092159, and the Notebaert Premier Fellowship.

REFERENCES

[1] A. Bifet and R. Gavalda. Adaptive learning from evolving data streams. *Advances in Intelligent Data Analysis VIII*, pages 249–260, 2009.

[2] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *JMLR*, 11:1601–1604, 2010.

[3] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *KDD*, pages 139–148. ACM, 2009.

[4] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[5] W. Buntine. Learning classification trees. *Statistics and Computing*, 2(2):63–73, 1992.

[6] D. Cieslak and N. Chawla. A framework for monitoring classifiers' performance: when and why failure occurs? *KAIS*, 18(1):83–108, 2009.

[7] T. Dietterich. Ensemble methods in machine learning. *MCS*, pages 1–15, 2000.

[8] P. Domingos and G. Hulten. Mining high-speed data streams. In *KDD*, pages 71–80. ACM, 2000.

[9] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *COLT*, pages 23–37, 1995.

[10] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. *SBIA*, pages 66–112, 2004.

[11] S. Grossberg. Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1(1):17–61, 1988.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[13] P. Halmos. *Measure theory*. Van Nostrand and Co., 1950.

[14] M. Harries. Splice-2 comparative evaluation: Electricity pricing. 1999.

[15] E. Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen (german). *Journal fur die Reine und Angewandte Mathematik*, 136:210–271, 1909.

[16] T. Ho. The random subspace method for constructing decision forests. *PAMI*, 20(8):832–844, 1998.

[17] W. Hoeffding. Probability inequalities for sums of bounded random variables. *JASA*, 58(301):13–30, 1963.

[18] S. Hoeglinger and R. Pears. Use of hoeffding trees in concept based data stream mining. In *ICIAFS*, pages 57 –62, Dec 2007.

[19] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *KDD*, pages 97–106. ACM, 2001.

[20] T. Kailath. The divergence and bhattacharyya distance measures in signal selection. *IEEE Transactions on Communication Technology*, 15(1):52–60, 1967.

[21] I. Katakis, G. Tsoumakas, and I. Vlahavas. Dynamic feature space and incremental feature selection for the classification of textual data streams. *Knowledge Discovery from Data Streams*, pages 107–116, 2006.

[22] I. Katakis, G. Tsoumakas, and I. Vlahavas. An Ensemble of Classifiers for coping with Recurring Contexts in Data Streams. In *ECAI*, pages 377–382, Patras, Greece, 2008.

[23] I. Katakis, G. Tsoumakas, and I. Vlahavas. Tracking recurring contexts using ensemble classifiers: an application to email filtering. *KAIS*, 22(3):371–391, 2010.

[24] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *ICML*. Citeseer, 2000.

[25] R. Kohavi and C. Kunz. Option decision trees with majority votes. In *ICML*, pages 161–169, 1997.

[26] J. Kolter and M. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *ICDM*, pages 123–130. IEEE, 2003.

[27] R. Lichtenwalter and N. V. Chawla. Adaptive methods for classification in arbitrarily imbalanced and drifting data streams. In *New Frontiers in Applied Data Mining*, volume 5669 of *Lecture Notes in Computer Science*, pages 53–75. Springer Berlin / Heidelberg, 2010.

[28] A. M. Oded Maron. Hoeffding races: Accelerating model selection search for classification and function approximation. In *NIPS*, volume 6, pages 59–66, April 1994.

[29] B. Pfahringer, G. Holmes, and R. Kirkby. New options for hoeffding trees. *AAI*, pages 90–99, 2007.

[30] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, September 2003.

[31] J. Quinlan. *C4.5: programs for machine learning*. 1993.

[32] C. Rao. A review of canonical coordinates and an alternative to correspondence analysis using hellinger distance. *Questiio (Quaderns d'Estadistica i Investigacio Operativa)*, 19:23–63, 1995.

[33] A. Tsymbal. The problem of concept drift: definitions and related work. *Informe técnico: TCD-CS-2004-15, Departament of Computer Science Trinity College, Dublin, https://www. cs. tcd. ie/publications/techreports/reports*, 4:2004–15, 2004.