Learning in Non-stationary Environments with Class Imbalance

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

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. In addition to the environments being non-stationary, they also often exhibit class imbalance. That is one class (the majority class) vastly outnumbers the other class (the minority class). This combination of class imbalance with non-stationary environments poses significant and interesting practical problems for classification. To overcome these issues, we introduce a novel instance selection mechanism, as well as provide a modification to the Heuristic Updatable Weighted Random Subspaces (HUWRS) method for the class imbalance problem. We then compare our modifications of HUWRS (called HUWRS.IP) to other state of the art algorithms, concluding that HUWRS.IP often achieves vastly superior performance.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms

Keywords

concept drift, class imbalance, non-stationary learning

1. INTRODUCTION

Two of the most common problems faced in data mining and machine learning research are the class imbalance problem and learning in non-stationary data streams. In spite of being well studied individually, the combination of the two problems has received considerably less attention. In this paper we discuss the confluence of the non-stationary data streams with class imbalance, and the complications that arise. We then provide a novel solution which we compare to the state of the art methods.

In traditional data mining classification contexts, algorithms have been developed such that given a training set, they are able to classify new instances into their appropriate classes. Recently, however, more and more problems have gone *online*, i.e., data arrives incrementally over time, and is therefore not all available at once. In order to solve this problem, various classifiers have been developed which adapt over time as more data becomes available.

One limitation of the naïve online model is that it assumes that the class boundary remains constant over time. In general, however, this assumption need not be true, and the underlying data generation function f from which the data is assumed to be drawn from may change over time. When a data stream experiences such a change (or *drift*), it is known as a *non-stationary environment* and is said to exhibit *concept drift*. In this paper when a sudden change in the underlying generation function f occurs, we say that the data stream has experienced concept drift. Alternatively, if the drift is occurring gradually over time, we say the data stream is experiencing concept drift.

When learning from non-stationary environments, a classifier must be able to respond to concept drift while ensuring that it retains all (relevant) past knowledge. These two goals bring rise to what is known as the stability-plasticity dilemma [9]. The problem becomes more pronounced if data cannot be stored due to space constraints, and thus all forgetting results in irretrievable loss of information.

While data in streams arrive over time, there are multiple different models for when and how they arrive. One popular model is to assume that instances first arrive without labels for testing, and then, once a prediction is given, a label is made available to update the classifier. While this assumes that instances arrive one at a time in this manner, another possible alternative is that the data may arrive in batches. That is, the data may arrive as a non-empty set of instances which are initially unlabeled for testing. Once predictions have been given for all instances in the batch, the labels for each of the instances are made available in order to update the classifier. For the sake of this paper, we only consider data streams for which instances arrive in batches.

While concept drift is a phenomenon only relevant in data streams, the class imbalance problem has a rich history of research in the traditional data mining literature [5, 4]. In the class imbalance problem one class (the positive class) is severely under represented in the dataset. This causes issues for traditional data mining algorithms, as under certain metrics it becomes optimal to merely predict the majority class, ignoring the minority class entirely. For instance, if the majority class makes up 99% of the dataset, the naïve classifier which always predicts the majority class will achieve 99% accuracy on testing data.

The problem of class imbalance is further exacerbated when learning from non-stationary data streams. This is for two main reasons. First, the distance — not only in time but also in distributional divergence — between observing positive class examples can become arbitrarily large. That is, there can be an arbitrarily long time — hours, days, or even months — between seeing successive minority class instances. Additionally, successive minority class instances may be drawn from arbitrarily different distributions. Such differences, however, can have very different meanings depending on the contexts.

For example, in one context the difference in minority class instances may indicate a drift in the minority class definition. Alternatively, the current minority class concept may merely define both instances as minority instances, even if there is no similarity between them (e.g., minority instances may be located in two pockets in the feature space, with no discernible similarity between the two types of minority class instances; such differences can be found in spam emails, where there are many different types of "categories" of minority class instances). Finally, the new minority class instance may just be the result of noise in the data, and not indicative of a more general trend.

While all three of the above alternatives are possible, it is impossible to know a priori the current scenario in any non-stationary data stream. This is due to the fact that the speed, type, and time of drift are assumed arbitrary and unknown. Therefore any algorithm developed for non-stationary data streams must be robust to any and all situations which may arise.

As a concrete example, consider the recent mortgage crisis. Models were built based on historical data which suggested certain properties in the rate of defaults (and therefore foreclosures) for various populations. Based on these models, loan recommendations were made which seemed sound. Due to a variety of factors outside of the model, however, the entire housing climate changed drastically, very quickly. As a result, the predictions of the models became based on faulty assumptions, and lead to a disaster.

By combining the class imbalance and non-stationary data problems, we see that the two problems together provide confounding effects. That is, in an imbalanced data stream undergoing concept drift, the time until the concept drift is detected can be arbitrarily long. This is due to the fact that since there are so few positive examples present in the stream, it is difficult to infer the source of the error for the positive class. In such instances, the misclassified positive class instance can merely be a result of noise in the data stream. In other cases, however, such a misclassification can signify a drift in concept which must be handled by the algorithm. Any algorithm designed to handle concept drift with class imbalance must account for this fact.

1.1 Contributions

In summary, the key **contributions** of this paper are:

1. A novel, non-temporal, instance selection method for the minority class in concept drifting data streams which exhibit class imbalance.
2. HUWRS with Instance Propagation HUWRS.IP as a method for overcoming class imbalance and concept-drift in data streams which perform well when compared against the state of the art on a variety of streaming datasets.

2. MOTIVATION

One prevailing unstated assumption of current state of the art non-stationary learning methods is that any detected feature drift requires model relearning. Specifically, if drift is detected in a feature, the entire model is often relearned, and past data discarded as belonging to a past concept [13]. There is no reason to believe, however, that this is necessarily true in practice. Thus while such a coarse grained approach to learning in non-stationary streams has proven effective, it may discard important information that can be used to improve performance.

Consider, for instance, a situation where certain features are drifting, while other features remain constant. This is a common case in concept drift injection mechanisms, and is most likely applicable in real world datasets. In such streams discarding past data about the stationary features, and forcing the model to be relearned, is unnecessary, and sacrifices performance while new instances are gathered. A more sophisticated approach may be able to still use the information available in the non-drifting features while also discarding the data from drifting features.

This observation provides the impetus for questioning the assumption of whether drift in one feature requires the model to change if the remainder of the features remain constant. This is especially true when the stream suffers from class imbalance, as discarding old instances can be costly, since it may require a lot of time (and potentially money) to observe new minority class instances. In addition to requiring more instances before the model can be retrained, one must also include the cost of rebuilding/retraining/modifying the classification algorithms once the data becomes available, and the opportunity cost associated with all instances which are misclassified before the model can be updated.

Currently there are very few methods available to overcome both the class imbalance and non-stationary problems in the literature. The most well known framework is due to Gao et. al. [8], which advocates propagating minority class examples from recent batches to the current batch in order to promote class balance. This attempts to overcome the previously mentioned issue of unnecessarily forgetting minority class instances by merely saving them all, and using them when necessary. The main drawback of this approach is that even if a change in the posterior distribution is detected, old instances are still used in order to increase the number of minority class instances. In the context of class imbalance in non-stationary data streams, however, these instances are likely drawn from a different concept than the current minority class concept, and may therefore inject improper bias into the classifier.

As a result, a more general approach to propagating past instances is required. The more general approach should be able to select only those instances which are relevant to the current minority class context. More specifically, only instances which are drawn from the same generating function as the current generating function should be selected. In this way, only the current concept defined by the minority class is strengthened, and the old, out of date concepts are forgotten.

While the novel instance selection mechanism will aid in not unnecessarily eliminating minority class instances, it is also advantageous to not unnecessarily eliminate majority class instances as well. In order to do this, we employ the Heuristic Updatable Weighted Random Subspaces method (HUWRS) [13] to build models only on a subset of the features. By combining HUWRS with the aforementioned instance selection mechanisms for propagating minority class instances when necessary, we retain the majority class benefits of HUWRS, and the minority class benefits of the propagation methods. The use of the novel instance selection method provides further benefit over the naïve instance propagation method, as depending on which features a classifier is built on, the selected instances may vary. That is, if there is a cyclic nature in which features drift, the method will be able to more accurately choose the instances relevant for each base learner.

With this goal in mind, we now introduce our instance selection technique. We then demonstrate how to incorporate it into the HUWRS framework to create HUWRS.IP for non-stationary environments which exhibit class imbalance.

3. METHOD

In this section we begin by describing the instance selection method. We then describe how to incorporate this method into the HUWRS framework to create HUWRS.IP.

3.1 Instance Propagation Method

In the framework proposed by Gao et. al [8], the most recent instances were selected to reinforce class balance when updating the ensemble on the current batch. The main drawback of the approach due to Gao et. al. is that it only selects a specific number of recently seen minority class instances. This inflexibility ignores the similarity of the minority class instance to the current concept, relying only on its similarity in time.

Selecting instances based on their temporal similarity to the current batch can pose serious drawbacks in practice. For instance, in data streams where concept drift occurs often and suddenly it will result in the selection of instances which used to be minority class instances, however after concept drift represent majority class instances. In order to combat this, we must devise a method that does not use temporal data about an instance to determine its similarity to the current concept.

We therefore define an instance selection strategy based on the probability that old instances are drawn from the same distribution as the new minority class concept. That is, let B_t (M_t) for $0 \leq t \leq T$ be the set of all instances (minority instances only) observed at time t, and B (M) be the set of all instances (minority instances, respectively) observed. Instead of merely selecting the last k instances observed, we can instead build a generative model on the most recent minority class instances.

We choose to use a generative model in our instance selection strategy as they, by definition, attempt to model $P(X|Y)$, i.e., they attempt to model the likelihood of a given feature vector X, given that the instance is class Y. This is valuable in our instance selection approach, as we are attempting to select only instances drawn from the same distribution as the current concept.

Another advantage of generative models is the fact that they are applicable in datasets which exhibit class imbalance, since the probability that instance X belongs to class Y is $P(Y|X) = P(X|Y) \cdot P(Y)$. This normalization by the class prior, $P(Y)$, allows the generative model to provide useful predictions in the face of class imbalance.

Therefore, we train a Naïve Bayes classifier C on the current batch of instances B_T. One important thing to note is that if B_T contains no minority class instances (which is not unusual in highly imbalanced environments), building C is not possible. As a result, for any batches B_i for which there are no minority class instances, we propagate the most recent set of minority class instances M_j (where j is $\arg\max_j |M_j| > 0$), to create the hybrid batch H_i. The classifier C can then be trained on this hybrid batch H_i in order to learn the distribution of the most recent batch of minority class instances. Note that this is the most naïve way of attempting to determine the instances which belong to the most recent batch. In general a more sophisticated approach can be employed in order to ensure that the hybrid batch contains the most recent concept.

We can then use C to classify all minority instances in M, and only use those for which the probability is above a threshold. An alternative approach is to use the same classifier C, but instead of choosing only those instances which score above a certain threshold, we can instead choose the k most likely instances.

The main advantage of this instance selection approach over the naïve approach in general is that it avoids the problem of determining where the boundary is between the old concept and the new concept. This allows it to more accurately accrue only instances which are drawn from the same concept as the current concept, and therefore provide better classification accuracy. This is the reason why we do not apply the approach of choosing the k most likely instances, since it, like the naïve approach, assumes that at least k instances are available from the new concept. Since this need not be the case in the case of a sudden change in the features, only selecting instances which are above a threshold should allow for better performance in the face of sudden concept drift.

One problem with the pure instance selection approach, however, is that it relies on finding instances which are similar to the current minority class context. In some cases of rapid drift, however, such instances may not be available. As a result, we allow our instance selection strategy to default to the naïve strategy if no previously seen minority class instances are propagated. In this way the instance selection strategy is able to remain robust in the face of rapid, non-reoccurring, concept drift.

As a result, we propose Algorithm 1. In Algorithm 1, instances are classified based on the probability they belong to the minority class as defined by the current batch of minority class instances. This is accomplished via the use of a Naïve Bayes classifier, which is a generative model that attempts to model $P(X|Y)$. To do this, it assumes each of the features are independent, and computes $P(X|Y) = \prod_{i \in features} P(X_i|Y)$, where $P(X_i|Y)$ is the probability of observing feature value X_i for feature i, given the class is Y. By using Bayes' theorem, the model can then use this to compute $P(Y|X) = P(X|Y) \cdot P(Y)$, where $P(Y)$ is the class prior, or the probability of observing class Y. If no instances are above the threshold, then the instance selection mechanism reverts to the naïve approach.

The use of the hybrid approach allows us to leverage the instance selection method when there are a sufficient number of instances available, but otherwise balances the class distribution with the recently available instances. This enables our hybrid approach to be robust to a wide variety of non-stationary data streams. In particular, the instance selection mechanism will be especially beneficial in environments in which contexts reoccur (e.g., seasonally, yearly, etc.). In such scenarios the instance selection mechanism can select the instances from the "correct" time period to augment the current batch. In rapidly changing environments where the instance selection mechanism is not applicable, the simple propagation method can aid in balancing out the class distributions.

Algorithm 1 *Instance_Select*

```
Require: Training batch $B_T$, minority batches to test $M_t$ for $0 \leq t < T$, and threshold $threshold$.
Ensure: $M_{threshold}$ are the selected instances to add to the base learner.
  Train a Naïve Bayes classifier $C$ on $B_T$.
  $numAdded \leftarrow 0$
  for $i = 0$ to $T - 1$ do
    for Each instance $instance \in B_i$ do
      Let $p$ be the probability that $instance$ is a minority instance according to $C$.
      if $p > threshold$ then
        Let $M_{threshold} \leftarrow M_{threshold} \bigcup \{instance\}$
        $numAdded \leftarrow numAdded + 1$
      end if
    end for
    if $numAdded = 0$ then
      Add the last $k = 0.2 \cdot |B_t|$ instances to $M_{threshold}$.
    end if
  end for
  Return $M_{threshold}$.
```

As a result, this method enables us to more accurately determine which instances are relevant to the current batch. This should lead to better performing base learners, and therefore better overall performance.

3.2 HUWRS for Non-stationary Learning and Class Imbalance

In the previous section we introduced a new instance selection mechanism for class imbalance in non-stationary environments. In this section we describe how to incorporate it into the Heuristic Updatable Weighted Random Subspaces (HUWRS) method [13] to produce a robust ensemble framework for concept drift and class imbalance.

3.2.1 Combining Instance Propagation and HUWRS

For completeness, before introducing HUWRS.IP, we first define the *Random_Subspace_Method* in Algorithm 2 as it is used in HUWRS. The important thing to note about the method is it generates an ensemble of classifiers, each built on a different (potentially differently sized) set of features.

The combination of the instance selection method defined above with HUWRS results in the Heuristic Updatable Weighted Random Subspaces with Instance Propagation (HUWRS.IP) (Algorithms 3 and 4). There are two main facets to HUWRS.IP. First, Algorithm 3 updates the classifier when labeled instances become available. Second, Algorithm 4 updates the classifier as new testing (i.e., unlabeled) instances become available. For the sake of space, we omit some of the details of HUWRS, and limit the discussion to the impact of class imbalance on the classifier weighting function, since it is the major change for HUWRS.IP. For more details on HUWRS, the reader can consult [13].

Algorithm 2 *Random_Subspace_Method*

```
Require: Training set $X$, a range of 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 $|F| \in P$.
    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
```

3.2.2 Hellinger Distance

One of the most important part of the HUWRS algorithm is the drift detection mechanism based on Hellinger distance [12]. The use of Hellinger distance as a method of detecting concept drift has been previously explored [7, 16].

Fundamentally, Hellinger distance is a distributional divergence measure [14, 20]. Let (P,B,ν) be a measure space [11], 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:

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

The Hellinger distance is derived using the Bhattacharyya coefficient as:

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

3.2.3 Hellinger Weight

In HUWRS, the Hellinger weight of a classifier was defined as follows. 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:

$$Classless\_HW(D_1, D_2) = \frac{1}{n} \sum_{f=1}^{n} \frac{\sqrt{2} - d_H(D_{1,f}, D_{2,f})}{\sqrt{2}} \quad (3)$$

Algorithm 3 $Train\_HUWRS.IP$

```
Require: Training batches $B$, previously seen minority instances $M$, range of number of features $p$, retraining threshold $threshold_{retrain}$, instance selection threshold $threshold_{select}$, number of bins $b$, and ensemble size $n > 0$.
Ensure: CLASSIFIER is the model trained on training set $B$ 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, and $M$ is a set of batches of minority class instances.
  $\text{CLASSIFIER}, \text{FEATURES} \leftarrow Random\_Subspace\_Method(B_0)$
  for $c = 1$ to $n$ do
    $\text{BINS}_c \leftarrow BIN(B_0, \text{FEATURES}_c, b)$
  end for
  $\text{CLASS\_WEIGHT} \leftarrow \vec{1}$
  while New time step $i$ is available do
    Let $M_i$ be the set of minority instances in batch $B_i$.
    $\text{CLASSLESS\_WEIGHT} \leftarrow \vec{0}$
    for $c = 1$ to $n$ do
      $temp \leftarrow BIN(B_i, \text{FEATURES}_c, b)$
      $\text{CLASS\_WEIGHT}_c \leftarrow Class\_HW(temp, \text{BINS}_c)$
      $\text{CLASSLESS\_WEIGHT}_c \leftarrow 0$
      if $\text{CLASS\_WEIGHT}_c < threshold$ then
        Let $X$ be the current batch $B_i$ with only features from FEATURES$_c$.
        $toAdd \leftarrow Instance\_Select(X, M, threshold_{select})$
        Train CLASSIFIER$_i$ on dataset $B_i \bigcup toAdd$ using features in FEATURES$_c$.
        $\text{CLASS\_WEIGHT}_c \leftarrow 1$
        $\text{BINS}_c \leftarrow BIN(B_i, \text{FEATURES}_c, b)$
      end if
    end for
  end while
```

Algorithm 4 $Test\_HUWRS.IP$

```
Require: CLASSIFIER as learned using $Train\_HUWRS.IP$, 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 $|X| \geq u$.
  $prob = \vec{0}$
  $num\_classes \leftarrow$ the number of classes in the dataset.
  $X \leftarrow X \bigcup \{x\}$
  for $c = 1$ to $n$ do
    if $|X| \geq u$ then
      $tmp\_bins \leftarrow BIN(X, \text{FEATURES}_c, b)$
      $\text{CLASSLESS\_WEIGHT}_c \leftarrow Classless\_HW(tmp\_bins, \text{BINS}_c)$
    end if
    $w \leftarrow \text{CLASS\_WEIGHT}_c + \text{CLASSLESS\_WEIGHT}_c$
    $prob \leftarrow prob + (w \cdot Test(\text{CLASSIFIER}_c, x)/n)$
  end for
```

where n is the number of features to consider, D_1 (D_2) the distribution of positive (negative, respectively) class instances, and d_H is the Hellinger distance between distributions $D_{1,f}$ and $D_{2,f}$.

Thus when classes are available, the Hellinger weight is calculated as the average of the minority class and majority class Hellinger distance between the two feature distributions. In this way we are giving equal weight to the Hellinger distance between the minority class distribution, and majority class distribution. By balancing the weight in this way, implicitly more weight is given to a shift in the minority class. In future work the combining function can be optimized for use in non-stationary data streams with class imbalance, however in practice this simple heuristic is very effective.

One final observation is that when classes are not available, the Hellinger weight is computed as the average of the Hellinger weights of each of the features, and thus the relative frequency of each class is not relevant.

3.2.4 Benefits of HUWRS

The value of using HUWRS in addition to the instance selection method is that we maintain all of the benefits HUWRS gives for non-stationary environments. That is, we retain the advantages of building classifiers which are robust in the face of a subset of features drifting, as well as high latency in obtaining class labels (since we can use the testing set to re-weight the ensemble, see [13] for details).

In addition to obtaining robustness from concept drift by using a subset of the features for each classifier, we also obtain robustness from class imbalance by using the instance selection method. The instance selection method is also further enhanced, as each member of the ensemble can individually determine which instances are most relevant in its feature space, and therefore obtain better overall performance.

4. 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 an unreasonable amount of memory. We now demonstrate that the memory usage of HUWRS.IP is reasonable, and can be easily controlled.

From our previous work [13], we know that memory usage of HUWRS is finite and bounded. Since the instance selection method requires a number of previously seen minority instances, we note that as the number of batches goes to infinity, the number of instances saved by the algorithm also goes to infinity. Since the only instances saved, however, are from the minority class, we note that this is not a substantial burden in general, as the minority class instances will only represent a manageable number of instances.

Therefore, we propose to allow the algorithm to maintain all minority class instances seen by the algorithm (as in the framework due to Gao et. al.[8]). If this limitation is not possible in practice, then two simple forgetting mechanisms can be employed. First, we can merely forget the oldest instances until we have dropped under the quota. Since this runs counter to the motivation of the method, however, an alternative approach is to instead keep a count of the number of times each instance has been reused by a classifier, dropping the instances with the smallest count. In order to ensure that older batches are not more likely to be kept than newer batches, a reweighting scheme can be used to ensure the selection method is not biased.

Thus, we can either assume that saving all minority instances is possible, and thereby satisfy the memory constraints trivially. If this assumption is not applicable in practice, we can instead use either of the two forgetting techniques to ensure that HUWRS.IP maintains a finite and bounded memory footprint.

5. 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.

5.1 Implementation Details

We implemented HUWRS.IP as an extension to HUWRS in MOA [1], a stream mining suite based off of Weka [10]. For our comparative analysis, we compare against the framework of Gao et. al [8]. We also present the results of two state of the art learners for non-stationary environments, namely: Adaptive Hoeffding Option Trees (AHOT) [2], and Dynamic Weighted Majority (DWM) [15] (further information about these methods can be found in Section 8). As HUWRS.IP is an ensemble applicable to any traditional base classifier, we chose C4.4 [18] decision trees, i.e., unpruned, uncollapsed C4.5 decision trees [19] with Laplace smoothing applied at the leaves. As outlined by Kolter and Maloof [15], the base learner used used for DWM was standard Naïve Bayes.

In addition to a base learner, HUWRS.IP requires several other parameters. As recommended for many ensembles, we set the ensemble size for HUWRS.IP to $n = 100$. As in HUWRS [13] we set $threshold_{retrain} = 0.7$, and $b = 30$, and the minority class instance propagation threshold was set to $threshold_{select} = 0.9$. For Gao, we use the recommended parameters of propagating all minority instances, and undersampling the majority class by 40%.

By setting the weights once, and not optimizing them for each dataset, we set a general guideline, and provides results which are not susceptible to random noise. 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 (and thus HUWRS.IP) 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 various subspace sizes.

When testing the algorithms, we considered a batch learning framework. That is, we partitioned the dataset into multiple equal sized chunks, testing each instance in a chunk without labels first. Once all instances in a chunk were tested, we computed the area under the ROC curve (AUROC) of the classifier over the chunk, introduced labels for all of the instances, finally presenting them to the classifier for updating. The AUROC presented for each algorithm is computed as the average chunk performance of the classifier on the dataset.

Note that we use AUROC as it is the de facto standard for performance in the class imbalance literature. One important thing to note is that for some chunks no positive class instances are available. In such cases, the AUROC for that chunk is not considered when computing the average performance of the classifiers.

5.2 Datasets

To ensure a fair comparison, we tested the algorithms on multiple real world datasets, as well as a synthetic dataset. The details of the datasets are available in Table 1. 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 brief detail.

Dataset	# Ftrs	# Insts	% Minority	WS
Can	9	443,872	1.88	4,439
Compustat	20	16,956	3.86	1,131
Ozone_1h	72	2,536	2.88	363
Ozone_8h	72	2,534	6.31	363
Stagger	3	12,000	44.19	1,000
Text	11,466	11,162	6.35	1,117
Wrds	41	99,200	49.22	3,307

Table 1: Statistics for the datasets used. # Ftrs is the number of features, # Insts is the number of instances, % Minority represents the percent of instances in the dataset that are minority class, and WS denotes the window size used for the dataset.

can: *Can* is a real world dataset derived from crunching a can over time. Domain experts then marked certain instances as salient or based on their expertise. Due to the nature of the problem, the vast majority of regions are not-salient, and thus salient regions represent the minority class.

compustat: *Compustat* contains information about the fiscal health of firms over time. The class represents the default status of the firm at the current time step.

ozone_1h: The *Ozone_1h* dataset was built from hourly ozone readings over the Houston, Galveston, and Brazoria area. The class value is whether or not the current readings represent an "ozone day", i.e., they have a higher concentration of ozone than normal.

ozone_8h: *Ozone_8h* is analogous to the *Ozone_1h* dataset, however readings were taken every eight hours.

STAGGER: *STAGGER* is a non-stationary data stream generator developed by Schlimmer and Granger [21] where each instance has three nominal features. The generator then simulates concept drift by changing the meaning of each of the features over time.

text: *Text* is a text document recognition dataset, where each feature is a word. The dataset represents medical documents, where the minority class is defined as documents pertaining to carcinoma.

wrds: *Wrds* contains customer buying history for 100,000 customers over time. The positive class represents customers who respond to a marketing campaign.

6. RESULTS

In this section we discuss the performance of HUWRS.IP against various other state of the art machine learning algorithms. In particular, we compare it to the method of Gao et. al. [8], which is the state of the art learning algorithm for non-stationary environments and class imbalance. In addition, we also include three non-stationary learning algorithms which don't compensate for class imbalance, namely: HUWRS, Adaptive Hoeffding Option Trees (AHOT), and Dynamic Weighted Majority (DWM).

From the results in Table 2 there are a few interesting observations. First, we note that HUWRS does extremely poorly on *Can*. This is due to the fact that in *Can*, there are multiple consecutive batches in a row which don't have any minority class instances. Consequently if any of these batches are determined to be drifting by the classifier, there may be no minority class instances in the retraining batch. This has a severe negative impact on the performance of the ensemble, as demonstrated by its very poor performance.

In HUWRS.IP, we avoid this issue by always propagating the last batch of minority class instances to be used for instance selection. Therefore if the current batch is batch B_i, but the last batch with minority instances was batch B_{i-5}, the minority instances from batch B_{i-5} (i.e., M_{i-5}) are added to the current batch, and the instance selection classifier is built using the new hybrid batch $H_i = B_i \bigcup M_{i-5}$. In this way the classifier is able to overcome long time periods of no minority class instances.

The efficacy of this approach is borne out in the results, as we see that HUWRS.IP always outperforms HUWRS, even on the datasets which are reasonably balanced, i.e., *STAGGER* and *Wrds*. This performance increase can be explained as HUWRS.IP are able to add more diversity to the ensemble by selecting relevant instances from the past to use to train new classifiers. This diversity allows the ensemble to obtain better performance.

When comparing HUWRS.IP to Gao, we see in Table 2 that HUWRS.IP (often drastically) outperforms Gao in average AUROC. On the *Can*, *Ozone_1h*, *Ozone_8h*, *STAGGER*, and, *Text* datasets, HUWRS.IP achieves significant improvements in performance over Gao (although only minor improvements over HUWRS on *STAGGER*). On *Compustat*, HUWRS.IP shows improvements over Gao, however they are much more modest. The only dataset on which HUWRS.IP is outperformed by another method is on the *Wrds* dataset, where Gao achieves the best performance. While Gao achieves the best performance, its improvement over HUWRS.IP is rather minor.

Such a drastic improvement in performance indicates that HUWRS, combined with our instance selection mechanism, is a valuable tool for combating class imbalance and non-stationary environments.

7. ANALYSIS OF THE INSTANCE PROPAGATION TECHNIQUE

In Section 6 we presented the results of HUWRS.IP as compared against a variety of other non-stationary learning algorithms. From the results, we observed that HUWRS.IP performed very well compared to the other state of the art learning algorithms, often providing drastic performance increases. Given its effectiveness, we now compare HUWRS.IP against its constituent instance propagation mechanisms in order to determine the effectiveness of the instance selection algorithm described in Section 3.1. In particular, we compare the instance propagation method to the naïve instance propagation method due to Gao, and to a modification of Algorithm 1 with the naïve instance propagation removed.

In Table 3, we present the results of each of the methods: HUWRS.IP, HUWRS.Prop (i.e., HUWRS with naïve instance propagation), and HUWRS.IS (i.e., HUWRS with the simple instance selection). From the table, we see that HUWRS.IP generally outperforms both HUWRS.Prop and HUWRS.IS. In fact, on the *Can*, and *Ozone_1h* datasets, HUWRS.IP provides significant performance improvements over the two other methods. This indicates that for certain datasets, the instance propagation method described leverages both of the constituent parts to provide much improved performance.

There are two datasets for which HUWRS.IP performs worse than one of the other methods, namely *Compustat*, and *STAGGER*. The decreased performance on *Compustat* is due to the fact that the minority class changes drastically over time. As a result, for most ensemble members on most of the batches, the naïve instance selection mechanism selects only a few instances to add to the current batch. Since the majority of the batches have less than 5 instances propagated, the class imbalance problem remains unaddressed, and performance suffers (as seen in the performance of HUWRS.IS on *Compustat*). HUWRS.Prop, on the other hand, consistently propagates the most recent minority class instances, and thereby demonstrates superior performance. HUWRS.IP, therefore, achieves performance gains over HUWRS.IS (since there are no batches for which no instances are propagated), however fails to meet the performance of HUWRS.Prop (since there are numerous classifiers for which only 1 or 2 instances are propagated).

While on *Compustat* HUWRS.Prop provides superior performance than both HUWRS.IP and HUWRS.IS, on *STAGGER* HUWRS.IS outperforms both other methods. This is due to the fact that *STAGGER* is a reasonably balanced dataset, and therefore does not require much rebalancing. Therefore HUWRS.IS is able to correctly and accurately choose only those instances which are relevant to the current classifier on the current batch, and ignore any others. HUWRS.Prop, on the other hand, continues to propagate instances, even if they're no longer relevant. HUWRS.IP, however, is unable to realize the same performance, as it cannot be as selective as HUWRS.IP, and therefore sees a drop in performance.

From these results, we see that our novel instance propagation mechanism is effective for most datasets. There are, however, some datasets on which the naïve method provides superior performance, and others where the instance selection approach achieves superior performance. Since, however, the performance gains of using any of the naïve approaches is small compared to that of using the instance propagation method, we recommend the use of the:instance propagation method defined in Algorithm 1.

One final thing to note is that while HUWRS.IP achieves better performance, in general than Gao, HUWRS.Prop, and HUWRS.IS, generally outperform Gao as well. This

Dataset	HUWRS	HUWRS.IP	AHOT	DWM	Gao
Can	0.50306	**0.88671**	0.55544	0.78939	0.81203
Compustat	0.75660	**0.79769**	0.53347	0.75360	0.79078
Ozone_1h	0.79009	**0.87032**	0.50000	0.81463	0.77199
Ozone_8h	0.84414	**0.88243**	0.50000	0.82924	0.79160
STAGGER	0.88122	**0.89762**	0.83462	0.86972	0.68460
Text	0.91606	**0.97134**	0.61539	0.73009	0.93835
Wrds	0.90154	0.97000	0.76189	0.89973	**0.97267**

Table 2: Average AUROC results for each of the datasets and methods.

Dataset	HUWRS.IP	HUWRS.Prop	HUWRS.IS
Can	**0.88671**	0.83349	0.83335
Compustat	0.79769	**0.80572**	0.78452
Ozone_1h	**0.87032**	0.81715	0.81642
Ozone_8h	**0.88243**	0.85727	0.88180
STAGGER	0.89762	0.89875	**0.89949**
Text	**0.97134**	**0.97134**	**0.97134**
Wrds	**0.97000**	0.96686	0.96983

Table 3: AUROC results for each of the datasets for the two different instance selection methods. HUWRS.Prop represents propagating the last 20% of instances, while HUWRS.IP represents selecting instances based on the instance selection criteria.

indicates that HUWRS building classifiers on only a subset of the features is a very powerful method for overcoming concept drift.

8. RELATED WORK

Developing classifiers to overcome concept drift and class imbalance is a relatively new field of study. As a result, there are not many techniques in the literature which are designed for to work in that environment. In this section we will give an overview of the current techniques which combat concept drift and class imbalance, as well as a brief overview of the state of the art in non-stationary learners.

8.1 Non-Stationary Learners

As an extension to the standard C4.5 decision tree due to Quinlan [19], Buntine introduced option trees [3]. 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 [17]. 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. [2]. 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.

Dynamic Weighted Majority (DWM) [15] is an alternative to building decision trees that creates an ensemble of classifiers. In order to test an instance, the ensemble combines the prediction of each base classifier using a weighted vote. In order to train DWM the ensemble begins by attempting to classify each training instance. Each base classifier that misclassifies the instance has its weight reduced by a multiplicative constant β. If the entire ensemble misclassifies the instance, the ensemble adds a new classifier with a weight of 1 after rescaling the weights of the ensemble (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 trains each base classifier on the new instance.

8.2 Non-Stationary Learners for Class Imbalance

The first technique developed which attempted to learn in non-stationary environments which also exhibit class imbalance is due to Gao et. al. [8]. Gao et. al. proposed a framework based on collecting positive class examples. In their ensemble algorithm, they break each incoming chunk into a set of positive (P) and negative (Q) class instances. One then selects all seen positive class instances (AP), and a subset of the negative class instances (O) which is determined randomly based on a distribution ratio. These two sets are then combined to form a complete dataset to train the new ensemble classifier C_i. By accumulating all positive class instances, this approach implicitly assumes, however, that the minority class is not drifting.

Building on this concept, Chen and He propose SERA [6], which is similar to the proposal of Gao et. al., however instead of using all past instances, the algorithm selects the "best" n minority class instances as defined by the Mahalanobis distance. Given these instances, the algorithm then uses all majority class instances and uses bagging to build an ensemble of classifiers. Thus SERA suffers from a similar, albeit less severe, concern as the method proposed by Gao et. al., as the algorithm may not be able to track drift in minority instances depending on the parameter n.

9. DISCUSSION AND FUTURE WORK

We introduced Heuristic Updatable Weighted Random Subspaces with Instance Propagation (HUWRS.IP) as a new method for dealing with concept drift in data streams which also suffer from class imbalance. In this method an ensemble of classifiers is built as in HUWRS, however when trained each classifier selects certain minority instances to be propagated. In HUWRS.IP, the instances selected to be propagated are based on their similarity to the current minority class instances when possible, and otherwise are those instances most temporally similar to the current batch.

In order to demonstrate the effectiveness of HUWRS.IP, we compared them to a variety state of the art methods in both the concept drifting environment (HUWRS, Adaptive Hoeffding Option Trees, and Dynamic Weighted Majority), and the concept drifting environment with class imbalance (Gao). For a majority of the datasets HUWRS.IP outperforms the other methods. This indicates that HUWRS, when combined with a method for propagating minority class instances, is very effective at overcoming class imbalance and concept drift.

In addition to comparing HUWRS.IP to other state of the art methods, we also compared it to HUWRS.Prop and HUWRS.IS, where HUWRS.Prop is HUWRS augmented with the simple instance propagation technique of Gao, and HUWRS.IS is HUWRS augmented with a naïve version of the instance selection technique in Algorithm 1. This comparison demonstrated that while the instance propagation technique defined in Algorithm 1 is, in general, a significantly more effective instance propagation technique than both the naïve instance propagation mechanism and the naïve instance selection mechanism.

One current shortcoming of HUWRS (and by extension HUWRS.IP) is its dependence on observing batches of instances. Since this is a common situation in practice, it is not a major limitation, however in future work this limitation can be overcome, resulting in a more robust method which is applicable in a wider range of environments.

Two other areas of future work relate to the instance propagation mechanism, and the classifier weighting mechanism. Currently, the instance propagation mechanism merely uses the previous batch of minority instances to build the Naïve Bayes classifier. In general, however, it might be possible to more accurately select instances corresponding to the current concept. This would allow the classifier to be trained on more instances, and thus more accurately learn the current minority class concept.

Additionally, the classifier weighting mechanism assumes concept drift in the minority class and majority class are equally as important. This may prove an unreasonable assumption in some cases, and various weighting strategies can be employed so remove this assumption.

Given all of these considerations, in scenarios where data arrives in batches and is subject to either gradual concept drift or sudden concept change, as well as class imbalance, we recommend using Heuristic Updatable Weighted Random Subspaces with Instance Propagation, as it makes less assumptions about the underlying data stream, and in general achieves better performance.

Acknowledgments

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

10. REFERENCES

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

[2] 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.

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

[4] N. Chawla. Data mining for imbalanced datasets: An overview. *Data Mining and Knowledge Discovery Handbook*, pages 853–867, 2005.

[5] N. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[6] S. Chen and H. He. SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 522–529. IEEE, 2009.

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

[8] J. Gao, W. Fan, J. Han, and P. Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, pages 3–14. Citeseer, 2007.

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

[10] 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.

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

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

[13] T. R. Hoens, N. V. Chawla, and R. Polikar. Heuristic updatable weighted random subspaces for non-stationary environments. In *ICDM*, pages 241–250. IEEE, 2011.

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

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

[16] 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, 2010.

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

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

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

[20] 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.

[21] J. Schlimmer and R. Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.