Temporal Data Mining in a Dynamic Feature Space

Brent K. Wenerstrom
Bingham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd
Part of the Computer Sciences Commons

BYU ScholarsArchive Citation
Wenerstrom, Brent K., "Temporal Data Mining in a Dynamic Feature Space" (2006). All Theses and Dissertations. 761.
https://scholarsarchive.byu.edu/etd/761

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in All Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen amatangelo@byu.edu.
TEMPORAL DATA MINING IN A DYNAMIC FEATURE SPACE

by

Brent Wenerstrom

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

May 2006
This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

<table>
<thead>
<tr>
<th>Date</th>
<th>Dr. Christophe Giraud-Carrier, Chair</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>Dr. Tony Martinez</td>
</tr>
<tr>
<td>Date</td>
<td>Dr. Robert Burton</td>
</tr>
</tbody>
</table>
As chair of the candidate’s graduate committee, I have read the thesis of Brent Wenerstrom in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Dr. Christophe Giraud-Carrier
Chair, Graduate Committee

Accepted for the Department

Dr. Parris Egbert
Graduate Coordinator

Accepted for the College

Dr. Thomas Sederberg
Associate Dean, College of Physical and Mathematical Sciences
ABSTRACT

TEMPORAL DATA MINING IN A DYNAMIC FEATURE SPACE

Brent Wenerstrom

Department of Computer Science

Master of Science

Many interesting real-world applications for temporal data mining are hindered by concept drift. One particular form of concept drift is characterized by changes to the underlying feature space. Seemingly little has been done to address this issue. This thesis presents FAE, an incremental ensemble approach to mining data subject to concept drift. FAE achieves better accuracies over four large datasets when compared with a similar incremental learning algorithm.
The author would like to thank his wife for her patience and support; his adviser for the time spent in discussion, editing, and advising; Matt Smith for his feedback on many different thoughts and plans; and his parents for their examples which helped him get to this point.

The author would also like to thank Keith Copsey for insightful discussions on incremental learning. This work is funded in part by a grant from the Byrne Foundation and the BYU Bookstore, who also contributed some of the data used in this work.
## Contents

Acknowledgments v  

1 Introduction 1  

2 Related Work 5  

3 The FAE Algorithm 9  
   3.1 FAE Description 9  
   3.2 FAE Illustration 13  

4 Experimental Results 17  
   4.1 Spam Assassin 18  
   4.2 CCERT 24  
   4.3 KDD Cup 2000 26  
   4.4 BYU Bookstore 30  
   4.5 Ensemble Size Analysis 39  

5 Conclusion and Future Work 41  

Bibliography 45
Chapter 1

Introduction

Data mining has been described as “The science of extracting useful information from large data sets or databases.” [1] The attainment of this useful information from raw data requires a number of steps be taken by the analyst. According to the CRISP-DM Process [2] these steps include: gaining a business understanding, understanding the data, preparing the data, modeling the data, evaluating the model and deploying that model (see also Figure 1.1). When building predictive models, or models built to predict future events or results, the success of the data mining process hinges on finding a good fitting model or a model which accurately predicts future events from observable data. Temporal data mining is simply the extension of data mining to incorporate some element of time into the data whether through the ordering of data points or as an explicit element of each data point.

One category of modeling algorithms is that of classification. Classification is the mapping of some set of input features to a single output class, where a class is a finite set of discrete values. The mapping relies on a generalization from the training data, where the output for the given set of data points is known. Most often generalization is achieved through the discovery of patterns in the data. For example one could build a model predicting whether a new customer to an e-commerce website would spend more than $50 during their first session. After examining the model built a sample pattern that might be found would be customers whose IP came from Florida and viewed your best selling product were likely to fall into the category of big spenders.

Typically prior to model building — although this may require more than one
iteration —, the preparation of the data, or pre-processing of the data is assumed to deliver 1) a set of relevant predictive features and 2) a rich set of data defined by these features, from which a model may be induced and deployed. These assumptions hold in cases where the number of features is limited or the task remains fairly static. Some examples of such tasks are weather predictions having a small set of sensors to work with or predicting the winner in a chess game based on the end game position. There are a number of situations, however, where this assumption leads to models whose applicability is hugely restricted in either time or space. For example, if the task is to predict which new visitors to an e-commerce web site are likely to become big spenders, it is clear that any model based on static data will have a limited life span as customer behavior is notoriously volatile and subject to changes over time. Similarly, if the task is spam filtering, where features capture the presence of words in the body of an email, any model based on static features will rapidly become unusable, as one would expect that different sets of words would be most relevant to spam filtering for an individual at different periods of time.
Applications where changes are the norm rather than the exception are subject to what has been termed concept drift [3, 4]. Concept drift refers to the situation where the prediction depends on some hidden, time-varying context, not available directly or immediately to the system, and which leads to a degradation in performance, until the system is able to adapt to the change. Concept drift can be defined along the dimensions of both time and space.

In terms of time, two types of concept drift have been defined [4], namely sudden and gradual. Sudden concept drift refers to an abrupt change generally made manifest through an immediate drop in accuracy. Gradual concept drift develops more slowly and is harder to detect, since it is not easily distinguishable from noise, at first. With time the impact of gradual concept drift on the learner’s performance is much more noticeable than the impact noise would have.

In terms of space, we also find it useful to consider two types of concept drift, namely descriptional and contextual. Descriptive concept drift refers to the case where the distribution of the classes change in relation to the values of the features but the set of features itself does not change over time. Contextual concept drift corresponds to situations where the set of relevant features shifts from one set to another over time. In extreme cases, the original set of features provides no information for subsequent use in the data mining task. Clearly, both descriptional and contextual concept drift are dependent on time and can be either gradual or sudden.

Because of the temporal nature of their data, classification data mining applications subject to concept drift are excellent candidates for incremental learning [5, 6, 7], in which the algorithm adapts by updating itself one training example at a time on streaming data. Even then, concept drift remains a challenge.

This thesis presents FAE (Feature Adaptive Ensemble), a novel incremental learning algorithm for adapting to both contextual and descriptional concept drift. FAE is an ensemble algorithm that acts intuitively like a kind of political committee, where members specialize on particular issues (i.e., contexts), and are voted in and out frequently. Decisions are made by the most popular committee members and new members are voted in over old members when the old members are out of touch with
society’s trends and issues (i.e., become poor predictors).

The thesis is organized as follows. Chapter 2 briefly discusses related work. The FAE algorithm is detailed and illustrated with a simple example in chapter 3. Chapter 4 presents experimental results on 4 large temporal data mining tasks. Finally, chapter 5 concludes the thesis and points to future work.
A number of papers have addressed the issue of *descriptional* concept drift. The approaches taken typically fall within the following categories [4]: instance selection and instance weighting, which use a single learner but alter its view of the data in some way; and ensemble learning, which uses multiple learners.

Instance selection consists of focusing learning only on the most recent instances. It is generally achieved through some type of either static or dynamic winnowing mechanism [3, 8], although some approaches also exist that essentially store all instances and simply give priority to the newest instance whenever a conflict arises [9]. Instance weighting, rather than removing old instances, weighs more heavily those instances which are heuristically determined to be most relevant. One such approach, in the context of text classification and using a support vector machine is in [8].

Ensemble learning consists of using multiple learners throughout the mining exercise. Two algorithms that use ensembles for the explicit purpose of combating descriptive concept drift are the Concept Drift Committee (CDC) algorithm [10] and the Dynamic Weighted Majority (DWM) algorithm [11]. In DWM, the ensemble dynamically grows and shrinks dependent upon performance. If the ensemble as a whole misclassifies an instance, then a new learner is spawned. An user-defined parameter, beta, is used to control the weights of each learner. A learner’s weight is multiplied by beta each time that learner incorrectly classifies an instance. When a learner’s weight falls below another user-defined threshold, theta, the learner is removed from the ensemble. In CDC, the ensemble starts with a single learner and new learners are added at each time step (i.e., after each training instance) until
a maximum size $N$ is reached. At each instance, all learners are updated and the ensemble is tested on a set of instances drawn from the same distribution as the current training instance. The content of the ensemble is controlled by the maturity and performance record of the individual learners. A learner which has an accuracy on the test data below a certain threshold, has learned on enough examples (i.e., is mature) and has the lowest accuracy in the ensemble is removed and replaced by a learner using only the current instance for training. Each learner’s classification is weighted by the accuracy it achieves on the test set. The ensemble is always of the same fixed maximum size. Both the DWM and the CDC algorithms highlight the advantage of not having an explicit window, but rather a form of implicit windowing through the ensemble.

Few papers have addressed the issues of contextual concept drift. Two such examples are Zhou et al. [12] using a “dynamic feature space” and Katakis et al. [13] using dynamic feature selection in connection with incremental learning. Zhou’s approach creates two Huffman trees of word frequencies for the spam and ham (non-spam) messages seen to date. New words may be added to these trees at any time. The two Huffman trees are then combined into a ranking for each message where the spam messages tend towards one end of the ranking and the ham the other end. Then a logistic regression model is created to find the cutoff at which a message will be declared spam. To account for descriptional concept drift, the logistic regression model is periodically recomputed in which time locality is captured through sampling past email rankings used to train the model based on an approximation of exponential aging. This approach has the drawback of not being incremental. The model used must be recalculated periodically from the past emails and all past email rankings must be kept in memory.

Katakis developed an incremental approach to text classification [13]. This approach requires the use of a learner that can dynamically add features, update all current features, and select a subset of features at classification time. Both $k$-Nearest Neighbors ($k$-NN) and Naive Bayes (NB) naturally satisfy these requirements, $k$-NN by simply changing the set of features over which similarity/distance is computed
and NB by simply updating counts (since each feature is independent of the others in terms of its contribution to the classification). By incrementing the feature selection algorithm and allowing for a different set of features to be used per instance, this algorithm is able to adapt to contextual concept drift. The current implementation uses NB as the learner and the $\chi^2$ statistic for feature selection.

The algorithm proposed in this thesis, Feature Adaptive Ensemble (FAE), may be viewed as an extension of the CDC algorithm in that it uses an ensemble, but with the same emphasis as Katakis’ on contextual concept drift. There are a number of significant distinctions between FAE and CDC. The learners in CDC are decision trees. Hence, the system must store all training instances and incremental learning is achieved at the cost of re-training all learners after each instance. By contrast, FAE uses NB learners which are inherently incremental. In CDC, all learners use all of the features, which assumes that all features must be available a priori. By contrast, each learner in FAE is specialized on some subset of features and new features may be added at any time. Finally, in CDC, performance is measured as accuracy on a test set whose distribution is the same as that of the current training instance. This assumes that the instance’s distribution is known (or can be approximated). With most applications, especially incremental ones where instances, in principle, become available one at a time over time, one does not have the luxury of additional test sets. Furthermore, one can argue that in the context of a truly incremental task, such a test set makes no sense. In incremental learning, the next instance is both a training and a test instance. And knowing the distribution of each training instance seems to beg the question of learning. By contrast, in FAE, performance is measured as the accuracy obtained from past classifications made on each instance before training on that instance. In addition to being an ensemble, FAE also differs from Katakis’ algorithm in that it may use any incremental learning algorithm in the ensemble (e.g., [6, 14, 15]). The requirements of Katakis’ algorithm as seen above are far more restrictive. Lastly, Katakis’ algorithm uses no memory saving mechanism while using every attribute seen to date in learning. This means that any task which could indefinitely derive or create new attributes, would eventually run out of resources.
Chapter 3

The FAE Algorithm

The FAE algorithm is inspired by the CDC algorithm [10]. FAE, like CDC, is able to adapt well to change by considering several learners of differing “ages.” By using learners of differing ages the most recent instances are weighted most. This is the case because all learners have trained on the most recent instances, with a diminishing number of learners having been trained on earlier instances. Also, FAE makes decisions on the removal of learners from the ensemble based on individual learners’ accuracy.

3.1 FAE Description

A high-level view of the FAE algorithm in the form of pseudo-code is shown in Figure 3.1 which will be explained in detail. The ensemble is initialized containing zero learners with each of its parameters (see Table 3.1) set by the user (line 1). The feature selection (FS) algorithm is initialized and updated on the first instance (line 2). The FAE algorithm is not tied to any specific feature selection algorithm, but the feature selection algorithm chosen should support incremental training.

For the experiments conducted in this thesis a $\chi^2$ statistic is used for feature selection. The $\chi^2$ statistic is obtained through the Weka library [16] using the `weka.core.ContingencyTables` class where a two dimensional array is passed into the function `chiVal(double[][], boolean useYates)`. This array contains counts of how many times a specific value was seen in the given attribute for a specific target class. Example data can be seen in Table 3.2 and the corresponding array can be seen in Table 3.3. Before creating the matrix for numeric values, the
1 Create empty ensemble with parameters: $m, p, f, r, N, M$ (see Table 3.1)
2 Update FS on $i_1$
3 Train $L_1$ on $i_1$
4 Add $L_1$ to ensemble
5 For instances $i_2..i_{last}$
6 Record accuracy of ensemble on $i_j$
7 Update FS on $i_j$
   # Each learner updates its model with the current instance,
   # either through retraining or incrementing
8 Train each $L$ in ensemble on $i_j$
   # $nfs$: the set of features currently selected by FS
   # $yfs$: the set of features used by the youngest learner
9 $\delta \leftarrow \frac{|nfs - yfs|}{M}$
10 If $\delta > f$ or (age of $L_{youngest} \geq r$ and accuracy on past $N <$
   # accuracy on past $2N$)
11 $threshold \leftarrow \frac{\min(a) + \max(a)}{2}$
   # Learners below threshold: increment probation time
   # Learners above/equal to threshold: probation time $\leftarrow 0$
12 Update probation time of each $L$ with $threshold$
13 Remove any $L$ with probation times $\geq p$
14 Create new $L$ using top $M$ features from FS, trained only on $i_j$
15 Add newly created $L$ to ensemble
16 End if
17 End for

Figure 3.1: Pseudo-code for the FAE Algorithm

values are put into five bins with an equal range. Unknown values are spread equally across the possible values that the attribute could have contained. By using some type of feature selection algorithm the number of inputs per learner in the ensemble is limited. Next, a new learner ($L_1$) is trained on the first instance and added to the ensemble (line 3 and 4). After priming FAE with a single instance, FAE is ready for incremental learning.

The FAE algorithm was created under the assumption that though we do not know future circumstances, our best guess is the immediate past. Therefore, learners are weighted and judged for removal based on the accuracies achieved per learner on the immediate past. At creation time, each learner is trained on a single instance and is given a default accuracy of 1 of 1.
<table>
<thead>
<tr>
<th>param.</th>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>maturity</td>
<td>The number of instances needed before a learner’s classifications are used by the ensemble.</td>
</tr>
<tr>
<td>$p$</td>
<td>probation time</td>
<td>The number of times in a row a learner is allowed to be under the removal threshold before being removed.</td>
</tr>
<tr>
<td>$f$</td>
<td>feature change threshold</td>
<td>The amount of change between the youngest learner’s set of features and the top $M$ set of features (see equation 3.1).</td>
</tr>
<tr>
<td>$r$</td>
<td>growth rate</td>
<td>The number of instances between when the last learner was added and when the ensemble’s accuracy is checked for the addition of a new learner.</td>
</tr>
<tr>
<td>$N$</td>
<td>number of instances</td>
<td>The number of instances over which to maintain an accuracy measure for use in the ensemble.</td>
</tr>
<tr>
<td>$M$</td>
<td>number of features</td>
<td>The number of features selected by the feature selection algorithm to be used by a newly created learner.</td>
</tr>
</tbody>
</table>

Table 3.1: A description of the parameters used in the FAE algorithm

<table>
<thead>
<tr>
<th>#</th>
<th>FREE</th>
<th>Spam?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>spam</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>ham</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>ham</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>spam</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>spam</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>ham</td>
</tr>
</tbody>
</table>

Table 3.2: $\chi^2$ example data set where “FREE” is the attribute and “Spam?” is the target class

<table>
<thead>
<tr>
<th></th>
<th>ham</th>
<th>spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.3: Resulting array from counting (attribute, class) pairs from Table 3.2 (count[attribute index][class index])
For tests conducted in this thesis we record the accuracy of the ensemble on each new instance (line 6), then update the FS algorithm and the ensemble. The performance of each learner is updated using its prediction on the current instance before each learner is trained on this same instance (line 8). Training each learner involves either incrementing the learner’s model in the case of incremental learners, or retraining the learner on all instances seen since the learners creation.

New learners are introduced to adapt to concept drift. To counter contextual concept drift, new learners are added when the FS algorithm has chosen a different set of features as most promising. More specifically, when the percentage change in features, or $\delta$ (see equation 3.1 and line 9) is greater than the feature change threshold, or $f$, then a new learner will be created (line 10). In equation 3.1, $nfs$ refers to the current set of features suggested by the FS algorithm and $yfs$ refers to the set of features used by the youngest learner in the ensemble. The parameter $M$ refers to the number of features to be used by each learning algorithm.

$$\delta = \frac{|nfs - yfs|}{M}$$  \hspace{1cm} (3.1)

To counter descriptional concept drift, where the distributions of values for some features as they relate to the output class may change, new learners are added when the performance of the ensemble as a whole is degrading (line 10). This algorithm defines degrading performance as the accuracy of the ensemble over the past $N$ instances is lower than the accuracy over the past $2N$ instances, where $N$ is a the window over which each learner maintains accuracy counts. To avoid adding a learner every instance, when perhaps little has changed in terms of concept drift, but the degradation is still true, a new learner cannot be added for degradation unless no new learner has been added for the number of instances equal to the growth rate period, or $r$.

When one of the conditions for the addition of a new learner is met, then we must update the probation times of each new learner (line 12), remove any learners who have been on probation long enough for removal (line 13), and finally add a new learner (line 15). To update the probation time of a learner, we first calculate
the threshold which determines the set of learners that will be put on probation. This threshold is calculated by taking the average of the best and worst accuracies of learners (see equation 3.2) that have attained maturity, or have been trained on \( m \) instances. All learners below this threshold have their probation count incremented, while the rest of the learners have their probation times set to zero. Any learner with a probation count greater than or equal to the maximum probation time, or \( p \), after their probation times are updated are removed.

\[
threshold = \frac{\min(a) + \max(a)}{2} \quad (3.2)
\]

When a new learner is created it is trained on the current instance and uses only those features prescribed by the FS algorithm. Never at any future time does a learner change that set of features that were used at its creation. This provides the flexibility of allowing any incremental learning algorithm to be part of the ensemble.

Classification using FAE is based on a weighted vote of the learners in the ensemble. All learners that have not reached maturity (trained on at least \( m \) instances) or have a non-zero probation time are given a weight of zero. From there, each learner is weighted according to the accuracy of that learner over the past \( N \) instances. The weight of each learner is added to the total weights for the class chosen by that learner. The class with the largest sum of weights is the output for the ensemble. In the case where none of the learners have reached maturity (all instances before the first learner has been trained on \( m \) instances), the learners are weighted by their ages, or the number of instances trained on per learner.

### 3.2 FAE Illustration

This section presents a simple example of how the FAE algorithm works. The example data set that will be used for this illustration is shown in Table 3.4. The target class or field to predict is the last column labeled “target.” The first column is not used for training but shows the sequence number of each example (i.e., the time at which the example became available). The other columns are used for training. A
Table 3.4: Example data set for the FAE illustration

<table>
<thead>
<tr>
<th>t</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.5: State of the ensemble over time where each tuple (<feature, weight, probability time>) represents a learner

<table>
<thead>
<tr>
<th>t</th>
<th>ensemble makeup</th>
<th>cum. acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[('a', 1, 0)]</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>[(‘a’, 1/2, 0), (‘c’, 1/1, 0)]</td>
<td>0/1</td>
</tr>
<tr>
<td>4</td>
<td>[(‘a’, 1/7, 1), (‘c’, 1/1, 0), (‘b’, 1/1, 0)]</td>
<td>0/2</td>
</tr>
<tr>
<td>5</td>
<td>[(‘c’, 1/3, 1), (‘b’, 2/3, 0), (‘d’, 1/1, 0)]</td>
<td>1/3</td>
</tr>
<tr>
<td>7</td>
<td>[(‘c’, 0/7, 1), (‘b’, 2/3, 0), (‘d’, 2/3, 0)]</td>
<td>2/4</td>
</tr>
</tbody>
</table>

‘-’ means that the corresponding attribute is not available to the ensemble allowing new attributes to become available over time.

For this example the parameters to FAE are set as follows: \( m = 1, p = 2, f = 0.1, r = 1, N = 3, \) and \( M = 1. \) These parameter settings are for simplicity’s sake and may lead to poor results in real-world settings. The feature selection algorithm that is used for this example ranks features by how accurately the output is predicted by the value of a given attribute in the past.

The base learner used in this example is a very simplistic learner. This learner outputs the value of its single input. In general more advanced learners would be used (e.g., Naive Bayes, artificial neural network, incremental decision tree).

The state of the ensemble after training on each instance is shown in Table 3.5. Before FAE trains on any of the instances the ensemble is empty. The feature selection algorithm is updated on the first instance. The values assigned to the two available attributes are ‘a’ = 1 and ‘b’ = 0 (or ‘a’ would be 100% accurate in predicting the output class by its value and ‘b’ would be 0% accurate). A learner is then trained on the first instance, and will use ‘a’ as its input as ranked by the FS algorithm and...
set its weight or past accuracy to $\frac{1}{2}$ and its probation time to 0 by default. The newly created learner is added to the ensemble (see the state of the ensemble at $t = 1$).

On the second instance ($t = 3$), the accuracy of the ensemble is first recorded. The ensemble has a single learner which classifies this instance as having a value of 1. The actual value for this instance’s target class is 0, so the cumulative accuracy is 0/1 or 0%. The feature selection algorithm is updated now such that ‘a’ = 0.5, ‘b’ = 0.5, and ‘c’ = 1. The accuracy of the first learner is updated to $\frac{1}{2}$.

On this second instance $\delta = \frac{1}{4} \text{ (or the current top } M = 1 \text{ features differ by one from the top feature when the last learner was added)}$ which is greater than $f$, which meets one of the conditions for adding a new learner. Before adding that new learner, we update the probation time for the existing learner in the ensemble. The threshold value is the weight of the only learner, meaning that this learner is not below the threshold and the probation time of the initial learner is set to zero. Then, we create a learner using the feature ‘c’ as its input adding it to the ensemble with other values set to default.

At time $t = 4$, we find that both learners predict the wrong class and the cumulative accuracy is now 0/2. We update the feature selection algorithm to ‘a’ $\approx 0.33$, ‘b’ $\approx 0.67$, and ‘c’ = 0.5, which means we have a new top feature, ‘b’, and will add a new learner after updating the accuracies of each learner. The accuracies of each learner are updated. Before adding a new learner, probation times must be updated. The threshold for probation is $\frac{1/3+1/2}{2} \approx 0.42$. The first learner added is below that threshold and has its probation time incremented to one. We add another learner to the ensemble which uses ‘b’ as input.

At time $t = 5$, the sum of weights for an output of zero is the sum of the weights for the learner using ‘a’ and ‘b’ ($0 + \frac{1}{4} = 1$) and for an output of one is the weight for the learner using ‘c’ ($\frac{1}{2}$). The learner using ‘a’ had a weight of zero, because it had a non-zero probation time. The correct output was zero, which was also the class with the highest weight and was predicted correctly. The FS algorithm is updated to ‘a’ = 0.5, ‘b’ = 0.75, ‘c’ $\approx 0.33$, and ‘d’ = 1. Each learners’ accuracy is updated ($a' = \frac{1}{3}$, ‘b’ = $\frac{3}{4}$, ‘c’ = $\frac{1}{3}$). Again, a new learner must be added with a new
feature taking the top rank. The threshold for probation is $\frac{1/3+1}{2} \approx 0.67$. Both the learners using ‘a’ and ‘c’ have their probation times incremented, both falling below the threshold. However, now that the learner using ‘a’ has a probation time equal to $p$ (which was set to 2) then this learner is removed from the ensemble. Again, a new learner is introduced using the feature ‘d’.

On the final instance, the sum of weights are class zero $= 0$ and class one $= \frac{3}{3} + \frac{2}{2} = 2$. Class one has the highest sum of weights and is also the correct class. As a result the final cumulative accuracy is 2/4, concluding this example.
Chapter 4

Experimental Results

This chapter presents a series of empirical results on four separate data sets, each of which is a live stream of data. Each data set exhibits contextual concept drift as well as the possibility of descriptional concept drift.

For each data set, the FAE algorithm is compared to Katakis’ algorithm, since Katakis’ algorithm is a general solution to contextual concept drift. In comparing the two algorithms the cumulative accuracy of each algorithm is plotted over time through all of the instances available. Cumulative accuracy was computed in the following way. First, each algorithm was trained on the first instance. Then, on each subsequent instance each algorithm predicted the class of the incoming instance. Counts were maintained of both the number of instances seen to date and the number instances classified correctly by each algorithm. The cumulative accuracy was the number of instances correctly classified divided by the number of instances seen to date. Additionally, some examples show a windowed accuracy in which counts were kept only on the last 50 instances. For example on the fifty-first instance the windowed accuracy would only include the accuracy over the past 50 instances and would not include the first example seen.

Both algorithms use the same $\chi^2$ statistic for feature selection. The parameter settings for FAE on all data sets are $m = 5, p = 3, f = 0.15, r = 10$, and $N = 50$. $M$ is equal to 250 on the Spam Assassin, CCERT and BYU Bookstore data sets and 50 on the KDD Cup 2000 data set (see Table 4.1). The values chosen for $m, p, f$, and $r$ showed slightly better results empirically than a small set of values for each parameter. To show the robustness of these parameters, the same parameters were used on all
Table 4.1: Parameter settings for experiments ($M = 250$ on Spam Assassin, CCERT and BYU Bookstore experiments and $M = 50$ on KDD Cup 2000 tasks (with the exception of $M$ on the KDD Cup 2000 data set). The parameters were not finetuned per task. FAE’s ensemble was composed of only Naive Bayes learners (as implemented in Weka [16]), for comparison with Katakis’ algorithm which in these experiments used the Naive Bayes algorithm as its base learner.

4.1 Spam Assassin

Spam filtering on live data was shown by Fawcett [17] to contain a number of challenges to data mining algorithms such as skewed and changing class distributions, unequal and uncertain error costs, complex text patterns, complex temporal characteristics, and adaptive adversaries. For the algorithms under test the most interesting aspect which would be found in a repository are that the class distributions are skewed and changing and that there are complex temporal characteristics. For example the Spam Assassin repository when streamed contains periods of extreme behavior such as periods of all ham and periods of all spam, as can be seen from Figure 4.1. Additionally, spam filtering is likely to contain contextual concept drift in which the set of most relevant features shifts from one set to another.

The Spam Assassin repository\textsuperscript{1} includes 6,047 email messages, 4,150 of which are labeled as ham and the remainder as spam. The first email includes 252 different tokens (where a token refers to a sequence of only characters, only numbers, or a Chinese character separated by space or punctuation). By the end of the data set,\textsuperscript{1}http://spamassassin.apache.org/publiccorpus/
Figure 4.1: Distribution of Spam Emails in the Spam Assassin Data Set, where each point represents the next 50 emails.
there are over 65,000 different tokens and an overlap of 72 tokens between the top $M$ tokens after the first instance according to $\chi^2$ and the top $M$ tokens after the last instance.

Before parsing out the tokens, we remove the header information (except the subject line), all HTML tags, attachments, and any token longer than 25 characters (to avoid accidentally including encoded binary files found in the email). Each token seen in an email to date becomes a binary feature with a value of ‘1’ for the presence of that token in the email and ‘0’ if the token is not present. For an example of this parsing in practice see Figures 4.2 and 4.3 parsed in order into instances in Table 4.2.

The emails contained in the Spam Assassin repository have been used by a number of researchers for differing purposes. Most notably, Katakis used these emails to demonstrate the usefulness of a dynamic set of features for spam classification versus both an incremental learner using a fixed set of features and a traditional

20
learner with a fixed set of features who stopped learning after having been trained on a small set of instances. Hence, this particular comparison may be regarded as the FAE algorithm versus the best of Katakis’ algorithm. Figure 4.4 shows the comparison of the cumulative accuracies over time of the FAE algorithm versus Katakis’ algorithm.

In comparing the graphs of cumulative accuracy (Figure 4.4) and the distribution of spam amongst the emails (Figure 4.1), one can see that both algorithms’ cumulative accuracies begin to take a dive around instance 1,000 where there is a shift from all incoming email being spam to all incoming email being ham. This dive continues as the majority class over the past 50 emails jumps back and forth between the ham and spam classes. This is the period of time where the FAE algorithm achieves its biggest advantage over Katakis algorithm with a six percentage point advantage demonstrating the FAE algorithm’s ability to adapt more rapidly. This edge narrows until the difference between the two algorithms is only two percentage points at the end. Depending on the user a six percentage point advantage could mean the difference between a good user experience and a bad user experience. Overall both algorithms achieve cumulative accuracies above 90%.

The windowed accuracy graph shown in Figure 4.5 shows the accuracies at each point in time achieved over the past 50 instances. Both algorithms achieve approximately equal performance except for two large dips by Katakis’ algorithm near instances 1,000 and 1,500, and a small dip by Katakis’ algorithm near instance 3,500 and a small dip by FAE near instance 6,000. These small differences in accuracy over short periods of time account for the large differences seen in the cumulative accuracy graph (Figure 4.4).
Figure 4.4: Cumulative Accuracy on Spam Assassin
Figure 4.5: Windowed Accuracy on Spam Assassin (window size = 50)
4.2 CCERT

CERNET Computer Emergency Response Team released two large sets of Chinese emails for the months of June and July of 2005\(^2\). We focus here on the larger of the two, June 2005. The spam and ham messages were unfortunately collected from different sources over different periods of time. The spam messages were collected using a honeypot technique where emails were collected which were sent to ‘anyone@ccert.edu.cn’ (where ‘anyone’ represents all possible strings), while the ham messages were collected from Chinese public forums. Due to the differences in date and time ranges between the two sets of messages, the ham messages were randomly distributed amongst the spam messages, maintaining an approximately equal ratio of 2.7 spam messages to every 1 ham message throughout the life of the simulated stream.

The features used are derived from the bodies of the emails using the same techniques used in cleaning and parsing the Spam Assassin repository (section 4.1). The number of features created in the first email was 105 and the number of features seen by the last email was 19,811. This is much smaller than the number of features seen in the Spam Assassin data set even though there are a great many more messages in the CCERT data set. This may be explained by the cleanliness of the data. Most of the features seen in this data set are valid Chinese characters, while there are a great number of invalid English words in the Spam Assassin data set.

The cumulative accuracies of the FAE algorithm and Katakis’ algorithm are shown in Figure 4.6. Both algorithms maintain similar cumulative accuracies above 90% for most of this data set. This high level of accuracy is likely due to class distributions keeping constant and each class coming from different sources.

If we zoom in and look at what takes place before these two algorithms reach 90% accuracy as shown in Figure 4.7, we can see that Katakis’ algorithm takes an early lead for a few instances but quickly loses that lead. However, the FAE algorithm achieves a cumulative accuracy over 90% around instance 200, while Katakis’ algorithm takes until just before instance 600 to reach the same level. After instance

\(^2\)http://www.ccert.edu.cn/spam/sa/datasets.htm
Figure 4.6: Cumulative Accuracy on CCERT
1,000 the two algorithms stay within about 1% of each other until the end. The windowed results are not shown for this data set, since there is no additional information contained in the windowed graph on the comparison between these two algorithms.

### 4.3 KDD Cup 2000

The KDD Cup 2000[^1] contains clickstream data from Gazelle.com, a web retailer that closed during the year 2000. The clickstream data included fields from user surveys. The questions used in the survey were changed from time to time. This lead to a number of fields in the database containing valid entries for certain periods of time, suggesting the existence of contextual concept drift. Additionally, during the time from which this data was collected different advertising campaigns came and went, each attracting a different distribution of buyers with differing preferences. With the population of user traffic changing over time due to different advertisements, it is expected that a certain amount of descriptional concept drift may also appear in this data set.

The data set created here is for question 2 of KDD Cup 2000: “Given a set of page views, which product brand will the visitor view in the remainder of the session?” The clickstream data had already been pre-aggregated into sessions. Amdocs, who were the winners for question 5 (characterize which product brand will be viewed rather than predict), created a multi-step process to classify each session (see Figure 4.8). The first step in the process screened out sessions likely to be robots by checking if the length of the session was one. Those sessions passing the first step were then classified as likely to view a product or not. A large majority (close to 95%) did not view one of the top brands. By filtering out those most likely to not view a top brand, the class distributions would allow a learning algorithm to more easily distinguish the difference between users viewing one top brand versus another. Both of these steps were performed on both the data trained on and the data points to classify. Additionally, the training data had sessions viewing multiple brands screened from it.

Figure 4.7: Cumulative Accuracy on CCERT from Instance 0 to Instance 700
We followed the same steps to create the simulated data stream. FAE and Katakis’ algorithm never saw any of the sessions of length one, viewing multiple brands, or viewing none of the top brands. There were 8,135 sessions remaining after filtering. Additionally, the two algorithms being tested were not allowed to know of the existence of a feature until a session contained a known value for that feature. The full set of features was available to the learning algorithm shortly after the stream began, due to the filtering. This led to much less potential for contextual concept drift. The distributions of the differing classes did make sudden changes through the test, which strongly suggests the presence of descriptive concept drift. Figure 4.9 shows the percentage of each class throughout time where each shaded area represents the relative percentage of each class on the previous 50 instances. On the first several examples before instance 1,000, the class “DK” was not present. From around instance 1,750 to instance 2,750 the majority class was “AE,” while that same class became a minority towards the end of the task.

Figure 4.10 shows cumulative accuracies. There are multiple crossing points along the two curves. The FAE algorithm peaks at the moment that the “DK” brand

Figure 4.8: Amdocs multi-step process for classifying user sessions for question 5 of the KDD Cup 2000
Figure 4.9: Distribution of Classes per 50 Instances on KDD Cup 2000 Data Set
is introduced around instance 1,000. At that time in which a new class appears that was never before seen, both algorithms take a hit, though the FAE algorithm surprisingly falls farther in accuracy at this point in time. However, from instance 2,000 on the general trend of the FAE algorithm is increasing, where during this same period Katakis’ algorithm has a negative slope showing the adaptive power of the FAE algorithm to descriptional concept drift. In summing the area under the curve, the FAE algorithm has a slight edge over Katakis’ algorithm, having longer periods of time where it dominates.

4.4 BYU Bookstore

The BYU Bookstore\(^4\) sells a variety of items online (e.g., text books, apparel, DVD’s). This experiment uses the clickstream data gathered for the online traffic to the BYU Bookstore from April 2005 until April 2006. This data includes 7,202,767 page views received from 297,024 different IP addresses.

For this experiment the clickstream data was organized by sessions. This process was helped by a session ID which was included with most hits (created using a cookie). Page views were bucketed by IP, then by session ID when available. When the session ID was not available then a page view was bucketed with the page view of the same IP with the closest time stamp. Buckets were ordered chronologically and any consecutive page views that were more than 30 minutes apart were separated into different sessions. Any page views not separated became a session. All sessions including more than 200 or less than three page views were removed from the data set, both to avoid uninteresting sessions and to remove bot sessions.

The idea for this task was taken from question 1 in the KDD Cup 2000 competition. The point of such an exercise is to find patterns which designate the ending of a session to find those pages which perhaps are leading to the early departure of customers from the site. The task designed for this experiment randomly assigns each session to one of two groups with a constant probability. The first group removed

\(^{4}\text{http://byubookstore.com}\)
Figure 4.10: Cumulative Accuracy on KDD Cup 2000
Table 4.3: Example parsing of sessions into attributes

<table>
<thead>
<tr>
<th>original session</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A  B  C</td>
</tr>
<tr>
<td>A → B</td>
<td>1  1  0</td>
</tr>
<tr>
<td>A → B → C</td>
<td>1  1  1</td>
</tr>
<tr>
<td>A → A</td>
<td>1  0  0</td>
</tr>
</tbody>
</table>

some random number of page views from the end of the session. The random number removed can be from one to all but two page views in the session. The second group left the sessions in it unchanged. In this specific case the attributes used were the pages visited in the session and patterns of length two signifying what page was followed by what other page [19]. The patterns found were very simple and an example extraction of these patterns is shown in Table 4.3. For example the session “A→B→C” includes two distinct patterns of length two: A→B and B→C. For the sake of simplicity, the pattern A→C was disallowed or any other pattern described by a page view eventually followed by another page view. Additionally, the final page view was included in the data set in the form of the length two pattern: C→end.

In addition to cleaning the data, the size of this task required some memory optimization. Both Katakis’ algorithm and the χ² statistic were altered such that attributes that were rarely true were “forgotten.” For this specific task if the percent of the time that an attribute was found true was less than $\frac{1}{250}$ (empirical results showed promise using this setting) then the attribute was removed from both the counts used for the χ² ranking and from Katakis’ algorithm. In this way, an ever growing number of attributes was less likely to cause memory problems than storing every attribute seen to date in each algorithm. This extension is not part of Katakis’ algorithm. It was implemented here for fairness, to avoid the ‘ever-growing’ problem of Katakis’ approach (see Related Work).

The cumulative accuracies for Katakis’ algorithm and FAE can be seen in Figure 4.11. Katakis’ algorithm starts out doing quite well, but monotonically decreases in accuracy towards an asymptote at around 63%. The FAE algorithm takes
an early dive, but recovers rapidly. However, its accuracy remains below the accuracy of Katakis’ algorithm throughout the task, even if only by 3%.

Our suspicion was that the constant class distribution favors Katakis’ algorithm seeing that Katakis’ algorithm maintains a higher accuracy than the FAE algorithm. This could be explained by the fact that with Katakis’ algorithm maintaining a single instance of Naive Bayes the class priors are likely to be more accurate than new learners introduced in the FAE algorithm. The distribution of shortened sessions over time is shown in Figure 4.12, with averaging used over 100 points where each point is the percentage of shortened sessions in 50 consecutive sessions, without overlapping between points. The full range of percentages of shortened sessions per 50 can be seen in Figure 4.13 which appears to be normally distributed around 0.5.

However, we were concerned that the large rate of change in the top 250 features might be a confounding factor that could also favor Katakis’ algorithm. When we compared the rate of change in the Spam Assassin data set and the current data set we found that the rate of change in the Spam Assassin data set was over 9 times as high for the first 6,000 instances. Yet the FAE algorithm performs better than Katakis’ algorithm on the Spam Assassin data set.

To test the hypothesis that a constant class distributions may indeed favor Katakis’ algorithm, a new experiment was run on this same data with the exception that every 1,000 instances the likelihood of shortening a session was randomly assigned a value between 0 and 1 inclusive. The resulting distribution of shortened sessions per 50 instances is approximately uniform (see Figure 4.14 or averaging the points over time see Figure 4.15).

After randomizing the distribution over time, the new cumulative accuracies can be seen in Figure 4.16. Over the first about 15,000 instances, Katakis’ algorithm has a better cumulative accuracy, due again probably to the rate of change of the top 250 features. However, from that point on the advantage that the FAE algorithm offers for adapting to local patterns, such as a differently skewed distribution for each set of 1,000 instances, is seen with the FAE algorithm taking the top cumulative accuracy. By the end the FAE algorithm is nearly 10% more accurate than Katakis’
Figure 4.11: Cumulative Accuracy on the BYU Bookstore clickstream
Figure 4.12: Class distributions on BYU Bookstore sessions over time averaging over 100 sets of 50 instances
Figure 4.13: Class distributions on BYU Bookstore sessions
Figure 4.14: Class distributions on BYU Bookstore sessions, when distributions randomized
Figure 4.15: Class distributions on BYU Bookstore sessions, when distributions randomized, over time averaging over 100 sets of 50 instances
algorithm on this data set. The results from the BYU Bookstore data suggest that the FAE algorithm is likely to perform better in situations where the class distributions are constantly changing, similar to the setup of the second experiment on the data. While a constant class distribution favors Katakis’ algorithm. We suspect that a highly variable class distribution occurs in many real-world applications (e.g., spam filtering).

4.5 Ensemble Size Analysis

Under certain conditions, the FAE algorithm will grow unrestrained. This is seen in both the Spam Assassin and CCERT data sets, while the size of the ensemble tended to stay under 20 for most of the task for the BYU Bookstore and KDD Cup 2000 tasks. In the Spam Assassin data set there exists a short period between instances 5,000 and about 6,000 in which the FAE ensemble grew to 383 learners. Even worse in the CCERT data set. This task is a simple task for the base learners allowing learners to escape removal before the end of probation time. In this task the ensemble steadily grew through out the task, peaking near the end of the task at 370 learners. In the case of the CCERT task it was found that setting the probation time ($p$) to 1, kept the size of the ensemble below 7 learners, with only a 2% drop in accuracy by the end of the task. However, after changing the probation time to 1 for the Spam Assassin data set, the same peak still existed. One may wish to impose a maximum size on the ensemble for certain data sets.
Figure 4.16: Cumulative Accuracy on the BYU Bookstore clickstream (second run)
Chapter 5

Conclusion and Future Work

This thesis presents a novel algorithm, Feature Adaptive Ensemble (FAE), for temporal data mining which successfully handles contextual concept drift. FAE uses a dynamically-sized ensemble algorithm composed of learners of differing “ages” and using different sets of features over time as selected by a feature selection algorithm. Learners are created when the top $M$ features, according to the feature selection algorithm, have changed sufficiently or when the ensemble’s performance appears to be degrading. A learner is removed when its accuracy over the past several instances remains among the lower half of accuracies in the ensemble for a probationary time.

The performance of the FAE algorithm was tested on four large temporal data sets and shown to be generally better than Katakis’ algorithm, with the exception of the constant class distribution found in the first run over the BYU Bookstore’s clickstream data, suggesting that the FAE algorithm is able to adapt readily to both contextual and descriptional concept drift. The results from the BYU Bookstore were especially useful in suggesting that the FAE algorithm would outperform Katakis’ algorithm if the class distributions were constantly fluctuating, which is a condition which we suspect happens often in real-world scenarios. These results demonstrate promise and lay the groundwork for future research in predictive modeling in temporal data mining.

The number of user defined parameters currently available in the FAE algorithm is more than would be desirable to most data miners. A number of approaches could be tested for minimizing the number of user defined parameters. For example the maturity or feature change threshold could be made to be adaptive to the task...
at hand, or perhaps a good $N$ could be found that works well for most tasks. Thus, making the algorithm more readily available for use without an understanding of the inner workings of the algorithm.

Currently, the feature selection algorithm tested had no element of windowing or adaptation. Creating a windowed version of the $\chi^2$ statistic for feature selection could improve the ability of the FAE algorithm to adapt, beyond that seen in the results.
Bibliography


