Dynamic Weighted Majority: An Ensemble Method for Drifting Concepts[*]

J. Zico Kolter   KOLTER@CS.STANFORD.EDU
Department of Computer Science
Stanford University
Stanford, CA 94305-9025, USA

Marcus A. Maloof[†]   MALOOF@CS.GEORGETOWN.EDU
Department of Computer Science
Georgetown University
Washington, DC 20057-1232, USA

Editor: Dale Schuurmans

Abstract

We present an ensemble method for concept drift that dynamically creates and removes weighted experts in response to changes in performance. The method, dynamic weighted majority (DWM), uses four mechanisms to cope with concept drift: It trains online learners of the ensemble, it weights those learners based on their performance, it removes them, also based on their performance, and it adds new experts based on the global performance of the ensemble. After an extensive evaluation—consisting of five experiments, eight learners, and thirty data sets that varied in type of target concept, size, presence of noise, and the like—we concluded that DWM outperformed other learners that only incrementally learn concept descriptions, that maintain and use previously encountered examples, and that employ an unweighted, fixed-size ensemble of experts.

Keywords: concept learning, online learning, ensemble methods, concept drift

1. Introduction

In this paper, we describe an ensemble method designed expressly for tracking concept drift (Kolter and Maloof, 2003). Ours is an extension of the weighted majority algorithm (Littlestone and Warmuth, 1994), which also tracks drifting concepts (Blum, 1997), but our algorithm, dynamic weighted majority (DWM), adds and removes *base learners* or *experts* in response to global and local performance. As a result, DWM is better able to respond in non-stationary environments.

Informally, concept drift occurs when a set of examples has legitimate class labels at one time and has different legitimate labels at another time. Naturally, over some time scale, the set of examples and the change they undergo must produce a measurable effect on a learner's performance. Concept drift is present in many applications, such as intrusion detection (Lane and Brodley, 1998)

---

[*]. Based on "Dynamic Weighted Majority: A new ensemble method for tracking concept drift", by Jeremy Z. Kolter and Marcus A. Maloof, which appeared in the Proceedings of the Third IEEE International Conference on Data Mining. © 2003 IEEE.

[†]. Corresponding author

and credit card fraud detection (Wang et al., 2003). Indeed, tracking concept drift is important for any application involving models of human behavior.

In previous work, we evaluated DWM using two synthetic data sets, the STAGGER concepts (Schlimmer and Granger, 1986) and the SEA concepts (Street and Kim, 2001), achieving the best published results (Kolter and Maloof, 2003). Using the STAGGER concepts, we also made a direct comparison to Blum's (1997) implementation of weighted majority (Littlestone and Warmuth, 1994).

Here, we present additional results for the STAGGER data set by making a direct comparison to our implementation of the STAGGER learning system (Schlimmer and Granger, 1986) and to a series of rule learners for concept drift that use the AQ algorithm: AQ-PM (Maloof and Michalski, 2000), AQ11-PM (Maloof and Michalski, 2004), and AQ11-PM+WAH (Maloof, 2003). We also present results for two real-world data sets. The first is the CAP data set (Mitchell et al., 1994), which Blum (1997) used to evaluate his implementation of weighted majority. The second is a data set for electricity pricing, which Harries (1999) used to evaluate Splice2. Finally, we include results for twenty-six UCI data sets (Asuncion and Newman, 2007) because we were interested in evaluating DWM's performance on static concepts.

Overall, results suggest that a weighted ensemble of incremental learners tracks drifting concepts better than learners that simply modify concept descriptions, that store and learn from examples encountered previously, and that use an unweighted ensemble of experts. Although DWM has no advantage over a single base learner for static concepts, results from 26 UCI data sets show that, overall, DWM performs no worse than a single base learner.

We cite two main contributions of this work. First, we present a general algorithm for using any online learning algorithm for problems with concept drift. Second, we present results of an extensive empirical study characterizing DWM's performance along several dimensions: with different base learners, with well-studied data sets, with class noise, with static concepts, and with respect to learners appearing previously in the literature.

The organization of the paper is as follows. In the next section, we survey work on the problem of concept drift, on ensemble methods, and at the intersection: work on ensemble methods for concept drift. In Section 3, we describe dynamic weighted majority, an ensemble method for tracking concept drift. In Section 4, we present the results of our empirical study. Section 5 concludes the paper, and it is here that we discuss directions for future research.

2. Background and Related Work

Dynamic weighted majority is an ensemble method designed expressly for concept drift. For many years, research on ensemble methods and research on methods for concept drift have intermingled little. However, within the last few years, researchers have proposed several ensemble methods for tracking concept drift. In the next three sections, we survey relevant work on the problem of tracking concept drift, on ensemble methods, and on ensemble methods for tracking concept drift.

2.1 Concept Drift

Concept drift (Schlimmer and Granger, 1986; Maloof, 2005) is an online learning task in which concepts change or drift over time. For example, with a professor's e-mail classification system, the concept of an important e-mail will change with semesters and with conference deadlines. Concepts may change suddenly or gradually, and if we view concepts as shapes in a representation space, then

they can change their shape, size, and location. In concrete terms, concept drift occurs when the class labels of a set of examples change over time. Researchers have used both real (e.g., Harries and Horn, 1995; Blum, 1997; Lane and Brodley, 1998; Harries, 1999; Black and Hickey, 2002; Gramacy et al., 2003; Wang et al., 2003; Gama et al., 2004; Delany et al., 2005; Gama et al., 2005; Scholz and Klinkenberg, 2005; Tsymbal et al., 2005) and synthetic (e.g., Schlimmer and Granger, 1986; Widmer and Kubat, 1996; Maloof and Michalski, 2000; Hulten et al., 2001; Street and Kim, 2001; Kolter and Maloof, 2003; Klinkenberg, 2004; Maloof and Michalski, 2004; Gama et al., 2005; Kolter and Maloof, 2005; Scholz and Klinkenberg, 2005) data sets as inspiration for and the evaluation of a variety of methods for tracking concept drift. There has also been theoretical work on this problem (e.g., Helmbold and Long, 1991; Kuh et al., 1991; Helmbold and Long, 1994; Auer and Warmuth, 1998; Mesterharm, 2003; Monteleoni and Jaakkola, 2004; Kolter and Maloof, 2005), but a thorough survey of this work is beyond the scope of the paper.

STAGGER (Schlimmer and Granger, 1986) was the first system designed to cope with concept drift. It uses a distributed concept description consisting of class nodes linked to attribute-value nodes by probabilistic arcs. Two probabilities associated with each arc represent necessity and sufficiency, and are updated based on a psychological theory of association learning (Rescorla, 1968). In addition to adjusting these probabilities when new training examples arrive, STAGGER can also add nodes corresponding to new classes and new features. To better cope with concept drift, STAGGER may decay its probabilities over time. Empirical results on a synthetic data set, now known as the STAGGER concepts, show the method acquiring three changing target concepts in succession. The STAGGER concepts have been a centerpiece of numerous evaluations (Widmer and Kubat, 1996; Widmer, 1997; Maloof and Michalski, 2000; Kolter and Maloof, 2003; Maloof and Michalski, 2004; Kolter and Maloof, 2005), and we discuss them further in Section 4.1.

Concept versioning (Klenner and Hahn, 1994) is a method for coping with gradual or *evolutionary* concept drift. It uses a frame representation and copes with such drift by either adjusting current concept descriptions or creating a new version of these descriptions. The system creates a new version when an instance's attribute values and ranges present in current concept descriptions are dissimilar, based on a measure that accounts for both quantitative (e.g., value differences) and qualitative (e.g., increasing trend) information. Empirical results, based on 124 examples of computers sold between 1987 and 1993, suggest that the system acquired three versions of concepts based on machines' clock frequency and memory size.

The FLORA systems (Widmer and Kubat, 1996) track concept drift by maintaining a sequence of examples over a dynamically adjusted window of time and using them to induce and refine three sets of rules: rules covering the positive examples, rules covering the negative examples, and potential rules that are, at present, too general. Examples entering and leaving the window cause FLORA to refine rules, to move them from one set to another, or to delete them. FLORA2 uses just these basic mechanisms, whereas FLORA3 is an extension that handles recurring contexts. FLORA4 extends FLORA3 with mechanisms for coping with noise, similar to those present in IB3 (Aha et al., 1991). On the STAGGER concepts, the method generally performed well in terms of slope and asymptote after concepts changed. (We further analyze these results in Section 4.1.)

Although not expressly designed for concept drift, the MetaL(B) and MetaL(IB) systems (Widmer, 1997) use meta-learning with naive Bayes and instance-based learning, respectively, to cope with *reoccurring contexts*. For example, the concept of warm is different in the summer than in the winter. In this scenario, *season* or *average temperature* is a contextual attribute that identifies the

relevant concept of warm. As an agent moves from one season to the next, it uses the contextual variable to better focus the base algorithm on those features relevant for learning and prediction.

Meta-learning mechanisms identify contextual attributes by maintaining co-occurrence and frequency counts over the learner's entire history and over a fixed window of time. Using a χ^2 test, the meta-learning algorithm identifies contextual features and those predictive features relevant for the given context. The base learner then uses the predictive features of the contextually relevant examples in the current window to form new concept descriptions. By adding a contextual variable to the STAGGER concepts (Schlimmer and Granger, 1986), experimental results using predictive accuracy suggest that the meta-learners were able use contextual features to acquire target concepts better than and more quickly than did the base learners alone. When compared to FLORA4 (Widmer and Kubat, 1996), which retrieves previously learned concept descriptions when contexts reoccur, MetaL(B) often performed better in terms of slope and asymptote, even though it relearned new concept descriptions rather than retrieving and modifying old ones.

Lane and Brodley (1998) investigated methods for concept drift in one-class learning problems (i.e., anomaly detection). For an intrusion detection domain, they used instance-based learning in which instances were fixed-length sequences of UNIX commands. As a user enters new commands, the system calculates a measure of similarity between the current instance and past instances. They investigated two methods for coping with concept drift. One fits a line to the sequence of similarity measures in a window and uses its direction and magnitude to adjust the decision threshold. A second uses the slope of the line to determine if there is a stable or changing trend and inserts new examples into a user profile only if the trend is changing. Both methods proved useful in striking the delicate balance between learning a user's changing behavior and detecting anomalous behavior.

CD3 (Black and Hickey, 1999) uses ID3 (Quinlan, 1986) to learn online from batches of training data. The method maintains a collection of examples from the stream, all annotated with a time stamp of *current*. It annotates examples in a new training batch with a stamp of *new*. For static concepts, the time stamp is an irrelevant attribute, but if drift occurs, then the time stamp will appear in induced trees. Moreover, one can use its position in the tree as an indicator of how much drift may have occurred—the more relevant the time stamp, the higher it will appear in the tree, and the more drift that has occurred. After pruning, the method produces a set of valid and invalid rules by enumerating paths in the tree with new and current time stamps, respectively. Rule conditions containing a time stamp are then removed.

The performance element uses the valid rules for prediction, and the method uses the invalid rules to remove outdated examples from its store. However, depending on parameter settings, if both an invalid and a valid rule cover a stored example, then the method removes the example. Results from an empirical study involving synthetic data and varying amounts of class noise suggest that CD3 copes with revolutionary and evolutionary concept drift better than—in terms of slope and asymptote—ID3 learning from only the most recent batch and better than ID3 learning from all previously encountered batches.

CD4 (Hickey and Black, 2000) extends CD3 by letting time stamps for a batch be numeric, although the method no longer uses invalid rules to remove stored examples. CD5 (Hickey and Black, 2000) extends CD4 by assigning numeric time stamps to individual examples, rather than to batches. In an empirical study with synthetic data and evolutionary and revolutionary changes in target concepts, all three systems performed comparably.

In subsequent work, Black and Hickey (2002) applied CD3 to twenty-seven months of customer-call data and were able to detect concept drift based on the number and height of time stamps in the

induced decision trees. Indeed, during one period, the time stamp was the most relevant attribute and at the root of the decision tree.

Syed et al. (1999) present an online algorithm for training support vector machines (Boser et al., 1992). The method adds previously obtained support vectors to the new training set and then builds another machine. When compared to a batch algorithm, the online method performed comparable on several UCI data sets (Asuncion and Newman, 2007). The authors pointed to drops and recoveries in predictive accuracy during the incremental runs as evidence that SVMs can handle drift.

The presence of drops and recoveries are necessary for detecting concept drift, but they are not sufficient. The drops in predictive accuracy could have been due to sampling, and in general, it may be difficult to determine if a learner is coping with concept drift or is simply acquiring more information about a static concept. This type of concept drift has been called *virtual concept drift*, as opposed to *real concept drift* (Klinkenberg and Joachims, 2000). Indeed, the data sets included in the evaluation (e.g., monks, sonar, and mushroom) are not typically regarded as having concepts that drift.

Kelly et al. (1999) define *population drift* as being "related to" concept drift, noting that the term *concept drift* has no single meaning. They define population drift as change in the problem's probability distribution. Such changes can occur in the prior, conditional, and posterior distributions, and using artificial data, they analyze the impact on performance of changes to each of these distributions.

The term *concept drift* has been applied to different phenomena, such as drops and recoveries in performance during online learning (Syed et al., 1999). However, in concrete terms, the term has historically meant that the class labels of instances change over time, established in the first paper on concept drift (Schlimmer and Granger, 1986). Such change corresponds to change in the posterior distribution and in the conditional distribution. However, the prior distribution may not change, and changes in the conditional distribution will not necessarily mean that concept drift has occurred.

Klinkenberg and Joachims (2000) used a support vector machine to size windows for concept drift. Rather than tuning parameters for an adaptive windowing heuristic (c.f. Widmer and Kubat, 1996), the algorithm sizes the window by minimizing generalization error on new examples. Upon receiving a new batch of examples, the method generates support vector machines with various sizes of windows using previously encountered batches of training examples, and selects the window size that minimizes the $\xi\alpha$-estimate, an approximation to the leave-one-out or jackknife estimator (Hinkley, 1983). Experimental results on a manually constructed data set derived from $2,608$ news documents suggest that exhaustively searching for a window's size was better, in terms of slope and asymptote of predictive accuracy, than a window of fixed size. Results from subsequent experiments indicate that selecting examples in batches or using an adaptive window yielded higher precision and recall than did weighting examples based on their age or their fit to the current model (Klinkenberg, 2004).

The AQ-PM systems (Maloof and Michalski, 2000, 2004; Maloof, 2003), like the FLORA systems (Widmer and Kubat, 1996), induce rules and use partial instance memory (PM) to maintain a subset of the examples from the input stream. However, rather than selecting a sequence, the AQ-PM systems select those examples that lie on the boundaries of concept descriptions, thus they store examples that are important but that do not necessarily reoccur in the stream. AQ-PM (Maloof and Michalski, 2000) uses the AQ algorithm (Michalski, 1969), a batch learning algorithm, to form rules by simply reapplying the algorithm when new examples arrive. It stores examples for a fixed period of time. AQ11-PM and GEM-PM are incremental versions that use the AQ11 (Michalski

and Larson, 1983) and GEM (Reinke and Michalski, 1988) algorithms, respectively, to form rules. Finally, AQ11-PM+WAH is an extension of AQ11-PM that uses Widmer and Kubat's (1996) window adjustment heuristic (WAH) to dynamically size the window of time over which examples are kept. The STAGGER concepts were the centerpiece in the evaluation of all of these systems.

The Concept-adapting Very Fast Decision Tree (CVFDT) learner (Hulten et al., 2001) extends VFDT (Domingos and Hulten, 2000) by adding mechanisms for handling concept drift. VFDT grows a decision tree only from its leaf nodes, so there is no restructuring of the tree (cf. sc iti, Utgoff et al., 1997). The method maintains a decision tree and maintains counts of attribute values for classes in each leaf node. New examples propagate through the tree to a leaf node with the algorithm updating the counts. If the propagated examples and the examples used to form a leaf node are not of the same class, then the method uses a splitting criterion and the Hoeffding (1963) bound to determine if the node should be split.

CVFDT extends VFDT by maintaining at each node in the tree a list of alternate subtrees and attribute-value counts for each class. Like VFDT, new examples propagate through the tree to a leaf node, but they also propagate through all of the alternate subtrees along this path. Periodically, the method forgets examples, and if an alternate subtree is more accurate than the current one, it swaps them. Empirical results on a synthetic data set consisting of a rotating hyperplane, five million training examples, and drift every 50,000 time steps, suggest that CVFDT produced smaller, more accurate trees than did VFDT.

2.2 Ensemble Methods

Ensemble methods maintain a collection of learners and combine their decisions to make an overall decision. Generally, an algorithm applied multiple times to the same data set will produce identical classifiers that make the same decisions, so for ensemble methods to work, there must be some mechanism to produce different classifiers. This is accomplished by either altering the training data or the learners in the collection.

Bagging (Breiman, 1996) is one of the simplest ensemble methods. It entails producing a set of bootstrapped data sets from the original training data. This involves sampling with replacement from the training data and producing multiple data sets of the same size. Once formed, a learning method produces a classifier from each bootstrapped data set. During performance, the method predicts based on a majority vote of the predictions of the individual classifiers.

Weighted majority (Littlestone and Warmuth, 1994), instead of using altered training sets, relies on a collection of different classifiers, often referred to as "experts." Each expert begins with a weight of one, which is decreased (e.g., halved) whenever an expert predicts incorrectly. To make an overall prediction, the method takes a weighted vote of the expert predictions, and predicts the class with the most weight. Winnow is a similar algorithm, but also increases the weights of experts that predict correctly (Littlestone, 1991).

Boosting (Freund and Schapire, 1996) is a method that generates a series of weighted classifiers. It iteratively weights training examples based on how well a classifier predicts them, and in turn, weighting the classifier based on the weights of the examples used to construct it. During performance, each classifier in the ensemble returns a prediction for an instance, and then the global method predicts based on a weighted-majority vote. This method constructs classifiers specialized at predicting examples in specific parts of the representation space. Arcing (Breiman, 1998) is similar to boosting, but uses a different weighting scheme and predicts with a majority vote.

Stacked generalization or *stacking* (Wolpert, 1992), like weighted majority, is an ensemble method for combining the decisions of different types of learners. However, stacking uses the predictions of the base learners to form training examples for a meta-learner. Stacking begins by partitioning the original examples into three sets: training, validation, and testing. The method trains the base learners using the training data and then applies the resulting classifiers to the examples in the validation set. Using the predictions of the examples as features and their original class labels, it forms a new training set, which it uses to train the meta-learner. After training the meta-learner with these examples, performance entails presenting an instance to the base classifiers, using their predictions as input to the meta-classifier. The method's overall prediction is that of the meta-classifier.

There have been numerous studies of ensemble methods (e.g., Hansen and Salamon, 1990; Woods et al., 1997; Quinlan, 1996; Zheng, 1998; Opitz and Maclin, 1999; Bauer and Kohavi, 1999; Maclin and Opitz, 1997; Dieterich, 2000; Zhou et al., 2002; Chawla et al., 2004), and we cannot survey them all. Generally, this research suggests that ensemble methods outperform single classifiers on many standard data sets (Opitz and Maclin, 1999). Boosting is generally better than bagging (Bauer and Kohavi, 1999; Opitz and Maclin, 1999; Dieterich, 2000), although bagging seems more robust to noise than is boosting (Dieterich, 2000). Analysis suggests that ensemble methods work by reducing bias, variance, or both (Breiman, 1998; Bauer and Kohavi, 1999; Zhou et al., 2002).

Popular ensemble methods, such as bagging and boosting, are off-line algorithms, but researchers have developed online ensemble methods. Winnow (Littlestone, 1988) and weighted majority (Littlestone and Warmuth, 1994) fall into this category, as do Blum's (1997) versions of these algorithms.

Online AdaBoost (Fan et al., 1999), when a new batch of examples arrives, weights the examples and then re-weights the ensemble's classifiers based on their performance on the new examples. The method then builds a new classifier from the new, weighted examples. For efficiency, the method retains only the k most recent classifiers.

Online arcing (Fern and Givan, 2003) uses incremental base learners and incrementally updates their voting weights. When a new training instance arrives, for each classifier in the ensemble, the method first increases the classifier's voting weight by one if the classifier correctly classifies the new instance. The method then computes an instance weight for the classifier based on the number of other classifiers in the ensemble that misclassify the instance. Finally, an incremental learning function uses the instance and the weight to refine the classifier. The method may use a weighted or unweighted voting scheme to classify an instance.

2.3 Ensemble Methods for Concept Drift

Ensemble methods for concept drift share many similarities with the online and off-line methods discussed in the previous section. However, methods for concept drift must take into account the temporal nature of the data stream, for a set of examples may have certain class labels at time t and others at time t'.

Clearly, ensemble methods for concept drift must process a stream of data. Some researchers have used repeated applications of off-line learning algorithms to process batches of training examples (Maloof and Michalski, 2000; Street and Kim, 2001; Wang et al., 2003; Scholz and Klinkenberg, 2005). Others have used incremental algorithms (Blum, 1997; Kolter and Maloof, 2003; Maloof and Michalski, 2004; Kolter and Maloof, 2005).

Any method for coping with changing concepts must have mechanisms for refining or removing knowledge of past target concepts. To achieve these effects, one ensemble method for concept drift builds two ensembles and selects the best performing one for subsequent processing (Scholz and Klinkenberg, 2006). Other methods replace poorly performing members of the ensemble (Street and Kim, 2001; Fan, 2004), decrease the effect these members have on the overall prediction (Blum, 1997; Scholz and Klinkenberg, 2006), or both (Kolter and Maloof, 2003; Wang et al., 2003; Kolter and Maloof, 2005).

Blum's (1997) implementation of the weighted majority algorithm (Littlestone and Warmuth, 1994) uses as experts pairs of features coupled with a history of the most recent class labels from the training set appearing with those features. When a new instance arrives, the expert for a given pair of features predicts based on a majority vote of the labels of past observations. The global algorithm predicts based on a weighted-majority vote of the expert predictions and decreases the weight of any expert that predicts incorrectly. Each expert then stores the correct prediction in its history.

Blum (1997) also investigated triples of features as experts and a variant of winnow (Littlestone, 1988) that lets experts abstain if their features are not present in an instance. On a calendar scheduling task, which we describe further in Section 4.3, these methods were able to track a professor's preferences for scheduling meetings across semester boundaries.

The Streaming Ensemble Algorithm (SEA) maintains a fixed-size collection of classifiers, each built from a batch of training examples (Street and Kim, 2001). When a new batch of examples arrives, SEA uses C4.5 (Quinlan, 1993) to build a decision tree. If there is space, SEA adds the new classifier to the ensemble. Otherwise, if the new classifier outperforms a classifier in the ensemble, SEA replaces it with the new one. Performance is measured on the current batch of examples. Overall, SEA predicts based on a majority vote of the predictions of the classifiers in the ensemble. An evaluation using a synthetic data set generated from shifting a hyperplane revealed that the method was able to acquire a series of four changing target concepts with accuracy of about 92%. We discuss this data set further in Section 4.2.

Herbster and Warmuth (1998) consider a setting in which an online learner is trained over several concepts and has access to n experts (which are fixed prediction strategies). They present an algorithm that performs almost as well as the best expert on each concept individually, paying an additional penalty of $\log n$ on each concept. Bousquet and Warmuth (2002) extend this setting to one where the best expert always comes from a smaller pool of $m \ll n$ experts. Here they show that the learner can pay a $\log m$ rather than a $\log n$ penalty on each concept, plus a one-time cost of $\log \binom{n}{m}$ to identify the best m experts.

The Accuracy-weighted Ensemble (AWE) also maintains a fixed-size collection of classifiers built from batches of training examples, but this method weights each classifier based on its performance on the most recent batch (Wang et al., 2003). If there is space in the ensemble, then AWE adds the new weighted classifier. Otherwise, it keeps only the top k weighted classifiers. AWE predicts based on a weighted-majority vote of the predictions of the ensemble's classifiers. The evaluation of this method involved a synthetic data set generated from a rotating hyperplane and a data set for credit card fraud detection. Although the evaluation consisted of many experimental conditions, such as how the problem's dimension affects error rate of the ensemble classifier, it did not include measuring the system's performance over time and over changing target concepts.

Kolter and Maloof (2005) present AddExp, an algorithm for additive expert ensembles. It is similar to dynamic weighted majority, but unlike DWM, AddExp submits to formal analysis due

to differences in their weighting schemes. DWM sets the weight of a new expert to one, whereas AddExp sets a new expert's weight to the total weight of the ensemble times some parameter $\gamma \in (0,1)$. In addition to empirical results, the authors present worst-case bounds for the algorithm's loss and number of mistakes, proving that AddExp performs almost as well as the best-performing expert. They also describe two pruning methods for limiting the number of experts maintained; one is useful in practice, the other gives formal guarantees on the number of experts that AddExp will create.

3. DWM: An Ensemble Method for Concept Drift

Dynamic weighted majority maintains a weighted pool of experts or base learners. It adds and removes experts based on the global algorithm's performance. If the global algorithm makes a mistake, then DWM adds an expert. If an expert makes a mistake, then DWM reduces its weight. If, in spite of multiple training episodes, an expert performs poorly, as indicated by a sufficiently low weight, then DWM removes it from the ensemble. This method is general, and in principle, one could use any online learning algorithm as a base learner. One could also use different types of base learners, although one would also have to implement control policies to determine what base learner to add.

The formal algorithm for DWM appears in Figure 1. The algorithm maintains a set of m experts, E, each with a weight, w_i for $i = 1, \ldots, m$. Input to the algorithm is n training examples, each consisting of a feature vector and a class label. The parameters also include the number of classes (c) and β, a multiplicative factor that DWM uses to decrease an expert's weight when it predicts incorrectly. A typical value for β is 0.5. The parameter θ is a threshold for removing poorly performing experts. If an expert's weight falls below this threshold, then DWM removes it from the ensemble. Finally, the parameter p determines how often DWM creates and removes experts. We found this parameter useful and necessary for large or noisy problems, which we discuss further in Section 4.2. In the following discussion, we assume $p = 1$.

DWM begins by creating an ensemble containing a single learner with a weight of one (lines 1–3 of Figure 1). Initially, this learner could predict a default class, or it could predict using previous experience, background knowledge, or both. DWM then takes a single example (or perhaps a set of examples) from the stream and presents it to the single learner to classify (line 7). If the learner's prediction is wrong (line 8), then DWM decreases the learner's weight by multiplying it by β (line 9). Since there is one expert in the ensemble, its prediction is DWM's global prediction (lines 12 and 24). If DWM's global prediction is incorrect (line 16), then it creates a new learner with a weight of one (lines 17–19). DWM then trains the experts in the ensemble on the new example (line 23). After training, DWM outputs its global prediction (line 24).

When there are multiple learners, DWM obtains a classification from each member of the ensemble (lines 6 and 7). If one's prediction is incorrect, then DWM decreases its weight (lines 8 and 9). Regardless of the correctness of the prediction, DWM uses each learner's prediction and its weight to compute a weighted sum for each class (line 10). The class with the most weight is set as the global prediction (line 12).

Since DWM always decreases the weights of experts, it normalizes the weights by scaling them uniformly so that, after the transformation, the maximum weight is one (line 14). This prevents newly added experts from dominating predictions. DWM also removes poorly performing experts by removing those with a weight less than the threshold θ (line 15), although it will not remove the

Dynamic-Weighted-Majority ($\{\vec{x},y\}_n^1, c, \beta, \theta, p$)

$\{\vec{x},y\}_n^1$: training data, feature vector and class label
$c \in \mathbb{N}^*$: number of classes, $c \geq 2$
β: factor for decreasing weights, $0 \leq \beta < 1$
θ: threshold for deleting experts
p: period between expert removal, creation, and weight update
$\{e,w\}_m^1$: set of experts and their weights
$\Lambda, \lambda \in \{1,\dots,c\}$: global and local predictions
$\vec{\sigma} \in \mathbb{R}^c$: sum of weighted predictions for each class

1.	$m \leftarrow 1$	
2.	$e_m \leftarrow$ Create-New-Expert$()$	
3.	$w_m \leftarrow 1$	
4.	**for** $i \leftarrow 1,\dots,n$	// Loop over examples
5.	$\quad \vec{\sigma} \leftarrow 0$	
6.	$\quad$ **for** $j \leftarrow 1,\dots,m$	// Loop over experts
7.	$\quad\quad \lambda \leftarrow$ Classify$(e_j, \vec{x}_i)$	
8.	$\quad\quad$ **if** $(\lambda \neq y_i$ **and** $i \bmod p = 0)$	
9.	$\quad\quad\quad w_j \leftarrow \beta w_j$	
10.	$\quad\quad \sigma_\lambda \leftarrow \sigma_\lambda + w_j$	
11.	$\quad$ **end**;	
12.	$\quad \Lambda \leftarrow \text{argmax}_j \sigma_j$	
13.	$\quad$ **if** $(i \bmod p = 0)$	
14.	$\quad\quad w \leftarrow$ Normalize-Weights(w)	
15.	$\quad\quad \{e,w\} \leftarrow$ Remove-Experts$(\{e,w\}, \theta)$	
16.	$\quad\quad$ **if** $(\Lambda \neq y_i)$	
17.	$\quad\quad\quad m \leftarrow m+1$	
18.	$\quad\quad\quad e_m \leftarrow$ Create-New-Expert$()$	
19.	$\quad\quad\quad w_m \leftarrow 1$	
20.	$\quad\quad$ **end**;	
21.	$\quad$ **end**;	
22.	$\quad$ **for** $j \leftarrow 1,\dots,m$	
23.	$\quad\quad e_j \leftarrow$ Train$(e_j, \vec{x}_i, y_i)$	
24.	$\quad$ **output** Λ	
	end;	
	end.	

Figure 1: Algorithm for dynamic weighted majority (DWM), after Kolter and Maloof (2005).

last expert in the ensemble. As mentioned previously, if the global prediction is incorrect (line 16), DWM adds a new expert to the ensemble with a weight of one (lines 17–19). Finally, after using the new example to train each learner in the ensemble (lines 22 and 23), DWM outputs the global prediction, which is the weighted vote of the expert predictions (line 24).

As mentioned previously, the parameter p lets DWM better cope with many or noisy examples. p defines the period over which DWM will not update learners' weights (line 8) and will not remove or create experts (line 13). During this period, however, DWM still trains the learners (lines 22 and 23).

DWM is a general algorithm for coping with concept drift. One can use any online learning algorithm as the base learner. To date, we have evaluated two such algorithms, naive Bayes and Incremental Tree Inducer, and we describe these versions in the next two sections.

3.1 DWM-NB

DWM-NB uses an incremental version of naive Bayes as the base learner. For this study, we used the implementation from WEKA (Witten and Frank, 2005), which stores for nominal attributes a count for each class and for each attribute value given the class. The count is simply the number of times each class or attribute value appears in the training set, and so the learning element increments the appropriate counts when processing a new example. The performance element uses these counts to compute estimates of the prior probability of each class, $P(C_i)$, and the conditional probability of each attribute value given the class, $P(v_j|C_j)$. It then operates under the assumption that attributes are conditionally independent and uses Bayes' rule to predict the most probable class:

$$C = \operatorname*{argmax}_{C_i} P(C_i) \prod_j P(v_j|C_i) \ .$$

For numeric attributes, it stores the sum of an attribute's values and the sum of the squared values. Given a value, v_j,

$$P(v_j|C_i) = \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-(v_j-\mu_{ij})^2/2\sigma_{ij}^2} \ ,$$

where μ_{ij} is the average of the jth attribute's values for the ith class, and σ_{ij} is their standard deviation. The performance element computes these values from the stored sums.

3.2 DWM-ITI

DWM-ITI uses as a base learner an incremental algorithm for inducing decision trees, called Incremental Tree Inducer, or ITI (Utgoff et al., 1997). ITI uses as its concept description a decision tree with only binary tests. (Note that we can represent multi-valued attributes with a binary tree by treating each value as a binary attribute.) A *decision tree* is a rooted tree with internal nodes corresponding to attributes. Edges correspond to attribute values. For numeric attributes, there are two edges corresponding to a *cut-point*: one edge for values above the cut-point, the other for values less than or equal to the cut-point. The external nodes of the decision tree correspond to class labels. To facilitate incremental learning, external nodes also store examples, while the internal nodes of ITI's decision trees store frequency counts for symbolic attributes and a list of observed values for numeric attributes.

Like standard decision tree algorithms, given an observation, the performance element uses the observation's attributes and their values to traverse from the root node to an external node. It predicts the class label stored in the node.

ITI updates a tree by propagating a new example to a leaf node. During the descent, the algorithm updates the information stored at each node—counts or values—and upon reaching a leaf node, determines if the tree should be extended by converting the leaf node to a decision node. A secondary process examines whether the tests at each node are most appropriate, and if not, restructures the tree accordingly.

As one can imagine, for large problems, storing all examples at leaf nodes and restructuring decision trees can be costly, especially when maintaining an ensemble of such trees. While we were judicious when applying DWM-ITI to large problems, as we see in the next section, where we discuss our experimental study, the learner performed well in spite of these potential costs.

4. Empirical Study and Results

In this section, we present experimental results for DWM-NB and DWM-ITI. We conducted five evaluations. The first, most extensive evaluation involved the STAGGER concepts (Schlimmer and Granger, 1986), a standard benchmark for evaluating how learners cope with drifting concepts. In this evaluation, we compared the performance of DWM-NB and DWM-ITI

1. to best- and worst-case base learners,

2. to our implementation of STAGGER (Schlimmer, 1987),

3. to AQ-PM (Maloof and Michalski, 2000), AQ11-PM (Maloof and Michalski, 2004), and AQ11-PM+WAH (Maloof, 2003), and

4. to Blum's (1997) implementation of weighted majority.

To the best of our knowledge, ours are the only results for Blum's algorithm on the STAGGER concepts.

In an effort to determine how our method scales to larger problems involving concept drift, our second evaluation consisted of testing DWM-NB using the SEA concepts (Street and Kim, 2001), a problem recently proposed in the data mining community. For the third evaluation, we applied DWM-NB to the CAP data set, a calendar scheduling task (Mitchell et al., 1994). Blum (1997) used this problem to evaluate weighted majority and winnow. For the fourth, we evaluated DWM-NB on the task of predicting the price of electricity in New South Wales, Australia, between May 1997 and December 1999, a problem originally introduced by Harries (1999).

Finally, although it is clear that DWM should have no advantage over a single learner when acquiring static concepts, for the sake of completeness, we evaluated DWM-NB on twenty-six data sets from the UCI Repository (Asuncion and Newman, 2007). The intent of this evaluation was to show that, on static concepts, DWM performs no worse than a single base learner.

4.1 The STAGGER Concepts

The STAGGER concepts (Schlimmer and Granger, 1986) comprise a standard benchmark for evaluating a learner's performance in the presence of concept drift. Each example consists of three attribute values: *color* ∈ {*green, blue, red* }, *shape* ∈ {*triangle, circle, rectangle*}, and *size* ∈ {*small,*
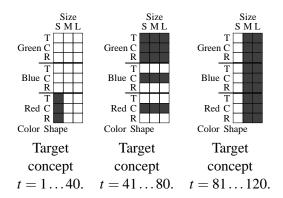
Figure 2: Visualization of the STAGGER Concepts (Maloof and Michalski, 2000). © 2000 Kluwer Academic Publishers. Used with permission.

medium, large}. The presentation of training examples lasts for 120 time steps, and at each time step, the learner receives one example. For the first 40 time steps, the target concept is *color = red ∧ size = small*. During the next 40 time steps, the target concept is *color = green ∨ shape = circle*. Finally, during the last 40 time steps, the target concept is *size = medium ∨ size = large*. A visualization of these concepts appears in Figure 2.

To evaluate the learner, at each time step, one randomly generates 100 examples of the current target concept, presents these to the performance element, and computes the percent correctly predicted. In our experiments, we repeated this procedure 50 times and averaged the accuracies over these runs. We also computed 95% confidence intervals.

We set the weighted-majority learners—DWM-NB, DWM-ITI, and Blum's (1997) with pairs of features as experts—to halve an expert's weight when it made a mistake (i.e., $\beta = 0.5$). For Blum's weighted majority, each expert maintained a history of only its last prediction (i.e., $k = 1$), under the assumption that this setting would provide the most reactivity to concept drift. For DWM, we set it to update its weights and create and remove experts every time step (i.e., $p = 1$). The algorithm removed experts when their weights fell below 0.01 (i.e., $\theta = 0.01$). Pilot studies indicated that these were the near-optimal settings for p and k; varying β affected performance little; the selected value for θ did not affect accuracy, but did reduce considerably the number of experts.

For the sake of comparison, in addition to these algorithms, we also evaluated naive Bayes, ITI, naive Bayes with perfect forgetting, and ITI with perfect forgetting. The "standard" or "traditional" implementations of naive Bayes and ITI provided a worst-case evaluation, since these systems have not been designed to cope with concept drift and learn from all examples in the stream regardless of changes to the target concept. The implementations with perfect forgetting, which is the same as training the methods on each target concept individually, provided a best-case evaluation, since the systems were never burdened with examples or concept descriptions from previous target concepts.

The left graph of Figure 3 shows the results for DWM-NB on the STAGGER concepts. As expected, naive Bayes with perfect forgetting performed the best on all three concepts, while naive Bayes without forgetting performed the worst. DWM-NB performed almost as well as naive Bayes with perfect forgetting, which converged more quickly to the target concept. Nonetheless, by time
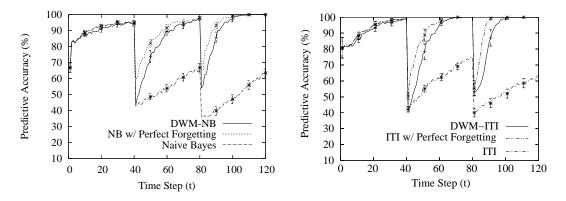
Figure 3: Predictive accuracy with 95% confidence intervals for DWM on the STAGGER concepts (Kolter and Maloof, 2003). Left: DWM-NB. Right: DWM-ITI. © 2003 IEEE Press. Used with permission.
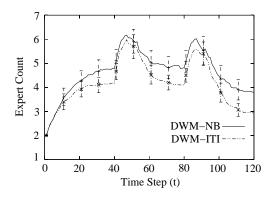

Figure 4: Number of experts maintained with 95% confidence intervals for DWM-NB and DWM-ITI on the STAGGER concepts (Kolter and Maloof, 2003). © 2003 IEEE Press. Used with permission.

step 40 for all three target concepts, DWM-NB performed almost as well as naive Bayes with perfect forgetting.

DWM-ITI performed similarly, as shown in the right graph of Figure 3, achieving accuracies nearly as high as ITI with perfect forgetting. DWM-ITI converged more quickly than did DWM-NB to the second and third target concepts, but if we compare the plots for naive Bayes and ITI with perfect forgetting, we see that ITI converged more quickly to these target concepts than did naive Bayes. Thus, the faster convergence is due to differences in the base learners rather than to something inherent to DWM.

In Figure 4, we present the average number of experts each system maintained over the fifty runs. On average, DWM-ITI maintained fewer experts than did DWM-NB, and we attribute this to the fact that ITI performed better on the individual concepts than did naive Bayes. Since naive Bayes
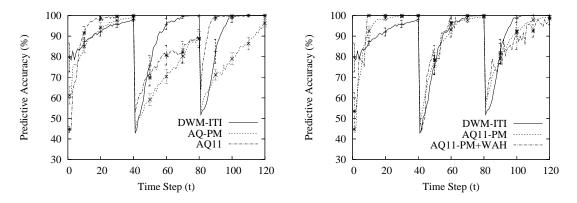
Figure 5: Predictive accuracy with 95% confidence intervals on the STAGGER concepts. Left: DWM-ITI, AQ-PM, and AQ11. Right: DWM-ITI, AQ11-PM, and AQ11-PM+WAH.
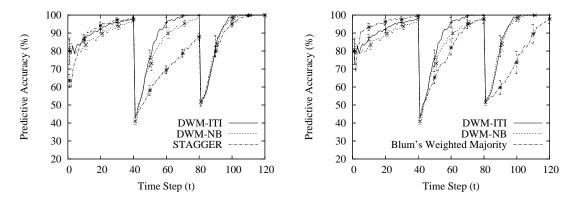

Figure 6: Predictive accuracy with 95% confidence intervals on the STAGGER concepts. Left: DWM-ITI, DWM-NB, and STAGGER. Right: DWM-ITI, DWM-NB, and Blum's weighted majority (Kolter and Maloof, 2003). © 2003 IEEE Press. Used with permission.

made more mistakes than did ITI, DWM-NB created more experts than did DWM-ITI. We can also see in the figure that the rates of removing experts were roughly the same for both learners.

The left graph of Figure 5 compares the performance of DWM-ITI to that of AQ-PM (Maloof and Michalski, 2000) and AQ11 (Michalski and Larson, 1983). DWM-ITI and AQ-PM performed similarly on the first target concept, but DWM-ITI significantly outperformed AQ-PM on the second and third concepts, again, in terms of asymptote and slope. AQ11, although not designed to cope with concept drift, outperformed DWM-ITI in terms of asymptote on the first concept and in terms of slope on the third, but on the second concept, performed significantly worse than did DWM-ITI.

The right graph of Figure 5 compares the performance of DWM-ITI to that of AQ11-PM (Maloof and Michalski, 2004) and AQ11-PM+WAH (Maloof, 2003). DWM-ITI did not perform as well as these other learners on the first target concept, performed comparably on the second, and converged more quickly on the third.

The left graph of Figure 6 compares the performance of our learners to that of our implementation of STAGGER (Schlimmer, 1987). Comparing to both DWM learners, STAGGER's performance was only slightly better on the first target concept, notably worse on the second, and comparable on the third.

Finally, the right diagram of Figure 6 shows the results from the experiment involving Blum's (1997) implementation of weighted majority, the implementation that uses pairs of features as experts. This learner outperformed DWM-NB and DWM-ITI on the first target concept, performed slightly worse on the second, and performed considerably worse on the third.

With respect to complexity of the experts themselves, the STAGGER concepts consist of three attributes, each taking one of three possible values. Therefore, this implementation of weighted majority maintained 27 experts throughout the presentation of examples, as compared to the maximum of six that DWM-NB maintained. Granted, pairs of features and their recent predictions are much simpler than the decision trees that ITI produced, but naive Bayes was quite efficient, maintaining twenty-one integers for each expert. There were occasions when Blum's weighted majority used less memory than did DWM-NB, but we anticipate that using more sophisticated classifiers, such as naive Bayes, instead of all combinations of pairs of features, will lead to scalable algorithms.

We focus our analysis on DWM-ITI, since it performed better than did DWM-NB on this problem. Researchers have built several systems for coping with concept drift and have evaluated many of them on the STAGGER concepts. FLORA2 (Widmer and Kubat, 1996) is a prime example, and on the first target concept, DWM-ITI did not perform as well as did FLORA2. However, on the second and third target concepts, DWM-ITI performed notably better than did FLORA2, not only in terms of asymptote, but also in terms of slope.

DWM-ITI outperformed AQ-PM (Maloof and Michalski, 2000), which had difficulty acquiring the second and third concepts (see Figure 5). AQ-PM maintains examples over a fixed window of time and, at each time step, relearns concepts from these and new examples. In this experiment, when the concept changed from the first to the second, the examples of the first concept remaining in this window prevented AQ-PM from adjusting quickly to the second. Decreasing the size of this window improved accuracy on the second concept, but negatively affected performance on the third.

Compared to DWM-ITI, AQ11 performed poorly on the second concept, but performed exceptionally well on the third concept. Our analysis suggests that its poor performance on the second concept was because of AQ11's expressive language for representing concepts. Notice that all of its rules achieved 100% on the first concept (see Figure 5, left). However, AQ11 produced twelve different rules for the first concept over the fifty runs. Some rules were similar, but others were quite different. AQ11 modifies its rules using only new examples, and in some cases, was able to transform rules for the first concept into rules with high accuracy on the second. Indeed, when we trained AQ11 only on examples of the second concept, it achieved 97%. However, in some trials, AQ11 was unable to make the required transformation and failed to learn adequately the second concept. We contend that if AQ11 produced simpler rules, provided that they were the right ones, it would have performed much better on the second concept. Indeed, it is well known that simpler models often perform better than do more complex ones.

AQ11-PM (Maloof and Michalski, 2004) and AQ11-PM+WAH (Maloof, 2003) performed comparably to DWM-ITI (see Figure 5), although DWM-ITI did appear to outperform the AQ learners on the third concept, especially in the later time steps. Although tangential, it is interesting to contrast the performances of AQ11 and AQ11-PM. The only difference between these two learners is that AQ11-PM maintains a fixed window of thirty examples that have appeared on the boundaries of

concept descriptions. When new examples arrive, AQ11-PM learns incrementally from these new examples and those present in the window. Although the presence of these examples may have decreased AQ11's (i.e., AQ11-PM's) performance on the third concept, their presence notably improved its performance on the second concept. If our hypothesis is correct—that AQ11 would have performed better had it produced simpler models—the examples held in the window may have constrained AQ11 such that it produced simpler models. Put another way, the examples in the window reduced the instability of the learner. We found this discovery intriguing and plan to investigate it in future work.

STAGGER (Schlimmer and Granger, 1986) performed comparably to DWM-ITI on the first and third target concepts, but did not perform as well as DWM-ITI on the second target concept. (See Figure 6, left.) Generally, acquiring the second concept after learning the first is the hardest task, as the second concept is almost a reversal of the first; the two concepts share only one positive example. Acquiring the third concept after acquiring the second is easier because the two concepts share a greater number of positive examples. Performing well on the second concept therefore requires quickly disposing of knowledge and perhaps examples of the first target concept. A learning method, such as STAGGER, that only refines concept descriptions will have more difficulty responding to concept drift, as compared to an ensemble method, such as DWM, that both refines existing concept descriptions and creates new ones.

Comparing DWM-ITI to Blum's weighted majority, DWM-ITI outperformed it on the STAGGER concepts. However, our analysis suggests that the difference in performance is due to the experts, rather than to the global algorithms. Recall that Blum's weighted majority uses as experts pairs of features with a brief history of past predictions. For the STAGGER concepts, pairs of features are useful for acquiring the first target concept (see Figure 2), which is conjunctive. Indeed, pairs of features are two-term conjunctions. However, the second and third concepts are disjunctive, and these are difficult to represent using only weighted pairs of features. As a result, on the STAGGER concepts, Blum's weighted majority did not perform as well as DWM.

Overall, we concluded that DWM-ITI outperformed these other learners in terms of accuracy, both in slope and asymptote. In reaching this conclusion, we gave little weight to performance on the first concept, since most learners can acquire it easily and doing so requires no mechanisms for coping with drift. On the second and third concepts, with the exception of AQ11, DWM-ITI performed as well or better than did the other learners. And while AQ11 outperformed DWM-ITI in terms of slope on the third concept, this does not mitigate AQ11's poor performance on the second.

We attribute the performance of DWM-ITI to the training of multiple experts on different sequences of examples. (Weighting experts also contributed, and we will discuss this topic in detail shortly.) Assume a learner incrementally modifies its concept descriptions as new examples arrive. When the target concept changes, if the new one is disjoint, then the best policy to learn new descriptions, rather than modifying existing ones. This makes intuitive sense, since the learner does not have to first unlearn the old concept, and results from this and other empirical studies support this assertion (Maloof and Michalski, 2000, 2004). Unfortunately, target concepts are not always disjoint, it is difficult to determine precisely when concepts change, and it is challenging to identify which concept descriptions (or parts of concept descriptions) apply to new target concepts. DWM addresses these problems by incrementally updating existing descriptions and, in parallel, by learning new concept descriptions.
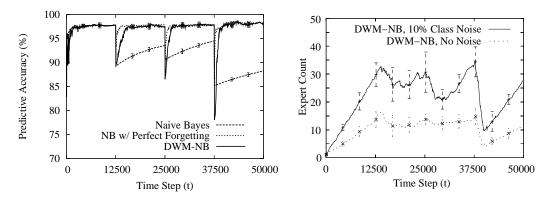
Figure 7: Performance with 95% confidence intervals of DWM-NB on the SEA concepts with 10% class noise (Kolter and Maloof, 2003). Left: Predictive accuracy. Right: Number of experts maintained. © 2003 IEEE Press. Used with permission.

4.2 Performance on a Larger Data Set with Concept Drift

To determine how well DWM-NB performs on larger problems involving concept drift, we evaluated it using a synthetic problem recently proposed in the data mining community (Street and Kim, 2001). This problem, which we call the "SEA concepts", consists of three attributes, $x_i \in \mathbb{R}$ such that $0.0 \leq x_i \leq 10.0$. The target concept is $x_1 + x_2 \leq b$, where $b \in \{7, 8, 9, 9.5\}$. Thus, x_3 is an irrelevant attribute.

The presentation of training examples lasts for $50,000$ time steps. For the first fourth (i.e., $12,500$ time steps), the target concept is with $b = 8$. For the second, $b = 9$; the third, $b = 7$; and the fourth, $b = 9.5$. For each of these four periods, we randomly generated a training set consisting of $12,500$ examples. In one experimental condition, we added 10% class noise; in another, we did not, and this latter condition served as our control. We also randomly generated 2,500 examples for testing. At each time step, we presented each method with one example, tested the resulting concept descriptions using the examples in the test set, and computed the percent correct. We repeated this procedure ten times, averaging accuracy over these runs. We also computed 95% confidence intervals.

On this problem, we evaluated DWM-NB, naive Bayes, and naive Bayes with perfect forgetting. We set DWM-NB to halve the expert weights (i.e., $\beta = 0.5$) and to update these weights and to create and remove experts every fifty time steps (i.e., $p = 50$). We set the algorithm to remove experts with weights less than 0.01 (i.e., $\theta = 0.01$).

In the left graph of Figure 7, we see the predictive accuracies for DWM-NB, naive Bayes, and naive Bayes with perfect forgetting on the SEA concepts with 10% class noise. As with the STAGGER concepts, naive Bayes performed the worst, since it had no direct method of removing outdated concept descriptions. Naive Bayes with perfect forgetting performed the best and represents the best possible performance for this implementation on this problem. Crucially, DWM-NB achieved accuracies nearly equal to those achieved by naive Bayes with perfect forgetting.

Finally, the right graph of Figure 7 shows the number of experts that DWM-NB maintained during the runs with and without class noise. Recall that DWM creates an expert when it misclassifies an
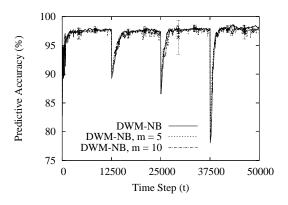
Figure 8: Predictive accuracy with 95% confidence intervals for DWM-NB on the SEA concepts with 10% class noise. The number of experts was capped at five and at ten and compared to DWM-NB with no limit on the number of experts created.

example. In the noisy condition, since 10% of the examples had been relabeled, DWM-NB made more mistakes and therefore created more experts than it did in the condition without noise.

As mentioned previously, DWM has the potential for creating a large number of experts, especially in noisy domains, since it creates a new expert every time the global prediction is incorrect. (The parameter p can help mitigate this effect.) A large number of experts obviously impacts memory utilization, and, depending on the complexity of the base learners, could also affect learning and performance time, since DWM trains and queries each expert in the ensemble when a new example arrives. An obvious scheme when resources are constrained is to limit the number of experts that DWM maintains.

To investigate this strategy's effect on DWM's performance, we produced a version of the algorithm that always keeps the k best performing experts. That is, after reaching the point where there are k experts in the ensemble, when DWM adds a new expert, it removes the expert with the lowest weight. In this version of DWM, we set the threshold for removing experts to zero, which guaranteed that experts were removed only if they were the weakest member of the ensemble of k experts.

We ran this modified version on the SEA concepts, capping the number of experts at five and at ten. We present these results in Figure 8. As one can see, for this problem, restricting the number of experts did not appreciably impact performance. Indeed, DWM-NB with five experts performed almost as well as the original algorithm that placed no limit on the number of experts created.

Comparing our results for the SEA concepts to those reported by Street and Kim (2001), DWM-NB outperformed SEA on all four target concepts. On the first concept, performance was similar in terms of slope, but not in terms of asymptote, and on subsequent concepts, DWM-NB converged more quickly to the target concepts and did so with higher accuracy. For example, on concepts 2–4, just prior to the point at which concepts changed, SEA achieved accuracies in the 90–94% range, while DWM-NB's were in the 96–98% range.

We suspect this is most likely due to SEA's unweighted voting procedure and its method of creating and removing new classifiers. Recall that the method trains a new classifier on a fixed number of examples. If the new classifier improves the global performance of the ensemble, then it

is added, provided the ensemble does not contain a maximum number of classifiers; otherwise, SEA replaces a poorly performing classifier in the ensemble with the new classifier.

However, if every classifier in the ensemble has been trained on a given target concept, and the concept changes to one that is disjoint, then SEA must replace at least half of the classifiers in the ensemble before accuracy on the new target concept will surpass that on the old. For instance, if the ensemble consists of 20 classifiers, and each learns from a fixed set of 500 examples, then it would take at least $5,000$ additional training examples before the ensemble contained a majority number of classifiers trained on the new concept.

In contrast, DWM under similar circumstances requires only $1,500$ examples. Assume $p = 500$, the ensemble consists of 20 fully trained classifiers, all with a weight of one, and the new concept is disjoint from the previous one. When an example of this new concept arrives, all 20 classifiers will predict incorrectly, DWM will reduce their weights to 0.5—since the global prediction is also incorrect—and it will create a new classifier with a weight of one. It will then process the next 499 examples.

Assume another example arrives. The original 20 experts will again misclassify the example, and the new expert will predict correctly. Since the weighted prediction of the twenty will be greater than that of the one, the global prediction will be incorrect, the algorithm will reduce the weights of the twenty to 0.25, and it will again create a new expert with a weight of one. DWM will again process 499 examples.

Assume a similar sequence of events occurs: another example arrives, the original twenty misclassify it, and the two new ones predict correctly. The weighted-majority vote of the original twenty will still be greater than that of the new experts (i.e., $20(0.25) > 2(1)$), so DWM will decrease the weight of the original twenty to 0.125, create a new expert, and process the next 499 examples. However, at this point, the three new classifiers trained on the target concept will be able to overrule the predictions of the original twenty, since $3(1) > 20(0.125)$. Crucially, DWM will reach this state after processing only $1,500$ examples.

Granted, this analysis of SEA and DWM does not take into account the convergence of the base learners, and as such, it is a best-case analysis. The actual number of examples required may be greater for both to converge to a new target concept, but the relative proportion of examples should be similar. This analysis also holds if we assume that DWM replaces experts, rather than creating new ones. Generally, ensemble methods with weighting mechanisms, like those present in DWM, will converge more quickly to target concepts (i.e., require fewer examples) than will methods that replace unweighted learners in the ensemble.

Regarding the number of experts that DWM maintained, we used a simple heuristic that added a new expert whenever the global prediction was incorrect, which intuitively, should be problematic for noisy domains. However, on the SEA concepts, while DWM-NB maintained as many as 40 experts at, say, time step $37,500$, it maintained only 22 experts on average over the 10 runs, which is similar to the 20–25 that SEA reportedly stored (Street and Kim, 2001).

If the number of experts were to reach impractical levels, then DWM could simply stop creating experts after obtaining acceptable accuracy; training would continue. Plus, we could easily distribute the training of experts to processors of a network or of a course-grained parallel machine. And as the results pictured in Figure 8 demonstrate, we can limit the number of experts that DWM maintains with potentially little effect on accuracy.

One could argue that better performance of DWM-NB is due to differences between the base learners. SEA was an ensemble of C4.5 classifiers (Quinlan, 1993), while DWM-NB, of course, used

naive Bayes as the base learner. We refuted this hypothesis by running both base learners on each of the four target concepts. Both achieved comparable accuracies on each concept. For example, on the first target concept, C4.5 achieved 99% accuracy and naive Bayes achieved 98%. Since these learners performed similarly, we concluded that our positive results on this problem were due not to the superiority of the base learner, but to the mechanisms that create, weight, and remove experts.

We did not evaluate DWM-ITI on the SEA concepts, since ITI maintains all training examples and all observed values for continuous attributes, and this would have led to impractical memory requirements. However, this does not exclude the possibility of using DWM on large data sets with a decision-tree learner as the base algorithm. For instance, we could use ITI, but implement schemes to index stored training examples, which would reduce memory requirements. We could also use a decision-tree learner that does not store examples, such as ID4 (Schlimmer and Fisher, 1986) or VFDT (Domingos and Hulten, 2000).

4.3 Calendar Scheduling Domain

The Calendar Apprentice (CAP) predicts user preferences for scheduling meetings in an academic institution (Mitchell et al., 1994). The task is to predict a user's preference for a meeting's location, duration, starting time, and day of the week. The data set consists of 34 features—such as the type of meeting, the purpose of the meeting, the type of attendees, and whether the meeting occurs during lunchtime—with intersecting subsets of these features for each prediction task. There are 12 features for location, 11 for duration, 15 for start time, and 16 for day of week. Although data are available for two users, we used the $1,685$ examples of the preferences for User 1 (Tom Mitchell).

For this experiment, we evaluated naive Bayes and DWM-NB. However, unlike previous designs, in which we tested the resulting classifiers using a test set, in this design, we measured performance on the next example (i.e., the next meeting to be scheduled). For this application, when another user proposes a meeting, the Calendar Apprentice predicts, say, the meeting's location, and the user either accepts or rejects the recommendation. The learner then uses this feedback to update its model of the user's preferences.

Table 1 shows the average performance of naive Bayes and DWM-NB for the calendar scheduling task. With the exception of predicting the preferred day of week for meetings, DWM-NB outperformed naive Bayes. Over the four prediction tasks, DWM-NB outperformed naive Bayes. As one can see, increasing the parameter p, which governs how often DWM updates its experts, generally decreased DWM's performance. Figure 9 shows accuracy versus the number of examples for the two learners on the four prediction tasks. According to Blum (1997), the sharp decreases in accuracy roughly correspond to the boundaries of semesters.

On this problem, Blum (1997) reported that, over the four prediction tasks, the CAP system averaged 53% and weighted majority with pairs of features averaged 57%. (Other algorithms in this study, such as winnow, performed even better.) DWM-NB averaged about 55%, which was better than the original CAP system, which used a decision tree to predict, but it was not better than Blum's weighted majority.

However, our analysis of this data set suggests that, again, these differences in performance were due to the base learners rather than to the global algorithms. Because of their complexity, we were not able to analyze the CAP concepts in the same manner that we analyzed the STAGGER concepts. We were nonetheless able to determine that pairs (and triples) of features were better suited to the prediction tasks than were all of the features. It is well known that naive Bayes can be sensitive to

Prediction Task	Naive Bayes	DWM-NB		
		$p = 1$	$p = 10$	$p = 50$
Location	62.14	65.69	65.16	62.43
Duration	62.37	64.44	64.62	63.03
Start Time	32.40	38.10	37.39	34.96
Day of Week	51.22	51.16	49.13	51.34
Average	52.03	54.85	54.07	52.84

Table 1: Percent correct of naive Bayes and DWM-NB on the CAP data set, using $1,685$ examples for User 1. The variance of $1,685$ Bernoulli trials is 0.0144%.
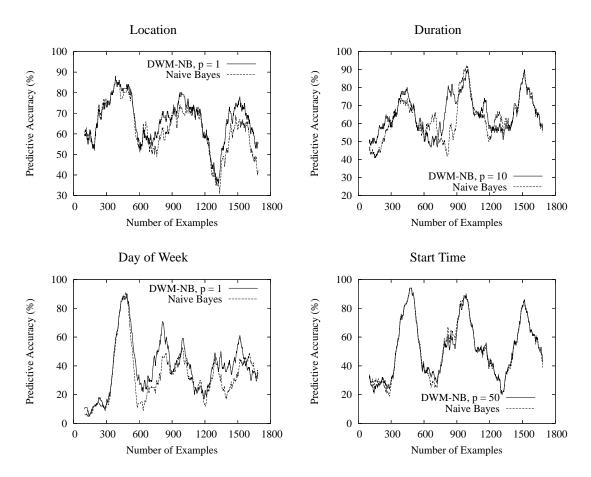

Figure 9: Accuracy on four calendar scheduling tasks for DWM-NB and naive Bayes. Measures are averages of the previous 100 predictions.

2776

Figure 10: Performance of DWM-NB on the electricity pricing task. Measures are averages of the previous $2,352$ predictions. Left: Predictive accuracy. Right: Number of experts maintained.

conditionally-dependent attributes, and we suspect that this is why Blum's (1997) implementation of weighted majority outperformed DWM-NB on this problem.

Since Blum's implementation forms concept descriptions consisting of weighted pairs of attribute values, we reasoned that conditionally-dependent attributes would have less affect on its performance than they would on naive Bayes'. Indeed, experts that predict based on pairs of attribute values should benefit from conditionally-dependent attributes. Furthermore, since the weighting scheme identifies sets of predictive pairs of attribute values, it should do so regardless of conditional dependence among those attributes.

4.4 Electricity Pricing Domain

As a second evaluation on a real-world problem, we selected the domain of electricity pricing (Harries, 1999; Gama et al., 2004). Harries (1999) obtained this data set from TransGrid, the electricity supplier in New South Wales, Australia. It consists of $45,312$ instances collected at 30-minute intervals between 7 May 1996 and 5 December 1998.[1] Each instance consists of five attributes and a class label of either up or down. There are two attributes for time: day of week, which is an integer in $[1,7]$, and period of day, which is an integer in $[1,48]$ (since there are 48 thirty-minute periods in one day). The remaining three attributes are numeric and measure current demand: the demand in New South Wales, the demand in Victoria, and the amount of electricity scheduled for transfer between the two states. The task is to use the attribute values to predict whether the price of electricity will go up or down.

Since predicting the price of electricity is an online task, we processed the examples in temporal order—the order they appeared in the data set. For each example, we first obtained predictions from DWM-NB and from naive Bayes, and then trained each learner on the example. (This is the same experimental design we followed for the calendar scheduling task.) As before, we set DWM to halve expert weights ($\beta = 0.5$), to update each time step ($p = 1$), and to remove an expert if its weight falls below 0.01 ($\theta = 0.01$).

Overall, naive Bayes averaged 62.32% and DWM-NB averaged 80.75%, maintaining an average of 22 experts. For reference, Harries (1999) used an online version of C4.5 (Quinlan, 1993) that

---

1. Our data set corresponds to the Elec2-3 data set (Harries, 1999).

built a decision tree from examples in a sliding window and then classified examples in the following week, reporting accuracies for various window sizes between 66% and 67.7%.

In Figure 10, we present performance curves for accuracy and for the number of experts that DWM maintained. To produce these curves, we averaged the raw performance metrics over a one-week period (i.e., $2,352$ observations).

As a data set derived from real-world phenomenon, we cannot know definitively if or when concept drift occurred. Nonetheless, DWM does appear to have been more robust to change present in the samples than was naive Bayes. One example is the period surrounding time step $10,000$. Naive Bayes' predictive accuracy fluctuates during this period, but DWM's remains nearly constant, on average.

Most illustrative is naive Bayes' sudden drop of roughly 12% in accuracy between time steps $35,000$ and $37,000$. DWM's accuracy decreased slightly and steadily during this period, but there was no comparable sudden decrease in performance.

The number of experts that DWM maintained fluctuated considerably between 1 and 90, but with an overall average of 22 experts. In terms of the weekly average, shown in the right graph of Figure 10, the average size of the ensemble never exceeded 29 experts, which we found encouraging for a problem with $45,312$ examples.

4.5 Evaluation on Static Concepts

Finally, we evaluated DWM-NB on 26 data sets from the UCI Repository (Asuncion and Newman, 2007). It is clear that, for static concepts, there is nothing inherent to the DWM algorithm that would make it more advantageous than a single learner. However, it is important to establish that DWM performs *no worse* than a single learner. Therefore, we selected data sets that varied in the number of classes, the number of examples, the types of attributes, and the number of missing values.

For each data set, we randomly selected 10% of the examples for testing.[2] We presented each example of the remaining 90% to naive Bayes and to DWM-NB with three settings of the parameter p: 1, 10, and 50. After each presentation, we evaluated the resulting concept descriptions on the examples in the testing set. We repeated this procedure ten times, averaging percent correct over these runs and computing 95% confidence intervals.

The results, presented in Appendix A, demonstrate that DWM-NB performed no worse than naive Bayes, the single base learner. As one can see in Table 2, naive Bayes outperformed DWM-NB on most of the tasks (16 of 26), but many of these differences are within 0.5%. For the tasks on which DWM-NB outperformed naive Bayes, many of the differences in performance were also within this range. Overall, the average difference in performance is $+0.35\%$—in DWM's favor—and so we concluded that, on these data sets, DWM-NB performed no worse than a single instance of naive Bayes.

In Figures 11–13, we present the learning curves for this experiment. For legibility, we did not include the curves for DWM-NB with $p = 1$. Limiting DWM-NB to ten experts reduced performance only slightly, and we omitted these results for brevity.

---

2. For data sets with predetermined training and testing sets, we used all available examples to create a single file of examples and proceeded as described.

5. Concluding Remarks

Tracking concept drift is important for many applications. Clearly, with learners for drifting concepts, there is a balance between learning that is completely reactive (e.g., predicting based only on the last example) and learning that is completely unreactive (e.g., predicting based on all encountered examples). If a learner knew when concepts had changed, it could discard its old descriptions and start learning anew. However, this may not always lead to optimum performance on a task because there may be knowledge of the old concept useful for acquiring the new concept. If the learner could appropriately leverage its relevant old knowledge, then performance on the new concept may be better, if not in terms of asymptote, then perhaps in terms of slope.

In this paper, we presented an ensemble method based on the weighted majority algorithm (Littlestone and Warmuth, 1994). Our method, dynamic weighted majority, creates and removes base algorithms in response to changes in performance, which makes it well suited for problems involving concept drift. We described two implementations of DWM, one with naive Bayes as the base learner, the other with ITI (Utgoff et al., 1997). On the problems we considered, a weighted ensemble of learners with mechanisms to add and remove experts in response to changes in performance provided a better response to concept drift than did other learners, especially those that relied on only incremental learning (i.e., STAGGER and AQ11), on the maintenance of previously encountered examples (i.e., FLORA2 and the AQ-PM systems), or on an ensemble of unweighted learners (i.e., SEA).

Using the STAGGER concepts, we evaluated DWM-NB and DWM-ITI our implementation of STAGGER, Blum's implementation of weighted majority, and four rule learners based on the AQ algorithm. To determine performance on a larger problem, we evaluated DWM-NB on the SEA concepts. Results on these problems, when compared to other methods, suggest that DWM maintained a comparable number of experts, but achieved higher predictive accuracies and converged to those accuracies more quickly. Indeed, to the best of our knowledge, these are the best overall results reported for these problems.

DWM, on a calendar scheduling task, outperformed the CAP system and performed comparably to Blum's weighted majority. On an electricity pricing domain, DWM outperformed a single base learner and an online version of C4.5 (Harries, 1999). Finally, on several problems with static concepts, DWM performed as well as a single base learner.

In future work, we plan to investigate more sophisticated heuristics for creating new experts: Rather than creating an expert when the global prediction is wrong, perhaps DWM should take into account the expert's age or its history of predictions. We would also like to investigate another decision-tree learner as the base algorithm, one that does not maintain encountered examples and that does not periodically restructure its tree; VFDT (Domingos and Hulten, 2000) is a likely candidate.

Although removing experts of low weight yielded positive results for the problems we considered in this study, it would be beneficial to investigate mechanisms for explicitly handling noise, such as those present in IB3 (Aha et al., 1991), or for determining when examples are likely to be from a different target concept, such as those based on the Hoeffding (1963) bounds present in VFDT (Domingos and Hulten, 2000) and CVFDT (Hulten et al., 2001).

Finally, the most intriguing prospect for future work is to explore the relationship between a learner's stability and its skill at coping with drifting concepts. Equally interesting is how certain mechanisms, such as learning from previously encountered examples in addition to new ones, af-

fects a learner's stability and its ability to cope in non-stationary environments. We anticipate that these investigations will lead to general, robust, and scalable ensemble methods for tracking concept drift.

Acknowledgments

Appendix A. Performance of DWM-NB on Selected UCI Data Sets

To establish DWM's performance on static concepts, we applied DWM-NB to twenty-six data sets from the UCI Repository (Asuncion and Newman, 2007). For each data set, we randomly selected 10% of the examples as a testing set, and used the remaining 90% for training. We presented a single training example to naive Bayes and to DWM-NB, and evaluated the resulting classifiers on the examples in the testing set, measuring percent correct. We proceeded in this manner until processing all of the training examples. We then repeated this procedure nine more times for a total of ten applications, after which we averaged the measures of performance and computed 95% confidence intervals. Table 2 lists these results. Although DWM-NB did not outperform naive Bayes, as expected, it also performed no worse. Over these twenty-six data sets, the average difference in performance is $+0.35\%$. Figures 11–13 contain learning curves for these data sets.

Data Set	NB	DWM-NB $p = 1$	DWM-NB $p = 10$	DWM-NB $p = 50$	Δ%
balance-scale	**90.27±0.13**	90.19±0.18	89.75±0.43	89.82±0.36	−0.08
breast-cancer	72.60±0.37	70.62±1.42	72.60±0.62	**72.88±0.30**	+0.28
breast-w	**95.99±0.04**	95.98±0.07	95.93±0.25	95.98±0.05	−0.01
colic	78.33±0.24	76.24±3.06	**79.29±1.05**	79.23±0.69	+0.96
credit-a	77.73±0.17	76.52±3.63	**78.70±0.67**	77.99±0.24	+0.97
credit-g	**74.97±0.30**	67.56±2.46	73.62±0.96	74.18±0.41	−0.79
diabetes	75.37±0.21	69.94±2.34	74.71±0.84	**75.51±0.38**	+0.14
glass	46.18±1.36	45.56±3.46	47.05±2.51	**47.97±0.85**	+1.79
heart-h	**83.89±0.42**	82.97±1.06	83.17±0.74	83.51±0.62	−0.38
heart-statlog	**84.11±0.46**	82.89±1.20	83.63±0.57	**84.11±0.48**	0.00
hepatitis	83.07±0.55	82.60±2.33	**84.07±0.76**	82.68±0.75	+1.00
hypothyroid	95.37±0.06	**95.64±0.33**	95.62±0.23	95.38±0.28	+0.27
ionosphere	**88.29±0.41**	87.60±3.24	88.26±0.72	87.81±0.62	−0.03
kr-vs-kp	**87.78±0.09**	78.50±7.69	86.65±1.03	87.15±0.41	−0.63
labor	**90.67±0.61**	89.40±1.33	89.27±0.96	90.33±1.00	−0.34
lymph	**77.34±1.87**	74.81±1.87	75.50±2.18	76.74±2.65	−0.60
mushroom	**95.75±0.03**	94.81±1.18	95.00±0.33	95.25±0.19	−0.50
primary-tumor	**48.62±0.63**	36.76±6.45	44.25±1.94	46.81±1.52	−1.81
segment	79.72±0.13	73.94±11.46	80.32±0.60	**80.38±0.37**	+0.66
sick	92.71±0.07	**94.44±0.32**	93.16±0.69	92.32±0.84	+1.73
soybean	**92.78±0.15**	92.76±0.15	92.50±0.27	90.03±2.14	−0.02
splice	**95.37±0.06**	**95.37±0.06**	95.31±0.14	95.23±0.24	0.00
vehicle	45.51±0.28	42.51±2.87	**46.33±1.82**	44.44±0.81	+0.82
vote	**90.09±0.11**	89.83±0.25	89.88±0.36	89.91±0.26	−0.18
vowel	**63.04±0.81**	49.07±10.16	58.60±3.05	61.01±1.92	−2.03
zoo	**88.27±0.20**	87.97±0.36	87.07±0.90	87.56±0.86	−0.30

Table 2: Performance of naive Bayes (NB) and DWM-NB on twenty-six selected UCI data sets. DWM-NB created experts every p examples. Measures are percent correct with 95% confidence intervals. Maximum accuracies are typeset in boldface; however, note that many of the differences are not statistically significant. The final column shows the difference in percentage between naive Bayes and the best performing DWM classifier.
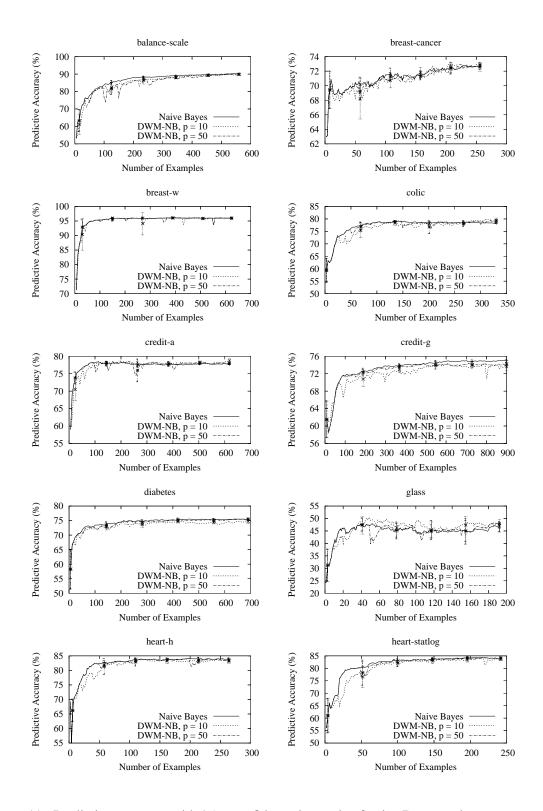
Figure 11: Predictive accuracy with 95% confidence intervals of naive Bayes and DWM-NB on selected UCI data sets, balance-scale to heart-statlog.
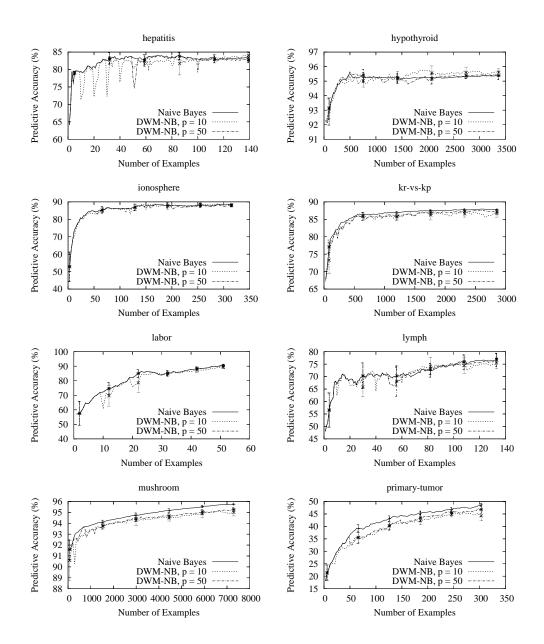
Figure 12: Predictive accuracy with 95% confidence intervals of naive Bayes and DWM-NB on selected UCI data sets, hepatitis to primary-tumor.
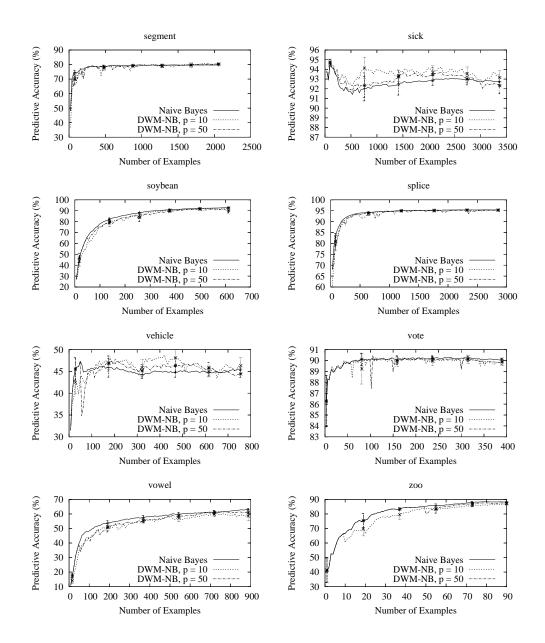
Figure 13: Predictive accuracy with 95% confidence intervals of naive Bayes and DWM-NB on selected UCI data sets, segment to zoo.

References

D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6: 37–66, 1991.

A. Asuncion and D. J. Newman. UCI Machine Learning Repository. Web site, Department of Information and Computer Sciences, University of California, Irvine, `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2007.

P. Auer and M. K. Warmuth. Tracking the best disjunction. *Machine Learning*, 32(2):127–150, 1998.

B. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1–2):105–139, 1999.

M. M. Black and R. J. Hickey. Maintaining the performance of a learned classifier under concept drift. *Intelligent Data Analysis*, 3(6):453–474, 1999.

M. M. Black and R. J. Hickey. Classification of customer call data in the presence of concept drift and noise. In *Soft-Ware 2002: Computing in an Imperfect World*, volume 2311 of *Lecture Notes in Computer Science*, pages 74–87. Springer, Berlin, 2002. First International Conference, Soft-Ware 2002, Belfast, Northern Ireland, April 8–10, 2002. Proceedings.

A. Blum. Empirical support for Winnow and Weighted-Majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26:5–23, 1997.

B. E. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fourth Workshop on Computational Learning Theory*, pages 144–152. ACM Press, New York, NY, 1992.

O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, 3:363–396, 2002.

L. Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4–5):187–195, 2005.

T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–158, 2000.

P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, New York, NY, 2000.

W. Fan. StreamMiner: A classifier ensemble-based engine to mine concept-drifting data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 1257–1260. Morgan Kaufmann, San Francisco, CA, 2004.

W. Fan, S. J. Stolfo, and J. Zhang. The application of AdaBoost for distributed, scalable and on-line learning. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 362–366. ACM Press, New York, NY, 1999.

A. Fern and R. Givan. Online ensemble learning: An empirical study. *Machine Learning*, 53: 71–109, 2003.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, San Francisco, CA, 1996.

J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Advances in Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer, Berlin, 2004. Seventeenth Brazilian Symposium on Artificial Intelligence, SBIA-2004, São Luis, Maranhõ, Brazil, September 29–October 1, 2004, Proceedings.

J. Gama, P. Medas, and P. Rodrigues. Learning decision trees from dynamic data streams. In *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC-2005)*, pages 573–577. ACM Press, New York, NY, 2005.

R. B. Gramacy, M. K. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *Advances in Neural Information Processing Systems 15*, pages 1465–1472. MIT Press, Cambridge, MA, 2003.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

M. Harries. Splice-2 Comparative Evaluation: Electricity Pricing. Technical Report UNSW-CSE-TR-9905, Artificial Intelligence Group, School of Computer Science and Engineering, The University of New South Wales, Sidney, Australia, July 1999.

M. Harries and K. Horn. Detecting concept drift in financial time series prediction using symbolic machine learning. In *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pages 91–98. World Scientific, Singapore, 1995.

D. P. Helmbold and P. M. Long. Tracking drifting concepts using random examples. In L. G. Valiant and M. K. Warmuth, editors, *Proceedings of the Fourth Annual Workshop on Computational Learning Theory (COLT'91)*, pages 13–23. Morgan Kaufmann, San Francisco, CA, 1991.

D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimising disagreements. *Machine Learning*, 14:27–45, 1994.

M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

R. J. Hickey and M. M. Black. Refined time stamps for concept drift detection during mining for classification rules. In *Temporal, Spatial, and Spatio-Temporal Data Mining*, volume 2007 of *Lecture Notes in Artificial Intelligence*, pages 20–30. Springer, Berlin, 2000. First International Workshop, TSDM 2000, Lyon, France, September 2000, Revised papers.

D. Hinkley. Jackknife methods. In S. Kotz, N. L. Johnson, and C. B. Read, editors, *Encyclopedia of Statistical Sciences*, volume 4, pages 280–287. John Wiley & Sons, New York, NY, 1983.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106. ACM Press, New York, NY, 2001.

M. G. Kelly, D. J. Hand, and N. M. Adams. The impact of changing populations on classifier performance. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 367–371. ACM Press, New York, NY, 1999.

M. Klenner and U. Hahn. Concept versioning: A methodology for tracking evolutionary concept drift in dynamic concept systems. In *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pages 473–477. John Wiley & Sons, London, 1994.

R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann, San Francisco, CA, 2000.

Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004. Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift.

J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the Third IEEE International Conference on Data Mining*, pages 123–130. IEEE Press, Los Alamitos, CA, 2003.

J. Z. Kolter and M. A. Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the Twenty-second International Conference on Machine Learning*, pages 449–456. ACM Press, New York, NY, 2005.

A. Kuh, T. Petsche, and R. L. Rivest. Learning time-varying concepts. In *Advances in Neural Information Processing Systems 3*, pages 183–189. Morgan Kaufmann, San Francisco, CA, 1991.

T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 259–263. AAAI Press, Menlo Park, CA, 1998.

N. Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using Winnow. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 147–156. Morgan Kaufmann, San Francisco, CA, 1991.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

N. Littlestone and M. K. Warmuth. The Weighted Majority algorithm. *Information and Computation*, 108:212–261, 1994.

R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551. AAAI Press, Menlo Park, CA, 1997.

M. A. Maloof. Concept drift. In J. Wang, editor, *Encyclopedia of Data Warehousing and Mining*, pages 202–206. Information Science Publishing, Hershey, PA, 2005.

M. A. Maloof. Incremental rule learning with partial instance memory for changing concepts. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2764–2769. IEEE Press, Los Alamitos, CA, 2003.

M. A. Maloof and R. S. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154:95–126, 2004.

M. A. Maloof and R. S. Michalski. Selecting examples for partial memory learning. *Machine Learning*, 41:27–52, 2000.

C. Mesterharm. Tracking linear-threshold concepts with Winnow. *Journal of Machine Learning Research*, 4:819–838, 2003.

R. S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing*, volume A3, pages 125–128. 1969.

R. S. Michalski and J. B. Larson. Incremental Generation of VL_1 Hypotheses: The Underlying Methodology and the Description of Program AQ11. Technical Report UIUCDCS-F-83-905, Department of Computer Science, University of Illinois, Urbana, 1983.

T. M. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, July 1994.

C. Monteleoni and T. S. Jaakkola. Online learning of non-stationary sequences. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1993.

J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 725–730. AAAI Press, Menlo Park, CA, 1996.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

R. E. Reinke and R. S. Michalski. Incremental learning of concept descriptions: A method and experimental results. In J. E. Hayes, D. Michie, and J. Richards, editors, *Machine Intelligence 11*, pages 263–288. Clarendon Press, Oxford, 1988.

R. A. Rescorla. Probability of shock in the presence and absence of CS in fear conditioning. *Journal of Comparative and Physiological Psychology*, 66:1–5, 1968.

J. C. Schlimmer. *Concept Acquisition through Representational Adjustment*. PhD thesis, Department of Information and Computer Science, University of California, Irvine, 1987.

J. C. Schlimmer and D. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501. AAAI Press, Menlo Park, CA, 1986.

J. C. Schlimmer and R. H. Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 502–507. AAAI Press, Menlo Park, CA, 1986.

M. Scholz and R. Klinkenberg. Boosting Classifiers for Drifting Concepts. Technical report, Collaborative Research Center on the Reduction of Complexity for Multivariate Data Structures (SFB 475), University of Dortmund, Dortmund, Germany, January 2006.

M. Scholz and R. Klinkenberg. An ensemble classifier for drifting concepts. In J. Aguilar and J. Gama, editors, *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*, pages 53–64. `http://www.liacc.up.pt/~jgama/IWKDDS/`, 2005. Held at the 16th European Conference on Machine Learning (ECML) and European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), ECML/PKDD-2005.

W. N. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 377–382. ACM Press, New York, NY, 2001.

N. A. Syed, H. Liu, and K. K. Sung. Handling concept drifts in incremental learning with support vector machines. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 272–276. ACM Press, New York, NY, 1999.

A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Dynamic Integration of Classifiers for Tracking Concept Drift in Antibiotic Resistance Data. Technical Report TCD-CS-2005-26, Trinity College Dublin, Dublin, Ireland, February 2005.

P. E. Utgoff, N. C. Berkman, and J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29:5–44, 1997.

H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235. ACM Press, New York, NY, 2003.

G. Widmer. Tracking context changes through meta-learning. *Machine Learning*, 27:259–286, 1997.

G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23:69–101, 1996.

I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 2005.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4): 405–410, 1997.

Z. Zheng. Naive Bayesian classifier committees. In *Proceedings of the Tenth European Conference on Machine Learning*, pages 196–207. Springer, Berlin, 1998.

Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1–2):239–263, 2002.