Machine Learning, 26, 5?? (1997)

Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain

AVRIM BLUM avrim@cs.cmu.edu
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

Editor:

Abstract. This paper describes experimental results on using Winnow and Weighted-Majority based algorithms on a real-world calendar scheduling domain. These two algorithms have been highly studied in the theoretical machine learning literature. We show here that these algorithms can be quite competitive practically, outperforming the decision-tree approach currently in use in the Calendar Apprentice system in terms of both accuracy and speed. One of the contributions of this paper is a new variant on the Winnow algorithm (used in the experiments) that is especially suited to conditions with string-valued classifications, and we give a theoretical analysis of its performance. In addition we show how Winnow can be applied to achieve a good accuracy/coverage tradeoff and explore issues that arise such as concept drift. We also provide an analysis of a policy for discarding predictors in Weighted-Majority that allows it to speed up as it learns.

Keywords: Winnow, Weighted-Majority, Multiplicative algorithms

1. Introduction

Multiplicative weight updating algorithms such as Winnow (Littlestone, 1988) and Weighted Majority variants (DeSantis et al., 1988; Littlestone and Warmuth, 1994; Cesa-Bianchi et al., 1993) have been studied extensively in the theoretical machine learning literature, in which a collection of strong properties have been proven. These algorithms could be said to fall into the category of "learning simple things really well." In particular, when the concept being learned is appropriately simple, they have been proven to have exceptionally good behavior in the face of irrelevant features, noise, or a target function changing with time (Littlestone, 1991; Littlestone and Warmuth, 1994). In this paper we add evidence of the practical importance of these algorithms. We show that these approaches achieve excellent performance in a natural learning task: the calendar scheduling domain of Mitchell et al. (Dent et al., 1992; Jourdan et al., 1991; Mitchell et al., 1994). In particular, our implementations give a substantial improvement along a number of lines, including accuracy and speed, over the results of Mitchell et al. (1994), who use a decision-tree based approach.

The main contributions of this paper are threefold. First, we describe how the Winnow and Weighted-Majority algorithms can be naturally applied to settings like the calendar domain having string-valued attributes and many potential string-valued classifications. In particular, we describe a new version of the Winnow algorithm especially suited to such conditions, that can be viewed as combining the opinions of "specialists" in analogy to the way in which the Weighted-Majority algorithm is viewed as combining the opinions of "experts". Second, we show that these approaches are useful in practice, achieving a good performance in terms of accuracy, speed, and accuracy vs. coverage tradeoffs. Finally, we provide a theoretical analysis of a number of issues raised by the experiments, such as what kind of performance guarantee can be placed on our Winnow variant, and when one can discard poorly performing "experts" in the Weighted-Majority algorithm.

1.1. Summary

Mitchell et al. (1994) describe results of the Calendar APprentice (CAP) system which uses a decision-tree based learning algorithm to predict certain aspects of a calendar event (location, duration, start time, and day of week) based on known facts about the attendees and the past history. We show here that Winnow and Weighted Majority achieve a good performance both in accuracy and in speed for this task. For example, on the 1685 data points made available for one user's (Tom Mitchell's) calendar, our Winnow variant achieves 75% accuracy in predicting a meeting's location, compared with 64% for CAP (see Table 1). Also, Winnow is quite fast due to its nature as an incremental algorithm. Even when there are many features, it can easily perform its loop of predicting, finding the correct answer, and then updating its internal state, in real time. In contrast, because the learning methods used by CAP were much slower, CAP had to restrict its learning to a batch job each night.

We also show how Winnow can be adapted to provide a good *accuracy vs. coverage tradeoff*, a topic not explored much in the theoretical literature, but important in practice.

One property of the Winnow and Weighted-Majority algorithms is that they are not very sensitive to the presense of extra, possibly irrelevant attributes (Littlestone, 1988; Littlestone and Warmuth, 1994). Mitchell et al. use a special "feature selection" procedure (Caruana and Freitag, 1994) that restricts the set of features presented to their learning algorithm. However, our experiments show that Winnow and Weighted-Majority experience only a very small loss in performance when a larger set is used, and in fact in some cases performance actually *improves*. The main performance loss is one of speed, but even this can be lessened, especially in the case of Weighted Majority, by a pruning method.

2. The learning problem

An *example* in the Calendar Apprentice domain is a calendar event to be scheduled on a user's calendar. It is described by a set of features such as:

- What type of event is it? (meeting, seminar, course, ...)
- What is the name of the seminar? (no-value, AI-seminar, ...)
- What is the position of the attendees? (graduate student, faculty, funder, ...)
- Are the attendees in the user's group? (yes, no)
- What are the names of the attendees in alphabetical order?

```
(req-event-type meeting)
(req-seminar-type nil)
(sponsor-attendees *inferred.novalue*)
(department-attendees cs)
(position-attendees faculty)
(group-attendees? no)
(req-course-name nil)
(department-speakers *inferred.novalue*)
(group-name *inferred.novalue*)
(lunchtime? no)
(single-person? yes)
(number-of-person 1)

(req-location dh4301c)
```

Figure 1. *One of the data points for User 1. Listed are the features used for predicting location by CAP, along with the correct location of this meeting.*

and so forth. A specific example is given in Figure 1. The quantities we are asked to predict are the event's location, duration, start time, and day of week. In the current CAP system, these quantities are then presented as default values to the user, who may decide to override them or not. The actual decisions made by the user are then given as feedback to the learner. The CAP project has made their data available, and in particular 1685 examples spanning two years of Tom Mitchell's calendar and 554 examples from a second user.[1]

A few aspects of this learning task worth noticing are:

1. The "target concept"[2] is something that changes with time. For instance, at a semester boundary, the times and days for a seminar or group meeting might change, the behavior of the user might change because he is now teaching a large class, and so forth.

   In fact, for this reason the CAP system each day builds a decision tree (which it then prunes into rules in a manner similar to C4.5) based on only the most recent 180 examples, where 180 is a value determined empirically to provide good performance.

2. The features in general are string-valued, and in fact the set of possible values held by some feature may not even be known at the start. For instance, we might not know at the start what the set of all seminar names might be. Similarly, the quantities we are asked to predict are also string-valued, and we also may not know what their possible values might be at the start: for instance, this is the case with location.

As mentioned above, the CAP system uses a decision-tree based learning method. More specifically, CAP maintains a database of rules, sorted by observed performance. Each night, CAP builds a decision tree using the most recent 180 example, prunes this tree into a list of rules, and merges these rules into its database, updating its statistics (Mitchell et al., 1994). A number of design choices are involved in this process, such as exactly what features to give to the decision-tree algorithm (CAP provides the algorithm only a restricted subset, based on the learning task), how to merge new and old rules, what length window to use, and so forth (see Mitchell et al. (1994) for details). In settling on their approach, the CAP designers also tried several other learning methods, including other decision tree variants and a neural-network algorithm, and found these to have equal or worse performance.

3. Description of the algorithms

We consider two multiplicative-weight-updating learning algorithms: the Weighted-Majority algorithm (Littlestone and Warmuth, 1994) and a version of Littlestone's Winnow algorithm (Littlestone, 1988). These are both *incremental* algorithms. Upon receipt of an unlabeled example they make a prediction. When they are told the correct answer, they use that information to adjust their hypothesis, and then are ready for the next example.

3.1. Weighted-Majority: combining experts

The simpler of the two algorithms implemented is essentially a straight Weighted-Majority algorithm as described in Littlestone and Warmuth (1994). It is based on the supposition that out of all the features given, perhaps there exists some small set of, say, two features, that just by themselves are enough to construct a good predictor. For instance, perhaps knowing only the event type and the seminar name is enough to predict well.

Specifically, the algorithm works as follows. For each pair of features we create a prediction strategy (an "expert") that examines only those two features and makes predictions based on their values. The global algorithm will receive predictions from all $\binom{n}{2}$ experts and then will decide based on a weighted-majority vote of those predictions. To fully specify this approach, we must say (1) how each individual expert works, and (2) how we will weight the votes.

We perform (2) by using the simple multiplicative rule discussed in Littlestone and Warmuth (1994). We begin with all experts having the same weight of 1. When we receive the example's correct labeling, we see which experts predicted incorrectly and cut the weights of those experts in half. We do not modify the weights of those that predicted *correctly*. The experimental results did not depend significantly on the exact constant (in this case, $1/2$) used to multiply the weights of the incorrect experts.

We perform (1) by having each expert perform a simple table lookup: given a pair of values for its two features (e.g., if its features are `event-type` and `position-of-attendees`, then it might see the pair of values `<meeting, grad-student>`) look at the last k times that that pair of values occurred and predict the outcome that occurred most often out of those k. We used $k = 5$. If the pair of values has never occurred before, then the expert predicts a global default (the most common outcome seen so far).

This algorithm clearly can be implemented using, say, triples of features instead of pairs. One could also imagine doing something more intelligent than just predicting a global default when the tuple of values seen does not exactly match anything in memory, but only the simple rule was used in the experiments.

A natural modification to the weighted majority algorithm is to discard experts if their weights drop too low, so that the algorithm *speeds up* as it learns more, in contrast to most learning algorithms that slow down as they learn. Discarding experts may be dangerous if one is too aggressive because the expert that turns out to be best in the end may not appear best near the start. However, one can provide theoretical guarantees for such strategies, which we do in Section 5.1. Our experimental results indicate that for a wide range of weight thresholds, one can achieve both a significant speedup and negligible loss in performance. (See Figure 7.)

3.2. Winnow: combining specialists

The Weighted-Majority algorithm and its variants are often thought of as ways of combining the opinions of "experts" that give predictions for each example. The Winnow algorithm, and in particular the variant we describe here, can be thought of as a method for combining the opinions of entities that each may choose to *abstain* instead of giving a prediction on any given example. We will call these entities *specialists*, because they are allowed to abstain when the current example does not fall into their "specialty".

Specifically, the Winnow variant we consider proceeds as follows. For each pair of (feature=value) conditions experienced so far, such as "`event-type = meeting` and `position-of-attendees = grad-student`," there will be one *specialist*. This specialist is extremely simple: it only wakes up to make a prediction if both its conditions are true, and in that case it predicts the most popular outcome out of the last five times it had the chance to predict.[3] (E.g., in the above case, it predicts the most popular outcome out of the last 5 times there was a meeting with grad students.) The very first time that the specialist appears, it abstains. The global algorithm makes a prediction based on a weighted majority vote over all predicting specialists.

One way to view the algorithm is that we are listing the antecedents for all the possible "rules of length 2", and determining their consequent based on their recent history. Each of the specialists corresponds to one of these rules. We then predict based on a weighted vote. In Section 5.2 we provide theoretical guarantees under the assumption that the target concept consists of a list of rules of length 2. Indeed, one original motivation for trying this approach was the fact that most of the rules created by the CAP algorithm (which runs ID3 and then prunes the decision tree into rules) are quite short.

How are the specialists weighted? When a specialist first appears (the pair of conditions occurs for the first time) the specialist is given weight 1 (and it abstains on this example). On further predictions, we use the basic "Winnow II" strategy described in Littlestone (1988). If the global algorithm predicts incorrectly, then if the specialist predicts incorrectly its weight is halved and if it predicts correctly its weight is multiplied by $3/2$. For a small benefit in accuracy, we halve the weight of a specialist when it makes a mistake *even if* the global algorithm predicted correctly. The reason we multiply by $3/2$ instead of, say, by 2 is for the benefit of our theoretical analysis (Section 5.2); however, tests reveal no significant difference between the two policies.

The reader familiar with the analysis of the Winnow algorithm will note that increasing the weights only when the global algorithm makes a mistake, and *not* when the global algorithm predicts correctly, is necessary for the theoretical analysis to go through. *In fact, in this calendar application, it turns out to be crucial empirically as well.* If one modifies the algorithm to multiply the weights of specialists that predict correctly even when the global algorithm predicts correctly too, then performance drops. Thus, one should not think of the algorithm as simply maintaining "percentage correct" figures for each specialist and weighting accordingly. Specialists are rewarded not so much for predicting correctly, but for predicting correctly in times when the global algorithm should have listened to them more carefully.

The algorithm as described above can be viewed as a variant on the "balanced" version of Winnow (Littlestone, 1989). In the "balanced" algorithm, one would maintain a vector of weights for each specialist, one weight for each possible output. One would then view the specialist as predicting all possible outputs, each with its associated weight, and weights would be updated as if each element of the vector was a separate (fixed) prediction algorithm. The advantage of our approach is (A) less computational overhead, and (B) faster adaptivity to changes. For example, if the AI-seminars are moved from Mondays to Fridays, in our approach this changes a specialist's behavior after 3 mistakes. In the balanced algorithm, this might cause a much larger number of mistakes, even if weights are lower-bounded as in (Littlestone and Warmuth, 1994).

One point to notice is that as learning progresses, the number of specialists in existence may become quite large. However, on any *individual* example, only a small number (number of features choose 2) actually make a prediction. Thus, this setting can be modeled by the *infinite attribute model* of (Blum, 1992; Blum et al., 1991).

4. Experimental results

We ran Winnow and Weighted-Majority on 1685 data points from one user and 554 data points from a second user of the CAP system. We presented the examples to the algorithms one by one in chronological order, recording when mistakes were made. Tables 1 and 2 compare the total percentage of correct predictions made by CAP, Winnow, and Weighted Majority (using both pairs and triples of features). As mentioned in the introduction, CAP filters the feature set and uses only a subset for each prediction task. For instance, 12 features are used for predicting location, 11 for duration, 15 for start time, and 16 for day of week. In the columns labeled "Winnow", "WM", and "WM-triples" we ran Winnow and Weighted-Majority (pairs and triples) on those same feature sets. Notice that Winnow achieves an average improvement over CAP of over 10 percentage points for User 1 and an even greater improvement (nearly 20 points) for User 2. Weighted Majority performs a bit worse than Winnow, but still does surprisingly well considering that the essence of the algorithm is just to quickly find the two (or three) best features. A plot of accuracy versus time for the Winnow algorithm's performance in predicting location is given in Figure 2.

Table 1. Comparison of fraction correct on 1685 data points from User 1 (Tom Mitchell). For reference, the standard deviation for 1685 fair Bernoulli trials is 0.012.

Prediction task	CAP	Winnow	Winnow-bigset	WM	WM-bigset	WM-triples
location	0.64	0.75	0.76	0.70	0.74	0.72
duration	0.63	0.71	0.74	0.64	0.73	0.66
start-time	0.34	0.51	0.53	0.39	0.50	0.45
day-of-week	0.50	0.57	0.57	0.56	0.56	0.57
AVERAGE	0.53	0.63	0.65	0.57	0.63	0.60

Table 2. Comparison of fraction correct on 554 data points from a second user. For reference, the standard deviation for 554 fair Bernoulli trials is 0.021.

Prediction task	CAP	Winnow	Winnow-bigset	WM	WM-bigset	WM-triples
location	0.33	0.71	0.72	0.67	0.66	0.69
duration	0.69	0.73	0.75	0.69	0.69	0.70
start-time	0.32	0.58	0.59	0.52	0.52	0.55
day-of-week	0.37	0.47	0.48	0.43	0.45	0.45
AVERAGE	0.43	0.62	0.63	0.58	0.58	0.60

In order to test the focusing abilities of these algorithms, we also ran Winnow and Weighted-Majority using a larger set of 34 features. This larger set was constructed by taking the entire feature set and just filtering out those whose values we could not represent well (such as bitmaps or high-precision real values) and those that could not legally be used for all 4 tasks (such as "day-of-week", which is in the CAP set of features for predicting start-time, but clearly should not be used for predicting day-of-week). We then ran our algorithms on the larger set. These results are in the columns marked "Winnow-bigset" and "WM-bigset" in Tables 1 and 2.

As expected, Winnow and Weighted Majority are not hurt significantly when the larger feature set is used, but interestingly, performance often *improves*, especially for Weighted-Majority. The reason for the improvement is that features that had been deemed not useful for ID3 may be quite useful for Winnow and Weighted Majority. In fact, in this larger set, for User 1 the performance of Weighted-Majority almost reaches that of our Winnow variant.

In addition to good predictive performance, Winnow and Weighted-Majority as implemented are both quite fast. On a SPARC10, Winnow takes on average about 30 seconds to make an entire sequence of 1685 predictions on the small feature sets, and about 90 seconds on the large feature set. Weighted-Majority is a bit slower without pruning, but a good bit faster with pruning (discussed in Section 4.5 below). Thus, at a speed of $1/60$ to $1/18$ seconds per example, both algorithms can be used in real time.

Theoretical analysis (Littlestone and Warmuth, 1994; Cesa-Bianchi et al., 1993) suggests using randomization instead of a strict weighted majority vote (weights on the outputs are normalized to sum to 1 and are then interpreted as probabilities). This produced no significant effect on prediction accuracy in the two datasets.

One point worth mentioning is that in these experiments, even though the total number of specialists in the Winnow-based learner can get quite large (by the time of the final example

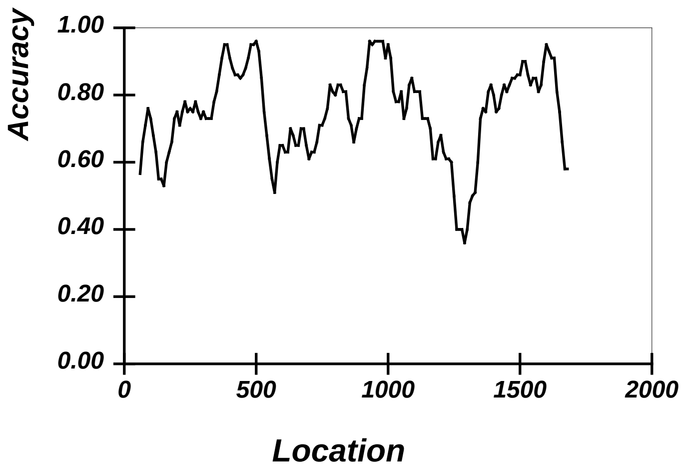

Figure 2. Accuracy versus example number for predicting location using Winnow. The larger dips correspond roughly to semester boundaries.

for User 1, there are 59731 specialists for the large feature set), on any individual example only a relatively small number actually make a prediction (561 in the large feature set). This smaller set of predicting specialists can be identified efficiently using a hash table, so the algorithm does not actually need to poll all of the specialists in existence on each example. In addition, the fact that only a relatively small number of specialists makes a prediction on each example is useful for the theoretical analysis (section 5.2).

4.1. Interpreting the weights: what the algorithms learned

For the Weighted Majority algorithm used, the weights assigned to the "experts" can be interpreted as answering the question: "if you were only allowed to look at two features, which two you choose?" In a sense, Weighted Majority can be viewed as an algorithm designed to answer this question while simultaneously making good predictions as it learns. Interestingly, in this experiment the best pair of features found was often not a subset of the best triple found when the algorithm was run with one expert per triple. For instance, for User 1, while there was a nice containment relationship for predicting location (the best single feature was the number of people in the meeting, the best pair was the number of people and the seminar type, and the best triple was that pair plus one more feature), for predicting duration the best pair and best triple had no intersection.

The Winnow algorithm used can be viewed as assigning weights to each possible "rule of length 2", indicating the extent to which that rule should be trusted. For instance, for predicting duration for User 1, some top-weighted rules are:

Table 3. Average fraction correct (over the 4 tasks) of predictions by CAP, Winnow, and a "brain-damaged" Winnow (WinnowDay) which was only allowed to update its internal state at day boundaries.

	User 1	User 2
CAP	0.53	0.43
Winnow	0.63	0.62
WinnowDay	0.59	0.56
Winnow-bigset	0.65	0.63
WinnowDay-bigset	0.62	0.51

"If there is a single attendee and he/she is from the ECE department, then 30 minutes."

"If there is more than one attendee and they are research programmers, then 60 minutes."

"If the attendees are faculty members and not from CMU then 60 minutes."

As discussed earlier, the weight of a rule does not describe the accuracy of that rule in isolation; rather, the method for updating weights depends on the relationship of that rule to the others that are firing at that time.

4.2. Incremental versus Batch

The Winnow and Weighted-Majority algorithms tested in these experiments were allowed to update their internal state after each example. In some sense, this is an unfair advantage over CAP which only updated its hypotheses overnight. (On the other hand, this is one of the advantages of using a fast, incremental algorithm; namely, real-time updating of hypotheses is possible.) To factor out this advantage, we also ran Winnow in a "brain-damaged" mode in which it was forced to make predictions using its state from the end of the previous day. Results are presented in Table 3. Notice that for User 1, average performance drops by 3% on the large feature set and 4% on the smaller feature sets, compared with the non "brain-damaged" version. For User 2, the drop is more substantial (and interestingly, this seems to cause greater damage to performance on the large feature set) but the improvement over CAP is more substantial as well.

4.3. How important is the weighting scheme?

The standard Winnow algorithm has the property that all of its learned information is stored in its weights: if one does not allow it to update weights, then it learns nothing. However, the "specialists" version of the algorithm used here learns in two ways. The global algorithm learns which specialists to pay attention to (which is stored in the weights), but also each individual specialist learns what output it should predict. In order to test the

Table 4. Testing the importance of the weighting scheme. Winnow-Day refers to the version that only updates its state at day boundaries (see Table 3). "Winnow-1" and "Winnow-Day-1" are the versions that force all weights to remain at 1. All of these results are on the smaller feature sets (the ones used by CAP).

Task	Winnow	Winnow-Day	Winnow-1	Winnow-Day-1
User 1: location	0.75	0.70	0.67	0.59
User 1: duration	0.71	0.68	0.63	0.56
User 1: start-time	0.51	0.45	0.32	0.22
User 1: day-of-week	0.57	0.53	0.47	0.37
AVERAGE	0.63	0.59	0.52	0.44
User 2: location	0.71	0.62	0.65	0.29
User 2: duration	0.73	0.70	0.66	0.44
User 2: start-time	0.58	0.49	0.51	0.18
User 2: day-of-week	0.47	0.42	0.44	0.37
AVERAGE	0.62	0.56	0.56	0.32

relative importance of the weighting scheme, we ran the algorithm in a mode in which weights were fixed to 1. In fact, the two algorithms (Winnow and Weighted Majority) become equivalent in this case, except for small technical differences.[4]

The results of this experiment are listed in Table 4. One can see that forcing all weights to remain at 1 causes Winnow to degrade significantly: a loss of 6-11 percentage points. For the version that is only allowed to update its internal state at day boundaries (the internal state includes the memory of the individual specialists) the loss is much more severe, ranging from 15 to 24 percentage points.

4.4. Accuracy versus coverage

A desirable property for learning algorithms in settings like the calendar domain is to be able to achieve a good *accuracy versus coverage tradeoff*. In other words, the algorithm should have some sort of adjustable parameter that allows it to trade off reduced coverage (the ability to not make a prediction on some examples) for increased accuracy on those examples on which it does predict.

The Winnow algorithm has a natural parameter of this sort. Namely, since the global algorithm is performing a vote, it can choose to predict only if that vote is sufficiently lopsided. However, we found that the algorithm makes a better tradeoff by performing the following slightly more detailed strategy. Each specialist's vote is split among the outcomes in its memory. For instance, if the last 5 times the specialist was "awake", three of the correct answers were Friday and two were Monday, then it gives $3/5$ of its vote to Friday and $2/5$ to Monday. Then the totals are tallied and a prediction is made only if the outcome with highest weight receives at least a specified fraction of the total vote. The advantage of this approach is that it allows individual specialists to claim differing degrees of confidence.

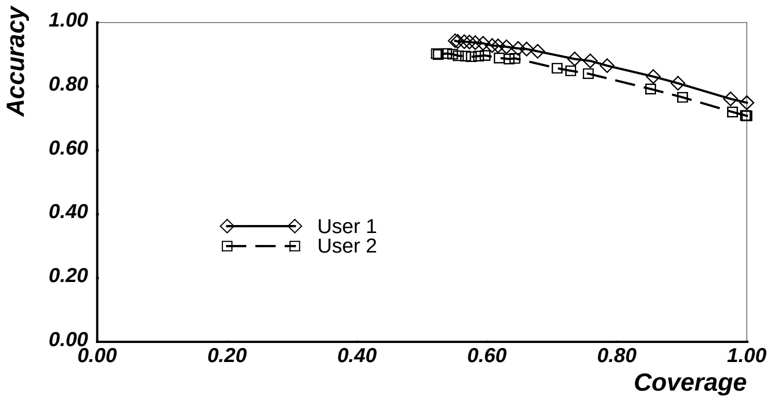

Figure 3. Location

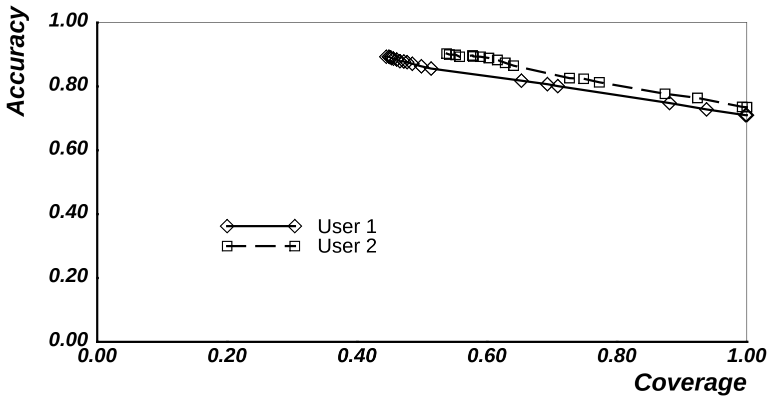

Figure 4. Duration

Modifying the algorithm in this way also does not significantly affect accuracy at the 100% coverage level.

Graphs showing the accuracy versus coverage tradeoffs achieved by Winnow are in Figures 3–6. "Coverage" is the fraction of examples on which a prediction is made, and "Accuracy" is the fraction correct on those examples. Strangely, for very high cutoffs, the performance in predicting day-of-week for User 1 decreases somewhat: the exact cause of this is unclear.

The performance of Winnow here compares quite well with that of CAP (Mitchell et al., 1994). For example, Winnow achieves 86% accuracy predicting duration for User 1 at 50% coverage, while CAP achieves a maximum of 74% accuracy in predicting duration

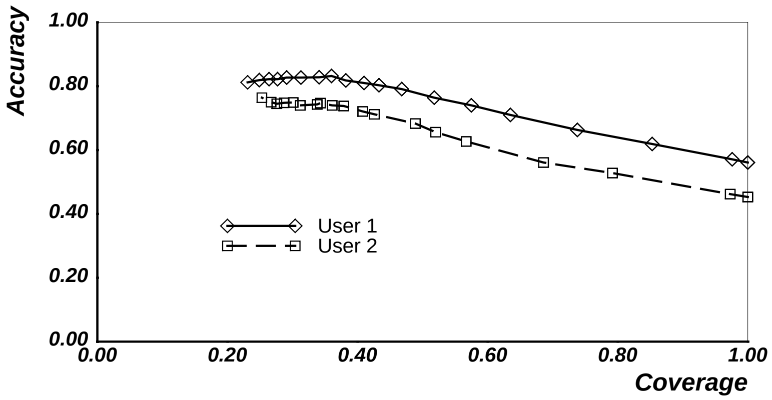

Figure 5. Day of week

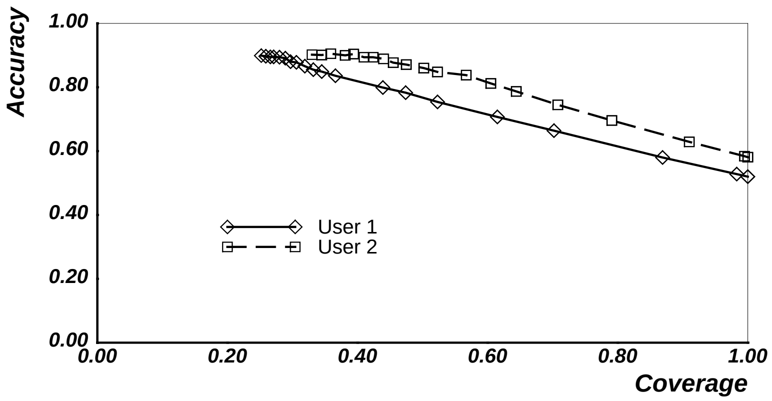

Figure 6. Start time

(and only about 70% accuracy at 50% coverage). In fact, Armstrong et al. (Armstrong et al., 1995) report similarly good accuracy-coverage performance using Winnow for a Web link-prediction task.

4.5. Pruning

The weighted-majority algorithm brings up the following exciting notion: by dropping poorly performing experts from consideration (neither asking for their predictions nor updating their weights), one can have an algorithm that speeds up as it learns more. This lies in contrast to the vast majority of algorithms that slow down as they learn.

The most natural method for pruning experts is simply to discard them when the ratio of their weight to the highest weight drops below some given threshold. In Figure 7, we plot for day-of-week prediction the number of experts remaining as a function of time for several different thresholds. Also given is the accuracy resulting from each policy. (Results for the other prediction tasks were similar.) Notice that pruning too aggressively may reduce effectiveness, but a wide range of policies exist that both do essentially no damage and prune quickly. As described in Section 5.1 below, the theoretical guarantees suggest using a cutoff that decreases polynomially with the number of experts. (This is the same as pruning when the number of mistakes made exceeds that of the current best expert by some quantity that depends *logarithmically* on the number of experts.)

5. Theoretical results

5.1. Pruning Weighted-Majority

In the standard worst-case analysis of weighted-majority (Littlestone and Warmuth, 1994; Cesa-Bianchi et al., 1993), pruning experts in the manner discussed in Section 4.5 is dangerous: an expert that currently has a low weight could potentially become important later. However, pruning *can* be justified under less pessimistic distributional assumptions. Specifically, suppose that examples are selected from a fixed distribution D and classified by some fixed target concept (as in the PAC model), and the experts are each fixed prediction algorithms. This means that the probability p_i that expert i predicts correctly on a random example is well-defined (though these p_i's need not be independent). In this case, one *can* provide theoretical guarantees. Of course in the calendar setting, each predictor is not a fixed function, but rather a learning algorithm itself. Nonetheless, the analysis in this model illuminates the type of behavior one can hope to expect.

We first need a small amount of notation. Let n be the number of experts and let opt be the (index of) the true optimal expert: the one with largest p_i. We will say that an expert i is *ϵ-optimal* if $p_i \geq p_{opt} - \epsilon$. Let us define the *observed optimal* expert to be the one that has made the fewest mistakes so far. We can now prove the following two simple theorems (the second is the more interesting one).

The first result, which follows immediately from Hoeffding bounds, states that a timid pruning policy is not likely to remove the true optimal expert.

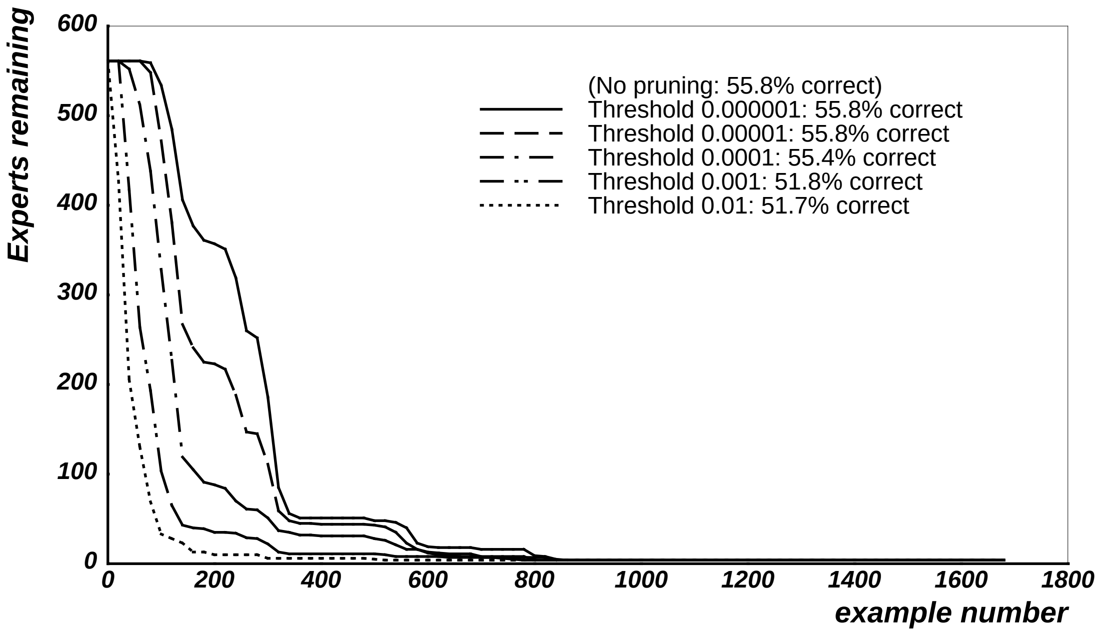

Figure 7. Pruning results for Weighted-Majority: number of unpruned experts as a function of time for day-of-week prediction using the large feature set (User 1).

THEOREM 1 *(Timid pruning) Consider the strategy that removes an expert if it has made at least* $\sqrt{2t\ln(2t^2n/\delta)}$ *mistakes more than the observed optimal, where t is the number of examples seen so far. Then with high probability* $(1-\delta)$*, the optimal expert will never be removed.*

Proof: Hoeffding bounds state that for a coin of bias p, the probability that after t coin flips the number of heads exceeds pt by more than $\sqrt{(t/2)\ln(1/\gamma)}$ is at most γ. The probability that the number of heads falls *below* pt by that amount is also at most γ.

Notice that to remove opt requires that at some time t, either opt has made at least $\frac{1}{2}\sqrt{2t\ln(2t^2n/\delta)}$ *more* mistakes than its expectation, or some other predictor has made that many *fewer* (or both). The probability that either of these events ever happens is at most

$$n\sum_{t=1}^{\infty}\delta/(2t^2n) \;=\; \delta/2\sum_{t=1}^{\infty}1/t^2 \;<\; \delta. \;\blacksquare$$

One can improve the dependence on t in the above bound somewhat by arguing as in the proof of the law of the iterated logarithm (Feller, 1968).

The second theorem states that with a more aggressive policy, even though one may remove the optimal expert, one still can ensure that with high probability at least some near-optimal expert remains. Notice that the number of examples t does not appear in this bound, and the dependence on the number of experts n is logarithmic. This motivates using a rule that discards experts when their weight drops to less than $1/p(n)$ times the maximum, for some polynomial p.

THEOREM 2 *(Aggressive pruning) Let* $\epsilon < 1/2$ *and consider the strategy that removes an expert if it has made at least* $(4/\epsilon)\ln[16n/(\epsilon^4\delta)]$ *more mistakes than the observed optimal. Then with high probability* $(1-\delta)$*, we will never remove all* ϵ*-optimal experts.*

Proof: We begin by showing that with probability at least $1-\delta/2$, opt will not be removed at any time $t \leq (4/\epsilon^2)\ln[\frac{16n}{\epsilon^4\delta}]$. This follows directly from Theorem 1 since in this range of times t, the quantity $\sqrt{2t\ln(4t^2n/\delta)}$ (taken from Theorem 1 but using $\delta/2$ instead of δ) is less than $(4/\epsilon)\ln[\frac{16n}{\epsilon^4\delta}]$, so long as $\epsilon < 1/2$.

Now consider the larger times t. We show that in this range, with probability at least $1-\delta/2$, none of the non-ϵ-optimal experts (henceforth called "bad" experts) has fewer mistakes than opt. This means that if opt is ever removed, then so are all the bad experts, proving the theorem. This again will follow from Hoeffding bounds.

Specifically, in this range of times t, we have $\epsilon t/2 \geq \sqrt{(t/2)\ln(4t^2n/\delta)}$. (This can be seen by first squaring, rearranging, and exponentiating to produce the inequality $e^{\epsilon^2 t/2} \geq 4t^2n/\delta$, then noticing that it suffices to verify the inequality using the specific value of $t = (4/\epsilon^2)\ln(\frac{16n}{\epsilon^4\delta})$, and finally using the fact that $x = \frac{16n}{\epsilon^4\delta}$ satisfies $x > 4(\ln x)^2$ for $\epsilon < 1/2$.) Thus, for any fixed time t in this range, the chance that either opt makes $\epsilon t/2$ mistakes *more* than its expectation, or some other expert makes $\epsilon t/2$ mistakes *less* than its expectation is at most $\delta/4t^2$. Notice that the expected number of mistakes by opt differs from the expected number of mistakes by any bad expert by at least ϵt, so for the procedure to fail, one of these two events must happen. Thus, summing over all t we have that with probability at least $1 - \delta/2$, opt never makes more mistakes than any of the bad experts. □

5.2. Mistake bounds for Winnow

We describe here an analysis of our winnow variant, which for convenience we refer to as Winnow-Specialist, in the following "infinite-attribute model" setting.

The setting is that we have some large (possibly infinite) set of *specialists*: predictors that on each example may either predict or abstain. Out of these specialists, r are infallible: whenever one of those r predicts it is always correct. In addition, on each example we have the following two guarantees: (1) at most n specialists make a prediction, and (2) at least one of those is one of the r infallible specialists. We call the r infallible specialists *relevant* and the others *irrelevant*.

In the calendar domain, this corresponds to thinking of the classification as being given by a list of r rules like:

If `event-type` = `meeting`, and `attendees` = `Joe`, then my office.

If `event-type` = `meeting`, and `attendees` = `the-dean`, then dean's office.

If `attendee-type` = `funder`,
and `single-attendee` = `yes`, then my office.

...

and the proviso that each example makes at least one rule fire, and that we never will see an example which makes two inconsistent rules fire. The quantity n is the number of feature-pairs. Alternatively, this setting can be viewed as assuming that each output corresponds to some disjunction of boolean variables, where the total number of relevant boolean variables is r, and we are guaranteed that each example satisfies exactly one of the disjunctions.

The algorithm Winnow-Specialist is as described in Section 3.2: Specialists have their weight initialized to 1 when they first predict. After that, they have their weight cut in half when they make a mistake, and their weight is multiplied by $3/2$ when they predict correctly *and* the global algorithm makes a mistake. The analysis is essentially the same as that for the standard Winnow algorithm, but one slight complication is that since we initialize specialists even when the global algorithm *does not* make a mistake, we cannot bound the total weight. For instance, many examples might pass in which no mistake is made, and yet a large number of initializations occur. Instead, we will bound the weight on specialists whose weight is greater than 1. (Alternatively, we could have modified the algorithm to only initialize new specialists when a mistake is made.)

THEOREM 3 *The* Winnow-Specialist *algorithm makes at most* $2r\log_{3/2}(3n)$ *mistakes under the conditions described above.*

Proof: Define a specialist to have "high weight" if its weight is greater than 1. Define $W_{high-irrel}$ to be the total weight on high-weight irrelevant specialists.

Consider what happens when the global algorithm makes a mistake. Let W_a be the total weight on high-weight specialists that predicted *incorrectly*. So, the total weight predicting incorrectly is at most $W_a + n$. Let W_b be the total weight on the irrelevant specialists that predicted *correctly* (whether their weight is high or not) and let W_r be the weight on the relevant specialists that predicted (and must have predicted correctly by definition). So, the total weight predicting correctly is $W_b + W_r$. Since the global algorithm made a mistake, this means that $W_b + W_r \leq W_a + n$. Now, notice that $W_{high-irrel}$ increases by *at most* $W_b/2 - W_a/2$, since the irrelevant specialists whose weights are multiplied by $3/2$ are exactly those included in W_b, and the specialists included in W_a (which all were high-weight) have their weights multiplied by $1/2$. Thus, the increase in $W_{high-irrel}$ due to this mistake is at most

$$W_b/2 - W_a/2 \leq \frac{1}{2}(n - W_r). \tag{1}$$

Now, consider mistakes in which $W_r < 2n$. There can be at most $r + r\log_{3/2}(2n)$ of these, since after that many such mistakes, all r relevant specialists will have weight at least $2n$. So, by Equation 1, these mistakes together add at most $\frac{1}{2}[nr + nr\log_{3/2}(2n)]$ to $W_{high-irrel}$.

Now consider mistakes that occur when $W_r \geq 2n$. Each of these removes at least $n/2$ from $W_{high-irrel}$. So at most $r + r\log_{3/2}(2n)$ mistakes of this sort can be made as well. Thus the total number of mistakes is at most $2r + 2r\log_{3/2}(2n) = 2r\log_{3/2}(3n)$. □

In fact, similar reasoning to the above shows that (as for the standard Winnow algorithm) Winnow-Specialist is fairly robust to noise. For instance, if a relevant specialist predicts incorrectly on an example, the result is that its weight will be cut in half, perhaps producing two more mistakes of the "$W_r < 2n$" form while its weight recoups, which in turn may produce up to two more mistakes of the "$W_r \geq 2n$" form.

6. Summary and conclusions

Winnow and Weighted-Majority are learning algorithms whose forte is in "learning simple things really well". If the learning setting has the property that one can hope to predict well by using a collection of simple rules, these algorithms have advantages of being fast and incremental, of being able to quickly focus on relevant features (Littlestone, 1988), of adapting well to target concepts that change with time, and (at least in the instance discussed here) having reasonable accuracy/coverage tradeoffs.

Several factors seemed to be influential in producing the good performance of these algorithms on the CAP data. One is that in the Winnow and Weighted Majority algorithms implemented, the question of how to discount old data in favor of new data is handled in a distributed manner. Most algorithms view examples in a monolithic, centralized way. For instance, CAP provides to its learning algorithm a fixed window of the past 180 examples. A disadvantage with this approach is that one is likely to forget rare but useful events, and yet not adapt quickly enough to small changes. In the algorithms used here, however, the memory is kept at the level of the experts/specialists. For instance, the "`event-type =`

`seminar` and `seminar-type = AI seminar`" specialist stores the 5 most recent AI seminars. Storing information at this more "local" level allows for recent events to take precedence over previous ones, while at the same time remembering the rare cases. For a frequent event, we benefit because only a few examples of a new outcome are needed to change behavior (e.g., the seminar is moved from Monday to Friday). For an infrequent event, we benefit as well because the associated specialist will still remember where and when that previous event occurred, no matter how long ago it was. Note that although there is a fair amount of "concept drift", the fact that pruning experts in the Weighted Majority algorithm worked so well suggests that in this domain, the type of drift that occurs is not one in which the set of "most important features" changes, but rather the change is in what one should do with those features. For instance, seminar-name is an important feature across semesters, but the start time and location of a given seminar may change with time.

A second factor influencing the performance of these algorithms is that the weighting scheme allows one to quickly focus in on a good set of prediction rules. As described in Table 4, performance drops substantially when the weighting mechanism is disabled. Moreover, in the Winnow algorithm it is important that the weights represent not the individual "percentage correct" figures for each specialist, but rather the extent to which the specialist is helpful in combination with the other specialists available. As mentioned in Section 3.2, if weights are increased for specialists even when the global algorithm is predicting correctly, performance drops significantly (though not quite as much as when the weights are completely disabled). Furthermore, the multiplicative weighting scheme allows the algorithms to perform well even when a large number of features are available without needing a separate feature-selection algorithm as in CAP. This ability to focus should be even more useful if the learning system itself is given the ability to probe the world for features (e.g., by fingering users, scanning the net, etc.).

In summary, this paper has demonstrated that Winnow and Weighted Majority can be useful algorithms in practice and has discussed some of the issues involved in their implementation. These are algorithms especially worth considering in situations where one suspects that good performance can be achieved by hypotheses of a relatively simple form but where other factors such as concept drift, noise, a need to be fast and incremental, and an abundance of features complicate the learning task.

Acknowledgements

I would like to thank Prasad Chalasani for a great deal of help in the early stages of this work, and the members of the CAP project, especially Rich Caruana, Dayne Freitag, Tom Mitchell, and David Zabowski (Stork), for helpful discussions and for making their data available.

This work was supported in part by NSF National Young Investigator grant CCR-9357793, by a Sloan Foundation Research Fellowship, and by ARPA under grant F33615-93-1-1330. The views and conclusions in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of ARPA, NSF, the Sloan Foundation, or the U.S. Government.

Data and source code

Data and source code are available via

`http://www.cs.cmu.edu/afs/cs/usr/avrim/Calendar/info.html`

Notes

1. Visit http://www.cs.cmu.edu/afs/cs/project/theo-5/www/cap-data.html.
2. Of course there is no true target concept — just labeled examples. Yet the notion of a target concept is always a convenient fiction.
3. Clearly one could implement this strategy using, say, triples of conditions; experimentally, in this domain using triples did not help much, and pairs performed much better than using just single conditions.
4. In the Weighted Majority algorithm, each expert is required to make a prediction on each example. So, if an example is seen that does not match anything in an expert's truth table, the expert just predicts a global default value. In the Winnow algorithm, there is no need to predict a default unless *none* of the specialists predict, which never happens in the two data sets (except for the first example). Running both algorithms with weights fixed to 1 on this data reveal no significant differences in performance between them.

References

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. In *1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments*.

Blum, A. (1992). Learning boolean functions in an infinite attribute space. *Machine Learning*, 9:373–386.

Blum, A., Hellerstein, L., and Littlestone, N. (1991). Learning in the presence of finitely or infinitely many irrelevant attributes. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 157–166, Santa Cruz, California. Morgan Kaufmann.

Caruana, R. and Freitag, D. (1994). Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*.

Cesa-Bianchi, N., Freund, Y., Helmbold, D., Haussler, D., Schapire, R., and Warmuth, M. (1993). How to use expert advice. In *Proceedings of the Annual ACM Symp. on the Theory of Computing*, pages 382–391.

Dent, L., Boticario, J., McDermott, J., Mitchell, T., and Zabowski, D. (1992). A personal learning apprentice. In *Proceedings of the 1992 National Conference on Artificial Intelligence*.

DeSantis, A., Markowsky, G., and Wegman, M. (1988). Learning probabilistic prediction functions. In *Proceedings of the* 29*th IEEE Symposium on Foundations of Computer Science*, pages 110–119.

Feller, W. (1968). *An Introduction to Probability and its Applications*, volume 1. John Wiley and Sons, third edition.

Jourdan, J., Dent, L., McDermott, J., and Zabowski, D. (1991). Interfaces that learn: A learning apprentice for calendar management. Technical Report CMU-CS-91-135, Carnegie Mellon University.

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

Littlestone, N. (1989). *Mistake bounds and logarithmic linear-threshold learning algorithms*. PhD thesis, U. C. Santa Cruz.

Littlestone, N. (1991). 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, Santa Cruz, California. Morgan Kaufmann.

Littlestone, N. and Warmuth, M. K. (1994). The weighted majority algorithm. *Information and Computation*, 108(2):212–261.

Mitchell, T., Caruana, R., Freitag, D., McDermott, J., and Zabowski, D. (1994). Experience with a personal learning assistant. *CACM*, 37(7):81–91.