# Active Learning using On-line Algorithms

Chris Mesterharm
Rutgers Computer Science Department
110 Frelinghuysen Road
Piscataway, NJ 08854, USA
mesterha@cs.rutgers.edu

Michael J. Pazzani
Rutgers Computer Science Department
110 Frelinghuysen Road
Piscataway, NJ 08854, USA
pazzani@rutgers.edu

## ABSTRACT

This paper describes a new technique and analysis for using on-line learning algorithms to solve active learning problems. Our algorithm is called Active Vote, and it works by actively selecting instances that force several perturbed copies of an on-line algorithm to make mistakes. The main intuition for our result is based on the fact that the number of mistakes made by the optimal on-line algorithm is a lower bound on the number of labels needed for active learning. We provide performance bounds for Active Vote in both a batch and on-line model of active learning. These performance bounds depend on the algorithm having a set of unlabeled instances in which the various perturbed on-line algorithms disagree. The motivating application for Active Vote is an Internet advertisement rating program. We conduct experiments using data collected for this advertisement problem along with experiments using standard datasets. We show Active Vote can achieve an order of magnitude decrease in the number of labeled instances over various passive learning algorithms such as Support Vector Machines.

## Categories and Subject Descriptors

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

## General Terms

Algorithms, Design, Performance, Theory

## 1. INTRODUCTION

In this paper, we study a type of inductive learning called active learning [28]. The model of induction we use is based on a set of instances $X$ where each instance $x \in X$ is given a label $y \in \{0, 1\}$. Our goal is to learn a function that correctly assigns labels to a large number of instances in $X$ using a small number of labeled instances. In active learning, we get labeled instances by selecting from a set $\hat{X} \subset X$. The labels for these selected instances can be determined by various procedures such as consulting a human expert. We assume there is a cost for each label.

We propose an active learning algorithm that works by combining several copies of an on-line learning algorithm. Our modification works with any on-line learning algorithm, but it benefits most by using an on-line algorithm that has a worst-case upper-bound on the number of mistakes. The main intuition of our algorithm is to actively select instances to force the on-line algorithms to make mistakes. If we can force enough mistakes then the on-line algorithms must learn the concept. What is more likely is that the algorithm will reach a point where it can no longer guarantee mistakes. In practice, at this point the algorithm is close to a good concept, but either way, the algorithm reverts to passive learning and does not suffer a penalty.

Previous inexpensive active learning algorithms such as uncertainty sampling with a probabilistic classifier [21] and Active Majority [22] often show an order of magnitude decrease in the number of labels required over passive learning but lack strong theoretical justification. More recent work has focused on strengthening the theory of active learning [11, 3, 12, 6, 18]. While these results are encouraging, the algorithms are often expensive and they do not have the dramatic decreases in label costs seen on practical problems by earlier algorithms. In this paper, we start with an efficient well-performing algorithm and attempt to better understand why it works in order to improve its performance and theoretical justification.

Our algorithm is strongly related to the Active Majority technique of Liere and Tadapalli [22]. In this paper, we present a new analysis that explains why Active Majority works, and based on that understanding, we present refinements to improve its performance. We call this new technique Active Vote. Active Vote can modify any on-line algorithm for active learning.

The motivating application for this research is to provide a content owner more control over the advertisements served on their website. These advertisements are typically served by an ad network such as Google's Adsense for content. While these ad networks have proven to be profitable, they sometimes place ads that the content owners believe their readers might find inappropriate. In this paper, we present an application that allows a content owner to provide feedback on the ads that appear on their website and uses that information to learn a classifier that is used to block ads that are deemed inappropriate. Active learning is important for this problem because most content owners will not take the time to provide feedback on thousands of ads.


*KDD'11,* August 21–24, 2011, San Diego, California, USA.

The paper is organized as follows. The next section defines different types of active learning. Section 2 motivates the use of on-line algorithms for active learning by describing how they relate in an exact model of learning. The main motivation is based on the fact that the number of mistakes made by the optimal on-line learning is a lower-bound for the number of label requests needed by any active learning algorithm. Section 4 describes our active learning technique, Active Vote, and Section 5 gives performance bounds for Active Vote. In Section 6, we give information on our Internet advertisement application, and Section 7 gives our experimental results. We perform experiments on four datasets including the previously mentioned Internet advertisement domain.

# 2. ACTIVE LEARNING MODELS

In this section, we explain several different types of active learning. We divide active learning models into two groups: batch learning and on-line learning. This section also reviews information on batch and on-line learning.

## 2.1 Batch Models

The most common type of active learning deals with batch learning. Batch active learning assumes that the learning algorithm has access to a large set of unlabeled instances. The algorithm requests labels for some small subset of these instances and uses this information to learn a hypothesis. The goal is that this hypothesis predicts the correct label for future instances. To make this problem solvable, one typically assumes that all the instances are drawn from the same distribution and that there are some restrictions on the form of this distribution. The goal of the algorithm is to maximize the accuracy of the learned hypothesis.

A common restriction on active batch learning is that the learning algorithm is presented the instances in a stream. To keep the algorithm efficient, the algorithm must decide as soon as it sees the instance whether or not it wants to request the label. If it does not request the label, then the instance is skipped. This means the algorithm does not have to keep track of large numbers of unlabeled instances. We instead focus on the situation where the algorithm is allowed to select the best instance from the set of instances, $\hat{X}$. In this way the algorithm can further optimize its choice. If there is a limit to the number of instances the algorithm can store or process, then one can easily convert to a stream algorithm by selecting the best instance in some fixed number of observed instances.

Another possible active batch model is based on transduction. This is the same as the previous active batch models except that the algorithm is evaluated on the remaining instances in $\hat{X}$. This is an interesting active learning model when $\hat{X}$ is finite because the learning algorithm is motivated to find difficult instances not just to learn the concept but also to remove them from evaluation. While we feel our techniques are applicable to this problem, we do not directly address transduction in this paper.

## 2.2 On-line Models

On-line learning is best understood by viewing the learning problem as a sequence of instances. The learning algorithm is presented an instance and the algorithm must predict the label. Soon after prediction, the algorithm receives the label and uses this information to update its prediction hypothesis. The goal of the algorithm is to minimize the number of incorrect predictions. Many on-line learning problems are based on predicting the future. For example, with the stock market the label is readily available after the market has closed.

For on-line active learning, we consider two possible models. The first model is based on bootstrapping. By this we mean that we have an early phase of learning where we try to learn as much as possible and we do not care about mistakes. Afterward we switch to on-line learning where we are penalized for mistakes. As an example, consider preallocation of resources for computer program scheduling. We don't know the resources until after executing the program, but we could set up a bootstrap period in which we test several programs. It would be useful to use active learning during the bootstrap to maximize how much we learn. Later, we shift to traditional on-line learning where we want to minimize mistakes. This is the on-line model that we consider in this paper.

The second model assumes that all labels have some cost. Therefore, the on-line algorithm must request labels to perform updates. To minimize costs the algorithm can actively learn by not requesting every label. However, the algorithm is always penalized for mistakes even if it does not request the label [9]. We do not consider this model in this paper because we are primarily interested in cases where we are not penalized for mistakes during active learning. However, this model can be extended to benefit from a bootstrap procedure as previously explained.

# 3. EXACT LEARNING

The main idea of this paper is to take an on-line algorithm and to convert it into an active algorithm by attempting to select instances that cause mistakes. In order to motivate this idea, we use the exact model of learning to show how on-line learning and active learning are related.

Exact learning assumes there is a finite set of hypotheses $H$ such that one hypothesis $h_{\text{opt}} \in H$ predicts all the instances in $X$ correctly. The hypotheses in $H$ are all distinct with respect to the instances in $X$. For on-line learning, an adversary selects the instances, and the goal of the algorithm is to minimize the number of mistakes it makes on this sequence of instances. For active learning the algorithm is allowed to ask for the label of any instance in $X$, and the goal of the algorithm is to find $h_{\text{opt}}$ using a minimal number of label requests. In this framework, the label requests are called membership queries [1].

Let $M(O)$ be the worst-case number of mistakes made by any on-line algorithm $O$ on all possible sequences, and let $O_{\text{opt}}$ be the optimal algorithm. Let $Q$ be any deterministic membership query algorithm that never queries the same instance more than once and returns when only a single hypothesis is consistent with all previous queries. Let $L(Q)$ be the maximum number of membership queries made by $Q$. Lastly, let $VC(H)$ be the VC dimension of the set of concepts [29]. The following bound gives a relation between active and on-line learning in the exact model [1].

$$VC(H) \leq M(O_{\text{opt}}) \leq \lg(|H|) \leq L(A) \leq |\hat{X}|. \qquad (1)$$

While dealing with the limited exact learning model, the previous bounds capture the main intuition of this paper. If we have a good on-line algorithm and we can force it to make mistakes on a significant number of the actively

selected instances, then this procedure should be effective for active learning.

Unfortunately, this will not always work. The easiest example is the singleton set [23, 1]. For this set of concepts, only a single instance in $X$ is labeled 1. The optimal on-line algorithm always predicts 0 until it gets the singleton instance. The algorithm might see a large number of instances, but it can only make one mistake. The membership query algorithm is allowed to request the instances, but it has no idea where to look. It makes $|X|$ label requests in the worst case.

Singleton sets are one of the reasons membership query learning is difficult. In essence, if only a small, arbitrary portion of the space has a specific label value, then the algorithm must, in the worst case, get labels for most of the instances to find a few instances. It is a needle in a haystack problem. We will return to this problem in Section 5.3.

# 4. ACTIVE LEARNING ALGORITHM

A large motivation for early research in active learning is the Query by Committee algorithm [16]. This algorithm keeps track of the version space of all consistent hypotheses. The Query by Committee algorithm samples two hypotheses from the version space and requests the label for any instance where the hypotheses do not agree. It then updates the version space based on the requested label.

Various authors have expanded on this research with the goal of making the algorithm more tractable. Our algorithm is most similar to the algorithm Active Majority [22]. Instead of storing the whole hypothesis space, Active Majority stores $k$ copies of an on-line algorithm. The algorithm receives a stream of unlabeled instances. For each instance, the algorithm randomly selects two of the hypotheses and requests the label if the hypotheses disagree. All $k$ algorithms use this labeled instance to update their hypothesis. After training, the algorithm makes predictions by taking the majority vote of the $k$ hypotheses.

We take a very similar approach but with a different motivation. The $k$ hypotheses stored by Active Majority do not approximate the version space. Instead they hold $k$ hypotheses that are hopefully converging to the optimal hypothesis in different ways. If those hypotheses make different predictions on some of the unlabeled instances, then we can use those instances to force mistakes. As long as the on-line algorithms do not depend on distributional assumptions for learning, they can learn even when the instances do not come from a distribution but are instead actively queried. This lack of distributional dependence is common for many on-line algorithms because they are designed to perform well even when an adversary is generating instances [25].

Just like Active Majority, our algorithm Active Vote can modify any on-line algorithm to the active setting. We give the on-line algorithm the prefix Active Voting (AV) to denote that the modification is being used. The pseudo code for the modification is given in Figure 1.

There are three main differences between Active Vote and Active Majority. When there is a disagreement between some of the $k$ hypotheses, Active Vote always uses an instance that it can guarantee causes the most mistakes. The best case is when $\lfloor k/2 \rfloor$ of the hypotheses predict one label. We call the set of instances that correspond to this best case the confusion set. The motivation for the algorithm is that anytime the label is requested for an element

**Active Vote**$(A, k, u, r, B, R)$
**Initialization**
Algorithm $A$ predicts labels in $\{-1, 1\}$
Create copies $A_1$ to $A_k$ of algorithm $A$.
Perturb each copy with $u$ random updates.
The integer $r$ partially controls random queries.
Instances in $B$ are for active queries.
Instances in $R$ are for random queries.
Set $t \leftarrow 0$.
**Active Learning**
**Select Instance**
If $t$ mod $r$ is 0 then
Select and remove $x$ from $R$.
Else let $x' = \arg\min_{x \in B} D(x) = |\sum_{i=1}^{k} A_i(x)|$.
If $D(x') = k$ then
Select and remove $x$ from $R$.
Else let $x \leftarrow x'$
Select and remove $x$ from $B$.
Request label $y$ for $x$.
**Update**
Update $A_1(x, y)$ to $A_k(x, y)$.
$t \leftarrow t + 1$.

**Figure 1: Pseudo-code for Active Vote.**

of the confusion set, a mistake must occur in at least $\lfloor k/2 \rfloor$ of the $k$ hypotheses. Active Majority is different because the instances it selects are not guaranteed to maximize the number of mistakes.

The second difference is based on what happens when there are no more unlabeled instances that are guaranteed to cause mistakes. The Active Majority algorithm does not consider this situation most likely because it is specified as an instance stream algorithm. This can cause problems for highly label skewed problems because it might take some time to converge on the area of the input space where the concept is located. Early in the learning, all the hypotheses might converge to predicting the majority label, and Active Majority will not request any additional labels.

Active Vote is designed to select unlabeled instances from a pool; however, if all the hypotheses agree, we do not want to randomly select instances from this same pool. The remaining instances are biased and are no longer a random sample. Therefore we keep an active pool, $B$, for active instances and a random pool, $R$, for random instances. After we query an instance, we permanently remove it. If we know ahead of time that the label budget is $c$ labels, then we can randomly select $c$ instances for the random pool. Furthermore, if we find at some point we cannot use all the instances in the random pool, we can move them to the active pool.

There are advantages to sometimes using random queries even when there are unlabeled instances that would cause mistakes. It is difficult for a user of the active learning program to have confidence in the results without some type of feedback. A small random sample might not give high statistical confidence of the accuracy, but it can give enough information to show the user that the algorithm is achieving useful accuracy. It is also possible that the user might require a better accuracy estimate. While this can cost the algorithm a large number of labels, it can still be significantly less than the number of labels needed for passive learning. We discuss this issue in Section 5. Parameter $r$ is used to

control these random queries. The algorithm performs a random query once every $r$ queries.

In order for Active Majority or Active Vote to work, the initial hypotheses of the $k$ algorithms must be different. Active Majority uses a technique that randomizes the weights of hyperplanes at the start of learning. We randomize in a more general way that can be applied to any on-line algorithm that is tolerant to noise. We force the algorithm to update on a small number of randomly generated instances. Since these can be considered noisy instances, the random updates do not have a large effect on the mistake bound.

The final difference is based on what the algorithms return after the active learning is complete. Active Majority just returns the $k$ hypotheses that are used in a batch setting to make predictions with a majority vote. Active Vote does not specify what it returns as it depends on the problem. In principal, it returns the $k$ algorithms and their history of hypotheses and instances; however, typically the algorithm does not need to store all that information. Section 5 and Section 7 give some specific examples.

Lastly, we want to address the cost of Active Vote. Let $C$ be the confusion set. A straightforward search of the active pool, $B$, to find the next instance to query would take $k(|B| - |C|)$ predictions. As long as $C$ is not too small, this can be significantly improved by a simple randomized search. If $C$ is empty, then the algorithm must make $k|B|$ predictions to find the next instance. However, if after a query from the random pool, $R$, none of the hypotheses has changed, then Active Vote can just proceed to take another query from $R$. This is typical on many problems because by the time the confusion set is empty the algorithm's $k$ hypotheses all have high accuracy. The cost of the predictions and the updates of the $k$ hypotheses depend on the choice of the on-line algorithm. The on-line algorithm that we consider in this paper is optimal in that its predictions and updates asymptotically take no more time than the time needed to read an instance. We discuss our choice for the on-line algorithm in Section 7.1. We also give information on features and modifications that combine beneficially with Active Vote.

# 5. BOUNDS

In this section, we give bounds on the performance of Active Vote. All of these bounds depend on keeping the confusion set non-empty. We start with a simple way to bound the number of mistakes made by at least one of the basic algorithm copies used by Active Vote. Let $S$ be the sequence of previous instance queries. Let $I_C$ be the subsequence of indexes that are queried from the confusion set, let $I_O$ be the other active indexes, and let $I_R$ be the subsequence of indexes sampled from the random pool. For example, if Active Vote performs the following sequence of queries: random, confusion, random, other, confusion, random, then $I_C = (2, 5)$, $I_O = (4)$, and $I_R = (1, 3, 6)$.

*Lemma 1.* The *AV-A* algorithm has at least one basic algorithm $A_i$ that makes at least $\lfloor \frac{k}{2} \rfloor \frac{|I_C|}{k}$ mistakes.

PROOF. Assume each of the basic algorithms makes less than $\lfloor \frac{k}{2} \rfloor \frac{|I_C|}{k}$ mistakes. Since there are $k$ basic algorithms there must be less than $|I_C| \lfloor \frac{k}{2} \rfloor$ total mistakes summed over all basic algorithms. This is a contradiction because every update from the confusion set causes at least $\lfloor \frac{k}{2} \rfloor$ basic algorithm mistakes and there are $|I_C|$ confusion set updates. □

## 5.1 On-line Bootstrap

Our next result applies to using Active Vote for the on-line bootstrapping problem. The Active Vote algorithm is only used during the bootstrap phase of learning. After the bootstrap is finished, we return the basic algorithm that has made the most mistakes. This algorithm is used for the remaining on-line trials. The idea behind this result is to simply subtract the mistakes during the bootstrap phase of learning from total mistakes the algorithm can make. See Section 2.2 for a description of the on-line bootstrap problem.

*Theorem 1.* When using Active Vote for bootstrap on-line learning, during the on-line phase of learning, the number of mistakes is at most $M(A) - \lfloor \frac{k}{2} \rfloor \frac{|I_C|}{k}$.

PROOF. Based on Lemma 1, at least $\lfloor \frac{k}{2} \rfloor \frac{|I_C|}{k}$ mistakes occur during the bootstrap. This leaves $M(A) - \lfloor \frac{k}{2} \rfloor \frac{|I_C|}{k}$ mistakes to occur during the on-line trials. □

The optimal bound for this theorem is achieved by any even $k$ value. Choosing a specific value of $k$ will depend on the learning problem. This requires giving a more precise definition of the bootstrap model. For example, when dealing with an adversary, how should one model the set of bootstrap instances? We plan to address this issue in future research.

## 5.2 Batch

For batch data, we assume the instances are independently sampled from a fixed distribution. First, we give a slight modification of a result by Cesa-Bianci *et al.* that bounds the average accuracy of the hypotheses an on-line algorithm learns when instances are sampled from a fixed distribution.

Notice that we are not bounding the accuracy of the final hypothesis. We are bounding the average accuracy of a sequence of hypotheses. This is often necessary when using on-line algorithms. Just because an on-line algorithm has a good upper-bound on mistakes does not guarantee that the final hypotheses of the algorithm will have a low error rate. Theoretically and practically, if one wants to use an on-line algorithm in a batch setting, it is important to use a technique that generates accurate hypothesis [24, 8]. We will give more details on this issue in the experimental section of the paper.

In this lemma, we introduce some new notation. Let $M(A, S, I)$ represent the number of mistakes algorithm $A$ makes when run on sequence $S$ if we only count mistakes for instances indexed by index set $I$.

*Lemma 2.* When running algorithm $A$ with instance sequence $S$, for any $\delta \in (0, 1]$,

$$\Pr\left(\sum_{i \in I_R} \frac{err_D(h_i)}{|I_R|} \geq \frac{M(A, S, I_R)}{|I_R|} + \sqrt{\frac{2 \ln\left(\frac{1}{\delta}\right)}{|I_R|}}\right) \leq \delta.$$

PROOF. We use a result of Cesa-Bianci *et al.* [8] but apply it to a subsequence of the hypotheses instead of all the hypotheses. The original result is an application of the Hoeffding-Azuma inequality. □

Next we apply Lemma 2 to upper-bound the average error of certain hypotheses used in Active Vote. The hypotheses we bound are from the basic algorithm that has made the

most mistakes during the previous trials. We only bound the hypotheses queried from the random pool, $R$.

*Theorem 2.* When running *AV-A*, let $h_i$ be the hypotheses generated by the basic algorithm that has made the most mistakes during random queries. For any $\delta \in (0, 1]$,

$$\Pr\left(\sum_{i \in I_R} \frac{err_D(h_i)}{|I_R|} \geq \frac{M(A) - \left\lfloor \frac{k}{2} \right\rfloor \frac{|I_C|}{k}}{|I_R|} + \sqrt{\frac{2 \ln\left(\frac{k}{\delta}\right)}{|I_R|}}\right) \leq \delta.$$

PROOF. We apply Lemma 2 with $\delta' = \delta/k$ to all $k$ algorithms run by *AV-A*. Using a union bound guarantees that the bound holds on all $k$ algorithms with probability $\delta$. Next, we use Lemma 1 to show that $M(A, S, I_R) \leq M(A) - \lfloor k/2 \rfloor |I_C|/k$ for the basic algorithm that made the most mistakes. □

Because we rarely know enough about the learning problem to generate upper-bounds on mistakes, the bound in Theorem 2 is more for guidance. Constraints on the learning problem can lead to choices for the algorithm to use with Active Vote. For actual feedback on performance, we can use the data dependent bound from Lemma 2. The data dependent bounds also motivate a change in the algorithm. As long as there are sufficient random samples, Active Vote should return the basic algorithm that made the fewest mistakes on instances selected from the random pool. The bound in Lemma 2 can be used to determine what constitutes sufficient.

## 5.3 Confusion Set

All the bounds in this section rely on keeping instances in the confusion set. Unfortunately, it is not always possible to keep the confusion set non-empty. As mentioned in Section 3, there are sets of concepts, such as the singleton in exact learning, that have small upper-bounds on mistakes yet require membership queries to request labels for every instance. If we could keep the confusion set non-empty for a constant fraction of the label queries, then we would break these lower-bounds.

When dealing with active learning and a distribution, things are not much better. In this setting, the goal is to create an algorithm that with $1-\delta$ probability returns a hypothesis that is $\epsilon$ close to the best hypothesis the algorithm can represent. The lower bounds for active learning and passive learning are essentially the same for both of these problems [5]. Therefore, in the worst-case active learning cannot help.

However, in practice, active learning is often effective. Many practical problems have a reasonable label split that removes the need to worry about the singleton set. In cases with large label skew, techniques to actively search for the minority label can be effective [2]. Also, as pointed out in [4], the lower-bounds on active learning for many concept classes are non-uniform in that they depend on changing the concept as $\epsilon$ goes to zero. A more careful analysis gives a much more favorable result for many popular concept classes.

The essence of the confusion set is to have a small number of algorithms that all converge in different ways to the target concept. As long as their disagreement contains unlabeled instances, then, on average, the algorithms can make rapid progress to the target function. However, one of the advantages of this active learning technique is that it does not degrade if the confusion set is empty. Instead it just reverts to random queries. Therefore, if one does encounter a difficult active learning problem, then, just as the active learning lower-bounds imply, the algorithm has an upper-bound based on passive learning.

# 6. INTERNET AD APPLICATION

Our motivation for researching active learning is an application for improving Internet advertisement. We are interested in the advertisements content owners (newspapers, blogs, etc...) place on their websites using an advertising network such Google's AdWords. While these ad networks have proven to be profitable, they sometimes place ads that the content owners might find inappropriate. Examples of inappropriate ads we have seen include the following.

- A web site for children's pet rabbits featuring an ad for the Rabbit personal massager.
- An advertisement for a product to get rid of woodpeckers on a bird watching site.
- An ad selling home tanning beds on a news web site discussing tanning beds as a cause of skin cancer.

To give an illustrative example, new-jersey-birds.com is a web site devoted to bird watching in new jersey and includes information and photographs of many birds, including woodpeckers. Due to simple keyword matching and bidding [14], ads might appear on the website such that describe products to get rid of woodpeckers while the web site is intended for those more interested in attracting woodpeckers. We worked with the owners of new-jersey-birds.com to alternate between two text ads. One ad read "Get Rid of Woodpeckers, Products, reviews and info on getting rid of woodpeckers. Free shipping at amazon.com." The other was identical but replaced "Get Rid of" with "Attract". Out of 4235 impression to get rid of woodpeckers only 2 were clicked while out of 4188 impressions to attract woodpeckers, 27 were clicked.

To analyze the extent of the problem, we wrote a program to download Google Ads placed on this website on a page featuring photos of a cute baby goose. 5.4% were about bird watching, 29.9% about hunting, 21.8% about removing geese. The remaining ads were more general ads or ads about other topics (e.g., New Jersey). While many ad networks do give some limited amount of control for the content owner, it normally requires the domain owner to block particular ads or high-level categories without fine-grained control.

While one of our research tracks is looking at a general semantic solution to inappropriate ads, in this paper, we explore allowing the content owner to create a classifier that blocks such ads using active learning. We have designed an application that uses active learning to learn a classifier from examples labeled by the content owner that can automatically block inappropriate ads. There are two motivations for blocking such ads. First, content owners might believe their readers will finds the ads offensive. Second, the content owner may have prior knowledge which leads him believe that his readers would prefer certain ads over others.

Our ad blocking application has several components. We use a daemon to collect ad information such as the ad URL, ad text, and the ad landing page HTML. Another component displays ads to the content owner and records feedback for a subset of the ads. See Figure 2 for a picture of the


**Figure 2: Ad Labeling Interface**

ad labeling interface. This component uses active learning to decide which ads to display for labeling. A final component classifies unlabeled ads, and if the ad is labeled as inappropriate, it is removed as a candidate for presentation.

For this problem, we need an algorithm that quickly converges to a good concept because we want to reduce the number of ad labels that a content owner must provide. Ideally, we want an algorithm that can learn a fairly accurate classifier with as few as ten labeled instances; however, at this stage of our research, our goal is an algorithm that reduces the number of required labels by a significant fraction over passive learning. We also need an algorithm that can select in real time which instances to label. It defeats the purpose of using active learning if the user of the application must wait several minutes to receive the next instance, particularly when each instance only requires seconds to label. Active Vote is easily able to fulfill these requirements on standard inexpensive personal computers.

# 7. EXPERIMENTS

In this section, we provide the results of our active learning algorithm on various datasets. We use four datasets for our experiments: the Reuters corpus dataset, our own dataset for Internet advertisement, the KDDCUP 99 network intrusion dataset, and the MNIST handwritten digit dataset. The experiments are focused exclusively on testing the batch performance of various algorithms.

## 7.1 Basic On-line Algorithm

The on-line algorithm we use for our experiments is a version of Alma$_2$ [17]. The algorithm represents its current hypothesis as a hyperplane. Let $n$ be the number of attributes and let $m$ be the maximum number of non-zero attributes in any instance. Alma$_2$ requires $O(n)$ storage but can be implemented to perform predictions and updates in $O(m)$ time [27]. Alma$_2$ is similar to the Perceptron algorithm [7], but Alma$_2$ adjusts its updates as it learns in a way that tends to improve generalization performance. We use a version of Alma$_2$ that only updates its hypothesis on mistakes, as this form of the algorithm does not require any problem specific tuning of parameters. This is important for active learning since any type of validation is difficult given the biased sample of instances generated by sampling from $\hat{X}$.[1]

A useful modification for on-line algorithm is instance recycling [27]. This technique saves a window of recent labeled instances. A parameter controls the maximum number of times an instance can be used in an update. In our experiments, we set the maximum update for an instance to be four to limit overfitting. After the algorithm makes a mistake, it loops over the recent instances looking for mistakes. Any mistake causes an additional update and increases the counter associated with that instance. The trial is over when all instances in the window that are below their maximum update count are predicted correctly.

For Active Vote, we make two modifications to instance recycling to help keep instances in the confusion set. First, we randomly perturb the maximum update count for the instances. This is implemented by flipping a fair coin for each newly labeled instance to decide whether to increase the maximum use count by one. The second modification randomizes the instance recycling iteration. When the algorithm starts to recycle over the old instances, the algorithm randomly selects a starting instance from the window. Both of these modifications are used to force the on-line algorithms to make slightly different updates with the goal of keeping the confusion set non-empty. We use the prefix RR to any on-line algorithm that has been modified to use this randomized recycling procedure.

## 7.2 Experimental Setup

Each experiment is averaged over 50 permutations of a training set. We use permutations instead of cross-validation to make it easier to adjust the size of the training set for learning curve experiments. For each experiment, we use a standard t-test to give 95% confidence intervals.

We compare the active learning algorithms to the Support Vector Machine SVM$^{\text{light}}$ [10, 20] and various passive forms of the Alma$_2$ algorithm. As explained in Section 5.2, the final hypothesis of Alma$_2$ can have a high variance in accuracy. To stabilize and improve performance, we use a slight modification of a hypothesis averaging technique given by Schapire *et al.* In [15], they predict with a hyperplane that is an average over all the previous hyperplanes. We find performance improves by taking a weighted average where the hypothesis from query $t$ has weight $t$. Intuitively, this is sensible because an effective learning algorithm is converging on the target concept and therefore the later hypotheses should be more accurate. We add the prefix $W$ to any on-line algorithm that is modified with this averaging technique.

For all of the experiments, we give the results for the active learning algorithms $AV$-RR-Alma$_2$ and $AM$-Alma$_2$. We set $k = 7$ for both algorithms following the choice for Active Majority found in [22]. The random recycling saves the most recent 500 instances and has a maximum update count for each instance of either 3 or 4. Note that instance recycling is independent for each of the $k$ algorithms. The update count of an instance for one algorithm is unrelated to the update count for the other algorithms. We modify Active Majority slightly by using our initial hypotheses randomization procedure. For both active algorithms, we initialize the hyperplanes with five random instances. Last, we make all predictions for both algorithms using the majority vote of the $k = 7$ hypotheses. This is the procedure used by Active Majority in Liere and Tadepalli [22], and we find that Active Voting with instance recycling is stable enough that it does not need anything more complicated.

The last parameter for Active Vote is $r$, which controls the frequency of forced random queries. We set $r = \infty$ and do not perform any forced random queries. Random queries are

[1] Another advantage of the Alma is that it allows kernel functions to expand its feature space. This could be important for active learning problems that need non-linear spaces.

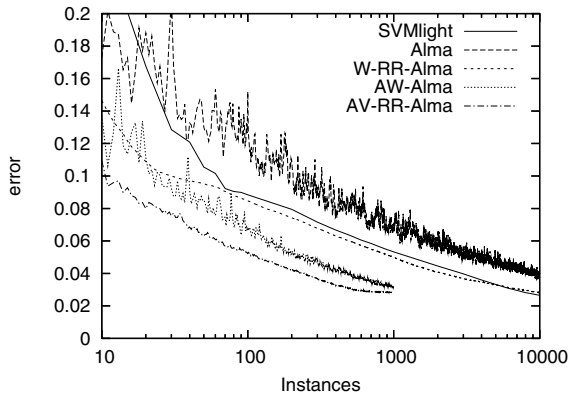


**Figure 3: Error rates for the Reuters earn concept**

useful to gauge performance or identify accurate hypotheses, but they tend to slow the learning. For our experiments, we use a test set to evaluate the algorithms and therefore do not need to use this feature.

## 7.3 Reuters

We use the Reuters-21578, Distribution 1.0 text collection [13]. We only use documents that either have topics specified or have no topics but which have a TOPICS tag value of yes. This latter group is defined to have no relevant topic. From this we create binary concepts based on the ten most frequent topics. Each document is represented as a bag of words with stemming [19] applied to each word. This gives a total of 13484 documents with 37086 words.

We train the algorithms using 10000 documents and test with the remaining 3484. We repeat this procedure using 50 permutations of the data. Active Vote divides its 10000 training examples into an active pool of size 9000 and a random pool of size 1000; therefore, we limit the algorithm to 1000 active queries. We also stop Active Majority after 1000 queries. Because it does not use a random pool, anytime all seven hypotheses agree on all remaining unlabeled instances, we just take a random query from all the instances. We include previously labeled instances in this random query because the remaining unlabeled instances might be biased based on the instances removed by the active queries.[2]

In Figure 3, we give the results for the most frequent concept, earn. Notice the high variability of Alma$_2$. The algorithm does tend to stabilize as the number of instances increases presumably because $Alma_2$ decreases the size of its updates as it learns. When we add instance recycling and weight averaging with the $W$-$RR$-Alma$_2$ algorithm, we see a large decrease in error rate and an improvement in stability. Again, it is important to use these or similar techniques when using these types of on-line algorithms in a batch setting. We see that the performance of $W$-$RR$-Alma$_2$ is comparable to SVM$^{\text{light}}$ for most of the plot.

During the early label queries, SVM$^{\text{light}}$ has the worst performance. This is a result of using the default parameters. While the default parameters tend to work well for a large number of instances, during the early queries it is preferable for SVM$^{\text{light}}$ to learn a concept that perfectly separates the data. Unfortunately, the optimal parameter choice for a small number of instances can give poor performance for larger training sizes. Since we are mostly interested in the performance of SVM$^{\text{light}}$ with all the training instances, we always use the default parameter.

For a small number of labeled instances, the two best performing algorithms are the active algorithms. However, one problem with $AM$-Alma$_2$ is that it has an intrinsic instability similar to the basic algorithm Alma$_2$. $AV$-$RR$-Alma$_2$ is much more stable because the instance recycling adds stability. We also see that Active Vote has more than a 20% reduction in error for most queries up to query 600. This is the point at which, on average, the algorithm starts to make a large number of queries from the random pool of instances. In practice, it can be useful to use this as a stopping criteria for the active learning because the rate of learning has slowed substantially. This is intuitive. Once an on-line algorithm has, on average, a high accuracy hypothesis, sampling from the problem distribution will require the algorithm to see many labeled instances before it makes a mistake and changes its hypothesis.

In Table 1, we give results for the first seven most frequent concepts.[3] In the heading for each column, we include the name of the label and the percentage of documents that contain that label. It is interesting that the performance of $WR$-Alma$_2$ at 10000 passive queries is almost identical to the performance of $AV$-$RR$-Alma$_2$ at 1000 active queries. This gives Active Voting an order of magnitude improvement in the number of labeled instances over a reasonably optimized version of Alma$_2$. In truth, the performance is actually better since $AV$-$RR$-Alma$_2$ starts making most of its queries from the random pool between 200 to 700 instances depending on the concept. After that point the improvement in accuracy is minimal.

The Support Vector Machine is consistently doing the best at 10000 queries, however, in many cases its improvement is not statistically significant over $AV$-$RR$-Alma. SVM$^{\text{light}}$ is more frequently statically tied with $WR$-alma$_2$ because $WR$-alma$_2$ has a confidence interval that is almost ten times larger than $AV$-$RR$-Alma$_2$. This large variability is also seen in Active Majority.

When dealing with a smaller number of queries, the active learning algorithms always perform best. Unfortunately at 100 queries, it is difficult to evaluate $AM$-Alma$_2$ because it has such a large confidence interval. This can occur with an on-line algorithm that uses its final hypothesis for batch learning. Even a majority vote with $k = 7$ hypotheses is not enough to add significant stability. However, the mean of $AV$-$RR$-Alma$_2$ is always lower after 100 queries.

At 1000 queries, either $AV$-$RR$-Alma$_2$ has a smaller mean or it is very similar to $AM$-Alma$_2$. In general, $AV$-$RR$-Alma$_2$ converges faster and with less variance than $AM$-Alma$_2$, but they appear to be converging to a similar accuracy. We do find it somewhat surprising that Active Majority does so well after 1000 queries. We also performed artificial data experiments with label noise as high as 10% where $AM$-Alma$_2$ has much greater problems with stability, while $AV$-$RR$-Alma$_2$ is able to quickly converge to a hypothesis that is close to optimal.

## 7.4 Internet Ads

The experiments we perform are based on ads found on

[2]This will slightly reduce the number of true label queries required by the algorithm.

[3]We removed the remaining three concepts because of space limitations. Their results are similar.

**Table 1: Reuters error rates**

| Alg | earn(.296) | acq(.182) | money(.059) | grain(.047) | crude(.047) | trade(.041) | interest(.038) |
|---|---|---|---|---|---|---|---|
| *100* queries | | | | | | | |
| W-RR-Alma | .0847±.0136 | .1309±.0490 | .0558±.0097 | .0448±.0081 | .0465±.0072 | .0406±.0073 | .0370±.0066 |
| AM-Alma | .0677±.0333 | .0994±.0630 | .0484±.0410 | .0403±.0930 | .0377±.0403 | .0417±.0426 | .0403±.0672 |
| AV-RR-Alma | .0522±.0098 | .0733±.0165 | .0397±.0103 | .0252±.0075 | .0272±.0090 | .0281±.0083 | .0279±.0083 |
| *1000* queries | | | | | | | |
| W-RR-Alma | .0499±.0090 | .0628±.0115 | .0380±.0074 | .0308±.0101 | .0323±.0102 | .0307±.0082 | .0295±.0069 |
| AM-Alma | .0314±.0091 | .0443±.0092 | .0244±.0056 | .0142±.0049 | .0189±.0050 | .0197±.0051 | .0199±.0051 |
| AV-RR-Alma | .0282±.0049 | .0374±.0063 | .0237±.0047 | .0134±.0041 | .0186±.0035 | .0197±.0048 | .0204±.0043 |
| *10000* queries | | | | | | | |
| W-RR-Alma | .0282±.0052 | .0382±.0063 | .0246±.0046 | .0144±.0038 | .0191±.0041 | .0195±.0041 | .0205±.0050 |
| SVM | .0265±.0007 | .0352±.0008 | .0227±.0007 | .0132±.0006 | .0181±.0006 | .0191±.0007 | .0186±.0006 |

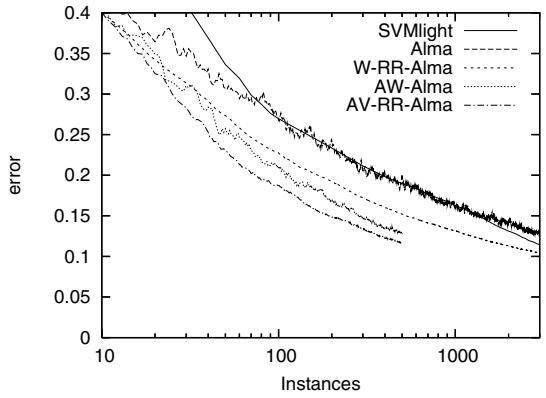


**Figure 4: Error rates for Internet ad experiments**

twelve websites that deal with various farm animal related topics.[4] We allow the content owner to specify one of five ratings for an ad. For these experiments, any ad that is rated below neutral is considered an inappropriate ad. The ads are represented based on the text of the ad and the text on the ad landing page. We use a bag of words representation with stemming on all words [19]. We also added extra attributes for words that appear in the ad and words in the titles and headers of the landing page.

In Figure 4, we include a plot of our results. The experiment is identical the experiment performed with the Reuters data except that we only have 4143 instances. We use 3000 of the instances for training and 1143 instances for the test set. The active experiments are limited to 500 queries. As can be seen, the results are similar with the *AV-RR*-Alma$_2$ algorithm giving the best performance with fewest label queries. While the active to passive label ratio is only 1/4, we speculate this is because we are on parts of the learning curve that have higher error rates compared to the Reuters data. We plan to explore this in future research.

## 7.5 KDDCUP and MNIST

The primary purpose for these two datasets is to compare against the active learning algorithm of Beygelzimer *et al* [6]. We believe this is an interesting comparison given the strong theoretical motivation behind this recent algorithm. However, we want to stress that the comparison is more of a qualitative comparison because the experiments performed

[4]The data will be made publicly available on a the UCI Machine Learning repository [13].

**Table 2: KDDCUP99 Error rates**

| Alg | 100 | 500 | 1000 | 10000 |
|---|---|---|---|---|
| AA-J48 | .032 | .0065 | | |
| J48 | .052 | .012 | .012 | |
| AM-Alma | .0115 | .00882 | .00705 | |
| AV-RR-Alma | **.00861** | **.00582** | **.00388** | |
| W-RR-Alma | .0253 | .0137 | .01026 | **.00339** |
| SVM | .0202 | .0124 | .0115 | .00574 |

in [6] uses a decision tree, J48, as their basic algorithm and use PCA to reduce both problems to 25 attributes. For these experiments, we do not report statistical significance because of space constraints; however, we follow a similar procedure as the previous experiments, using 50 permutations to compute the average performance. The statistical significance is similar to the Reuters experiments.

The KDDCUP99 dataset is a large set of network intrusion data from a previous KDDCUP competition. Each instance consists of 41 symbolic or continuous attributes. To improve performance when learning with a hyperplane, we expand the attribute space by using a binning technique on the continuous attributes. This technique breaks the attribute up into at most 30 intervals and creates a new attribute for each of these intervals. A binning attribute has a value of 1 if the original continuous attribute has a value greater than the end point of the interval, otherwise the binning attribute has a value of 0. This allows the algorithm to learn concepts that represent a range of values using only two binning attributes [26].

The results for 100, 500, 1000, and 10000 label queries are reported in Table 2. We use the notation *AA*-J48 to refer to the learning algorithm of Beygelzimer *et al.* when applied to the J48 decision tree algorithm. The results show that at 100 queries *AV-RR*-Alma$_2$ has roughly 1/3 the error rate of *W-RR*-Alma$_2$, and that for some error rates, *W-RR*-Alma$_2$ uses over ten times as many labels as Active Vote. While the improvement of *AA*-J48 is not as dramatic, it still gives a large decrease in error rate with roughly 50% of the error rate of J48 at 500 queries.

The MNIST dataset is a dataset of handwritten digits with 256 bit grayscale information for a grid of 28 by 28 pixels. Since this dataset already has a large number of attributes, we just use the 728 continuous values normalized to the [0,1] interval. The results are reported in Table 3. While the results are not as good as the KDDCUP99 experiment, we still see that *AV-RR*-Alma$_2$ has close to half the error rate of *W-RR*-Alma$_2$ at 100 queries. The results for *AA*-J48 show minimal improvement. In [6], the authors speculate

**Table 3: MNIST Error rates**

| Alg | 100 | 500 | 1000 | 10000 |
|---|---|---|---|---|
| AA-J48 | .18 | .107 | .082 | |
| J48 | .18 | .107 | .093 | |
| AM-Alma | .104 | .0549 | .0477 | |
| AV-RR-Alma | **.0633** | **.0421** | **.0408** | |
| W-RR-Alma | .108 | .0580 | .0497 | **.0373** |
| SVM | .0818 | .0501 | .0452 | .0374 |

that the large amount of noise in the MNIST dataset causes the poor performance of $AA$-J48. This might also explain the somewhat poor performance of $AM$-Alma$_2$.

# 8. CONCLUSION

This paper revisits some of the previous research in active learning. which showed promise because the algorithms are computationally efficient and the results gave dramatic decreases in label queries on practical problems. We provide a new way to understand one of these algorithms in terms of on-line learning, and we show how to take this new understanding to give a more effective way to convert on-line algorithms into active learning algorithms.

While not a focus of the paper, our results naturally apply to a new type of active learning problem that is based on bootstrapping an on-line problem so that it can maximize its learning. We plan to expand our understanding of this framework in future research.

Another area that requires more research is the confusion set. While we did not have a problem keeping elements in the confusion set, we are interested in algorithms/problems in which we can prove that the confusion set will be non-empty until the algorithm achieves an accurate hypothesis. This might require changes such as selecting instances for labeling based on how they affect the next confusion set.

# 9. REFERENCES


[1] D. Angluin. Queries revisited. *Theor. Comput. Sci.*, 313:175–194, February 2004.

[2] J. Attenberg and F. Provost. Why label when you can search? In *KDD*, 2010.

[3] M. F. Balcan, A. Beygelzimer, and J. Langford. Agnostic active learning. In *ICML*, 2006.

[4] M. F. Balcan, S. Hanneke, and J. W. Vaughan. The true sample complexity of active learning. *Machine Learning*, 80(2-3):111–139, 2010.

[5] A. Beygelzimer, S. Dasgupta, and J. Langford. Importance weighted active learning. In *ICML*, 2009.

[6] A. Beygelzimer, D. Hsu, J. Langford, and T. Zhang. Agnostic active learning without constraints. In *NIPS*, 2010.

[7] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34(1):123–135, 1962.

[8] N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

[9] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Worst-case analysis of selective sampling for linear classification. *Journal of Machine Learning Research*, 7:1205–1230, 2006.

[10] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[11] S. Dasgupta. Coarse sample complexity bounds for active learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS*, pages 235–242, Cambridge, MA, 2006. MIT Press.

[12] S. Dasgupta, D. Hsu, and C. Monteleoni. A general agnostic active learning algorithm. In *NIPS*, 2007.

[13] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998; http://archive.ics.uci.edu/ml/.

[14] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second-price auction *American Economic Review*, 97(1):242–259, 2007.

[15] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *COLT*, 1998.

[16] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–1680, 1997.

[17] C. Gentile. A new approximate maximal margin classification algorithm. *Machine Learning*, 2:213–242, 2001.

[18] S. Hanneke. Rates of convergence in active learning. *The Annals of Statistics*, 39(1):333–361, 2011.

[19] M.-C. Jenkins and D. Smith. Conservative stemming for search and indexing. In *SIGIR*, 2005.

[20] T. Joachims. Making large-scale support vector machine learning practical. In A. S. B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1999.

[21] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *SIGIR*, pages 3–12. Springer-Verlag, 1994.

[22] R. Liere and P. Tadepalli. Active learning with committees for text categorization. In *AAAI*, pages 591–596, 1997.

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

[24] N. Littlestone. From on-line to batch learning. In *COLT*, pages 269–284, 1989.

[25] N. Littlestone. *Mistake Bounds and Linear-threshold Learning Algorithms*. PhD thesis, Computer Science, University of California, Santa Cruz, 1989. Technical Report UCSC-CRL-89-11.

[26] W. Maass and M. K. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141:378–386, 1997.

[27] C. Mesterharm. *Improving On-line Learning*. PhD thesis, Computer Science, Rutgers, The State University of New Jersey, 2007.

[28] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[29] V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.