Active Learning with Boosting for Spam Detection

Enela Pema

December 11, 2009

Abstract

Spam detection algorithms have been developed to train in a large enough set of labeled data and predict with a high accuracy of 95% if an email is spam or not. A problem that arises in this setting is that labeling examples is a costly process. It requires humans to read them one by one and classify them. Active learning is a learning approach developed to address this problem. It learns a small set of labeled examples and implements strategies to improve learning by labeling additional examples. Its goal is to provide a good classifier with as small as possible a number of examples. Boosting algorithms intuitively fit well into the active learning framework since they are good at distinguishing significant examples. In this report, I study performance and behavior of active learning in combination with four boosting algorithms: AdaBoost, AdaBoost*, LPBoost, ERLPBoost.
1. Introduction

Spam filters detect whether a new incoming mail is a spam or not. An accurate classifier can be obtained from training on a big set of batch data. An issue related to spam filtering is that labeling examples is too costly. The only way to label an email as spam or not spam is by having the user to look at its content. Labeling a large set of examples is very time consuming. The set of labeled examples that can be feasibly provided is small, while the set of unlabeled examples is large. Therefore, in this setting there is place for applying active learning techniques. Active learning is a learning process that operates initially on a small set of labeled examples and iteratively increases the training set by asking the user to label more examples. So, part of the learning process is also choosing significant examples to label in order to learn fast. Boosting fits naturally into active learning because it focuses learning on difficult examples. My goal in this project is to analyze the behavior and performance of active learning using four versions of boosting with different characteristics. I give a short analysis of each algorithms. The organization of this report is as follows: In Section 2 I describe the Active Learning algorithm. In Section 3 I describe the dataset I used for this project. In Section 4 I describe the four Boosting algorithms I implement and combine with active learning. In Section 5 I give my results from plugging these algorithms into the active learning framework and I compare them with each other. In Section 6 I recapitulate my results.
2. Description of Active Learning

Active learning operates on a dataset that consists of a small number of labeled examples and a large number of unlabeled ones. It iteratively asks the user to label $k$ examples at a time. It stops sampling when it reaches the maximum allowed size of labeled data. The number of initially labeled examples, the number of examples labeled in each iteration and the maximum number of labeled examples are determined based on outside factors. I have implemented active learning with Confidence-Based sampling. The hypothesis provided from training on the current training set is used to score the unlabeled examples. The score is a measure of how much the weak learners agree on the prediction of a certain examples. The lower the score, the more difficult the example is. We pick the $k$ most difficult examples to add to the training set. The key idea of this approach is that it is using boosting to distinguish the significant examples of the set. By adding the most significant examples first we can achieve an accurate classifier really fast. I show that this approach works by comparing it to Active Learning with random-sampling. This approach chooses randomly what examples to label. I describe the two frameworks below. Later on I plug in various boosting algorithms into these frameworks.
Active Learning Framework with Confidence-Based Sampling (using Boosting)

Given a dataset of labeled examples \((x_1, y_1), \ldots, (x_{N_{tr}}, y_{N_{tr}})\) and unlabeled examples \((x_{N_{tr}+1}), \ldots, (x_{N})\)

Train on \((x_1, y_1), \ldots, (x_{N_{tr}}, y_{N_{tr}})\) with boosting and provide a strong classifier \(H\)

Compute scores for \((x_{N_{tr}+1}), \ldots, (x_{N})\) using the formula:

\[
score(x_i) = \frac{\left| \sum_{j=1}^{T} \alpha_j h_j(x_i) \right|}{\sum_{j=1}^{T} \alpha_j}
\]

Ask the user to label the \(k\) examples with smaller score.

Add the new labeled examples to the training set.

\(N_{tr} = N_{tr} + k\)

If \(N_{tr}\) smaller than the maximum size of the training set (given as input), repeat from the beginning.

| Table 1: Active Learning Framework with Confidence-Based Sampling |

Active Learning Framework with Random Sampling

Given a dataset of labeled examples \((x_1, y_1), \ldots, (x_{N_{tr}}, y_{N_{tr}})\) and unlabeled examples \((x_{N_{tr}+1}), \ldots, (x_{N})\)

Train on \((x_1, y_1), \ldots, (x_{N_{tr}}, y_{N_{tr}})\) with boosting and provide a classifier \(H\)

Pick randomly \(k\) unlabeled examples. Ask the user to label the \(k\) chosen examples.

Add the new labeled examples to the training set.

\(N_{tr} = N_{tr} + k\)

If \(N_{tr}\) smaller than the maximum size of the training set (given as input) then repeat from the beginning.

| Table 2: Active Learning Framework with Random Sampling |
3. Dataset

I use the dataset provided to us in class consisting of 2000 emails and 2000 features. Features here are key words found in emails. The presence or non-presentation of certain words in an email determines whether it is spam or not. The original dataset has labels \( \{0, 1\} \). I update those values to \( \{-1, 1\} \). Moreover, I generate the complements of the features. After applying these updates, the dataset I am working on has 2000 examples and 4000 features. I separate \( \frac{3}{4} \) as a training set and \( \frac{1}{4} \) as a testing set. The space of hypotheses that the boosting algorithms use is the space of features. In all experiments, the algorithms were run 5 times and the results are averaged among 5 runs.

4. Boosting

Boosting is a learning algorithm that iteratively constructs a strong hypothesis as a combination of weak hypothesis. The weak hypothesis are considered to be at least 50% accurate. The key idea of the algorithm is that it improves learning by putting more focus on the difficult examples. Difficult examples are considered to be more significant. The algorithm can learn more from them. In every iteration, a weak learner with highest edge is chosen. The edge of a weak learner shows how good it is in predicting the examples. The algorithm maintains a distribution over the examples as a measure of how difficult an example is and the distribution is updated in every boosting round. The weight of an example is increased if the weak learner does not predict well on it and is decreased if the weak learner predicts well. While learning, the edge of the weak learners decreases and the minimum error approaches \( \frac{1}{3} \). Every weak learner is associated a weight \( \alpha_t \) which expresses the "strength" of the weak learner. Boosting produces a linear combination of base hypothesis. This linear combination assigns a positive or negative value to every example. The sign is the prediction \((-1,+1)\). Most boosting algorithms differ on how they update the distribution on the examples or the weight vector of weak learners. I briefly describe the four boosting algorithms I have implemented.
4.1 AdaBoost

AdaBoost algorithm

Given \((x_1, y_1), \ldots, (x_N, y_N) \in S_t\) where \(y_i \in \{-1, +1\}\)

Initialize distribution over the examples \(D_1 = \frac{1}{N}\) for \(i = 1..N\)

For \(t=1\) to \(T\) (number of boosting rounds)

Get weak hypothesis \(h_t\) such that \(h_t = \text{argmin}\{\epsilon_j\}\)

where \(\epsilon_j\) is the error of weak learner \(h_j\) given distribution \(D_t\), \(\epsilon_j = \sum_{i=1}^{N} D_t[i \neq y_i]\).

Update distribution \(D_{t+1}(x_i) = \frac{D_t(x_i)e^{-\alpha_t y_i h_t(x_i)}}{Z}\) where \(Z\) is a normalizing factor.

and \(\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}\)

Output strong classifier \(H(x_i) = \text{sign}(\sum_{k=1}^{t} \alpha_k y_k h_k(x_i))\)

| Table 3: Table 1: AdaBoost Algorithm |

AdaBoost does an exponential update on the distribution over the examples. It increases the weights by a factor \(e^{\alpha_t}\) for misclassified examples and decreases them by a factor \(e^{-\alpha_t}\) for correctly classified examples. The following plot shows performance of AdaBoost in the training and testing set of the spam email dataset.
4.2 AdaBoost*

In AdaBoost it has been observed that even when all examples are classified correctly, the algorithm keeps learning and the margins of the examples keep improving. In fact AdaBoost has not been proved to maximize the margins. Based on this observation, AdaBoost* maximizes the minimum margins up to a certain precision. To achieve this it ensures that the weak learner returns hypothesis with edge at least $\gamma^*$ (minimum edge). If this holds, then there exists a linear combination of these hypothesis that achieves margin $\gamma^* = \rho^*$. This is guaranteed by the following theorem:

Theorem: Von Neumann’s Min-Max-Theorem

$$\gamma^* := \min_D \max_{m=1,M} \sum_{n=1}^N D_n y_n h_m(x_n) = \max_{\alpha} \min_{n=1,N} y_n \sum_{m=1}^M \alpha_m h_m(x_n) = \rho^*$$

The minimum edge $\gamma^*$ that can be achieved over distributions $D$ is equal to the maximum margin $\rho^*$ of any linear combination of $M$ hypothesis.
AdaBoost* Algorithm

Given \((x_1, y_1), \ldots, (x_N, y_N) \in S_t\) where \(y_i \in \{-1, +1\}\)
Initialize distribution over the examples \(D_1 = \frac{1}{N}\) for \(i = 1..N\)

For \(t=1\) to \(T\) (number of boosting rounds)
Get weak hypothesis \(h_t\) such that \(h_t = \text{argmax}\{\gamma_j\}\)
where \(\gamma_j\) is the edge of weak learner \(h_j\) given distribution \(D_t\).

\[\gamma_j = \sum_{i=1}^{N} D_t^i y_i h_t(x_i)\]

If \(|\gamma_t| = 1\), then \(\alpha_1=\text{sign}(\gamma_t), h_1=h_t, T=1\), break.

\(\gamma_{\text{max}} = \min\{\gamma_r\}\) for \(r = 1..t\), \(p_t = \gamma_{\text{max}} - \nu\)
\(\alpha_t = \frac{1}{2} \ln \frac{1 + \nu}{1 - \nu}\)

Update distribution \(D_{t+1}(x_i) = \frac{D_t e^{-\alpha_t y_i h_t(x_i)}}{Z_t}\) where \(Z_t\) is a normalizing factor.

Output strong classifier \(H(x_i) = \text{sign}\left(\sum_{k=1}^{t} \alpha_k y_k h_k(x_i)\right) / \sum_{r=1}^{T} \alpha_r\)

Table 4: AdaBoost* Algorithm

For this algorithm it is proved that it achieves margin at least \(\rho \ast -\nu\), where \(\nu\) is a precision parameter, in at least \(T = \frac{2\ln N}{\nu^2}\). The step size in updating alpha will depend on the progress of the algorithm towards the desired margin. The parameter \(\nu\) can be tuned.

The following plot shows performance of AdaBoost* trained with the spam dataset for \(T=350\) and \(\nu = 0.2\). For tuning this parameters I picked values of \(T\) close to 200 (as in AdaBoost) and gave the parameter \(\nu = \sqrt{\frac{2\ln N}{T}}\). I observed the performance with various values of \(T\) and \(T=350, \nu = 0.2\) seemed like a good choice.
Figure 2: Performance of AdaBoost* in the training set of 1500 examples and in the testing set of 500 examples

4.3 LPBoost

LPBoost is a boosting algorithm defined based on a linear programming problem. It updates the weights $d^t$ at iteration $t$ to any distribution that minimizes the maximum edge of the hypothesis seen so far. This program is solved using linear programming. The problem has multiple solutions and depending on the optimizer a different solution can be used. Here I implement the hard margin version of LPBoost.
LPBoost Algorithm

Given \((x_1, y_1), \ldots, (x_N, y_N) \in S_t\) where \(y_i \in \{-1, +1\}\), and accuracy parameter \(\epsilon\)

Initialize: \(d^0\) to the uniform distribution and \(\gamma^0\) to 1.

Do for \(t=1, \ldots, T\)

Send \(d^{t-1}\) to oracle and obtain hypothesis \(h^t\)

Set \(u^t_n = h^t(x_n) y_n\)

Assume \(u^t d^{t-1} \geq g\), where \(g\) need not be known

Update the distribution to any \([d^t, \gamma_{LP}] \in \text{argmin} \ \gamma\)

such that \(u^q d \leq \gamma\), for \(1 \leq q \leq t\), \(d_n \leq 1\), for \(1 \leq n \leq N\), and \(\sum_n d_n = 1\)

If \(\min_{q=1 \ldots t} u^q d^{t-1} - \gamma_{LP} \leq \epsilon\) then set \(T = t\) and break.

Output: \(f_w(x) = \sum_{q=1}^{T} w_q h^q(x)\), where the coefficients \(w\) maximize the soft margin over the hypothesis set \(\{h^1, h^2, \ldots, h^T\}\) using the LP problem.

Table 5: LPBoost Algorithm
The plot below shows the generalization error of LPBoost algorithm in the training set and in the testing set.

Figure 3: Performance of LPBoost in the training set of 1500 examples and the testing set of 500 examples

It can be noticed that it is very unstable.
4.4 ERLPBoost

As described in the section above, LPBoost is very unstable. Entropy Regularized LPBoost adds entropy regularization to the objective function in LPBoost. The optimization problem here is:

\[
\min_{d \in S} \max_{t=1..T} u^t d + \frac{1}{\eta} \Delta(d, d^1)
\]
\[\text{s.t. } d \leq \frac{1}{\eta} 1\]

I use sequential quadratic programming to solve the optimization problem. The iteration bound for ERLPBoost is proven for \(\eta \geq \frac{1}{\epsilon} \ln \frac{N}{\nu}\).

**ERLPBoost Algorithm**

Given \((x_1, y_1), \ldots, (x_N, y_N) \in S_t\) where \(y_i \in \{-1, +1\}\), accuracy parameter \(\epsilon\) and capping parameter \(\nu \in [1, N]\)

Initialize: \(d^1\) to the uniform distribution.

Do for \(t=1, \ldots, T\):

Send \(d^t\) to oracle and obtain hypothesis \(h^t\)

Set \(u^t_n = h^t(x_n)y_n\)

Set \(\delta^t = \min_{q=1..t} P^q(d^q) - P^{t-1}(d^t)\) If \(\delta^t \leq \frac{\epsilon}{2}\) then set \(T = t - 1\) and break

Else update distribution to:

\([d^{t+1}, \gamma^t] \in \arg\min_{d, \gamma} \gamma + \frac{1}{\eta} \Delta(d, d^t)\]
\[\text{s.t. } du^m \leq \gamma, \text{ for } 1 \leq m \leq t; \ d \in P^N, \ d \leq \frac{1}{\eta} 1\]

Assume \(u^t d^{t-1} \geq g\), where \(g\) need not be known

Update the distribution to any \([d^t, P_{LP}^t] \in \arg\min \gamma\)

such that \(u^q d \leq \gamma\), for \(1 \leq q \leq t\), \(d_n \leq 1\), for \(1 \leq n \leq N\), and \(\sum_n d_n = 1\)

If \(\min_{q=1..t} u^q d^{t-1} - P_{LP}^t \leq \epsilon\) then set \(T = t\) and break.

Output: \(f_w(x) = \sum_{q=1}^{T} w_q h^q(x)\), where the coefficients \(w\) maximize the soft margin over the hypothesis set \(\{h^1, h^2, \ldots, h^T\}\) using the Linear Programming problem.
The following plot shows performance of ERLPBoost in the spam email dataset. The parameters used are: $\epsilon = 0.01 \eta = \frac{1}{\epsilon} \ln N$.

Figure 4: Performance of AdaBoost* on the training set of 1500 examples and the testing set of 500 examples
5. Active Learning with Boosting

I take each of the previously described algorithms and combine it with active learning. In every experiment I separate the dataset into a training and sampling set of size $\frac{3}{4}N = 1500$ and a testing set of size $\frac{1}{4}$. I start with a training set of size 200 and add 50 examples at a time. The sample of 50 is chosen from the training and sampling set of 1500 examples. I will refer to the training and sampling set as the full set in my plots. In every iteration I measure accuracy in both sets. In the full set, I treat the examples on which the algorithm has not trained yet as testing examples. I average the error on both the labeled part and the unlabeled part. I also measure accuracy in the testing set of 500 examples. I report both results in different plots. In the analysis of active learning with each boosting algorithm, I compare performance of active learning with confidence-based sampling with the performance of active learning with random sampling, both described in Section 2.

In active learning we never and up labeling the full set. My motivation for doing that in my experiments was to study the behavior of the algorithms in every iteration. It allows for more significant observations to be made.

5.1 Active Learning with AdaBoost

I plug the AdaBoost algorithm into the Active Learning framework. I tune $T=200$ rounds. In section 4.1 AdaBoost achieves accuracy 95% in the testing set earlier around the 80th iteration, but it stabilizes after 200 iterations. The following plots show performance of AdaBoost in the full dataset and in the testing dataset.
As we can see from the plots, AdaBoost succeeds in scoring the difficult examples. The accuracy achieved with confidence-based sampling is higher as compared to the accuracy achieved with random sampling. The difference is more noticeable in early sampling iterations. We notice that when the training set reaches 700 examples, accuracy achieved from confidence-based sampling algorithm is 95%-96% which is as good as we can possibly get. From this point, adding more examples does not contribute much in the learning process. At the end when we have labeled the full dataset, both lines meet as they should.

There is a concern related to tuning the number of boosting rounds. I let AdaBoost run the same number of rounds in every iteration of active learning. Since the size of the training set changes, one might wonder if we are overfitting. The high performance of 95% in the testing set in the region [550 examples, 1500 examples] indicates that the algorithm is not overfitting. But in the region [100 examples, 550 examples] we might be overfitting. To check this I ran active learning with AdaBoost with different values of T. I show the result in the plot below.
As this plot suggests, in the region [100 examples, 550 examples], accuracy is comparable. A careful eye can denote that in fact the performance is a just little bit better when the number of rounds is smaller. This indicates that we might be overfitting a little. The reasons why we do not get a more obvious overfitting effect in early iterations is because the training set is too small to even learn from it. Also, under the assumption that we are adding significant examples to the training set, the amount of noise is small. If $T$ is chosen smaller, we might gain very little better performance in the region [100 examples, 550 examples], but for larger training sets $T$ will not be enough training and accuracy will drop. This is the best tuning I could do for AdaBoost. A more sophisticated solution would be to make the number of boosting rounds smaller initially and increase it by some factor after every sampling iteration.
5.2 Active Learning with AdaBoost*

For AdaBoost* I tuned the precision parameter $\nu$ to 0.2. If the precision value is very small, it takes too long for the algorithm to converge to the maximum margin. If $\nu$ is larger it takes less iterations to achieve margin $\rho = -\nu$ but the achieved margin has larger distance from the optimal margin. For $\nu = 0.2$ and $T = 350$ the algorithm performs very well. The error in the training set is minimal and no overfitting is observed in the testing set. When I plug AdaBoost* into the Active Learning framework, I change the number of rounds after every sampling iteration. I update $T = \frac{2\ln N_i}{\nu^2}$ where $N_i$ is the size of the training set after the i-th iteration of active learning. The following plots show the performance of active learning with AdaBoost* in the full dataset and in the testing set.
Figure 8: Active Learning with AdaBoost*: confidence-based vs random sampling
5.3 Comparison of Active Learning with AdaBoost and Active Learning with AdaBoost*

The following plot compares active learning using AdaBoost and AdaBoost*.

Figure 9: Learning with AdaBoost*: confidence-based vs random sampling
As the plots show, AdaBoost\(^*\) performs much better in the full dataset and is comparable to AdaBoost in the testing set. In the testing set, in early sampling iteration performs slightly better than AdaBoost.

### 5.4 Active Learning with LPBoost

The plot below shows accuracy of active learning with LPBoost.
5.5 Active Learning with ERLPBoost

I use the parameters $\epsilon = 0.01$, $\nu = 1$, $\eta = \frac{1}{7}\ln N$ when running ERLPBoost with active learning and I produce the following plot.
I was not able to perform enough tests with ERLPBoost. The plots were produced using $\nu = 1$ and theoretical value of $\eta$. It does not show good performance and the highest accuracy it achieves is 85%. I believe this is due to bad tuning. The regularization effect as compared to active learning with LPBoost can be observed in early iterations.

6. Conclusions

I experimented with active learning using various boosting algorithms. A convincing result is that active learning with boosting works. In effect the achieved performance was high in small sets of labeled data. AdaBoost* seemed to perform better than all the algorithms. Remains to be studied ERLPBoost due to incomplete experimental results in this report.
References


