Assignment 1 Report, Group #3

Catherine Slaughter, Fazel Mohammadi, Dehan Zhang

Leiden Institute of Advanced Computer Science, The Netherlands

1 Introduction

In machine learning, a binary classification problem is one of the simplest types of problems students are taught to solve. In these, a set of data with an arbitrary number of features are assigned one of two labels (usually mathematically indicated as either 0 and 1 or -1 and 1), where the label assigned is dependent in some way on the features. There are many clever ways to go about solving a binary classification problem. One handy and standard tool in machine learning is ensemble learning. Ensemble learning works based on the principle of the wisdom of the crowd, which tells us that a large group of people (if acting logically and without some group-wide bias) are asked to estimate a value or answer a question. The group's average answer is often an excellent guess. In machine learning, this can be done by creating a group of learners, having them each attempt an answer, and taking the average of all the outputs. There are several different ways to implement an ensemble learning algorithm. For this project, we are tasked with writing a linear half-space Perceptron algorithm, combining a number of these into an ensemble, and training them based on an AdaBoost boosting algorithm. With these, we test our ensemble on several toy data sets to sort them via binary classification.

2 Background

2.1 Linear Half-spaces Weak Learner

One way to tackle a binary classification problem is with a linear halfspace learner. In linear halfspaces, the goal is to divide the feature space into two with a flat hyperplane. For example, for a data set with two features of importance (a 2-dimensional set), this looks like dividing the data with a line. An nth-dimensional vector parameterizes the dividing hyperplane, $\mathbf{w}$ and a bias term, b, where the dimensionality n is, again, equal to the number of features being trained on [2]. The label prediction for a data point $\mathbf{X}$ given by a single linear halfspace classifier is equal to the sign of

$$\langle \mathbf{w}, \mathbf{X} \rangle + b \tag{1}$$

[2]

Linear half-spaces are a straightforward way to implement a binary classifier. However, what is gained in simplicity may be lost in efficacy for data sets that are not linearly separable. For example, data for which the "boundary" between one label and another is circular (or hyperspherical, in higher dimensions) may not be well classified by such an algorithm [2]. Additionally, data with several boundaries between the different labels may cause the linear half-spaces method to falter. We hope that by creating an ensemble of weak linear half-spaces, we can create predictors that regularly perform better than random chance (50%), even for these nonlinear and complicated cases.

Implementation We train our linear half-space predictors via a Perceptron algorithm for our project. Using a Perceptron is one way to implement Empirical Risk Minimisation while training our weak learners. It does this by iteratively adjusting the values in $\mathbf{w}$ and b to attempt to include data points that the model finds incorrect in previous iterations. Because we cannot assume our data is linearly separable, we introduce a maximum on the number of allowed iterations. For our purposes, this is set to 200, but the user can change it. In plain language, the algorithm works as follows [6]:

1. Initialize $\mathbf{w}$ and b, and the learning rate r
2. For each data point in the training set:
   - Predict the label given with current values of $\mathbf{w}$ and b
   - Compare prediction to the real value
   - If incorrect, update $\mathbf{w}$ and b accordingly
3. Decrease r
4. Repeat steps 2 and 3 until all training points are correctly classified, or the maximum number of iterations is reached

Each time a point $\mathbf{X}_i$ is found to be incorrectly classified, the values in $\mathbf{w}$ and b are updated as follows:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + r\mathbf{X}_i y_i \tag{2}$$

and

$$\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} + r y_i \tag{3}$$

Where y_i is the real, known label for training point $\mathbf{X}_i$. This algorithm finds the minimum for the training error loss function, which is equal to the number of points that are wrongly classified over the total number of points:

$$L_S(h) = \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m} \tag{4}$$

It minimizes this loss by adjusting to fit each wrongly classified data point. The learning rate r tells us how significant the adjustment step(s) is. Picking the value of r is essential but tricky. Too large a value will cause the algorithm to overshoot the minimum, and too small a value will take many iterations to approach the minimum. For this reason, it is common to decrease the value of the learning rate with each Perceptron iteration, the specific method with which we do this is discussed further in subsection 3.1. By convention, the learning rate takes values between zero and one, so we initialize it to one.

We randomly initialize $\mathbf{w}$ instead of initializing it to zeroes. We found early on that our choice made very little difference in our ensemble outcomes. In theory, for a less complicated data set and small learning rate, it is possible that starting with the same values every time could lead the Perceptron algorithm to get "stuck" in some local minima when attempting to minimize loss. While this is not a concern for this project, we choose to randomly initialize $\mathbf{w}$ regardless, as there is no significant difference in memory or computing power used between np.rand() and np.zeros().

2.2 AdaBoost Ensemble Learning

There are generally two ways to construct an ensemble in ensemble learning: bagging or boosting. Several weak learners are created separately in a bagging algorithm, and their predictions are averaged to determine the ensemble prediction. By comparison, in a boosting algorithm, each new weak learner is trained based on some output from the learner that came before it–old learners "boost" new ones [1].

Implementation Our project uses an Adaptive Boosting ("AdaBoost") algorithm to construct our ensemble. In an AdaBoost algorithm, the data that are used to train each constituent learner are selected based on how well previous learners have predicted them. For each learner, a subset of training data is selected. For our purposes, we randomly select 60% of the total given data for our training data, and of this training data, 50% is used to train each weak learner in the ensemble. These subsets necessarily overlap often [1].

Additionally, a weighted average of individual learner predictions is computed for the ensemble to predict a class. The weights for these averages (α) are based on the learner's loss:

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$$

Where t in this context is the learner number and is unrelated to the iteration variable used in the Perceptron algorithm description. ϵ_t is a simple loss, the number of misclassifications over the total number of training set data points, the same as in Equation 4 [5]. This loss may go to zero in cases with linearly separable data, generating a divide-by-zero error. In these cases, α is calculated with $\epsilon = .00001$, an arbitrarily small value.

For each weak learner, the subset of training data used to teach it is randomly selected according to a probability distribution. This distribution starts uniformly. With each new weak learner added to the ensemble, the probability distribution is changed based on if the new learner correctly predicts the associated data point. Data that are incorrectly predicted are given a higher probability, so they are more likely to be used to train the next weak learner(s) and vice versa. The equation to update each probability is

$$D_{(t+1)} = D_t exp(-\alpha_t y_i \hat{y}_i)$$

Where y_i is the objective, known label value and $\hat{y}_i$ is the value predicted by the t-th learner. If these labels are the same, the exponent becomes negative, decreasing the probability [5]. If they are different, the exponent becomes positive, increasing the probability. After all the new probabilities are calculated, they are divided by the sum of all the new probabilities to normalize to one. This is necessary for the weights in NumPy's random selection function.

Once all the weak learners are trained, a prediction is made by taking the weighted average of all the ensemble's predictions. If this average is more significant than zero, the ensemble prediction is 1; otherwise, it is -1.

2.3 Theoretical Multi-class Extension

In principle, our ensemble could be extended to a multi-class classifier with more than two possible labels for the data. The most obvious way to do this would be to change the style of our base learner from a linear half-space algorithm to an alternative option. Decision trees are one relatively simple and common alternative for this purpose.

Another alternative would be to directly extend our Perceptron algorithm to the multi-class case. A multi-class Perceptron algorithm for halfspaces creates a set of linear boundaries–one for each possible label–and tunes the weights of each of them simultaneously. Given a data set with K possible labels and T training points, the algorithm [3] generally works as follows:

1. Initialise a set of vectors $\mathbf{w}_i$ where i ranges from 0 to K
2. Predict $\hat{y}_t$
   - The prediction is the label i whose corresponding weight vector maximises $\langle \mathbf{w}_i, \mathbf{x}_t \rangle$
3. If the prediction is incorrect ($\hat{y}_t ! = y_t$), update the weight vectors corresponding to the predicted label ($\hat{y}_t$) and the real label (y_t)
   - $\mathbf{w}_{\hat{y}_t}^{(t+1)} = \mathbf{w}_{\hat{y}_t}^{(t)} - \mathbf{x}_t$
   - $\mathbf{w}_{y_t}^{(t+1)} = \mathbf{w}_{y_t}^{(t)} + \mathbf{x}_t$
4. if The prediction is correct, the weights remain the same.
5. repeat steps 2-4 T times for all data points $x_{0...T}$

This general algorithm can be adjusted to include a learning rate in the weight update step as the user sees fit.

3 Experiments

To better understand the efficacy and shortcomings of our linear half-spaces AdaBoost ensemble, we calculate the loss according to Equation 4 for several different circumstances for each of our four toy data sets.

3.1 Learning Rate

While we knew we wanted to decrease the learning rate of our Perceptron algorithm with each iteration, the choice of how to do so was somewhat more complex. Our second test compares loss minimization for two different methods of decreasing the learning rate. For the nth Perceptron iteration, the first method sets the learning rate to $r_n = 1/(n-1)$, the latter multiplies the previous learning rate by this value, so $r_n = r_{(n-1)}/(n-1)$ (Figure 1). Ideally, the loss will decrease monotonically with each iteration. If the learning rate is too large and we overstep our minimum, we see the loss value "bouncing" up and down [4]. If the learning rate is too small, the loss value will not decrease meaningfully with each iteration [4]. We do this with data set 2 to use a non-linearly separable data set but still minimize the needed computing time.

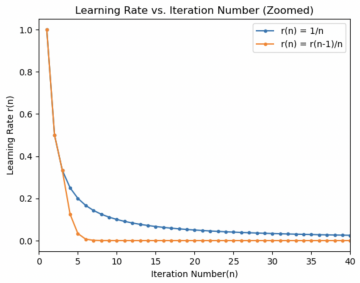

Fig. 1: The two methods for decreasing the learning rate for the Perceptron algorithm. The second drops off more quickly than the first.

3.2 Probability Distribution Update

Lastly, we look at how choosing from a probability distribution when selecting a given learner's training data subset impacts loss as the ensemble is constructed. By comparing the loss vs. number of learners at each iteration of the AdaBoost algorithm for subsets of data that are randomly selected and those selected from a probability distribution as described in subsection 2.2. As in our experiment with the learning rate, we utilize data set 2.

3.3 Number of Learners in the Ensemble

The first circumstance we want to test is how increasing the number of weak learners in the ensemble impacts the ensemble loss. Ideally, more learners will minimize loss (Equation 4) over the ensemble. However, we expect diminishing returns on this after some time. We create and test ensembles with 1, 5, 10, 15, 20, 30, 50, and 100 individual weak learners. We create 15 individual ensembles for each ensemble size and average their losses. While more time-consuming, doing so allows for strange values due to randomness to be averaged out nicely. We conduct these experiments on all four data sets.

4 Results

4.1 Plotted Data

Example outputs for our ensemble can be found in Figures 2, 3, 4, 5. Data set 1 shows an additional plot for a 5-learner ensemble. In each, the point color indicates the expected label, and the shape indicates whether or not the ensemble predicted the point correctly.

4.2 Learning Rate

An example result for the loss vs. learning rate throughout training a single weak learner on data set 2 for the two learning rate functions can be found in Figure 6.

The first learning rate method, $r_n = 1/(n-1)$, starts relatively flat before beginning to bounce around. While the second learning rate method, $r_n = r_{(n-1)}/(n-1)$, starts flat, then shows a clear, steep drop.

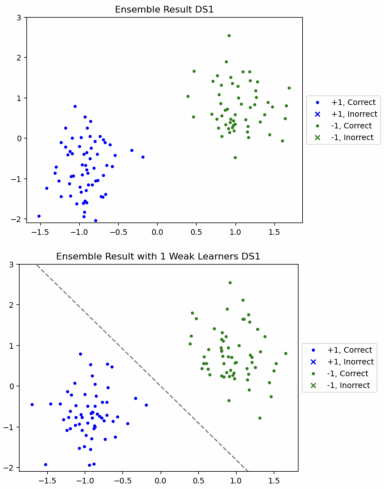

Fig. 2: Example outputs for the ensemble (top, 100 learners) and the base learner(bottom) for data set 1. Data point color indicates the correct label, and shape indicates if the ensemble predicted the label correctly. In the bottom plot, the grey dashed line represents the base learner.

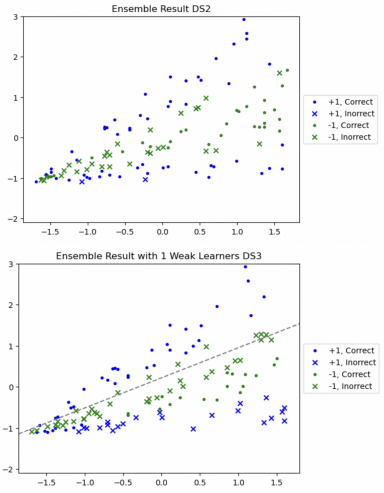

Fig. 3: Example outputs for the ensemble (top, 100 learners) and the base learner(bottom) for data set 2. Data point color indicates the correct label, and shape indicates if the ensemble predicted the label correctly. In the bottom plot, the grey dashed line represents the base learner.

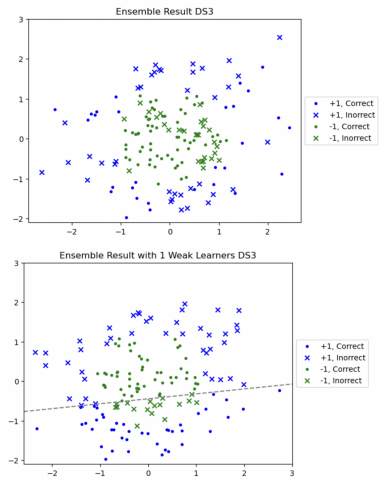

Fig. 4: Example outputs for the ensemble (top, 100 learners) and the base learner(bottom) for data set 3. Data point color indicates the correct label, and shape indicates if the ensemble predicted the label correctly. In the bottom plot, the grey dashed line represents the base learner.

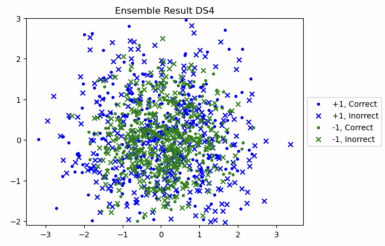

Fig. 5: Example ensemble outputs for 100 weak learners for data set 4. Data point color indicates the correct label, and shape indicates if the ensemble predicted the label correctly.n

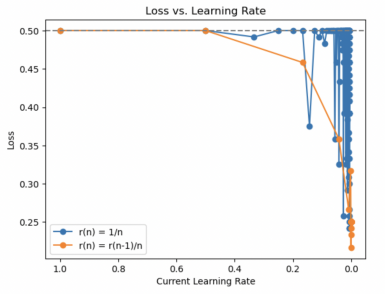

Fig. 6: The loss for the learner vs. the learning rate for the two learning rate functions outlined in subsection 3.1. $r_n = 1/(n-1)$ is in blue, and $r_n = r_{(n-1)}/(n-1)$ is in orange. Dots represent each iteration. Lines are shown to better express the data trends. The x-axis is inverted for ease of understanding so that later iterations are further right.

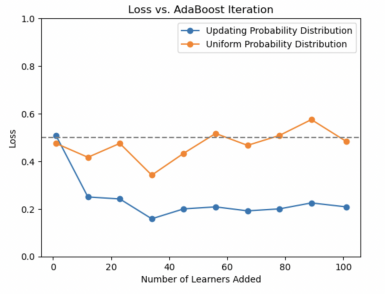

Fig. 7: The loss from the ensemble as it is is being built from our typical AdaBoost algorithm (blue) and by selecting the training data for each new learner with uniform probability (orange).

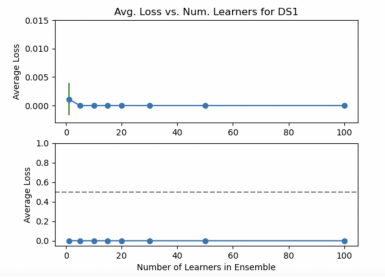

Fig. 8: The average loss vs. size for data set 1. Green lines represent the standard deviation of the 15 iterations at each value. The grey dashed line (bottom) represents the expected loss for random label predictions (.5). We have zoomed in near loss = 0 (top) to better show features.

4.3 Probability Distribution Update

We plot the loss as the ensemble forms in Figure 7 for data set 2. We can see that for the standard AdaBoost algorithm, which updates the probability distribution over the training data so incorrect points are more likely to be used in training, the loss initially decreases then plateaus as the ensemble is built. There is no clear trend for the algorithm that selects training data from a static uniform probability distribution.

4.4 Number of Learners in the Ensemble

We calculate the average loss over 15 iterations for varying ensemble sizes for each data set. The results can be seen in Figures 8, 9, 10, 11. For the first data set, most ensembles predict the values of the data with zero loss. The single learner is close but imperfect. For data sets two and three, the single learner incorrectly predicts labels at around 50% of the time. For both, the loss begins to drop as ensemble size increases before reaching a plateau. The second data set plateaus around 0.25, and the third around 0.4.

5 Conclusions

We have successfully implemented and tested a binary classifier machine learning code using linear halfspaces Perceptron weak learner and AdaBoost ensemble algorithms. A few essential results can be gleaned from these tests to better our code.

5.1 Learning Rate

First, we have demonstrated how different choices in learning rate can vastly impact Perceptron's

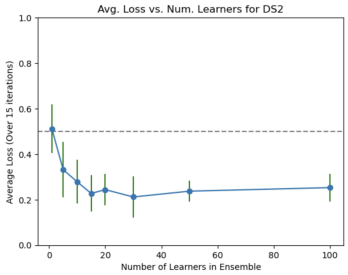

Fig. 9: The average loss vs. size for data set 2. Green lines represent the standard deviation of the 15 iterations at each value. The grey dashed line represents the expected loss for random label predictions (.5).

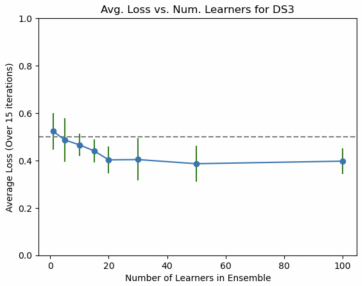

Fig. 10: The average loss vs. size for data set 3. Green lines represent the standard deviation of the 15 iterations at each value. The grey dashed line represents the expected loss for random label predictions (.5).

ability to minimize loss effectively. Our plot (Figure 6) clearly shows that for the learning rate, which drops off more slowly ($r_n = 1/(n-1)$), the loss for the learner begins to jump around wildly at later iterations, indicating the minimum is being overshot. As such, we chose to implement a learning rate function that drops off very quickly ($r_n = r_{(n-1)}/(n-1)$). By comparison, this learning rate does not miss the minimum nearly as quickly. Future research could delve into even more complicated ways of setting the learning rate.

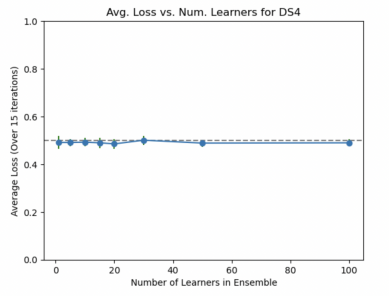

Fig. 11: The average loss vs. size for data set 4. Green lines represent the standard deviation of the 15 iterations at each value. The grey dashed line represents the expected loss for random label predictions (.5).

5.2 Probability Distribution Update

Second, we have shown clearly the importance of updating the probability distribution for the AdaBoost algorithm in Figure 7. As we can see, updating the probability distribution gives us our desired result, wherein the loss for the ensemble decreases as new learners are trained. In the bagging version, where the probability distribution is always static and uniform, the loss tends to hover around 0.45. We get much better performance through boosting. Updating the sample distribution in this way dramatically improves performance compared to the default.

5.3 Number of Learners in the Ensemble

Lastly, we have shown that the ensemble of weak learners works much more effectively than the single base learner in most cases. Even the results for our linearly separable data (data set 1) benefited from the ensemble (Figure 8). Figures 3 and 4 show the shortcomings of a singular linear halfspace learner on non-linearly separable data. However, Figures 9 and 10 show that, even for some nonlinear data, we can regularly achieve a loss below the random-assignment limit with an ensemble of linear halfspaces. That being said, even for many learners, we cannot achieve zero loss for these data sets. In addition,

data set 4 (Figures 5 and 11) further illustrates the shortcomings of this ensemble. This data is 10-dimensional, whereas all the others are 2-dimensional. It is clear from Figure 11 that the effects of nonlinearity become much more pronounced in higher-dimensional spaces. Even with many learners, we cannot predict the labels of data set 4 with greater accuracy than the random-assignment limit. This is not unexpected, given that the weak learner used is linear.

References

1. Anna Kononova, D.V.: Ensemble learning (October 2022), https://brightspace.universiteitleiden.nl/d2l/le/lessons/168895/topics/2121523
2. Anna Kononova, D.V.: Supervised learning 2 (October 2022), https://brightspace.universiteitleiden.nl/d2l/le/lessons/168895/topics/2113564
3. Beygelzimer, A., Pál, D., Szörényi, B., Thiruvenkatachari, D., Wei, C.Y., Zhang, C.: Bandit Multiclass Linear Classification: Efficient Algorithms for the Separable Case. arXiv e-prints arXiv:1902.02244 (Feb 2019)
4. Jordan, J.: Setting the learning rate of your neural network. (March 2018), https://www.jeremyjordan.me/nn-learning-rate/
5. McCormick, C.: Adaboost tutorial (December 2013), https://mccormickml.com/2013/12/13/adaboost-tutorial/
6. Wu, X.: Advanced machine learning lecture 7: Linear predictors (March 2019), https://www.eecis.udel.edu/~xwu/class/ELEG867/Lecture7.pdf