Concept of Adaboost

Adaboost is an ensemble learning method that implements boosting such that it trains a set of weak classifiers in a progressive manner such that each weak classifier tries to reduce the errors of misclassifying data from the previous weak classifier. In doing so, it combines the series of weak classifiers into a strong classifier. The way Adaboost go about reducing the errors and improving the accuracy of samples classification is by using the metric "weight" that is assigned to each of the samples in the training dataset. Let's say we have a base classifier that incorrectly classified a set of samples, in that, the weight of the samples that were misclassified in the base classifier will be boosted so that the subsequent classifier can focus on correctly classifying these misclassified samples.

The idea of a weak classifier is that it can only produce a subpar accuracy of at least 50 per cent and above whereas a strong classifier can produce an accuracy of at least 95 per cent and above.

The weak classifier in this case would be the Decision Stump. Decision Stump is a single level decision tree that has only one root node and two leaf nodes and has a maximum depth of 1. The reason why Decision Stump is considered as a weak classifier is due to the fact that it makes prediction or decision using only a single input feature or variable as opposed to a full-sized Decision Tree that can take advantage of more than one input features or variables to make a prediction or decision. As a result, it has a low accuracy rate.

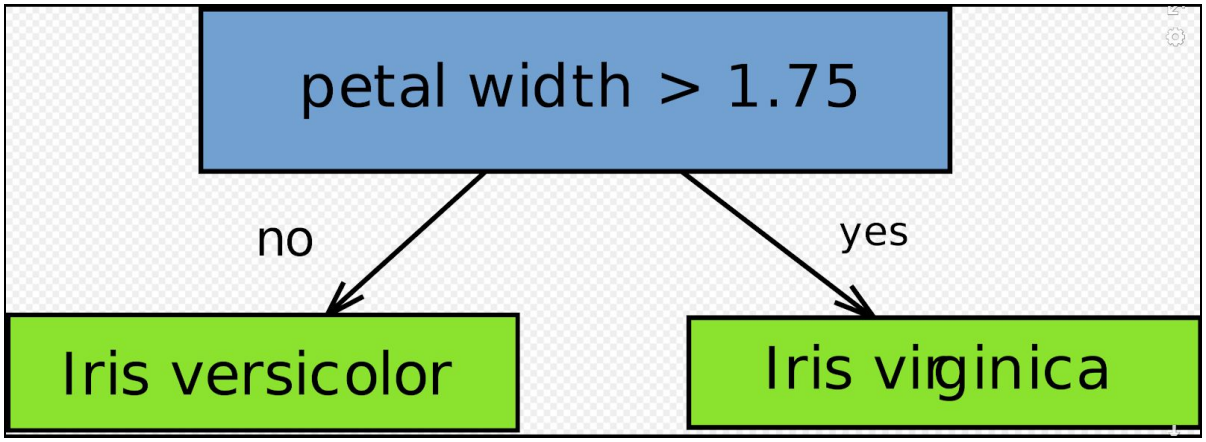

Figure 1: Decision Stump (Source = https://en.wikipedia.org/wiki/Decision_stump#/media/File:Decision_stump.svg)

Algorithmic Description of Adaboost

Step 1: We need to create a weak classifier (Decision Stump) from a given training set, $x_i \in X$ based on the weight of each of sample, $x_{i\ to\ n}$. The weight for each of the sample in the training set, $x_{i\ to\ n}$ indicates how important it is for each of them to be correctly classified. In the first

iteration, the weight for each sample in the training set, X is equal to 1/N. Therefore, the samples are equally as important as one another in the beginning.

$$D_1(i) = \frac{1}{N},$$

a) D_t *is the weight distribution for the samples in the training set* X

b) i *is the sample in the training set* X

c) N *is the size of the training set* X

Step 2: At this point, Adaboost algorithm will create a decision stump for each of the features in the given training set forming a forest of stumps and then we need to select the decision stump that best classifies the samples in the training set in the current iteration. The way we would go about choosing the best decision stump is by measuring and comparing the "information gain" of each of the decision stump based on the "**entropy**". The "**information gain**" give us the measure of how well a feature can split a dataset into, such that the resulting split datasets are as pure as possible meaning it doesn't have impurities like a mixture of the target feature values. So if the impurity of the resulting target values is very high then the entropy for that feature is very high as well and vice versa. The formula for calculating entropy is,

$$H(x) = -\sum_{for\ k \in target(true, false)} (P(x = k) * log_2(P(x = k)))$$

a) x *is the feature*

b) k *is the label value*

c) $P(x = k)$ *is the probablity that the label value of Mammal or Reptile takes a specific target value*

The formula for calculating information gain is

$$InfoGain(feature, D) = Entropy(D) - \sum_{t \in feature} \left(\frac{|feature=t|}{|D|} * H(feature = t)\right)$$

a) $feature$ *is the desciptive feature in the dataset*

b) D *is the current dataset*

c) $H(feature = t)$ *is the entropy for the feature*

For instance, say that we are given a dataset as shown in figure 2 below.

	toothed	hair	breathes	legs	species
0	True	True	True	True	Mammal
1	True	True	True	True	Mammal
2	True	False	True	False	Reptile
3	False	True	True	True	Mammal
4	True	True	True	True	Mammal
5	True	True	True	True	Mammal
6	True	False	False	False	Reptile
7	True	False	True	False	Reptile
8	True	True	True	True	Mammal
9	False	False	True	True	Reptile

Figure 2: Dataset (Source = https://www.python-course.eu/Decision_Trees.php)
-The species is the Label Y(Mammal and Reptile)
-The tooth, hair, breathes and legs are the feature X

As shown below in figure 3, the feature "legs" has the highest information gain out of all of the decision stumps that we have and so it is then selected as the decision stump for the current iteration.

Now we will calcuate the Information gain for each descriptive feature:

toothed:

$$H(toothed) = (\frac{8}{10}(-((\underbrace{\frac{5}{8} * log2(\frac{5}{8})}_{\text{toothed = True ; Mammal}}) + (\underbrace{\frac{3}{8} * log2(\frac{3}{8})}_{\text{toothed = True ; Reptile}}))) + \frac{2}{10}(-((\underbrace{\frac{1}{2} * log2(\frac{1}{2})}_{\text{toothed = False ; Mammal}}) + (\underbrace{\frac{1}{2} * log2(\frac{1}{2})}_{\text{toothed = False ; Reptile}})))$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\text{toothed = True}} \qquad\qquad \underbrace{\qquad\qquad\qquad\qquad}_{\text{toothed = False}}$$

= **0.963547**

$InfoGain(toothed) = 0.971 - 0.963547 =$ **0.00745**

breathes:

$H(breathes) = (\frac{9}{10} * -((\frac{6}{9} * log_2(\frac{6}{9})) + (\frac{3}{9} * log_2(\frac{3}{9}))) + \frac{1}{10} * -((0) + (1 * log_2(1)))) =$ **0.82647**

$InfoGain(breathes) = 0.971 - 0.82647 =$ **0.1445**

legs:

$H(legs) = \frac{7}{10} * -((\frac{6}{7} * log_2(\frac{6}{7})) + (\frac{1}{7} * log_2(\frac{1}{7}))) + \frac{3}{10} * -((0) + (1 * log_2(1))) =$**0.41417**

$InfoGain(legs) = 0.971 - 0.41417 =$ **0.5568**

Figure 3: The information gain value for all three feature (Source = https://www.python-course.eu/Decision_Trees.php)

Step 3: Determine how much influence or say a decision stump has in the final classification based on the total error incurred in the current iteration. The amount of say is equivalent to the alpha value of the following formula :

$$Alpha = \frac{1}{2} * (\frac{1-\varepsilon}{\varepsilon}) > 0$$

a) ε *is the total error rate which is equivalent to the sum of the weights for the misclassified samples*

The amount of say or alpha is large when the total error, ε is small and vice versa.

Step 4: Update the weights of the samples in the training set in the current iteration. If the current decision stump correctly classifies a training sample then we need to decrease its sample weight and if it incorrectly classifies a training sample then we need to increase its sample weight so that it can be correctly classified by the next decision stump.

$$D_{t+1}(i) = \frac{D_t(i)}{Sum(D_t)} * exp(-alpha_t * y_i * h_t(x_i)$$

a) D_t *is the weight of the t sample data*

b) $Sum(D_t)$ *is the normalization constant*

c) $-alpha_t$*is the amount of say of the current decision stump*

d) y_i *is the label of the training dataset* $(1,-1)$

e) $h_t(x_i)$ *is the prediction value of the current decision stump*

Step 5: Sum up all of the prediction value of the current iteration to produce the final classifier. For example, if the iteration is 3 that means we would have 3 different prediction value each for the decision stump that was created in each iteration. In this case, the decision stump 3 is created based off of the errors of decision stump 2 and decision stump 2 is created based off of the errors of decision stump 1. Since each of these decision stumps has their own prediction value, we can combine all three of their prediction values into a single prediction value to serve as the final classifier for iteration 3.

$$H_{final}(x) = sign(\sum_t alpha_t * h_t(x))$$

a) *sign here makes sure that the final prediction value is either* 1 *or* -1

b) $alpha_t$*is the amount of say of the t decision stump*

c) $h_t(x)$ *is the prediction value of the t decision stump*

Step 6: Repeat step 2 to step 5 until you have achieved a satisfying accuracy rate.

My Analysis

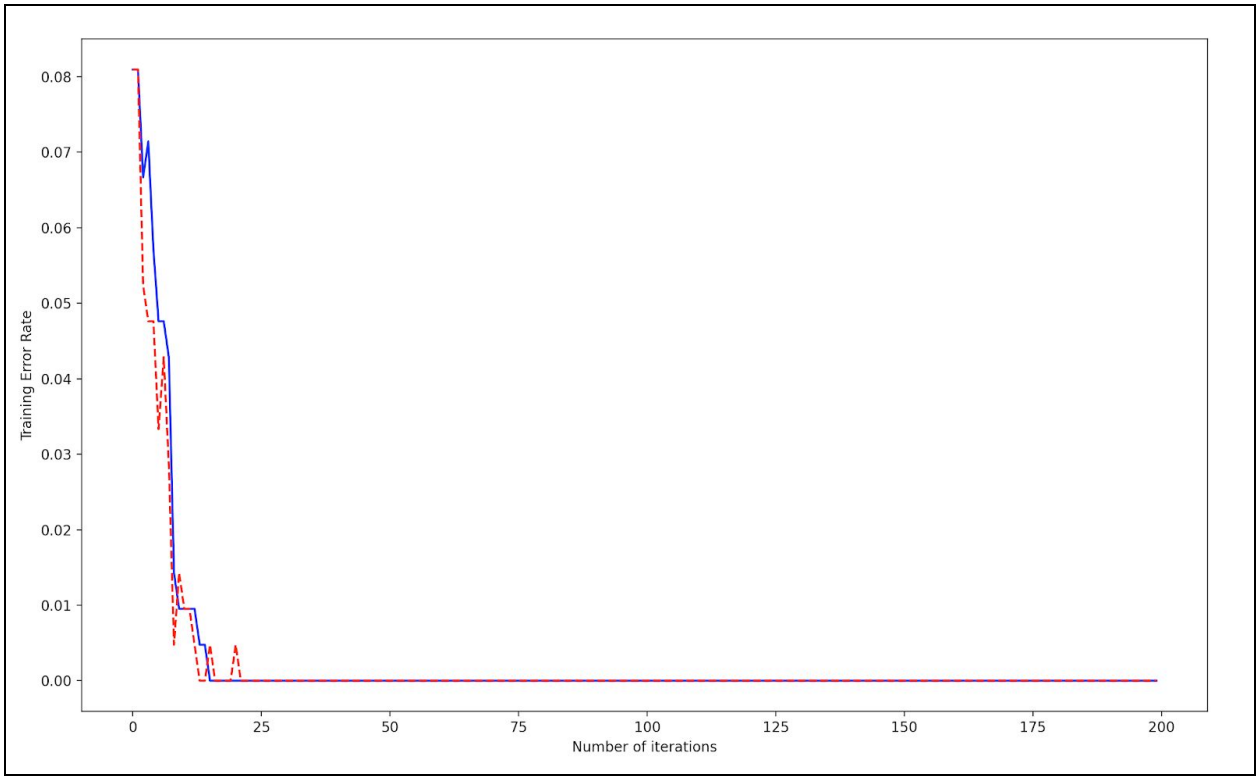

Figure 4: (Blue) My training error curve vs (Red) Sklearn Adaboost training error curve

Based on figure 4 above, it looks like my Adaboost implementation has a fairly consistent training error with that of the Sklearn Adaboost implementation. The training error in both implementations trails to zero at roughly the same rate and number of iterations. This is indicative that my Adaboost implementation is not faulty as it meets the expected value.

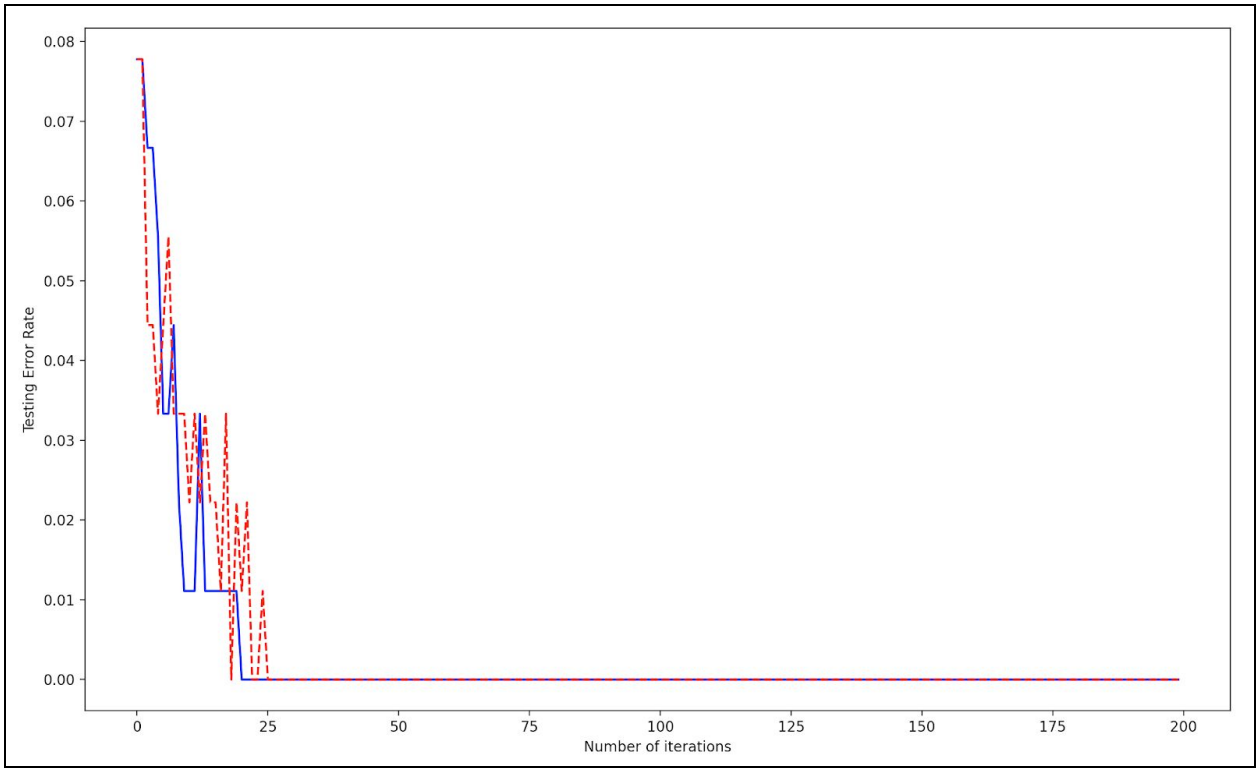

Figure 5: (Blue) My testing error curve vs (Red) Sklearn Adaboost testing error curve

Based on figure 5 above, again it looks like my Adaboost implementation has a fairly consistent testing error with that of the Sklearn Adaboost implementation. The testing error in both implementations trails to 0.00 at roughly the same rate and number of iterations.

Since both my testing and training error are fairly consistent with one another where both of them trails to zero at around the same rate and number of iterations, therefore, I conclude that my model is balanced where it is neither overfitting nor underfitting.

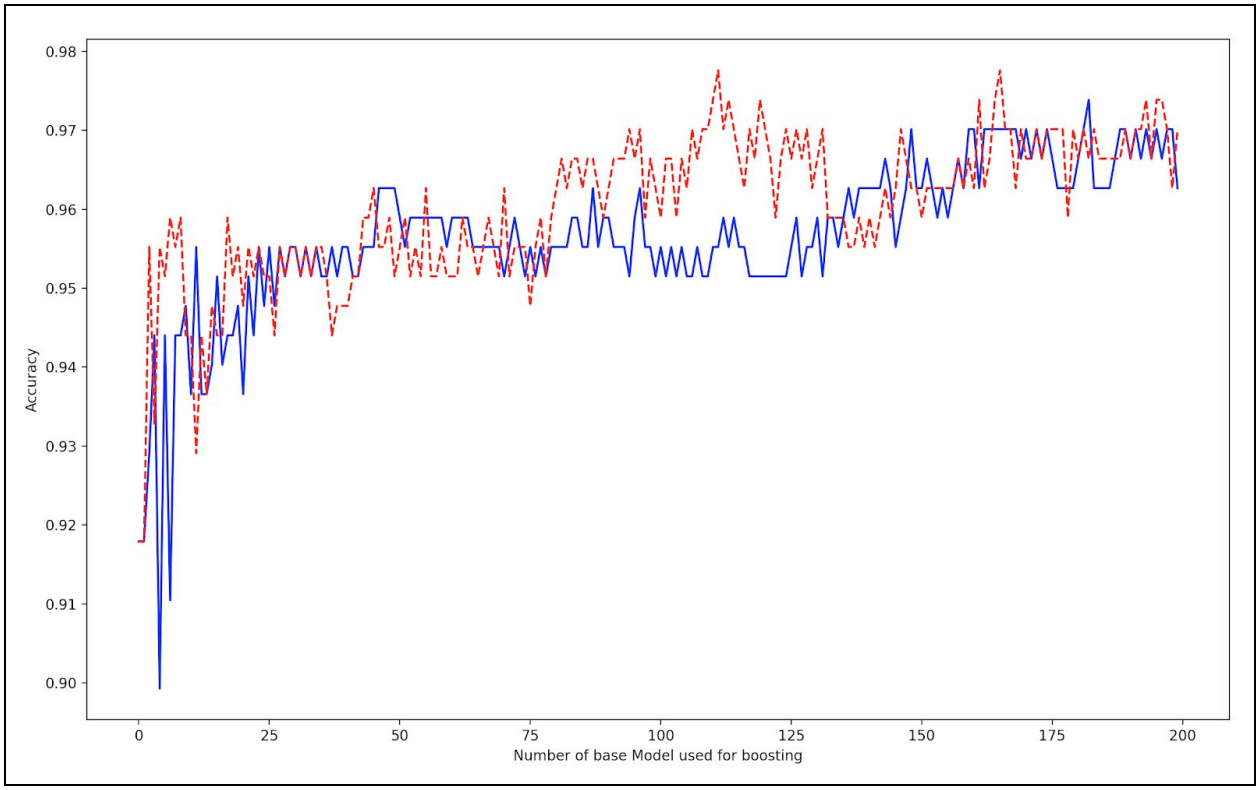

Figure 6: (Blue) My Adaboost performance/accuracy rate vs (Red) Sklearn Adaboost performance/accuracy rate

Based on figure 6 above, it looks like my Adaboost performance/accuracy rate matches that of the Sklearn Adaboost performance/accuracy rate although it differs slightly at around 0 to 10 and 75 to 135 number of the base model used. But the overall trend seems to suggest that they are both consistent with one another. Since both of the Adaboost performances/accuracies continue to increase throughout the increasing number of the base model used for boosting, I can conclude that it is correctly implemented.

(Interesting observation)

The zero testing and training errors seem to occur at around iteration 20 and above based off of my own and Sklearn Adaboost implementations. In theory, zero testing and training errors are indicative of an overconfident model that degrades in its test performance but that is not the case here with what is being observed in figure 6 below as the accuracy rate of my Adaboost implementation and the Sklearn Adaboost implementation seems to increase after iteration 20 which directly opposes with what is observed in figure 4 and 5 as those two observations lead to the conclusion of a balanced generalised model at around iteration number 20.