ABSTRACT

Cancer is one of the major causes of human death in the world and breast cancer is the most common form of cancer in women. It may be a deadly disease but if diagnosed early on, it can be treated. Machine learning is growing at a very fast pace which has led to a time where we can even use it to predict cancer. This field has expanded by combining different methods to maximise the accuracy of prediction. Boosting is a machine learning model where we combine many weak classifiers to obtain one high-performance prediction model. AdaBoost is the most popular boosting algorithm. Apart from AdaBoost, Support Vector Machines (SVMs) is another successful classification method which is essentially similar as they both try to maximize the minimal margin on a training set. We have tried to implement a learning algorithm on the Wisconsin Diagnostic Breast Cancer (WDBC) by measuring their classification test accuracy. For the implementation of our AdaBoost model, the first 300 samples were used for training and the rest for testing. As a result of the experiment, it was observed that AdaBoost is the most accurate model with error as low as 0. 1269.Support vector machine did a pretty decent job with an accuracy of 0.902 when we used a non-linear kernel.

INTRODUCTION

Boosting is a methodology in Machine Learning where the core principle is to make accurate prediction from weak and inaccurate models. Adaptive Boosting also known as AdaBoost, is the first boosting algorithm proposed by Freund and Schapire in 1996. The intuitive idea behind AdaBoost is that it combines predictions to come up with a final prediction. It does somewhat a similar job like Random forest but has a unique feature of Adapting (and thus explains it name). It gives higher weightage to the wrongly classified examples during the training process. Thus, it sequentially adapts for misclassified training samples. Therefore, it also known as sequential ensemble method. The weak classifiers in AdaBoost are called Decision stumps. Decision stump is simply a one level decision tree. It splits a data sample into two subsets based on some feature. To find the best fitted stumps, we try all the features and choose the one that gives the best accuracy.

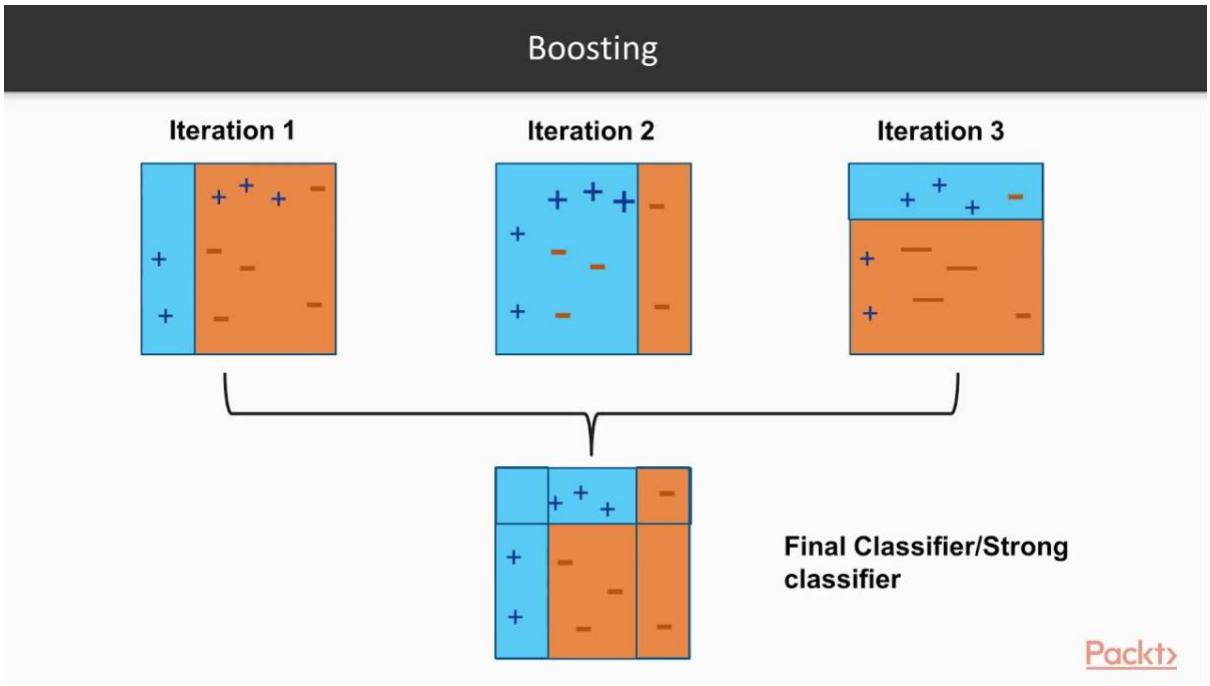

Figure 1: Intuitive idea on how AdaBoost works.

ALGORITHM AND EXPLANATION

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$.
- Update, for $i = 1, \ldots, m$:

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

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Figure 2: The boosting algorithm AdaBoost

Each training sample in the training dataset is weighted. The initial weight is set to:

weight(x_i) = 1/N

where x_i is the i_{th} training sample and N is the number of training samples.

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first- or second-class value.

The misclassification rate is calculated for the trained model. Traditionally, this is calculated as:

error = (correct – N) / N

where error is the misclassification rate, correct are the number of training sample predicted correctly by the model and N is the total number of training samples.

For example, if the model predicts 78 out of 100 training samples correctly the error or misclassification rate would be (78-100)/100 or 0.22.

This is modified to use the weighting of the training samples, which is the weighted sum of the misclassification rate:

error = sum(w(i) * training_error(i)) / sum(w)

where w is the weight for training sample i and training_error is the prediction error for training sample i which is 1 if misclassified and 0 if correctly classified.

For example, if we had 3 training samples with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the training samples were -1, 1 and -1, then the training_errors would be 0, 1, and 0. The misclassification rate would be calculated as:

error = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2)
or
error = 0.704

An alpha value is calculated for the trained model which provides a weighting for any predictions that the model makes. The alpha value for a trained model is calculated as follows:

alpha = ln((1-error) / error)

where alpha is the alpha value used to weight predictions from the model, ln is the natural logarithm and error is the misclassification error for the model.

The effect of the alpha weight is that more accurate models have more weight or contribution to the final prediction. The training weights are updated giving more weight to incorrectly predicted training samples, and less weight to correctly predicted training samples.

The weight of one training sample (w) is updated using:

w = w * exp (alpha * training_error)

where w is the weight for a specific training sample, exp is the numerical constant e or Euler's number raised to a power, alpha is the misclassification rate for the weak classifier and training_error is the error the weak classifier made predicting the output variable for the training sample, evaluated as:

if (y == p),
training_error = 0
otherwise 1

where y is the output variable for the training sample and p is the prediction from the weak learner.

This has the effect of not changing the weight if the training sample was classified correctly and making the weight slightly larger if the weak learner misclassified the training sample.

IMPLEMENTATION

Machine Learning methods have been instrumental in detecting a disease. In this experiment we have been provided with Wisconsin Diagnostic Breast Cancer (WDBC) data set. This data set contains 569 data samples and 32 features. The objective of this experiments is to train an AdaBoost model and predict between benign and malignant on the test data.

Implementation of AdaBoost using Freund and Schapire algorithm:

The method *def __init__* has been used for initialising the weak learners and the weights.

The method *def train* is the actual method that trains the model and it calls method *def add_learner* for updating the data weighting coefficient for next learner. It also calls def predict to make the prediction.

The *def prediction_error* is used to calculate the error in our prediction and *def plot_error_rate* is used to plot a graph between the number of iterations and error.

Implementation of Decision Stumps:

We have implemented a simple decision stump classifier.

Here, *dim* is used for dimension on which to split, value for the value of dimension and *op* for the comparator function to use while comparing with the value of the dimension.

def fit data is used for finding an optimal dimension and the value of that dimension which will best split the given data depends on the split that gives the minimum error. Here X is a n x d data matrix, n number of samples with d dimension, Y is a n dimensional array containing label of each observation, label = {-1,1}, sample_weights is used for the weight of each observation and num_splits is used for the number of split value to be tested randomly.

The *def fit_dim* fits a one-dimensional Decision stump classifier and finds the optimal value of a particular dimension which results in the best split i.e. if split is performed on that dimension, then what value of the dimension will result in the best split. This function is called by the *def fit_data* for every dimension to find the optimal dimension.

EXPERIMENTS AND RESULTS

The data set provided is applied to the AdaBoost model that was implemented. Out of 569 data samples, 300 samples were used for training and rest were used for testing.

The initial training error was 0.1533 and testing error was 0.1157 while the final training error was 0.1033 and the testing error was 0.1269.

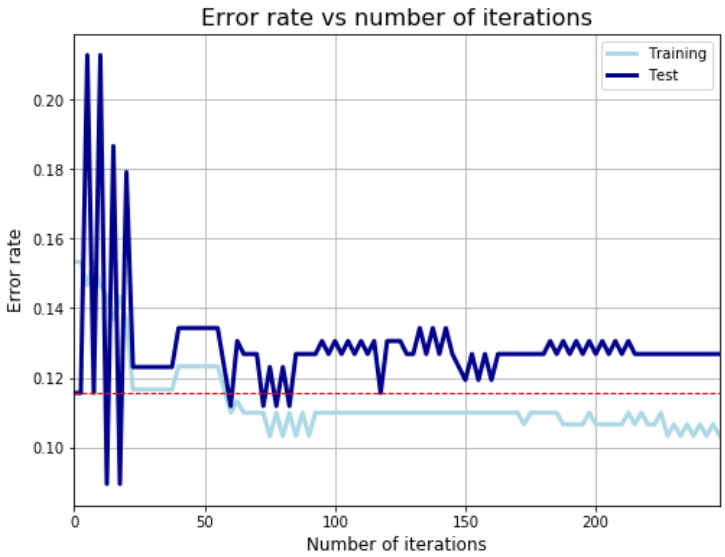

Figure 3: Error rate Vs Iteration plot

The above plot demonstrates the training and test error rates obtained using the AdaBoost model on our dataset with the decision stumps as the base learner. The top and bottom curves are test and training error, respectively. It can be observed that the adaboost model very quickly drives down the training error. The model performs very well on test data. Thus, we can appreciate a significant reduction in the error rate as we increase the number of iterations. It can also be observed that the test error goes down after we already have zero training error. An explanation for this is that even when the training error is zero, the margin can still be improved by further boosting iterations. In other words, adding classifiers tends to increase the margin size. This practically makes it robust to Overfitting. It is nonetheless possible to overfit with AdaBoost, by adding too many classifiers. The solution that is normally used practice is a procedure called early stopping.

The second experiment conducted was to compare the accuracy between the adaboost algorithm and SVM. The prediction accuracy for SVM was 0.87313.AdaBoost with decision stumps as weak learners outperforms linear SVMs (depends on the dataset ,not a general rule).An explanation for this would be that the AdaBoost can learn non-linear decision boundaries, as the our hypothesis function uses a Sigmoid function which is a non-linear function, thus giving better results especially if your data cannot be linearly separated. When the SVM uses non-linear kernel, it classify the data better and gives a better accuracy of around 0.902.

We have also trained our data with some inbuilt ensemble model like AdaBoostClassifier and GradientBoostingClassifier. We get an almost similar prediction accuracy of 0.8731in both of them. Both the boosting algorithms learn from previous errors and make a weighted sum at the end. Both of them are pretty similar except their loss function. From the results, it can be said the inbuilt ensemble models has lesser accuracy.

Finally, we have compared the SVM model with gradient boosting using ROC curve. AUC–ROC curve is the model selection metric for classification problem. ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability. The area covered by the curve is the area between the blue line (ROC) and the axis. The larger the area covered, the accurate the machine learning models is at distinguishing the given classes. In the experiment conducted, it can be observed that SVM performs slightly better than Gradient Boosting.

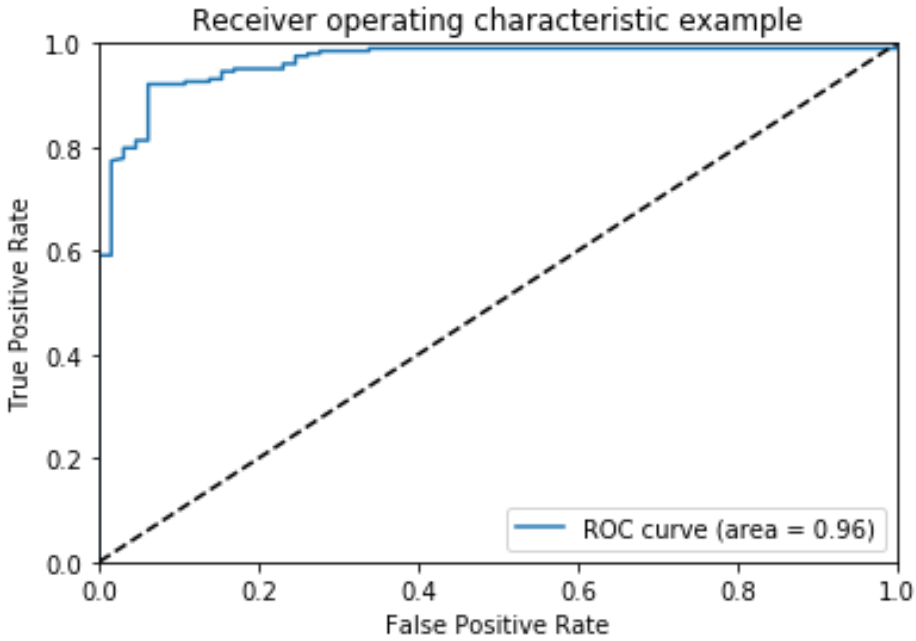

Figure 4: ROC curve for Gradient Boost

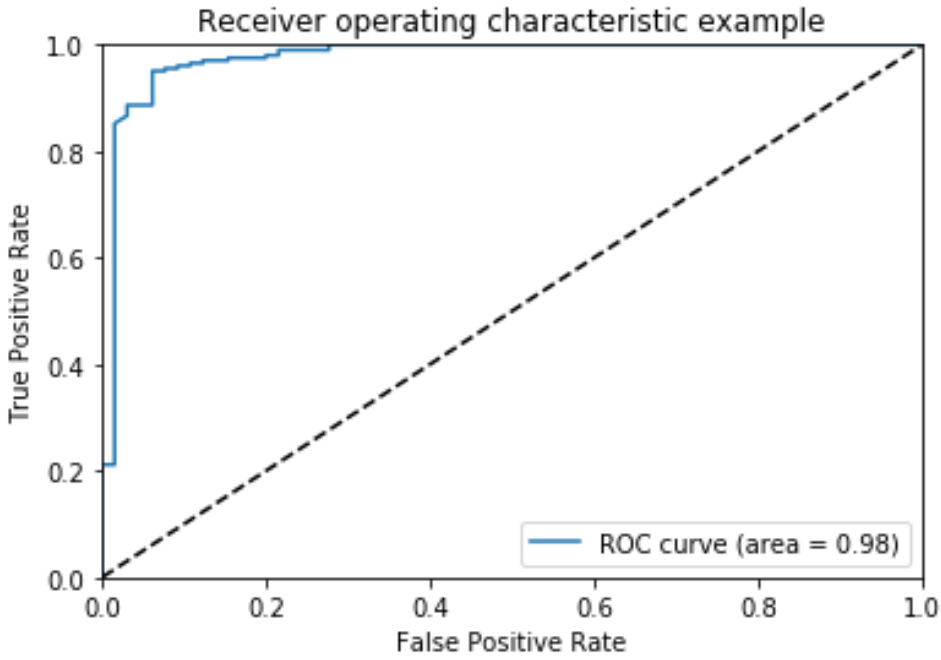

Figure 5: ROC curve for SVM

CONCULSION

In this study, we used the Wisconsin Diagnostic Breast Cancer (WDBC) dataset to investigate the most successful breast cancer classification model. AdaBoost, SVM and Gradient Boosting were used in the classification.

It can be concluded that that combining weak classifiers can give a very strong classifier. The resulting strong classifier can eventually provide zero training error. Also, boosted model with decision stumps are very easy to implement and very effective in comparison to SVM. Although, SVM gives pretty much accurate predictions for linear data, Boosting has an edge when it comes to classify non-linear dataset. On comparing inbuilt ensemble methods like AdaBoost and Gradient Boost, we found pretty similar accuracy. From the aforementioned comparisons, the AdaBoost model showed the best classification performance in terms of accuracy and speed as we are using Decision stumps as a weak classifier.

REFERENCES

E. Schapire, R 2013, Explaining AdaBoost, pp. 37–52.

Robert E. Schapire, Yoav Freund, et al. Boosting the margin: A new explanation for the effectiveness of voting methods.

S Wang, Boosting Methods, Lecture Notes, Cmput466/551, delivered 30 March 2005

Tan, Yandan & Zhao, Guangcai & Dai, Houde & Lin, Zhirong & Cai, Guoen. (2018). Classification of Parkinsonian Rigidity Using AdaBoost with Decision Stumps. 170-175. 10.1109/ROBIO.2018.8665303.

J, Haebchan, 2018, Adaboost for Dummies: Breaking Down the Math (and its Equations) into Simple Terms, Towards DataScience, viewed 30 September 2019
https://towardsdatascience.com/adaboost-for-dummies-breaking-down-the-math-and-its-equations-into-simple-terms-87f439757dcf

MrWayne, AdaBoost: Why does test error decrease even after training error hits zero? viewed 3 October 2019
https://stats.stackexchange.com/q/171128

Guest Contributor, Understanding ROC Curves with Python, viewed 3 October 2019
https://stackabuse.com/understanding-roc-curves-with-python/