# AdaBoost Algorithm using Gini-Index method for Decision Stumps

Name: B Sai Sahana Bhargavi
username on miner : word2vec

1.Introduction: Implemented AdaBoost algorithm through Gini index based approach for learning decision stumps for encoding of the problem of recognizing handwritten digits,and to distinguish between the digits 3 and 5 of NIST dataset .

Approach:

2.Data extraction: Read the training dataset containing 1214 records where the first column in the training dataset is the label either 3 or 5, and following 256 continuous features are a pixel encoding and extracted the testing dataset of 326 records.
Training dataset is further split into training and validation data sets for comparing training and estimated testing error. Where X denotes the set of records containing features and Y denotes the class labels 3 or 5.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=15)
```

The test file data is read into the `X_test_file` data frame.

3.Decision Stump using Gini-index as weak learners:

Gini-index: `def Gini_index(groups, classes)`: Gini-score determines the node impurity for the given set of groups (left and right) and class labels.
Below formulas are used to compute the respective proportions and gini_index.
proportion = count(class_value) / count(rows)
gini_index = sum(proportion * (1.0 - proportion))
gini_index = 1.0 - sum(proportion * proportion)

Weighted gini index for each group is calculated as follows relative to all the parent samples
gini_index = (1.0 - sum(proportion * proportion)) * (group_size/total_samples)

The scores are then added across each child node at the split point to give a final Gini score for the split point that can be compared to other candidate split points.

Test-split: `def test_split(index, value, X)`: for each row of the training dataset, if the attribute(index) is less or greater than the split value then that attribute is shifted to left or right groups accordingly.

Decision stump: `def decisionStump(X,Y):based upon` the groups(left and right) returned from test_split() and the gini-score from Gini_index() functions , it creates a dictionary that contains index of the selected attribute used for best split, the value to be used for comparing the each attribute to get classified, the two left and right groups, left and right class values.

4.Adaboost algorithm:

`def theta_init(X):` initialized the each record in the dataset to same weight of 1/len(X).

The below steps from 1 to 6 run for the 200 - 250-300 iterations.

1. created new training data set **x1** using sample data (with replacement) from original X_tr according to 'theta' weights.

2.1 With stump: trained a base classifier using the decisionstump() created as mentioned above.

2.2 without stump:

```
DecisionTreeClassifier(criterion='gini' max_depth = 1, random_state = 1)
 pred_train_i = clf.predict(X_tr) (for training data)
 pred_test_i = clf.predict(X_tst) (for testing data)
 pred_test_file = clf.predict(X3) (for testing data of file)
```

3. applied the base classifier to all the instances to the original training data set X_tr, testing (validation) data set X_tst and testing file data set X_tst_file and the respective class labels are determined from the predict(node,row) function.

`def predict(node, row):` the base classifier returned from the decision stump acts as a node and checks if the child node is terminal or if it is a dictionary node containing another level of tree to be considered and thus classifies the each attribute into either 3 or 5.

4. calculates the error rate value

err1=classified_err(pred,Y)*theta

where classified_err(pred,Y) = sum(pred != Y) / float(len(Y))

err1=np.sum(err)

5. If the error value (err) calculated is greater than 0.5 then the weights are intialised again using def theta_init(X) and the procedure starts from step 1. else, weights of the each instance are updated through `theta_normalize(X,Y,theta,predict,alpha)` using below formula.

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \begin{cases} \exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

where $Z_i$ is the normalization factor

6. computes the importance of algorithm alpha 0.5 * log((1-err)/(err))

7. final prediction is done by obtaining the sign of the weighted sum of final predicted value.

Final prediction | sign (weighted sum) = $\sum (\alpha_i * (\text{predicted value at each iteration}))$

5.Decision tree classifier using sklearn:

defined decision tree model for the given data set using

```
from sklearn.tree import DecisionTreeClassifier
```

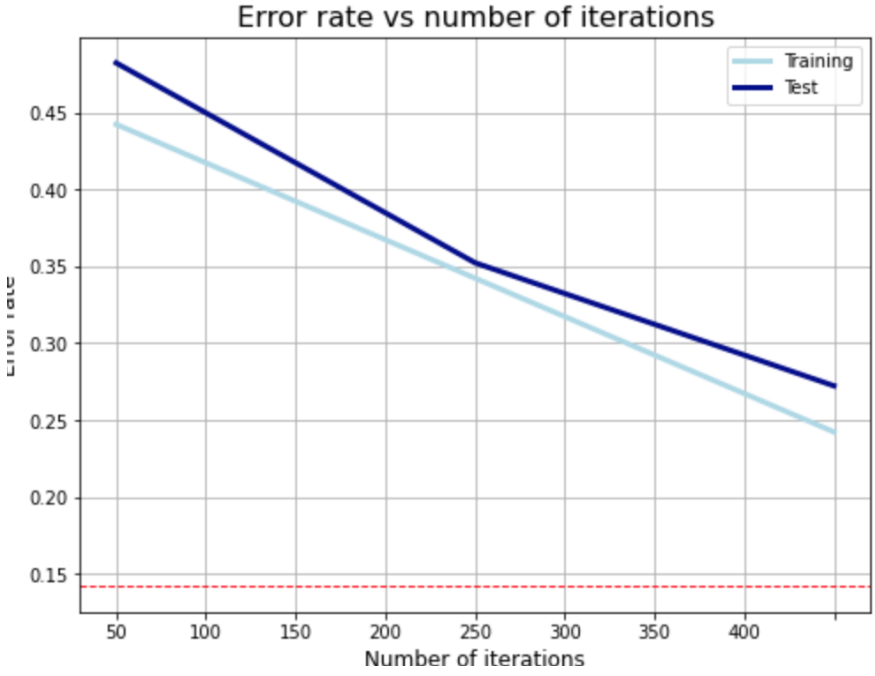
clf_tree = DecisionTreeClassifier(max_depth = 1, random_state = 1)

Computes the training error, testing error and returns the predicted outputs for the test data file

er_tree = generic_clf(Y_train, X_train, Y_test, X_test, clf_tree,X_test_file)

| | Adaboost with gini-based stump | Adaboost with Sklearn decision stump | Sklearn Decision tree |
|---|---|---|---|
| Training error | 0.243 | 0.1623 | 0.1623 |
| Testing error | 0.153 | 0.144 | 0.144 |
| Estimated testing error on miner | 0.35 | 0.15 | 0.15 |

On plotting the graph for obtained the training error and estimated testing error it can be observed that the values are decreasing as the number of iterations increase when tried running for 200,250,300 iterations, the dotted red line represents estimated testing error for sklearn based decision tree, where it shows constant performance in each set of iteration.
As Adaboost uses weak classifiers as they are weak, it can lead to low margins and overfitting, which can be seen in the graph as there is slight difference between training error and testing error.




6.Conclusion: AdaBoost aids in the selection of the training set for each subsequent classifier that is trained using the preceding classifier's results. It also defines how much weight should be given to each classifier's recommended response while merging the results. It combines weak learners to produce a strong one to correct classification errors. In Gradient Boosting(GB), the

shortcomings of weak learners are identified with gradient computation technique when compared to high-weighted datapoints technique in Adaboost.  While, adaboost is confined to binary classification problems, GB is for both classification and regression problems.