

CSE 601 Data Mining and Bio Informatics

Project 3

**Classification Algorithms**



**University at Buffalo**

*The State University of New York*

Falguni Bharadwaj - 50163471

Malavika Reddy Tappeta - 50169248

Debika Dutt – 50170009

## Introduction

In this project, we implemented three different classification algorithms Nearest Neighbor, Decision Tree and Naïve Bayes. We also implemented Random Forest and Boosting based on the implementation of Decision Tree.

There are two forms of data analysis that can be used for extracting models describing important classes or to predict future data trends. These two forms are:

1. Classification
2. Prediction

## What is Classification?

Classification consists of predicting a certain outcome based on a given input. In order to predict the outcome, the algorithm processes a training set containing a set of attributes and the respective outcome, usually called goal or prediction attribute. The algorithm tries to discover relationships between the attributes that would make it possible to predict the outcome. Next the algorithm is given a data set not seen before, called prediction set, which contains the same set of attributes, except for the prediction attribute – not yet known. The algorithm analyses the input and produces a prediction. The prediction accuracy defines how “good” the algorithm is.

Following Metrics are used for performance evaluation:

	PREDICTED CLASS		
		Class=Yes	Class=No
	ACTUAL CLASS		
	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+TN}$$

$$\text{F1-measure} = \frac{2*TP}{(2*TP)+TN+FP}$$

# 1. Nearest Neighbor

## **Algorithm (for n fold cross validation)**

1. Split the data into n partitions
2. Iterate n times where in use each of the n partitions as test data and the remaining n-1 partitions as training data
3. In each iteration find the distance between each of the test data sample from each and every training data sample and find the top k nearest neighbors based on the distance.
4. Find the majority of points that belong to a particular class and classify the testing point as belonging to that class
5. Compare the predicted class value obtained and the actual class values for each testing data point
6. Calculate the performance by evaluating Accuracy, Precision, Recall, and F-1 measure for each of the n iterations

## **Implementation**

For this algorithm we first load the input CSV file and read each line of the input file using the `readLine()` function. For each line we split the words using the comma separator, convert it into double values and store it in an array. In the case of Dataset2 for feature values present and absent we replace them with 1 and 0 respectively. These are the attributes/ features of the data sample. We then store the features in an `ArrayList` and then create a `HashMap<Integer, ArrayList<Integer>>` called 'data' where we store the <Key,Value> pairs as the sample ID(in this case we take the line number) and the `ArrayList` of features of that data sample. We store all the data samples of the file in the above format. We also store the class values of each sample in a `HashMap` `actualClass` where the key is the sample ID and value is the class it belongs to.

Next, we iterate over the dataset n times for n cross validation. In every iteration we first split the data into n partitions by using  $(\text{size of dataset}/n)$  as the size for each partition and then for each iteration the testing data will be one of these partitions eg. for iteration 1 the testing data is row 1 to n and the training data is the rest of the samples i.e n+1 to the last row. For this we maintain 2 `ArrayLists` `training` and `testing` which stores the sample ids of their corresponding data.

After the data is split into training and testing data we iterate over the testing `ArrayList` to find the distance from the testing sample `ts` and all the training samples. For this we call the function `euclidean()` where we calculate the euclidean distance between `ts` and all the training samples. Here, we look up the `HashMap` 'data' to find the attribute values of the testing sample and each of the training samples. We iteratively calculate the euclidean distance between `ts` and each of

these training samples and then store all the distances in an array 'distance' and also in a HashMap 'neighboursDist' where key is the distance and value is the training sample ID. Euclidean distance for 2 samples is calculated as  $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_p - y_p)^2}$  where  $x_1$  to  $x_p$  are the attributes of testing sample and  $y_1$  to  $y_p$  are the attributes of training sample. Once we have the distance array that contains all the distances we sort it in ascending order. Then the first  $k$ , i.e. the number of nearest neighbors to be considered (value can be varied), values of the array are considered the closest neighbors. We find the sample IDs of these distances by looking up the HashMap 'neighboursDist' and return these  $k$  IDs.

Next, we send these  $k$  nearest neighbors of testing sample  $ts$  to the function `findClass()` to determine which class  $ts$  belongs to. In this function we keep a count of how many of the nearest neighbors belong to class 0 and how many belong to class 1. We find this by looking up the `actualClass` HashMap for each of the nearest neighbors and get the corresponding class value. Out of the count for class 0 and 1, whichever count is higher we assign that class to the testing sample  $ts$ . This testing sample and its predicted class values are stored as keys and values respectively in the HashMap `predictedClass`.

Finally, we calculate the performance metrics. We do this by calling the `calculatePN()` function. In this function we compare the values of predicted class and the actual class for each of the testing samples. If both values for a particular sample are 1 then we increment the TruePositive (TP) value, if both values are 0 then we increment TrueNegative (TN). If predicted class is 1 and actual class is 0 we increment FalsePositive (FP) and if predicted class is 0 and actual class is 1 we increment FalseNegative (FN). Then we calculate the Accuracy  $[(TP + TN) / (TP + TN + FP + FN)]$ , Precision  $[TP / (TP + FP)]$ , Recall  $[TP / (TP + FN)]$  and F1-measure  $[(2 * TP) / ((2 * TP) + TN + FP)]$ .

We implement the above steps for each iteration of  $n$  cross validation and then calculate the average accuracy, precision, recall and f1-measure values.

### Parameter setting

There are 3 parameter settings in this algorithm i.e.  $k$ ,  $n$ -fold cross validation and distance. For distance calculation we have many options like hamming distance, euclidean distance, Manhattan distance, etc. We choose euclidean distance as our metric for this implementation. We can use any value  $n$  for  $n$ -fold cross validation and based on the performance of different values we can select the best value for  $k$ .

### **Pros and Cons**

This algorithm is easy to implement and can handle multi-class data/cases. It makes no assumptions about the data and the accuracy is pretty high. Disadvantages are that it is computationally expensive as it needs to compute the distance of each test sample with every training sample given. Also, it more space is needed to store all the training samples. It is sensitive to outliers.

## Results

The results obtained for 10 fold cross-validation when  $k = 9$  for dataset1 are as shown below. We observe that for different values of  $k$  and  $n$  in  $n$ -fold cross validation the accuracy varies from 0.91 to 0.93.

```
Precision: 0.95238096
Recall: 0.3773585
F1-measure: 0.5405405

-----Performance Metrics for Cross-validation 6-----
Accuracy: 0.91071427
Precision: 0.9444444
Recall: 0.33333334
F1-measure: 0.49275362

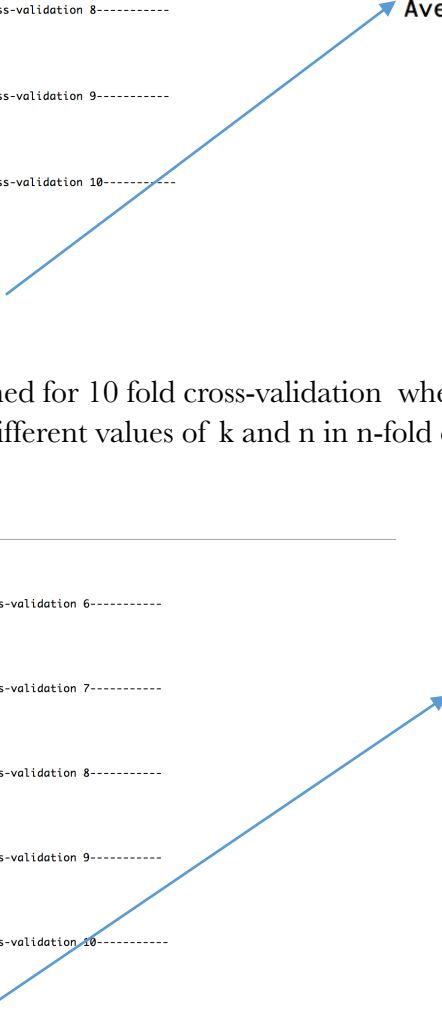
-----Performance Metrics for Cross-validation 7-----
Accuracy: 0.9285714
Precision: 0.9230769
Recall: 0.46153846
F1-measure: 0.61538464

-----Performance Metrics for Cross-validation 8-----
Accuracy: 0.9464286
Precision: 0.9285714
Recall: 0.24528302
F1-measure: 0.3880597

-----Performance Metrics for Cross-validation 9-----
Accuracy: 0.9464286
Precision: 0.9285714
Recall: 0.49056605
F1-measure: 0.6419753

-----Performance Metrics for Cross-validation 10-----
Accuracy: 0.89285713
Precision: 0.9166667
Recall: 0.44
F1-measure: 0.5945946

Average Accuracy = 0.93035716
Average Precision = 0.93400896
Average Recall = 0.3556942
Average F1 measure = 0.50697243
```



Average Accuracy = 0.93035716  
Average Precision = 0.93400896  
Average Recall = 0.3556942  
Average F1 measure = 0.50697243

The results obtained for 10 fold cross-validation when  $k = 9$  for dataset2 are as shown below. We observe that for different values of  $k$  and  $n$  in  $n$ -fold cross validation the accuracy varies from 0.60 to 0.66.

```
Precision: 0.625
Recall: 0.18518518
F1-measure: 0.2857143

-----Performance Metrics for Cross-validation 6-----
Accuracy: 0.5652174
Precision: 0.41666666
Recall: 0.1923077
F1-measure: 0.2631579

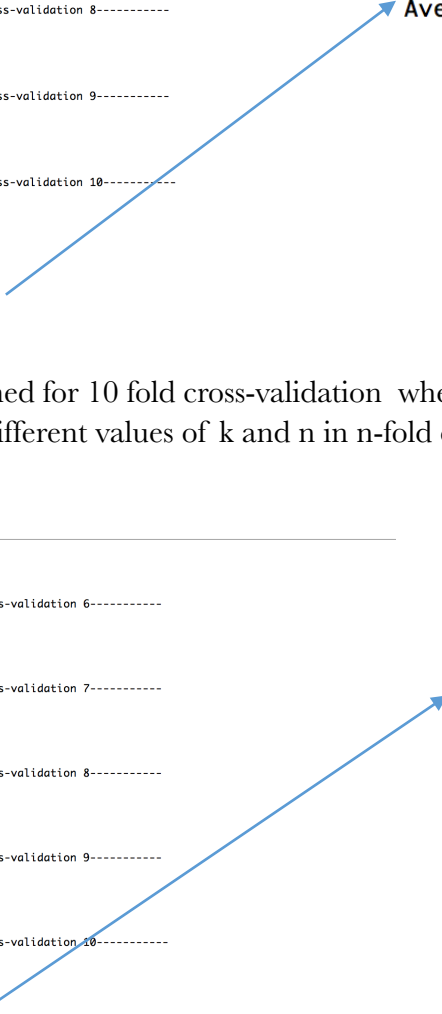
-----Performance Metrics for Cross-validation 7-----
Accuracy: 0.73913044
Precision: 0.54545456
Recall: 0.1764706
F1-measure: 0.26666668

-----Performance Metrics for Cross-validation 8-----
Accuracy: 0.82608694
Precision: 0.625
Recall: 0.13157895
F1-measure: 0.2173913

-----Performance Metrics for Cross-validation 9-----
Accuracy: 0.5869565
Precision: 0.36363637
Recall: 0.14814815
F1-measure: 0.21052632

-----Performance Metrics for Cross-validation 10-----
Accuracy: 0.6086956
Precision: 0.25
Recall: 0.071428575
F1-measure: 0.11111111

Average Accuracy = 0.6652174
Average Precision = 0.5255128
Average Recall = 0.17658126
Average F1 measure = 0.26217213
```



Average Accuracy = 0.6652174  
Average Precision = 0.5255128  
Average Recall = 0.17658126  
Average F1 measure = 0.26217213

## 2. Decision Tree

### **Algorithm**

GrowTree(D, F) – grow a feature tree from training data.

Input : data D; set of features F.

Output : feature tree T with labelled leaves.

if Homogeneous(D) then return Label(D) ;

S = BestSplit(D, F) ;

split D into subsets  $D_i$  according to the literals in S;

for each i do

if  $D_i$  not empty then  $T_i = \text{GrowTree}(D_i, F)$  else  $T_i$  is a leaf labelled with Label( $D_i$ );

end

return a tree whose root is labelled with S and whose children are  $T_i$

BestSplit(D, F) – find the best split for a decision tree.

Input : data D; set of features F.

Output : feature f to split on.

$I_{\min} = 1$ ;

for each  $f \in F$  do

split D into subsets  $D_1, \dots, D_l$  according to the values  $V_j$  of f;

if  $\text{Impurity}(\{D_1, \dots, D_l\}) < I_{\min}$  then

$I_{\min} = \text{Impurity}(\{D_1, \dots, D_l\})$ ;

$f_{\text{best}} = f$ ;

end

end

return  $f_{\text{best}}$

### **Implementation**

Implemented decision tree by following k-fold cross validation by dividing the dataset in this way  $\text{test} = \text{data}[j:q,:]$  where j and q= vary from 0 to n at the interval of “divider” where divider = number of rows in dataset divided by “k”. This happens k times. Our training set is the difference of original dataset with test set.

We have a “DecisionTree” class which has attributes like dataset (what dataset this node is storing), isLeaf (which is true only when a node is leaf), parentNode (keeps the parent node of

this node stored), column (this is the feature on which node is split), children (stores all the children after splitting).

We start the algorithm by building the tree from function “startTreeBuild” whose parameters are dataset, empty list[] and number of columns. Inside this, we check if the label of all the elements in the dataset of this node are exactly same. If yes, we assign this node as leaf node. If not, we proceed to split the data.

For this, we iterate through every column and call the function which calculates information gain for each of them. Inside that function, if the column values are nominal, then we store them in a dictionary “features\_dict”. For each item in dataset, we check and update out features\_dict with the count of those features. For example, in dataset4, first column’s features dict was of the form: {'overcast': 4, 'sunny': 4, 'rain': 4} for train dataset with k value as 5. Say T1: overcast, T2: sunny and T3: rain.

Now we just iterate our dataset and each time calculate the individual entropy using the formula

```
entropy = -1 * first * log_first - second * log_second
where first = float (class1_count) / float (total_count)
      second = float (class2_count) / float (total_count)
      log_first = np.log2(first)
      log_second = np.log2(second)
```

to find the information gain of that particular column.

total = T1+T2+T3

Info(column1) = T1/total \* entropy(T1) + T2/total \* entropy(T2) + T3/total \* entropy(T3)

This way we calculate it for every column and then subtract it from org\_info which stores the original information gain of the data without splitting.

We split that node based on the highest value of information gain difference. That node dataset is split into splitFeature list and each separate entry is assigned to a new child node and a tree is built on this child node. This whole process is repeated till encountered with a leaf node.

This way, our tree stores the dataset & column split values and we use this to predict the label for test cases. We start with the root.column and find if label exists, if not, we move to the roots children and continue the process till we find the label. Then we check if its the same as actual label or not. We calculate the performance metrics Accuracy, Precision, Recall and F1-measure using the above labels.

## **Pros and Cons**

Its advantages are its interpretability i.e its output can be expressed in a readable format. They are applicable for both continuous and categorical values. Decision tree forces you to consider all possible outcomes of a decision. It can easily handle irrelevant attributes and missing data and therefore requires less data cleaning. It is very fast at testing time where the time complexity is  $O(\text{depth})$ .

Disadvantages are the tree might be too large even after pruning and trees with many decisions nodes may have limited value and the outcome could be less accurate. Also, they are time

consuming. Small changes in the dataset will need rebuilding of the entire tree and generated tree would be very different from the previous formed tree and this is not desirable. It can only perform axis-aligned splits of data. Overfitting is one of the most practical difficulty for decision trees.

## Overfitting

Overfitting is a key challenge in decision trees. If we don't set a limit on the decision tree we could get 100% accuracy on training set where it will end up making 1 leaf for each observation. To handle this case we set constraints on the tree size and for this algorithm we set it to the maximum depth of the tree. This is because higher depth will allow model to learn relations very specific to a particular sample.

## Results

The performance metrics for each iteration of the cross validation and their average for dataset1 is as shown in the left image. In the right screenshot it shows at each point which column the node is being split at .

```
~~~~~ Cross Validation: 1
PRECISION, RECALL AND F-MEASURE for 0
Precision: 96.25
Recall: 97.4683544304
F measure: 96.8553459119
PRECISION, RECALL AND F-MEASURE for 1
Precision: 93.9393939394
Recall: 91.1764705882
F measure: 92.5373134328
Accuracy is 95.5752212389%
~~~~~ Cross Validation: 2
PRECISION, RECALL AND F-MEASURE for 0
Precision: 95.5223880597
Recall: 94.1176470588
F measure: 94.8148148148
PRECISION, RECALL AND F-MEASURE for 1
Precision: 91.3043478261
Recall: 93.3333333333
F measure: 92.3076923077
Accuracy is 93.8053097345%
~~~~~ Cross Validation: 3
PRECISION, RECALL AND F-MEASURE for 0
Precision: 97.2602739726
Recall: 97.2602739726
F measure: 97.2602739726
PRECISION, RECALL AND F-MEASURE for 1
Precision: 95.0
Recall: 95.0
F measure: 95.0
Accuracy is 96.4601769912%
~~~~~ Cross Validation: 4
PRECISION, RECALL AND F-MEASURE for 0
Precision: 88.8888888889
Recall: 93.3333333333
F measure: 91.0569105691
PRECISION, RECALL AND F-MEASURE for 1
Precision: 92.0
Recall: 86.7924528302
F measure: 89.3203883495
Accuracy is 90.2654867257%
~~~~~ RESULTS!!
Total Average Accuracy is: 93.2743362832%
Total Average Precision for 0 as positive is: 94.7019572431%
Total Average Recall for 0 as positive is: 94.2441409371%
Total Average F-Measure for 0 as positive is: 94.4371853658%
Total Average Precision for 1 as positive is: 90.8931927975%
Total Average Recall for 1 as positive is: 91.7604513504%
Total Average F-Measure for 1 as positive is: 91.2448435239%

##### TREE #####
NODE-ID -> NODE_NAME/FEATURE
[ node children ]
0 -> 7
1 -> 23
2 -> 1
3 -> 15
4 -> 20
5 -> 4
6 -> 3
7 -> 28
8 -> 10
9 -> 13
10 -> 21
11 -> 10
12 -> 1
13 -> 5
14 -> 9
15 -> 0
16 -> 27
17 -> 22
18 -> 21
19 -> 1
20 -> 1
21 -> 25
22 -> 21
23 -> 20
24 -> 16
25 -> 0
PRECISION, RECALL AND F-MEASURE for 0
Precision: 88.8888888889
Recall: 93.3333333333
F measure: 91.0569105691
PRECISION, RECALL AND F-MEASURE for 1
Precision: 92.0
Recall: 86.7924528302
F measure: 89.3203883495
Accuracy is 90.2654867257%
~~~~~ RESULTS!!
Total Average Accuracy is: 93.2743362832%
Total Average Precision for 0 as positive is: 94.7019572431%
Total Average Recall for 0 as positive is: 94.2441409371%
Total Average F-Measure for 0 as positive is: 94.4371853658%
Total Average Precision for 1 as positive is: 90.8931927975%
Total Average Recall for 1 as positive is: 91.7604513504%
Total Average F-Measure for 1 as positive is: 91.2448435239%
```

The performance metrics for each iteration of the cross validation and their average for dataset2 is shown below:



```

PRECISION, RECALL AND F-MEASURE for 0
Precision: 73.4375
Recall: 81.0344827586
F measure: 77.0491803279
PRECISION, RECALL AND F-MEASURE for 1
Precision: 60.7142857143
Recall: 50.0
F measure: 54.8387096774
Accuracy is 69.5652173913%
=====
PRECISION, RECALL AND F-MEASURE for 0
Precision: 59.0909090909
Recall: 73.5849056604
F measure: 65.5462184874
PRECISION, RECALL AND F-MEASURE for 1
Precision: 46.1538461538
Recall: 30.7692307692
F measure: 36.9230769231
Accuracy is 55.4347826087%
=====
PRECISION, RECALL AND F-MEASURE for 0
Precision: 75.8620689655
Recall: 63.768115942
F measure: 69.2913385827
PRECISION, RECALL AND F-MEASURE for 1
Precision: 26.4705882353
Recall: 39.1304347826
F measure: 31.5789473684
Accuracy is 57.6086956522%
=====
PRECISION, RECALL AND F-MEASURE for 0
Precision: 74.5454545455
Recall: 66.1290322581
F measure: 70.0854708055
PRECISION, RECALL AND F-MEASURE for 1
Precision: 43.2432432432
Recall: 53.3333333333
F measure: 47.7611940299
Accuracy is 61.9565217391%
=====
Total Average Accuracy is: 61.3043478261%
Total Average Precision for 0 as positive is: 71.2025711358%
Total Average Recall for 0 as positive is: 69.704663256%
Total Average F-Measure for 0 as positive is: 70.0881351984%
Total Average Precision for 1 as positive is: 44.8163926693%
Total Average Recall for 1 as positive is: 46.1617512922%
Total Average F-Measure for 1 as positive is: 44.6313445039%
===== RESULTS! =====

```

### 3. Random Forest

#### **Algorithm**

Each tree is constructed using the following algorithm:

1. Let the number of training cases be  $N$ , and the number of variables in the classifier  $M$ .
2. We are told the number  $m$  of input variables to be used to determine the decision at a node of the tree;  $m$  should be much less than  $M$ .
3. Choose a training set for this tree by choosing  $n$  times with replacement from all  $N$  available training cases(i.e take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
4. For each node of the tree, randomly choose  $m$  variables on which to base the decision at that node. Calculate the best split based on these  $m$  variables in the training set.
5. Each tree is fully grown and not pruned

For prediction , a new sample is pushed down the tree. It is assigned the label of the training sample in the terminal node it ends up in. This procedure is iterated over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction.

#### **Implementation**

For this algorithm we iterate over  $n$ -fold cross validation wherein we split the data into  $n$  partitions and for each iteration we use one of the partitions as testing data and the remaining  $n-1$  partitions as training data. This algorithm is similar to decision tree, but here we reduce the size of the training dataset not just horizontally but vertically as well i.e. we reduce the rows(data samples) as well as the columns(attributes/features). For e.g. if we have 80 records as training data with 30 columns each that represent the attributes, and if we want 60% of these records as training data for the first iteration then we do

$$\text{row\_length} = 80 * 0.6$$

and then when building the tree using the attributes we select a small number of random features. This is given as shown below in the startBuildTree function

```
i 0 to (col_length-1)*0.2
```

```
    r = random(0 to col)
```

```
    func(node,r)
```

So, the new training data would now have 50 rows and 6 columns(store in a list) and tree would be build for this dataset. For each cross-validation iteration the algorithm forms  $k$ -trees(integer value which can be changed) and for each tree different combinations of features/columns are taken. Once we have the predicted class values for each of these days, we append them to a `global_predict_list`. We compare these values for each test records and pick the class that has a

majority by using the “mode” function and store that as the final predicted class. We then compare this final predicted class with the actual class values and calculate the performance metrics.

### Parameter setting:

- Splits are chosen according to a purity measure. E.g. squared error(regression), Gini index(classification)
- To select N we build the tree until the error no longer decreases
- To select M we try to recommend defaults, half of them and twice of them and pick the best

### Pros and cons

Advantages are that it is one of the most accurate learning algorithms available. For many datasets it produces a highly accurate classifier. It is efficient for large databases. It gives an estimation of which variables are important for classification. It can estimate missing data and maintains accuracy when large proportion of the data is missing. Generated forests can be saved and used for other data.

Disadvantages are that overfitting is a problem for some datasets with noisy regression/classification tasks. You have very little control over what the model does. The best we can do is try different parameters and random seeds.

### Results

We run this algorithm for dataset1. We took only 60% rows and 40% columns from training dataset for the results shown below. We observe that it has overall improved result, but there have also been cases where the accuracy is much below what we usually get from normal decision tree. It all depends on the feature column we include in those 40% columns. Since we won't have prior knowledge of which column might be more important, we can't control the random forest generation.

```
PRECISION, RECALL AND F-MEASURE for 0
Precision: 69.9115044248
Recall: 100.0
F measure: 82.2916666667
PRECISION, RECALL AND F-MEASURE for 1
Precision: 0.0
Recall: 0.0
F measure: 0
69.9115044248
===== Cross Validation: 2
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 3
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 4
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== RESULTS!!
Total Average Accuracy is: 93.982300885%
Total Average Precision for 0 as positive is: 93.982300885%
Total Average Recall for 0 as positive is: 100.0%
Total Average F-Measure for 0 as positive is: 96.458333333%
Total Average Precision for 1 as positive is: 80.0%
Total Average Recall for 1 as positive is: 80.0%
Total Average F-Measure for 1 as positive is: 80.0%

PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 2
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 3
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 4
PRECISION, RECALL AND F-MEASURE for 0
Precision: 53.0973451327
Recall: 100.0
F measure: 69.3641618497
PRECISION, RECALL AND F-MEASURE for 1
Precision: 0.0
Recall: 0.0
F measure: 0
53.0973451327
===== RESULTS!!
Total Average Accuracy is: 90.6194690265%
Total Average Precision for 0 as positive is: 90.6194690265%
Total Average Recall for 0 as positive is: 100.0%
Total Average F-Measure for 0 as positive is: 93.8728323699%
Total Average Precision for 1 as positive is: 80.0%
Total Average Recall for 1 as positive is: 80.0%
Total Average F-Measure for 1 as positive is: 80.0%
```

```

PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 2
PRECISION, RECALL AND F-MEASURE for 0
Precision: 60.1769911504
Recall: 100.0
F measure: 75.138121547
PRECISION, RECALL AND F-MEASURE for 1
Precision: 0.0
Recall: 0.0
F measure: 0
60.1769911504
===== Cross Validation: 3
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== Cross Validation: 4
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
===== RESULTS!!
Total Average Accuracy is: 92.0353982301%
Total Average Precision for 0 as positive is: 92.0353982301%
Total Average Recall for 0 as positive is: 100.0%
Total Average F-Measure for 0 as positive is: 95.0276243094%
Total Average Precision for 1 as positive is: 80.0%
Total Average Recall for 1 as positive is: 80.0%
Total Average F-Measure for 1 as positive is: 80.0%

```

```

PRECISION, RECALL AND F-MEASURE for 0
Precision: 85.8695652174
Recall: 100.0
F measure: 92.3976608187
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 61.7647058824
F measure: 76.3636363636
88.4955752212
~~~~~ Cross Validation: 2
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
~~~~~ Cross Validation: 3
PRECISION, RECALL AND F-MEASURE for 0
Precision: 100.0
Recall: 100.0
F measure: 100.0
PRECISION, RECALL AND F-MEASURE for 1
Precision: 100.0
Recall: 100.0
F measure: 100.0
100.0
~~~~~ Cross Validation: 4
PRECISION, RECALL AND F-MEASURE for 0
Precision: 53.0973451327
Recall: 100.0
F measure: 69.3641618497
PRECISION, RECALL AND F-MEASURE for 1
Precision: 0.0
Recall: 0.0
F measure: 0
53.0973451327
~~~~~ RESULTS!!
Total Average Accuracy is: 81.4159292035%
Total Average Precision for 0 as positive is: 80.8290963557%
Total Average Recall for 0 as positive is: 100.0%
Total Average F-Measure for 0 as positive is: 88.1361483175%
Total Average Precision for 1 as positive is: 80.0%
Total Average Recall for 1 as positive is: 52.8529411765%
Total Average F-Measure for 1 as positive is: 56.2483370288%

```

## **4. Boosted Decision Trees**

### **Algorithm:**

1. Initialize the observation weights  $w_i = 1/N$ ,  $i = 1, 2, \dots, N$ .
2. For  $m = 1$  to  $M$  repeat steps (a)–(d):
  - (a) Fit a classifier  $C_m(x)$  to the training data using weights  $w_i$ .
  - (b) Compute weighted error of newest tree

$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^N w_i}.$$

- (c) Compute  $\alpha_m = \log[(1 - \text{err}_m)/\text{err}_m]$ .
  - (d) Update weights for  $i = 1, \dots, N$ :
$$w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]$$
and renormalize to  $w_i$  to sum to 1.
3. Output  $C(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m C_m(x) \right]$ .

### **Implementation:**

- Initially, set all weights on all the records as  $1/N$  where  $N$  = number of rows in our training dataset
- At each round,  $m$ 
  - Create a sample based on the weights (like 60% of training data)
  - Train a classifier  $C_m$  on the sample and apply it on the original training set (all of training data)
  - Records that are wrongly classified will have their weights increased &
  - Records that are classified correctly will have their weights decreased according to this formula: We do this in the updateWeights function

This will be the weighted error:

$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq C_m(x_i))}{\sum_{i=1}^N w_i}.$$

- Calculate  $\alpha_m = \log[(1 - \text{err}_m) / \text{err}_m]$ .
- Update weights according to  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq C_m(x_i))]$

- ### Pros and Cons:

Disadvantages are that it takes longer to build the trees because it is done sequentially and therefore takes time to train or score. It is harder to tune than other models because of the hyper parameters and is therefore more prone to overfitting. Compared to linear classifiers it is less interpretable.

We see how our system is learning from each tree classifier ultimately reaching 100%.

```
FOR TREEEEE 0
Accuracy is: 88.9473684211%
FOR TREEEEE 1
Accuracy is: 93.6842105263%
FOR TREEEEE 2
Accuracy is: 98.1578947368%
FOR TREEEEE 3
Accuracy is: 98.6842105263%
FOR TREEEEE 4
Accuracy is: 99.4736842105%
ModeResult(mode=array([[ 0.00264538],
[ 0.00264538],
[ 0.00264538],
[ 0.00264538]]), count=array([[378],
[378],
[378],
[378]]))
===== Cross Validation: 1

FOR TREEEEE 0
Accuracy is: 96.3157894737%
FOR TREEEEE 1
Accuracy is: 94.4736842105%
FOR TREEEEE 2
Accuracy is: 97.1052631579%
FOR TREEEEE 3
Accuracy is: 98.6842105263%
FOR TREEEEE 4
Accuracy is: 99.2105263158%
ModeResult(mode=array([[ 0.00265233],
[ 0.00265233],
[ 0.00265233],
[ 0.00265233]]), count=array([[377],
[377],
[377],
[377]]))
===== Cross Validation: 2
```

```
misc:nermanaged:downloadas talgunionnaraadawaj$ python boostingui.py
STARTING!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
(569, 31)
~~~~~ Cross Validation: 0
FOR TREEEEE 0
Accuracy is: 88.9473684211%
FOR TREEEEE 1
Accuracy is: 94.4736842105%
FOR TREEEEE 2
Accuracy is: 99.4736842105%
FOR TREEEEE 3
Accuracy is: 99.7368421053%
FOR TREEEEE 4
Accuracy is: 100.0%
~~~~~ Cross Validation: 1
FOR TREEEEE 0
Accuracy is: 93.9473684211%
FOR TREEEEE 1
Accuracy is: 98.4210526316%
FOR TREEEEE 2
Accuracy is: 98.9473684211%
FOR TREEEEE 3
Accuracy is: 99.7368421053%
FOR TREEEEE 4
Accuracy is: 100.0%
~~~~~ Cross Validation: 2
FOR TREEEEE 0
Accuracy is: 88.6842105263%
FOR TREEEEE 1
Accuracy is: 96.3157894737%
FOR TREEEEE 2
Accuracy is: 97.3684210526%
FOR TREEEEE 3
Accuracy is: 99.7368421053%
FOR TREEEEE 4
Accuracy is: 100.0%
```

## 5. Naïve Bayes

### What is Naïve Bayes Classifier technique?

The Naive Bayes Classifier technique is based on Bayesian theorem and is particularly used when the dimensionality of the inputs is high. The Bayesian Classifier is capable of calculating the most possible output based on the input. It is also possible to add new raw data at runtime and have a better probabilistic classifier. A naive Bayes classifier considers that the presence (or absence) of a particular feature(attribute) of a class is unrelated to the presence (or absence) of any other feature when the class variable is given.

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Likelihood                      Class Prior Probability

↓                                      ↓

Posterior Probability                      Predictor Prior Probability

- $P(c | X)$  is the posterior probability of class (target) given predictor (attribute) of class.
- $P(c)$  is called the prior probability of class.
- $P(x | c)$  is the likelihood which is the probability of predictor of given class.
- $P(x)$  is the prior probability of predictor of class.

### Implementation

**PerformanceCalc():** The predicted labels from the NaïveBayes() is sent to this method for the evaluation of performance metrics such as accuracy, precision, recall and f1 measure. These values are then averaged and printed in the console.

**NaiveBayes():** After preprocessing the data, for us to make a prediction we calculate the Gaussian Probability Density Function on the test dataset for classifying it into different class labels. The mean and standard deviation is calculated for the attribute estimated from the training data. We are plugging our known details into the Gaussian and returning the likelihood that our attribute belongs to the class.



**CrossValidation():** We have implemented the K-fold Cross Validation so that all the values in the dataset are eventually used for both training and testing. For a dataset with N rows, N iterations are performed. For each iteration, we use N-1 rows and the remaining rows for testing.

**Main():** The input data is read from the console. To classify the data into categories of 0 and 1 which denotes the presence or absence of an attribute in the dataset by using the LabelEncoder in sklearn.

### **How to avoid Zero probability problem?**

The Laplacian correction (or Laplace estimator) is a way of dealing with Zero probability values, which means the descriptor posterior probability becomes 0, if any probability is 0. We assume that our training set is so large that by incrementing by one in each count we need, will hardly make a difference in the estimated probabilities, yet will avoid the case of probability values. Using the Laplacian correction for the probabilities, we pretend that we have 1 more sample for each calculated probability. The “corrected” probability estimates are close to their “uncorrected counterparts”, yet the zero probability value is avoided.

### **Pros and Cons**

Advantages are that it's relatively simple to understand and easy to build due to no complex iterative parameter estimation. It's easily trained even with a small dataset. It exhibits high accuracy and speed when applied to large datasets. By using Bayesian Classifier, new data can be added at run time to achieve a better probabilistic classifier. It's not sensitive to irrelevant features. Disadvantages are that it assumes that every feature is independent, which is not the case always. It does not work with correlated data. Most real-life data are correlated so it does not serve the best purpose.

### **Results**

Following are the performance metrics results for dataset1

Average Accuracy :	0.934899749373
Average Precision :	0.917870936253
Average Recall :	0.904442635683
Average F1_measure :	0.910031838126

Following are the performance metrics results for dataset2

Average Accuracy :	0.707724329325
Average Precision :	0.575085073166
Average Recall :	0.620701549912
Average F1_measure :	0.591568145206

For dataset1, the Naive Bayes Classifier gives a high accuracy of approximately 93%. But since accuracy can be misleading, additional measures like precision, recall and f1 score are evaluated in order to avoid the accuracy paradox. Precision, recall and f1 score of more than 90% confirms that the prediction results of the classifier are indeed correct.

Likewise, for dataset2, the average accuracy is evaluated as 70% approximately but the other performance metrics precision, recall and f1 score are relatively lower which correctly predicts the class labels and its probability.