CSE 601- DataMining and BioInformatics

**Project 3- Classification Algorithms**

Uttara Asthana-50168804

Contents

[**1. INTRODUCTION** 3](#_Toc469138674)

[**2.DATASET** 4](#_Toc469138675)

[**3.TOOLS USED** 4](#_Toc469138676)

[**4. IMPLEMENTATION DETAILS** 4](#_Toc469138677)

[**4.1 PERFORMANCE MEASURES** 5](#_Toc469138678)

[**4.1.] KNN –K Nearest Neighbor** 5](#_Toc469138679)

[**4.2] DECISION TREES:** 8](#_Toc469138680)

[**4.3] RANDOM FOREST IMPLEMENTATION** 13](#_Toc469138681)

[**4.4] BOOSTING IMPLEMENTATION** 17](#_Toc469138682)

[**4.5] NAÏVE BAYES CLASSIFIER** 18](#_Toc469138683)

[**5 PERFORMANCE OF ALGORITHMS ON VARIOUS DATA SETS AND THEIR COMPARISON** 21](#_Toc469138684)

[**6.CONCLUSION:** 25](#_Toc469138685)

[**7.REFERENCES** 26](#_Toc469138686)

# **1. INTRODUCTION:**

In machine learning and statistics ‘Classification’ is the problem of identifying to which of a set of categories a new observation belongs. This categorization is on the basis of a training set of data containing observations whose category of membership is known. The training set contains a set of attributes, one of which is the category the record belongs to. A model for the class attribute as a function of values of other attributes is created. The goal is to assign class for previously unseen records as accurately as possible. A test set is used to determine the accuracy of the model. A popular approach is dividing the given dataset into training and test sets, with the training set used to build the model and the test set used to validate it.

Classification algorithms are commonly used in predicting tumor cells as benignant or malign, detecting fraudulent credit card transactions, classifying emails as spam or not.

In our project on classification algorithms we have implemented three classification algorithms which include Naïve Bayes, Nearest Neighbor and Decision Tree. We have implemented Random Forests and Boosting based on the decision tree that we have implemented in this project.

We have also evaluated the performance of our model for all of the given datasets by calculating the accuracy, precision, recall and F-1 measure.

# **2.DATASET:**

The datasets we had to initially train and test our model on were project3\_dataset1.txt and project3\_dataset2.txt. We divided both these datasets into train and test using the 10 - fold cross validation approach and built models for each of the classification algorithms. We were able to evaluate the performance of classification on these based on standard measures.

In case of Project3\_Dataset2 we have handled discrete values Absent/Present by assigning unique floating point numbers to them 1.0 (Present) and 0.0 (Absent).

# **3.TOOLS USED:**

The implementation of the classification algorithms was done in Python 2.7.11. We used Canopy as our choice python editor as a part of this project.

**K FOLD CROSS VALIDATION:**

We have adopted the K-fold cross validation technique for accessing the results in terms of accuracy and other performance measures. In a prediction problem, a model is usually given a dataset of known data on which training is run (training data set) and unknown data set against which the model is tested (testing data set). The main goal of this technique is to define a data set to ‘test’ the model in the ‘training phase’.

# **4. IMPLEMENTATION DETAILS:**

We have adopted 10 fold cross validation technique. One round of cross validation involves partitioning the sample data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the testing set). Multiple such rounds are implemented using different set of partitions and the validation results are averaged over the rounds.

Since we have adopted 10 fold cross validation, in each round a different combination of 10 percent of the total data is chosen as testing database and the rest of the tuples are considered for training purposes on which basis the classifier is built and results are obtained. 10 such rounds will be further conducted keeping the split ratio same (10 percent in our case) only the tuples choses=n this time would change.

Def classify\_Algorithm()

Return learned\_classifier and test tuples classifiedS

Def k\_fold(k)

New combination of

Train\_Database, Test\_Database

Performance\_Measure\_single\_fold=Classify\_Algorithm(Train\_Database,Test\_Database)

Performance\_measures.append(Performance\_Measure\_single\_fold)

Average\_performance=Average(Performance\_measures)

N\_fold=k

K\_fold(k)

## **4.1 PERFORMANCE MEASURES:**

We have used the following four measures to evaluate performance of our algorithm,

1.] Accuracy

Accuracy is defined as the percentage ratio of the total number of correctly classified samples to the total number of samples (#right/#total).

2.] Precision

Precision looks at the ratio of correct positive observations. The formula is True Positives / (True Positives + False Positives).

3.] Recall

Recall is also known as **sensitivity**or **true positive rate**. It’s the ratio of correctly predicted positive events. Recall is calculated as True Positives / (True Positives + False Negatives).

4.] F1 Measure

The F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. The formula for F1 Score is 2\*(Recall \* Precision)/ (Recall + Precision)

## **4.1.] KNN –K Nearest Neighbor**

K-NN is a classification algorithm that outputs the class membership of the testing data set. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its *k* nearest neighbors (*k* is a positive [integer](https://en.wikipedia.org/wiki/Integer), typically small).

Following are the implementation details for KNN:

**Data Pre-processing:**

We have scaled project3\_Dataset1 // in order to prevent the distance measure from one attribute to dominate over the other attributes. As we can see in Project3\_Dataset1 all the attribute values for column 2 are 2 digit numbers however those in column 4 mostly the attribute values are less than 1. Hence Data scaling would be beneficial in such case.

Input🡪 Training Data Set, Testing Data Set (In case of K fold cross validation single Data set file)

Required computation on Training data set, k i.e. the nearest neighbors to be considered for deciding the label of test data.

1. Required computation on Training Data set:
   * The set of stored records i.e. Training Data set
   * Distance Metric to compute distance between records

We have calculated the Euclidian distance between two data tuples in order to find out distance between them.



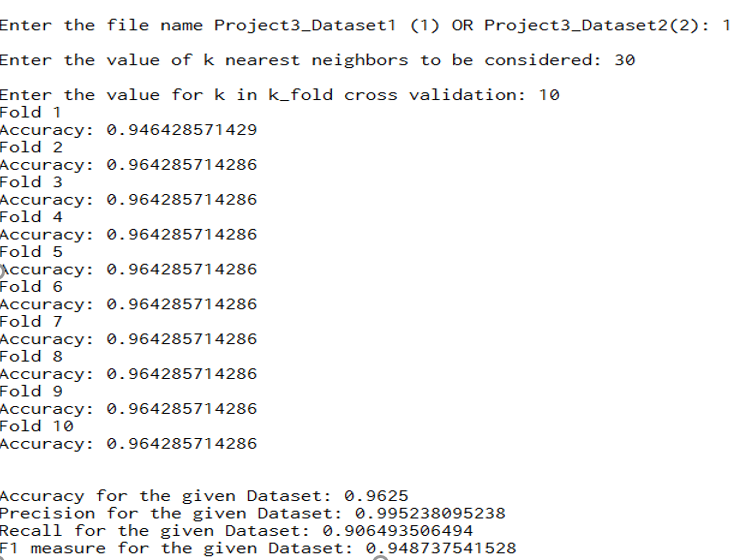
* + The value of *k*, the number of nearest neighbors to retrieve

1. To classify an unknown record:
   * + Compute its distance to other training records.
     + Identify k nearest neighbors.
     + Use class labels of nearest neighbors to determine the class label of unknown record by taking majority vote.

**Outputs for KNN for 10 Fold cross validation:**

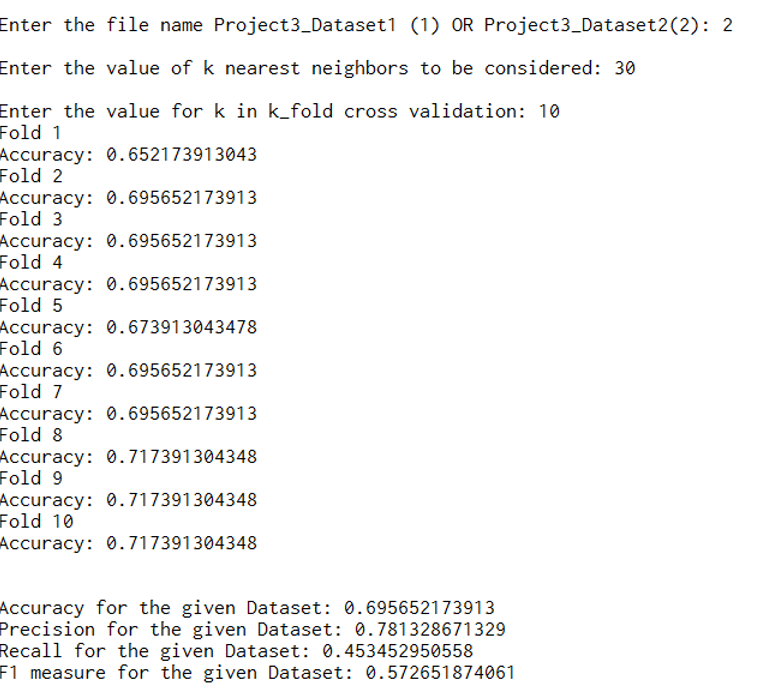
**Project3\_Dataset1:**

**K=30 (nearest neighbors)**



**Project3\_Dataset2:**

**K=30 (nearest neighbors)**



## **4.2] DECISION TREES:**

Decision trees is one of the predictive modelling techniques used in classification which maps observations about an item to conclusion to items target value.

**Data Pre-processing:**

**Data shuffling:**

Shuffling the data is useful when the data comes from different sources and is to be split into train/validation sets. We obtained improved accuracy performance after shuffling the data.

**Implementation Details:**

Splitting Method: We have used the binary splitting method, which divides data set into two subsets depending on the attribute.

We have adopted the greedy strategy records are split based on attribute test that optimize certain criterion. For splitting the best split value we have computed the Gini index for each value present in data set and the value with the lowest Gini is selected as splitting value and the column it corresponds to gives us the spitting attribute to be considered. The gini inex basically calculates the purity of the groups of data created by the split point.

We have used the Gini Index to measure the node impurity at each level. The node with the lowest Gini is given priority. When a node p is split into k partitions (children), the quality of split is computed as,



where, ni = number of records at child i,

n = number of records at node p.

Also at every iteration besides we are checking if the node is terminal or not i.e. if all the tuples in the node have the same class labels.

We have maintained a dictionary for tree traversal of the form-- //

**Pseudo Code:**

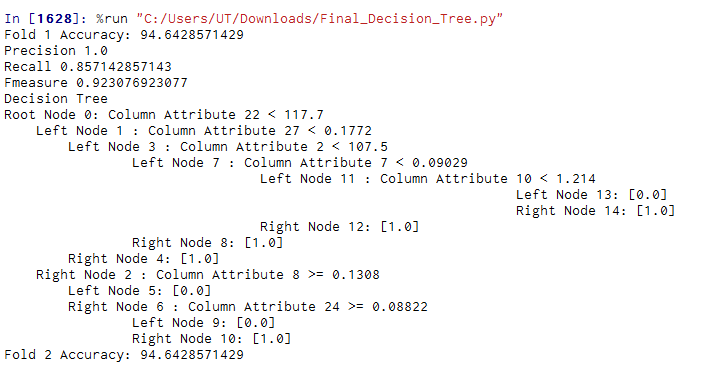
|  |
| --- |
| **define algorithm\_decision\_tree(training data):**  **datalist= empty list**  **add to datalist(training data)**  **parentnode=0**  **while sizeOf datalist > 0**  **child1, child2, split-attributes= call function get\_split(training data)**  **if sizeOf child1 != 0 and sizeOf child2 !=0 then**  **new datalist=update datalist(child, child2)**  **key=update parent node for next iteration**  **value=split-attributes**  **attribute-dictionary <- key, value**  **association-dictionary <- child1, child2**  **datalist=new datalist**    **define function get\_split(dataset):**  **if sizeOf dataset==1 OR count(labels=0) in dataset ==sizeOfdataset OR count(labels=1) in dataset ==sizeOf dataset then**  **assign leaf node and class label with max count**  **else**  **then**  **for column in columns:**    **gini index, set1, set2=calculate gini index(dataset, column, at each column value)**    **find minimum(gini index)**  **bestcolumn =column with min gini index**  **bestvalue= value in bestcolumn with min gini index**  **best-attributes=[ bestcolumn, bestvalue]**    **return best-attributes, set1, set2**    **define function calculate gini index(dataset, column, column\_value)**  **test by split (dataset at column)**  **set1= if value < column\_value**  **set2= if value >= column\_value**  **return set1, set2**  **define validate(test data, association-dictionary, attribute-dictionary):**  **begin:**  **for test in test data:**  **node=0**  **if node is in attribute-dictionary.keys then**  **if node is leaf node then**  **predict\_list <-class label**  **else**  **then**  **column, value=attribute-dictionary[node]**  **if testvalue at test[column] < value then**  **node=association-dictionary[node][left node]**  **else**  **then**  **node=association-dictionary[node][right node]**      **define function accuracy(actuallist, predictlist):**  **correct =0**  **for i in (0, sizeOf actuallist):**  **if actuallist[i] equals predictlist[i] then**  **increment correct by 1**    **accuracy percent= (correct/total number) \* 0.01**  **return accuracy percent**    **define function kfoldcrossvalidation(k, database):**  **window=sizeOf database/ k**  **split data= k portions of database**  **for i in range(0, k):**  **testdata = 1 portion of split data**  **training data = k-1 portions of split data**  **call algorithm\_decision\_tree(training data)**  **call validate(testdata, association-dictionary, attribute-dictionary)**  **call accuracy(actuallist, predictedlist)**  **append score to scores list**  **return scores**  **call kfoldcrossvalidation(k, database)**  **Final accuracy= sum of scores(i=0 to sizeOf scores)/ sizeOf scores** |

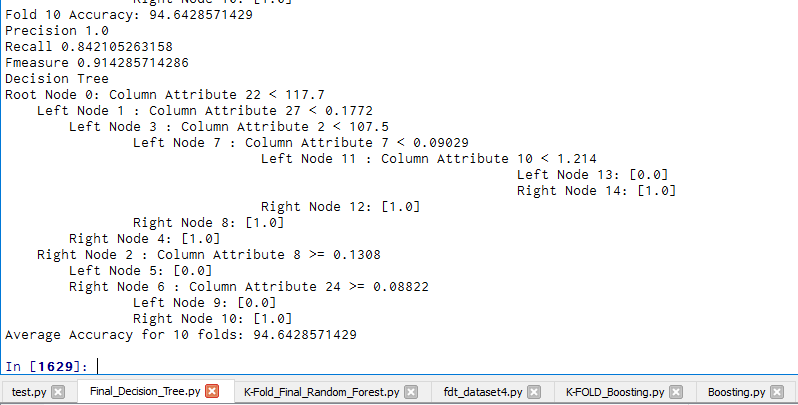
**OUTPUT FOR DECISION TREE for 10 FOLD CROSS VALIDATION**

For the purpose of convenience, we have shown the snapshot of tree learned from only one fold. Not that in the outputs the values for same measure may vary as we have randomized our choice of training data set resulting in a slight difference.

**PROJECT3\_DATASET1,**

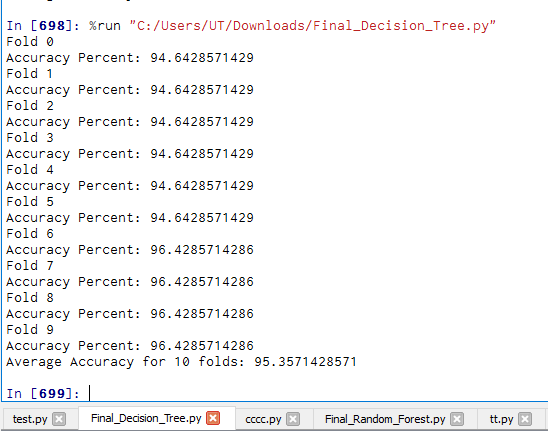
**Following is the learnt tree for Dataset1,**

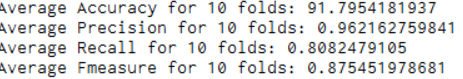




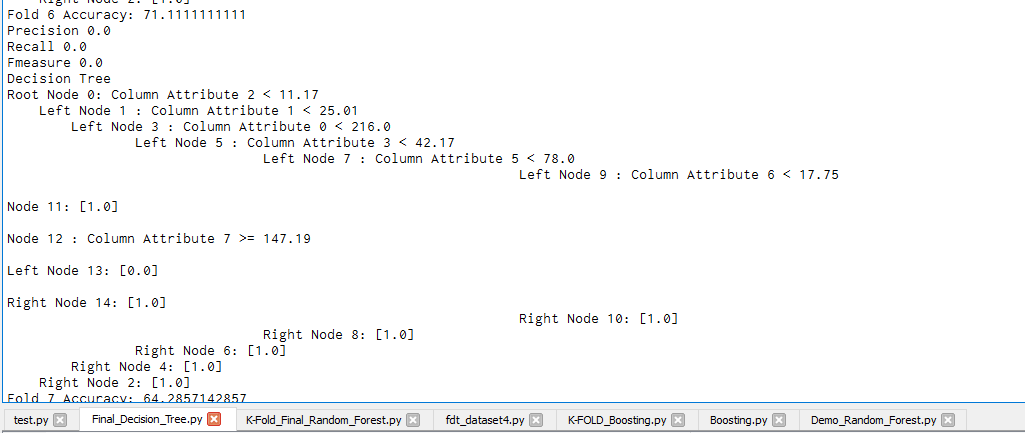
**PROJECT3\_DATASET1,**

Performance Measures,

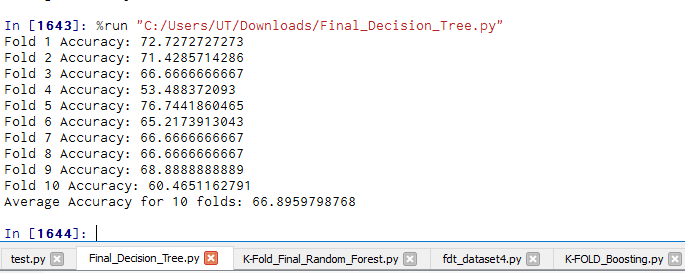


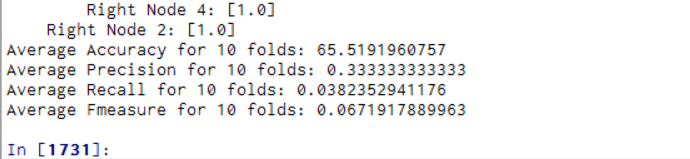


**Following is the tree learnt for the data set2**



**Performance Measure for Project3\_Dataset2,**





## **4.3] RANDOM FOREST IMPLEMENTATION:**

Random forest is basically one of the ways of implementing bagging in Decision Tree classifier. This algorithm makes decision trees susceptible to high variance if they are not pruned. This high variance can be harnessed and reduced by creating multiple trees with different samples of the training dataset (different views of the problem) and combining their predictions. This approach is called bootstrap aggregation or bagging for short. Our implementation of Random forest is based on our basic decision tree implementation.

Bagging is mainly implemented in 2 steps

**Step1**: **Bootstrap**

-Sampling with replacement

-Contains around 63.2% original records in each sample

**Step2: Bootstrap Aggregation**

- Train a classifier on each bootstrap sample

-Use majority voting to determine the class label of ensemble classifier

Following is the pseudocode for our implementation of Random forest:

**Step1**: Choose *T*—number of trees to grow

**Step 2**: Choose *m<M* (M is the number of total features)—number of features used to calculate the best split at each node (typically 20%)

**Step 3**: For each tree

- Choose a training set by choosing *N* times (*N* is the number of training examples) with replacement from the training set

- For each node, randomly choose *m* features and calculate the best split

- Fully grown and not pruned

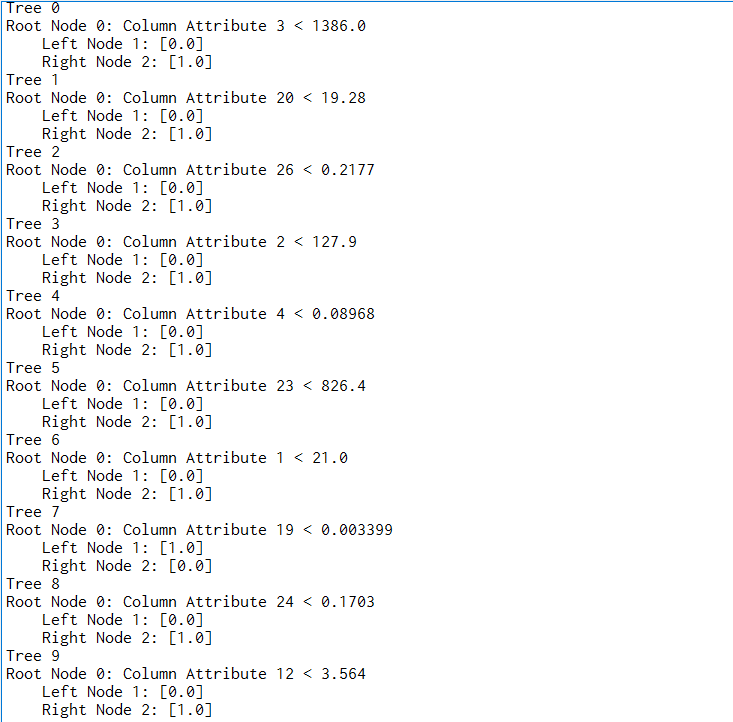
**Step 4**: Use majority voting among all the trees

**OUTPUT OF RANDOM FOREST FOR 10 FOLD CROSS VALIDATION**

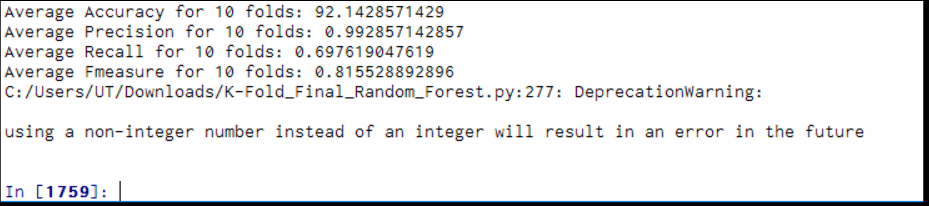
**For Project\_Dataset1,**

**Following is the decision tree learned,**

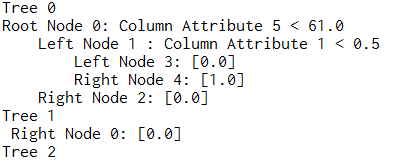
**(For fold 1)**

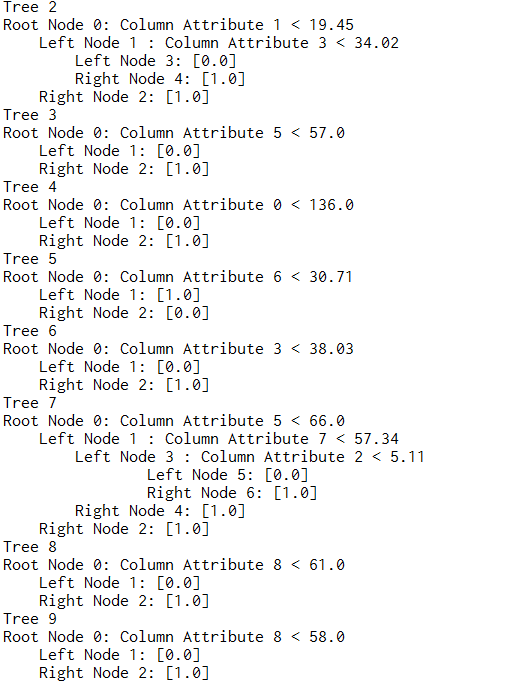
****

**Following are the performance measures obtained,**

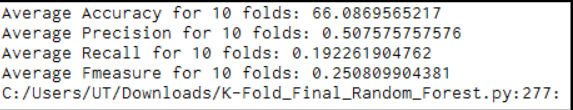
****

**Following is a snapshot the decision tree learned for Project3\_Dataset2,**

****

****

**Performance measures for Dataset2,**

****

## **4.4] BOOSTING IMPLEMENTATION:**

Boosting is a method of improving the classifier performance by boosting a set of weak learners to strong learners. It basically makes the records miss classified more important in the further iterations by assigning them greater weights in order to improve its learning.

Say if a particular record is hard to classify it is more likely to be chosen in the sub sequent rounds.

We have implemented the Ada boost method for improving the performance of our Decision Tree classifier.

Following are the implementation details,

-Initially, set uniform weights on all the records i.e. each instance in training data set is assigned a uniform weight . The weight is initially set to,

weight(xi) = 1/n

-At each round

-Create a bootstrap sample based on the weights i.e. for each instance

-Train a classifier on the sample and apply it on the original training set.

-Records that are wrongly classified will have their weights increased

For example, the weight of one training instance (w) is updated using:

w = w \* exp(stage \* terror)

-Records that are classified correctly will have their weights decreased

-If the error rate is higher than 50%, start over

The misclassification rate for each round is calculated by using the formulae

Error= Sum(Weights for incorrect predictions)/Total weight i.e.

Error = sum(w(i) \* terror(i)) / sum(w) where terror =1 if incorrect prediction is there and 0 otherwise.

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

stage = ln((1-error) / error) this is further used for calculation of new weights.

-Final prediction is weighted average of all the classifiers with weight representing the training accuracy

## **4.5] NAÏVE BAYES CLASSIFIER:**

Naïve Bayes is a classification technique that assumes attribute independence, it is a statistical classifier that predicts class member probabilities.

We have implemented the Naïve Bayes classifier in two different ways, one for Discrete attributes (as given in project3\_Dataset1 and Project3\_Dataset2) and one for continuous attributes (to handle strings e.g. as given in Project3\_Dataset4). Projject3\_Dataset2 also has continuous attributes Present/Absent in the fourth column however majority of its attributes are in discrete for.

Implementation details for Project3\_Dataset1 and PROJECT3\_Dataset2:

We have implemented Naïve Bayes using the concept of Gaussian probability Density function. We can use the Gaussian function to estimate the probability of a given attribute value, given the known mean and standard deviation for the attributes estimated from training data. Knowing the probability of an attribute belonging to a class, we can combine the probabilities of all of the attribute values for a data instance and come up with a probability of the entire data instance belonging to the class.

Following are the steps implemented:

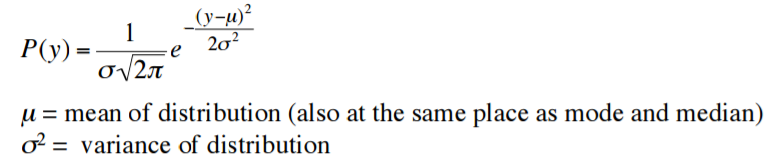
Input🡪 CSV file

On each iteration of K-fold following steps are implemented,

The data is split in training and testing,

For a test tuple to be classified,

1. Separation for class0 and class1: The training data set is divided into 2 classes depending on the class label the belong to 0 or 1.
2. Set probability0 🡪1 and probability1🡪 0
3. For each column of the training data set:
4. Calculate the Mean for all values in that column
5. Calculate the standard deviation
6. Calculate the gaussian probability for that particular attribute for class 0 and class1 using the formula below



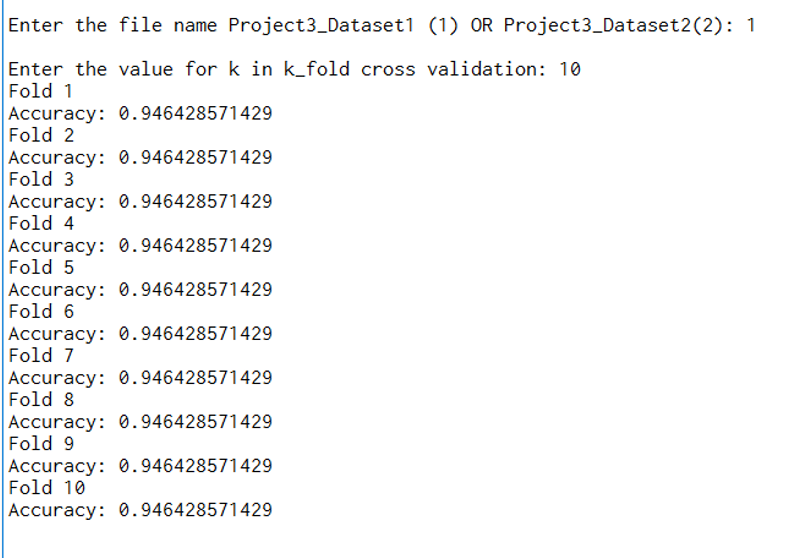
1. Set Probability0🡪 probability0\*p(0), Probability1🡪 probability1\*p(1),
2. Compare the probability 0 with probability 1 and assign the tuple to the class having greater probability
3. The probability that the tuple belongs to class0 is proability0/(probability0+probabilty1) and the probability that the tuple belongs to class 1 is proability1/(probability0+probabilty1)

**Handling divide by Zero Probability Error:**

In case of continuous value’s we can handle this error using laplacian correction however in case of Guassian probability density the probabilities are multiplied and handled in logarithmic space.

**OUTPUT FOR NAÏVE BAYES FOR 10 FOLD CROSS VALIDATION:**

**For Project3\_Dataset1:**

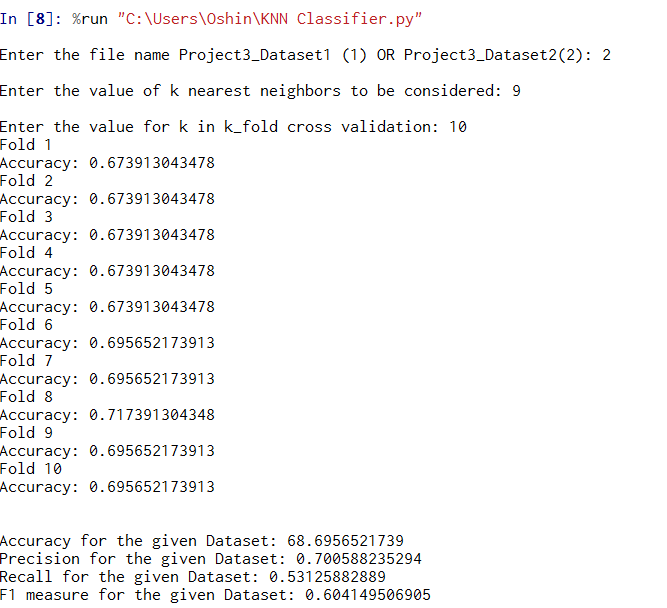


# **5 PERFORMANCE OF ALGORITHMS ON VARIOUS DATA SETS AND THEIR COMPARISON**

**K-NN**

1.] As we can see the algorithm works very efficiently for the Project3\_Dataset1 giving an accuracy of 96.25%. Precision, Recall and F1\_Measure are also relatively high as compared to Project3\_Dataset2 which gives an accuracy of 69.56%. The first Dataset has all independent discrete values and hence the algorithm works efficiently in this case. However, for the second data set there is a column attribute having string Absent/Present, this might be the reason for the algorithm to not work that well on this data set. Scaling the dataset1 made a considerable impact on the performance measure.

2.] Also we can see in the figure below that Project3\_Dataset2 should a better performance at k=9 as compared to k=30. Also the project3\_Dataset1 showed a very good performance at k=30. This indicates that the parameter k is very crucial for performance measure of the algorithm. Dataset1 seems to be widely distributed and hence a greater value of k seems to be a good fit for it, however Dataset2 showed a better performance for lesser K value hence this data set seems to be very sparsely distributed.



**DECISION TREES:**

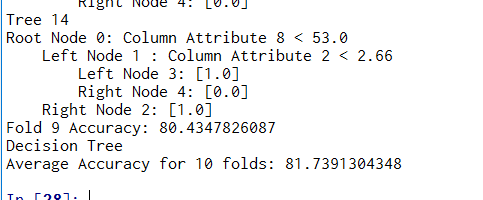
1.] In many cases Decision tree classifier is preferred over other methods as it is an in expensive methods unlike K-NN it is not a lazy classifier and is extremely fast in classifying unknown records. The Algorithm for Decision tree runs faster on both the data sets as compared to K-NN.

2.] Again the Decision Tree Algorithm works well for Project3\_Dataset1 giving an accuracy of 95.357% and gives a comparatively lesser accuracy of 66.89% on dataset2. They are very easy to understand, flexible and easy to debug. They work well with classification as well as regression problems. They can handle categorical values like (Absent/present) as well as continuous values like (2.9,3.4 e.tc.) very well. They only need a table of data and they will build a classifier directly from that data without needing any up front design work to take place. To some degree properties that don't matter won't be chosen as splits and will get eventually pruned so it's very tolerant of nonsense. To start it's set it and forget it.

3.] The Random forest is method of implementing bagging on decision trees and is used to improve the overall performance of tree in terms of accuracy and also used to provide an optimal solution by considering limited attributes. They are comparatively faster to train but very slow for prediction when trained.

4.] In case of Project3\_Dataset1, they show a slight improvement as in some cases the accuracy for this data set comes out to be 90% and random forest increases it to almost 93% however in some cases the improvement is not remarkable.

5.] For Project3\_Dataset2 by changing the parameter of number of trees (increasing it) the accuracy is improved in most of the cases and goes as high as 81% also fot No\_of Tress=15, which is usually in the range of 60-70% for the same data set for a basic decision tree implementation. We consider usually 20% of the total features for splitting at each node and hence this parameter is set 2.



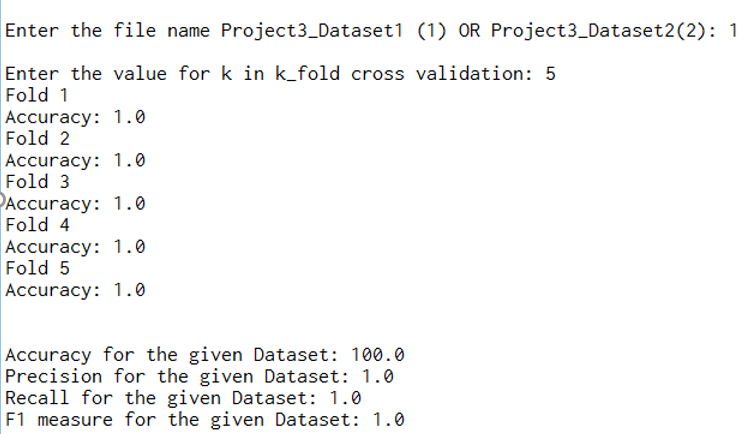
6.] Unlike Random forest gradient boosting run trail after trial and hence their parallel implementation is not possible. Gradient Boosted Trees algorithm additionally tries to find optimal linear combination of trees (assume final model is the weighted sum of predictions of individual trees) in relation to given train data.

**NAÏVE BAYES:**

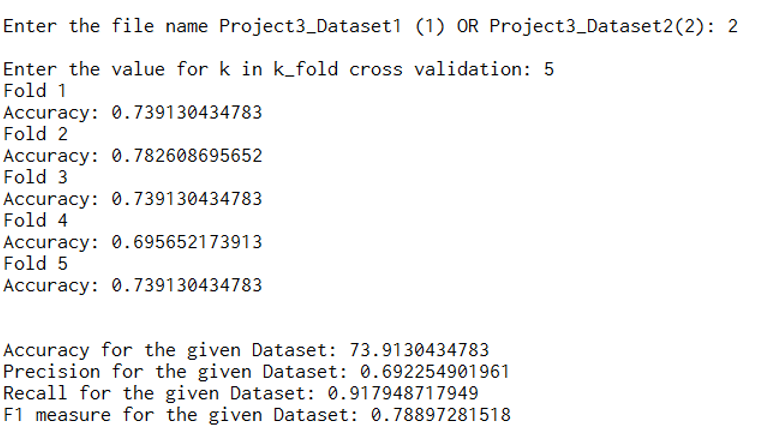
1.] The Naïve Bayes algorithm works efficiently for large data sets like Project3\_Dataset giving a greater accuracy of 94.64 as compared to the accuracy 66.95% obtained for the second data set. The first data set consists of completely discrete values unlike second data set which consists of string attribute (Present/Attribute) which could be the reason for the algorithm to work better on the first data set as we have followed the Guassian probability density method which heavily relies on data statistics like mean and standard deviation. Unlike decision tree this algorithm requires you to build classifier by hand.

2.] This algorithm is fastest from implementation time point of view and runs in a very less time as compared to other classification algorithm even on large data sets like Project3\_Dataset1.

3.] As we can see in the figures below if we decrease the value of k fold in cross validation technique the accuracy increases for both the data sets.



For 5-fold for Project3\_dataset1 100 percent accuracy is obtained.



For 5fold cross validation the accuracy increased to almost 74% which was 68.69% for the second data set for 10 fold cross validation.

# **6.CONCLUSION:**

We have successfully implemented Nearest Neighbor, decision tree and Naïve Bayes algorithms for classification of the given datasets. We have evaluated the performance of each of the algorithms based on the set criteria. We have implemented Random Forests and Boosting based on our implementation of the Decision Tree. For constructing the train and test data set we have used the 10 fold cross validation. We have also observed how the size of the dataset affects the performance of each algorithm.

# **7.REFERENCES**