CSE 601
Data Mining and Bioinformatics
Project 3: Classification Algorithms

Karthikeyan Ramachandriya Subramanian - 50289080
Riya Hazra - 50290600
Shwetasree Chowdhury - 50296995

What is classification?

Classification is a form of data analysis that extracts models describing important data classes.

Data classification is a ***two-step process***, consisting of a **learning step** (where a classification model is constructed) and a **classification step** (where the model is used to predict class labels for given data). In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels. The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The learning step is also considered a ***supervised learning*** (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with unsupervised learning (or clustering), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance.

The **accuracy of a classifier** on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple

Why do we need classification?

Classification on a large dataset will help predict labels based on same kind of data and can be found extremely useful in predicting the likelihood of a particular test condition. It can be found useful in the areas of banking, medicine, real estate and almost all walks of life.

A bank loans officer needs analysis of her data to learn which loan applicants are "safe" and which are "risky" for the bank. A marketing manager needs data to help guess whether a customer with a given profile will buy a new computer. A medical researcher wants to analyze breast cancer data to predict which one of three specific treatments a patient should receive. In each of these examples, the data analysis task is classification, where a model or classifier is constructed to predict class.

Some of the algorithms we have implemented and discussed are:

1. **K- Nearest Neighbour**
2. **Naïve Bayes**
3. **Decision Tree**
4. **Random Forest**
5. **Kaggle Competition**

1. K- Nearest Neighbor

1.1 DESCRIPTION

Nearest-neighbor classifiers are based on learning by analogy, that is, by comparing a given test tuple with training tuples that are similar to it. The training tuples are described by n attributes. Each tuple represents a point in an n-dimensional space. In this way, all the training tuples are stored in an n-dimensional pattern space.

When given an unknown tuple, a k-nearest-neighbor classifier searches the pattern space for the k training tuples that are closest to the unknown tuple. These k training tuples are the k “nearest neighbors” of the unknown tuple.

KNN is a ***lazy learner***. In contrast, ***Eager learners***, when given a set of training tuples, will construct a generalization (i.e., classification) model before receiving new (e.g., test) tuples to classify. We can think of the learned model as being ready and eager to classify previously unseen tuples.

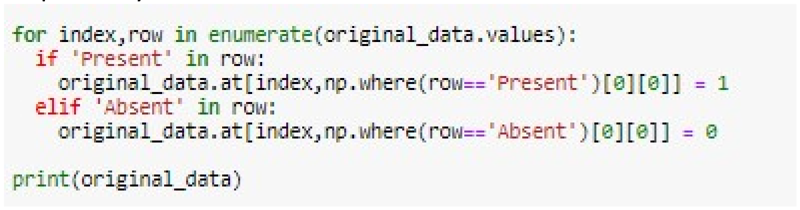
In a Lazy learning approach, the learner instead waits until the last minute before doing any model construction to classify a given test tuple. That is, when given a training tuple, a lazy learner simply stores it (or does only a little minor processing) and waits until it is given a test tuple.

1.2 PREPROCESSING ON DATASET

Each line represents one data sample in the dataset. The last column of each line is class label, either 0 or 1. The rest of the columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type). For the purpose of our project, we are not preprocessing the datasets used. No normalization is being used on them. However, measures to handle continuous/ nominal data have been taken.

Handling continuous/nominal attributes
For nominal/ string-type data values, we use label encoder functionality from **sklearns.preprocessing.** Label encoder groups similarly occurring values and assigns a common value/label for each occurrence. If we were to compare elements across multiple columns, we would just be calculating the Euclidean distance between the encoded label of the string representation.

For the purpose of this project, and in dataset2, we manually encode ‘Present’ and ‘Absent’ as 1 and 0 respectively.



```
for index,row in enumerate(original_data.values):
  if 'Present' in row:
    original_data.at[index,np.where(row=='Present')[0][0]] = 1
  elif 'Absent' in row:
    original_data.at[index,np.where(row=='Absent')[0][0]] = 0

print(original_data)
```

1.3 IMPLEMENTATION

Algorithm:

1. Load the Data.
2. Initialize the value of K.
3. For predicting the output class for the test data, iterate from 1st data point to the total number of data points.
    3.1 Calculate distance between test data and each row of training data by the help of Euclidean distance.

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

    3.2 Sort the calculated distance in ascending order.
    3.3 Get the top K rows from the sorted array.
    3.4 Now find out the most frequent class of the rows.
    3.5 Return the predicted class for the test data.

Parameter Setting:

- **k = number of neighbors**
  A small value of k means that noise will have a higher influence on the result and a large value make it computationally expensive. Data scientists usually choose as an odd number if the number of classes is 2 and another simple approach to select k is set **k=sqrt(n).**

- **distance function** : **Euclidean distance** is the most used similarity function. Manhattan distance, Hamming Distance, Minkowski distance are different alternatives.

- **K = no of folds for K-fold cross-validation**
  K can be any number, but **K=10** is generally recommended. This has been shown experimentally to produce the best out-of-sample estimate

  Advantages of cross-validation:
  1. More accurate estimate of out-of-sample accuracy
  2. More "efficient" use of data as every observation is used for both training and testing

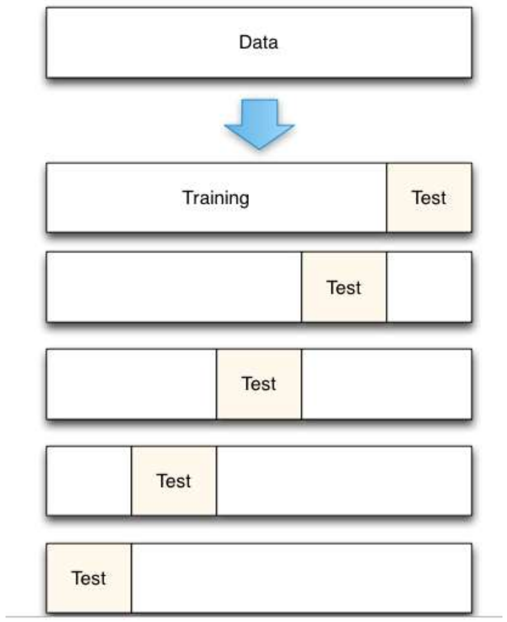


Illustration with k=5 where train split = 4 and test split =1

1.4 RESULTS

<u>Accuracies for k=5 and no of folds= 10</u>

project3_dataset1.txt (%)	**project3_dataset2.txt (%)**
96.42857142857143	52.17391304347826
89.28571428571429	69.56521739130434
96.42857142857143	65.21739130434783
91.07142857142857	56.52173913043478
94.64285714285714	56.52173913043478
91.07142857142857	52.17391304347826
91.07142857142857	63.04347826086957
96.42857142857143	76.08695652173914
92.85714285714286	54.347826086956516

90.625	54.166666666666664

Average Performance Metrics

	project3_dataset1.txt	**project3_dataset2.txt**
Average Accuracy	92.99107142857143	59.981884057971016
Average Precision	0.9292971094241512	0.4032550782550782
Average Recall	0.8772603910250968	0.2913103270998008
Average F1-Measure	0.901348817711653	0.3305024782183958

1.5 INFERENCES

1. Dataset 1 has a higher accuracy than dataset 2.
2. KNN is unsuitable for bulk datasets and test datasets which are unrelated to training sets.
3. Label encoding has helped achieve an accuracy of roundabout of 70% on dataset 2. Hence Label Encoding can be used as a pre-processing measure for nominal data.
4. When the number of instances is much larger than the number of attributes, a R-tree or a kd-tree can be used to store instances, allowing for fast exact neighbor identification
5. Scaling and/or feature selection as a measure of Pre-processing can be typically used to eliminate outliers.
6. balanced sampling of the dataset, can be of use to reduce bias in the dataset selection

Advantages:

- Very simple implementation.
- Robust with regard to the search space; for instance, classes don't have to be linearly separable.
- Classifier can be updated online at very little cost as new instances with known classes are presented.
- Few parameters to tune: distance metric and k.

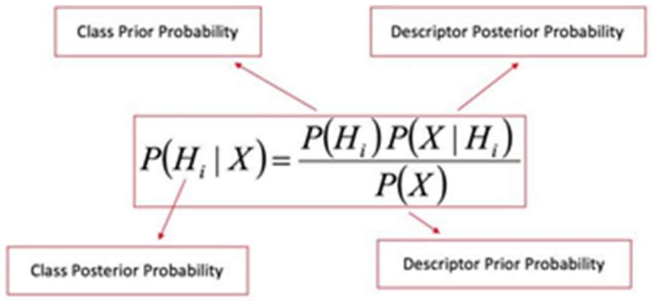
Disadvantages:

- Expensive testing of each instance, as we need to compute its distance to all known instances. -problematic for datasets with a large number of attributes.
- Sensitiveness to noisy or irrelevant attributes, which can result in less meaningful distance numbers.
- Sensitiveness to very unbalanced datasets, where most entities belong to one or a few classes, and infrequent classes are therefore often dominated in most neighborhoods.

2. Naïve Bayes

2.1 DESCRIPTION

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.


Class Prior Probability
Descriptor Posterior Probability
$P(H_i \mid X) = \frac{P(H_i)P(X \mid H_i)}{P(X)}$
Class Posterior Probability
Descriptor Prior Probability

- P(H|X) is the posterior probability of class (c, target) given predictor (x, attributes).
- P(H) is the prior probability of class.
- P(X|H) is the likelihood which is the probability of predictor given class.
- P(X) is the prior probability of predictor.

2.2 PREPROCESSING ON DATASET

Each line represents one data sample in the dataset. The last column of each line is class label, either 0 or 1. The rest of the columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type). For the purpose of our project, we are not preprocessing the datasets used. No normalization is being used on them. However, measures to handle continuous/ nominal data have been taken.

Handling continuous/nominal attributes

Continuous feature: In order to check the continuous, we first convert each value to float, if it gives the value error then the feature is categorical. In continuous, we first calculate column-wise mean and standard deviation of all the samples and use it for Gaussian probability for test data.

In categorical data, we found discrete probability separately using count on posterior and prior probability of a particular test sample.

Zero probability: Here, if the posterior probability is zero then we use Laplacian correction, but it was not necessary for our datasets.

2.3 IMPLEMENTATION

Algorithm:

1. Check whether the data contains how many categorical and continuous features and extract the columns for handling them separately.
2. Calculate number of samples in the respective classes into a dictionary taking key as classes and values as the sample rows.
3. For continuous data segment calculate column-wise mean and standard deviation in order to get Gaussian distribution. Similar for categorical calculate prior and posterior probability.
4. In testing phase, for each sample calculate conditional probability for each class and multiply continuous and categorical probability for getting total class-wise probability.
5. Check which class has higher probability which in turn will be class of that sample.

Parameter Setting:

- **K = no of folds for K-fold cross-validation**
  K can be any number, but **K=10** is generally recommended. This has been shown experimentally to produce the best out-of-sample estimate

  <u>**Cross Validation:**</u>
- We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.
- We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.
- We then extract this test set and move the remaining elements into another matrix and call them as train set.
- We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.
- We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algorithm is performed on this.
- The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.
- Finally, we calculate the average performance metric across the 10-folds.

Advantages of cross-validation:

3. More accurate estimate of out-of-sample accuracy
4. More "efficient" use of data as every observation is used for both training and testing

2.4 RESULTS

<u>Accuracies for no of folds= 10</u>

project3_dataset1.txt (%)	project3_dataset2.txt (%)
94.64285714285714	67.3913043478261
92.85714285714286	67.3913043478261
96.42857142857143	80.43478260869566
91.07142857142857	71.73913043478261
89.28571428571429	69.56521739130434
96.42857142857143	58.69565217391305
96.42857142857143	76.08695652173914
96.42857142857143	73.91304347826086
92.85714285714286	58.69565217391305
89.23076923076924	66.66666666666666

<u>Average Performance Metrics</u>

	project3_dataset1.txt	project3_dataset2.txt
Average Accuracy	93.56593406593407	69.05797101449274
Average Precision	92.02835908053301	54.60347501319369
Average Recall	90.64424464424465	69.7260217852323
Average F1-Measure	91.21166773605798	60.415460257053006

2.5 INFERENCES

1. Dataset 1 has a higher accuracy than dataset 2.
1. The data is first separated into continuous and categorical so that both of them can be handled separately and effectively.
2. It performs well in case of categorical input variables compared to numerical variable(s)
3. If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction.
4. Naive Bayes has no variance to minimize, hence boosting or optimizing algorithms used normally, won’t help.

Advantages:

- Easy to implement since each label is independent to each other.
- Faster and less complex than other classification algorithms
- This algorithm is also well known for multi class prediction feature

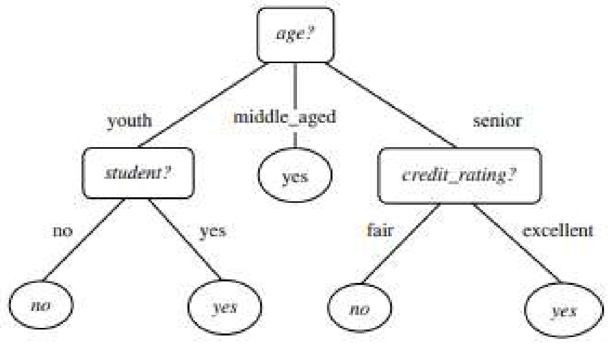
Disadvantages:

- Chances of facing zero posterior probability issues is high in case of categorical data
- Not a good choice if attributes of the data is highly dependent on each other.

3. Decision Tree

3.1 DESCRIPTION

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.



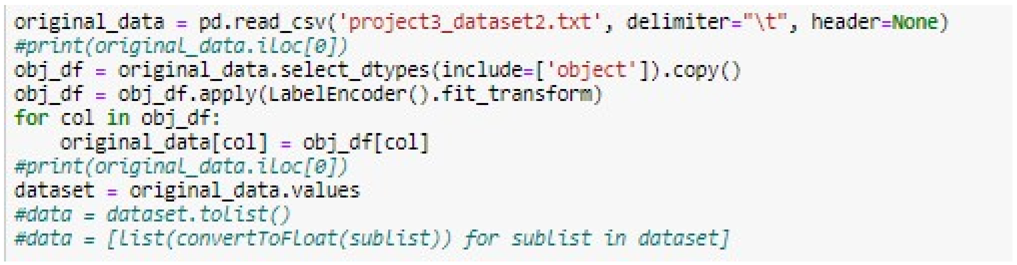
A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).

Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules

3.2 PREPROCESSING ON DATASET

Each line represents one data sample in the dataset. The last column of each line is class label, either 0 or 1. The rest of the columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type). For the purpose of our project, we are not preprocessing the datasets used. No normalization is being used on them. However, measures to handle continuous/ nominal data have been taken.

Handling continuous/nominal attributes
In order to handle all nominal values, we extracted columns with nominal values and performed label encoding on the same columns. Old columns are then replaced by the new columns in the original dataset



```
original_data = pd.read_csv('project3_dataset2.txt', delimiter="\t", header=None)
#print(original_data.iloc[0])
obj_df = original_data.select_dtypes(include=['object']).copy()
obj_df = obj_df.apply(LabelEncoder().fit_transform)
for col in obj_df:
    original_data[col] = obj_df[col]
#print(original_data.iloc[0])
dataset = original_data.values
#data = dataset.tolist()
#data = [list(convertToFloat(sublist)) for sublist in dataset]
```

Continuous feature: We take specific continuous feature value of each row as a split and compute impurity using that split. So whichever rows have feature value less than the split are considered to be in 'left' and whichever rows have a feature value >= split are considered to be in 'right'. Gini index is computed for both sets and then overall gini (impurity) is calculated for the split. We try to minimize this impurity by choosing the split which gives least impurity.

3.3 IMPLEMENTATION

Algorithm:

1. Read the text file and change all the inputs to a float value which can be handled by the algorithm.

2. We do the classification based on the Gini cost function. This provides an indication of how pure the nodes are , where node purity is how mixed the training data assigned to each node is. This helps in evaluating the splits.

3. A split in the dataset involves one input attribute and one value for that attribute. It can be used to divide training patterns into two groups of rows.
4. A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split. A perfect separation results in a Gini score of 0, whereas the worst case split that results in 50/50 classes in each group result in a Gini score of 0.5 (for a 2 class problem).

5. Given a dataset, we must check every value on each attribute as a candidate split, evaluate the cost of the split and find the best possible split we could make. Once the best split is found, we can use it as a node in our decision tree.

6. To store the split, we have used a dictionary to represent a node in the decision tree as we can store data by name. When selecting the best split and using it as a new node for the tree we will store the index of the chosen attribute, the value of that attribute by which to split and the two groups of data split by the chosen split point.

   CART ALGORITHM:

- For each predictor, all possible splits of the predictor values are considered
- For each predictor, the best split is selected. (we'll get to best split criteria)
- With the best split of each predictor is determined, we pick the best predictor in that group.

7. We use three cases for deciding whether the tree must grow or not. The parameters are :

- **max_depth** : This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes.
- **min_size :** This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes.

8. It is possible that all nodes belong to one group. In that case we kind of terminate the process of splitting the group.

9. prediction step involves implementation using a recursive function, where the same prediction routine is called again with the left or the right child nodes, depending on how the split affects the provided data.

Parameter Setting:

- **maxDepth**: This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes. For the purpose of our project, we have taken **maxDepth=3**
- **minimumSize:** This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes. **MinimumSize=1**
- **criterion** : which cost function for selecting the next tree node. Mostly used ones are gini/entropy.
- **numPartitions= no of folds for K-fold cross-validation**
  K can be any number, but **K=10** is generally recommended. This has been shown experimentally to produce the best out-of-sample estimate

  <u>**Cross Validation:**</u>

- We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.
- We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.
- We then extract this test set and move the remaining elements into another matrix and call them as train set.
- We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.

- We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algo is performed on this.
- The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.
- Finally, we calculate the average performance metric across the 10-folds.

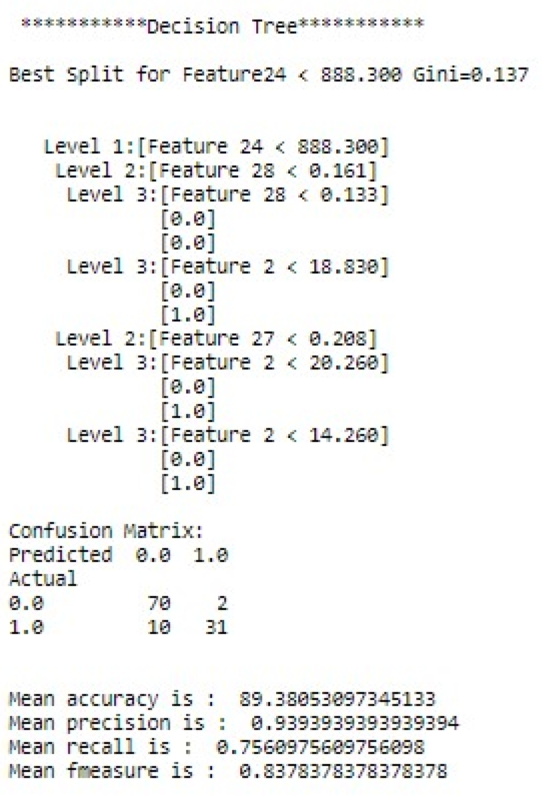
Advantages of cross-validation:

5. More accurate estimate of out-of-sample accuracy
6. More "efficient" use of data as every observation is used for both training and testing

3.4 RESULTS

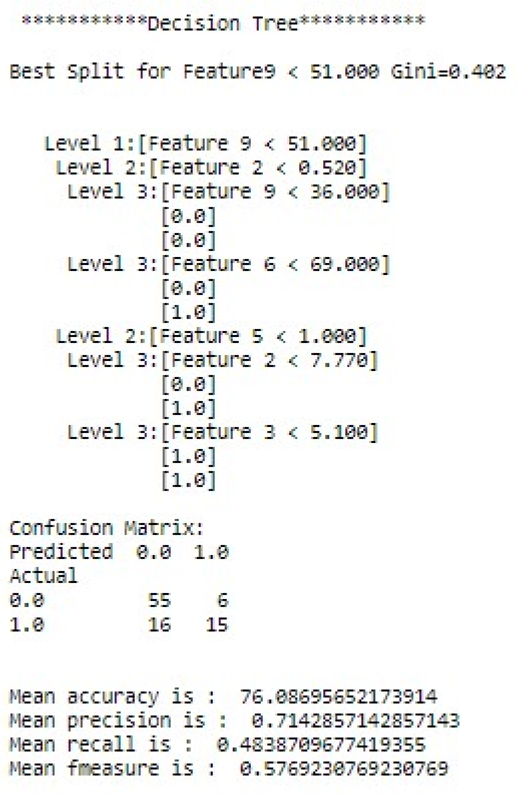
Decision Trees Constructed: maxDepth=3, numPartitions=10, minimumSize=1

Dataset1



```
 ***********Decision Tree***********

Best Split for Feature24 < 888.300 Gini=0.137


   Level 1:[Feature 24 < 888.300]
    Level 2:[Feature 28 < 0.161]
     Level 3:[Feature 28 < 0.133]
             [0.0]
             [0.0]
     Level 3:[Feature 2 < 18.830]
             [0.0]
             [1.0]
    Level 2:[Feature 27 < 0.208]
     Level 3:[Feature 2 < 20.260]
             [0.0]
             [1.0]
     Level 3:[Feature 2 < 14.260]
             [0.0]
             [1.0]

Confusion Matrix:
Predicted  0.0  1.0
Actual
0.0         70    2
1.0         10   31


Mean accuracy is :  89.38053097345133
Mean precision is :  0.9393939393939394
Mean recall is :  0.7560975609756098
Mean fmeasure is :  0.8378378378378378
```

Dataset 2



```
 ***********Decision Tree***********

Best Split for Feature9 < 51.000 Gini=0.402


   Level 1:[Feature 9 < 51.000]
    Level 2:[Feature 2 < 0.520]
     Level 3:[Feature 9 < 36.000]
             [0.0]
             [0.0]
     Level 3:[Feature 6 < 69.000]
             [0.0]
             [1.0]
    Level 2:[Feature 5 < 1.000]
     Level 3:[Feature 2 < 7.770]
             [0.0]
             [1.0]
     Level 3:[Feature 3 < 5.100]
             [1.0]
             [1.0]

Confusion Matrix:
Predicted  0.0  1.0
Actual
0.0         55    6
1.0         16   15


Mean accuracy is :  76.08695652173914
Mean precision is :  0.7142857142857143
Mean recall is :  0.4838709677419355
Mean fmeasure is :  0.5769230769230769
```

Accuracies for maxDepth=3, numPartitions(no of folds)=10, minimumSize=1

project3_dataset1.txt (%)	**project3_dataset2.txt (%)**
96.49122807017544	80.85106382978722

91.22807017543859	85.1063829787234
91.22807017543859	76.08695652173914
96.49122807017544	63.04347826086957
96.49122807017544	67.3913043478261
84.21052631578947	76.08695652173914
98.24561403508771	76.08695652173914
89.47368421052632	80.43478260869566
92.98245614035088	56.52173913043478
92.85714285714286	73.91304347826086

Average Performance Metrics

	project3_dataset1.txt	project3_dataset2.txt
Average Accuracy	93.32080200501254	73.5522664199815
Average Precision	0.9313720818162349	0.6336663852143729
Average Recall	0.8803344598709048	0.5335286131996659
Average F1-Measure	0.9036687199921228	0.5711660901204839

3.5 INFERENCES

1. Dataset 1 has a higher accuracy than dataset 2.
2. The data is first separated into continuous and categorical so that both of them can be handled separately and effectively.
3. Decision Tree so produced is visible in results above, with different levels emphasizing the structure of the tree
4. At every split, the best split is the value with the lowest Gini value. A gini value close to 0 emphasizes accurate split.
5. Decision tree is a far better classification algorithm when compared to KNN and Naïve Bayes as seen in performance/accuracy on both the datasets.
6. Although the dataset 1 does not show significance rise in accuracy, dataset 2 shows a sharp rise in average accuracy to close to 73%.

Advantages:

- Inexpensive to construct.
- Extremely fast at classifying unknown records.
- Easy to interpret for small-sized trees

- Accuracy comparable to other classification techniques for many simple data sets.
- Excludes unimportant features.
- Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
- A decision tree does not require normalization of data.
- A decision tree does not require scaling of data as well. Missing values in the data also does NOT affect the process of building decision tree to any considerable extent.

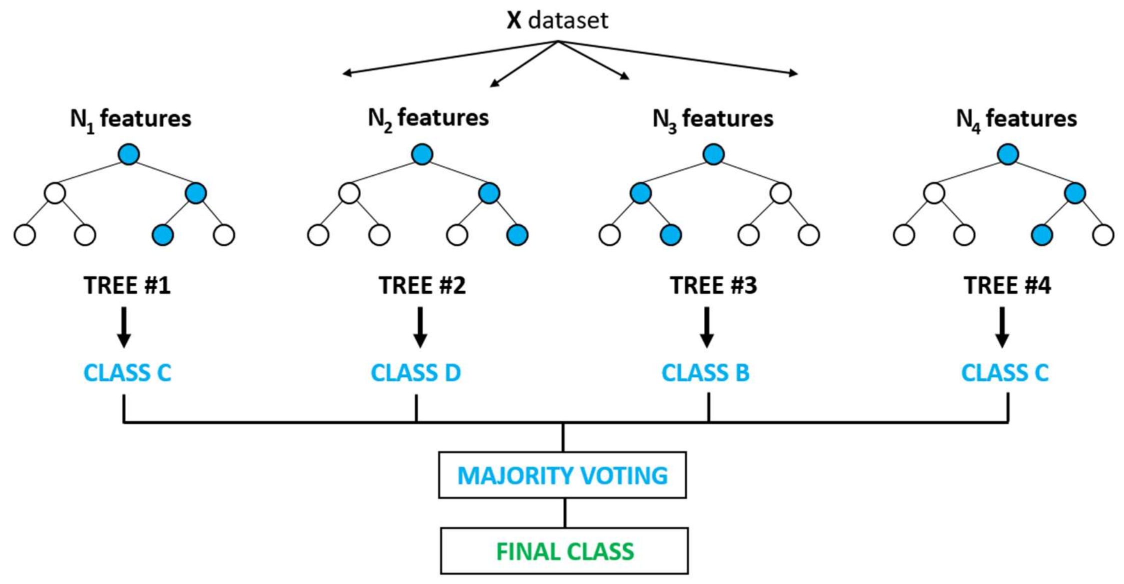
Disadvantages:

- Easy to over-fit.
- Decision Boundary restricted to being parallel to attribute axes.
- Decision tree models are often biased toward splits on features having a large number of levels.
- Small changes in the training data can result in large changes to decision logic.
- Large trees can be difficult to interpret and the decisions they make may seem counter intuitive.
- Decision tree often involves higher time to train the model.

4. Random Forest Classifier

4.1 DESCRIPTION

We now present another ensemble method called random forests. Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of techniques to Improve Classification Accuracy is a "forest." The individual decision trees are generated using a random selection of attributes at each node to determine the split. More formally, each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. During classification ,each tree votes and the most popular class is returned.



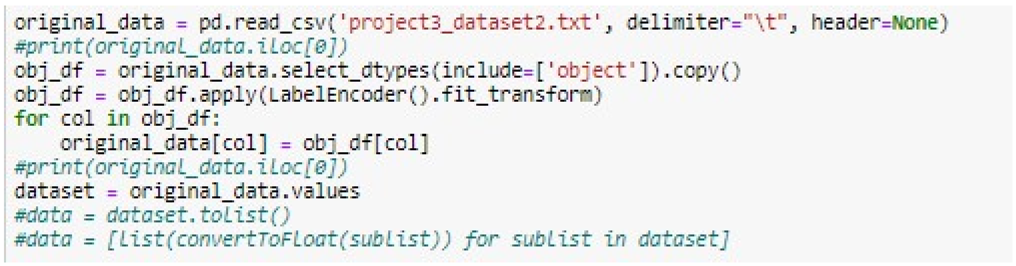
Random forests can be built using bagging in tandem with random attribute selection. A training set, D, of d tuples is given.

1.2 PREPROCESSING ON DATASET

Each line represents one data sample in the dataset. The last column of each line is class label, either 0 or 1. The rest of the columns are feature values, each of them can be a real-value (continuous type) or a string (nominal type). For the purpose of our project, we are not preprocessing the datasets used. No normalization is being used on them. However, measures to handle continuous/ nominal data have been taken.

Handling continuous/nominal attributes

In order to handle all nominal values, we extracted columns with nominal values and performed label encoding on the same columns. Old columns are then replaced by the new columns in the original dataset



```
original_data = pd.read_csv('project3_dataset2.txt', delimiter="\t", header=None)
#print(original_data.iloc[0])
obj_df = original_data.select_dtypes(include=['object']).copy()
obj_df = obj_df.apply(LabelEncoder().fit_transform)
for col in obj_df:
    original_data[col] = obj_df[col]
#print(original_data.iloc[0])
dataset = original_data.values
#data = dataset.tolist()
#data = [list(convertToFloat(sublist)) for sublist in dataset]
```

Continuous feature: We take specific continuous feature value of each row as a split and compute impurity using that split. So whichever rows have feature value less than the split are considered to be in 'left' and whichever rows have a feature value >= split are considered to be in 'right'. Gini index is computed for both sets and then overall gini (impurity) is calculated for the split. We try to minimize this impurity by choosing the split which gives least impurity.

General Algorithm:

1. Random subsets are created from the original dataset (bootstrapping).
2. At each node in the decision tree, only a random set of features are considered to decide the best split.
3. A decision tree model is fitted on each of the subsets.
4. The final prediction is calculated by averaging the predictions from all decision trees.

1.3 IMPLEMENTATION

Algorithm:

1. Read the text file and change all the inputs to a float value which can be handled by the algorithm.
2. ***RandomForest(sampleSize, n_trees,n_features, maxDepth,numPartitions,minimumSize)*** function is called with the following parameters. It creates a list of decision trees from

subsamples of the training dataset and then uses them to make predictions. Other supporting functions are:

- **subsample()-** to make a subsample of the training dataset and
- **bagging_predict()-** to make a prediction with a list of decision trees.

3. We do the classification based on the Gini cost function. This provides an indication of how pure the nodes are, where node purity is how mixed the training data assigned to each node is. This helps in evaluating the splits.
4. A split in the dataset involves one input attribute and one value for that attribute. It can be used to divide training patterns into two groups of rows.
5. A Gini score gives an idea of how good a split is by how mixed the classes are in the two groups created by the split. A perfect separation results in a Gini score of 0, whereas the worst case split that results in 50/50 classes in each group result in a Gini score of 0.5 (for a 2 class problem).
6. Given a dataset, we must check every value on each attribute as a candidate split, evaluate the cost of the split and find the best possible split we could make. Once the best split is found, we can use it as a node in our decision tree.
7. To store the split we have used a dictionary to represent a node in the decision tree as we can store data by name. When selecting the best split and using it as a new node for the tree we will store the index of the chosen attribute, the value of that attribute by which to split and the two groups of data split by the chosen split point.
8. We use three cases for deciding whether the tree must grow or not. The parameters are :
9. **max_depth** : This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes.
10. **min_size** : This is the minimum number of training patterns that a given node is responsible for. Once at or below this minimum, we must stop splitting and adding new nodes.
11. It is possible that all nodes belong to one group. In that case we kind of terminate the process of splitting the group.
12. prediction step involves implementation using a recursive function, where the same prediction routine is called again with the left or the right child nodes, depending on how the split affects the provided data.

Parameter Setting:

- **sampleSize: 1** The size of the samples so bagged
- **n_features:** total number of features in the dataset
- **n_trees:5** Number of random forests generated
- **maxDepth**: This is the maximum number of nodes from the root node of the tree. Once a maximum depth of the tree is met, we must stop splitting adding new nodes. For the purpose of our project, we have taken **maxDepth=3**
- **numPartitions**= no of folds for **K-fold cross-validation**
- K can be any number, but K=10 is generally recommended. This has been shown experimentally to produce the best out-of-sample estimate

<u>**Cross Validation:**</u>

- We are performing a 10-fold cross validation in order to get a good estimate of the accuracy measures of the algorithm. Below is the way this is implemented.

- We first shuffle the dataset and then identify the size of the test data set required for a 10-fold validation.
- We then extract this test set and move the remaining elements into another matrix and call them as train set.
- We perform regular algorithm on this test and train dataset pairs and compute the performance metrics.
- We next go on to the next fold, i.e, the next set of test data set elements are extracted and now the remaining data is the train data set, and algo is performed on this.
- The process is repeated 10 times until different segments of the dataset is considered as test dataset and performance metrics are evaluated for each of the folds.
- Finally, we calculate the average performance metric across the 10-folds.

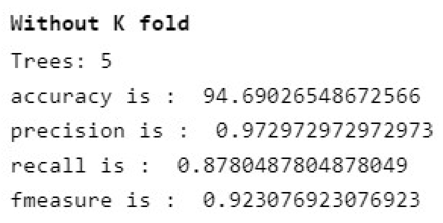
Advantages of cross-validation:

7. More accurate estimate of out-of-sample accuracy
8. More "efficient" use of data as every observation is used for both training and testing

1.3 RESULTS

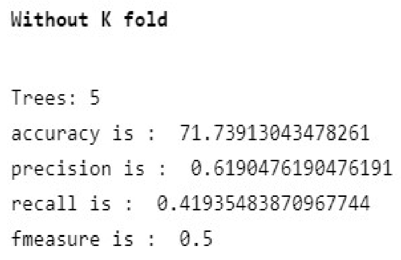
Final Performance Metric of trees constructed

Dataset1



```
Without K fold
Trees: 5
accuracy is :  94.69026548672566
precision is :  0.972972972972973
recall is :  0.8780487804878049
fmeasure is :  0.923076923076923
```

Dataset 2



```
Without K fold

Trees: 5
accuracy is :  71.73913043478261
precision is :  0.6190476190476191
recall is :  0.41935483870967744
fmeasure is :  0.5
```

Accuracies for **k fold cross validation with k=10**

project3_dataset1.txt (%)	project3_dataset2.txt (%)
95.57522123893806	71.73913043478261
96.46017699115043	68.47826086956522
96.46017699115043	72.82608695652173
94.69026548672566	70.65217391304348
96.46017699115043	68.47826086956522
96.46017699115043	77.17391304347827

96.46017699115043	81.52173913043478
92.03539823008849	76.08695652173914
96.46017699115043	69.56521739130434
93.80530973451327	72.82608695652173

Average Performance Metrics

	project3_dataset1.txt	project3_dataset2.txt
Average Accuracy	95.4867256637168	72.93478260869566
Average Precision	0.9471055844740055	0.5037434787434787
Average Recall	0.9184210526315789	0.472
Average F1-Measure	0.931954174528075	0.48490371366516405

3.6 INFERENCES

1. Dataset 1 has a higher accuracy than dataset 2.
2. The data is first separated into continuous and categorical so that both of them can be handled separately and effectively.
3. Random Forest Classifier has all the ***hyperparameters*** of a Decision Tree Classifier to control the tree's growth, with few exceptions, plus all hyperparameters from **Bagging Classifier** to control the ensemble
4. instead of searching for the best feature when splitting a node, it searches for the best feature among a random subset of features
5. Random Forest has a comparable accuracy metric when compared to Decision Forest, but way higher than KNN and Naïve Bayes as seen in performance/accuracy on both the datasets.

Advantages:

- It is robust to correlated predictors.
- It is used to solve both regression and classification problems.
- It can be also used to solve unsupervised ML problems.
- It can handle thousands of input variables without variable selection.
- It can be used as a feature selection tool using its variable importance plot.
- It takes care of missing data internally in an effective manner.

Disadvantages:

- The Random Forest model is difficult to interpret.

- It tends to return erratic predictions for observations out of range of training data.
- It can take longer than expected time to computer a large number of trees.

CONCLUSION

Final Performances across all classifiers and hyper-parameters can be tabulated and compared as follows:

KNN	**project3_dataset1.txt**	**project3_dataset2.txt**
Average Accuracy	92.99107143	59.98188406
Average Precision	0.929297109	0.403255078
Average Recall	0.877260391	0.291310327
Average F1-Measure	0.901348818	0.330502478

Naïve Bayes	**project3_dataset1.txt**	**project3_dataset2.txt**
Average Accuracy	93.56593407	69.05797101
Average Precision	92.02835908	54.60347501
Average Recall	90.64424464	69.72602179
Average F1-Measure	91.21166774	60.41546026

Decision Tree	**project3_dataset1.txt**	**project3_dataset2.txt**
Average Accuracy	93.32080201	73.55226642
Average Precision	0.931372082	0.633666385
Average Recall	0.88033446	0.533528613
Average F1-Measure	0.90366872	0.57116609

Random Forest	**project3_dataset1.txt**	**project3_dataset2.txt**
Average Accuracy	95.48672566	72.93478261
Average Precision	0.947105584	0.503743479
Average Recall	0.918421053	0.472
Average F1-Measure	0.931954175	0.484903714

From the above comparisons, we can infer:

- Naive bayes is much faster than KNN due to KNN's real-time execution and is parametric while KNN is non parametric
- Random Forest is a collection of decision trees and average/majority vote of the forest is selected as the predicted output.
- Random Forest model will be less prone to overfitting than Decision tree, and gives a more generalized solution.
- Random Forest is more robust and accurate than decision trees.
- Decision trees are more flexible and easy.
- Decision tree pruning may neglect some key values in training data, which can lead the accuracy for a toss.
- Random Forest is a complex and large model whereas Naive Bayes is a relatively smaller model.
- Random Forest has an accuracy of 95% for dataset 1 (most optimum algorithm)
- Decision Tree has an accuracy of 73% for dataset 2 (most optimum for this dataset)
- Therefore, most preferred algorithm for the datasets could be Random Forest and Decision tree
- According to other performance metrics:
  - **Precision** is analogous to accuracy, hence is the same as the accuracy
  - **Recall** is also known as sensitivity or is a measure of relevance. For dataset2, **Naïve Bayes** has demonstrated high sensitivity
- **F1-score** is a better metric when there are imbalanced classes which may or may not be in the case of dataset1 and dataset 2.
  As per F1 score, the ideal classifier for dataset 2 should be **Naïve Bayes**, thus showing some speculation towards the nature of the data.
- The higher the F1 scores, the better the performance.

The ***Performance metrics- Accuracy, Precision, Recall, f1-Score*** have been calculated using a confusion matrix and through the use of true positives, true negatives, false positives, and false negatives generated by the same.
The distribution of training data is the key criteria for selecting a suitable algorithm. Yet, we make some general assumptions during the selection of algorithms, based on training size, type of features, number of features, computational and space complexity etc.

KAGGLE COMPETITION

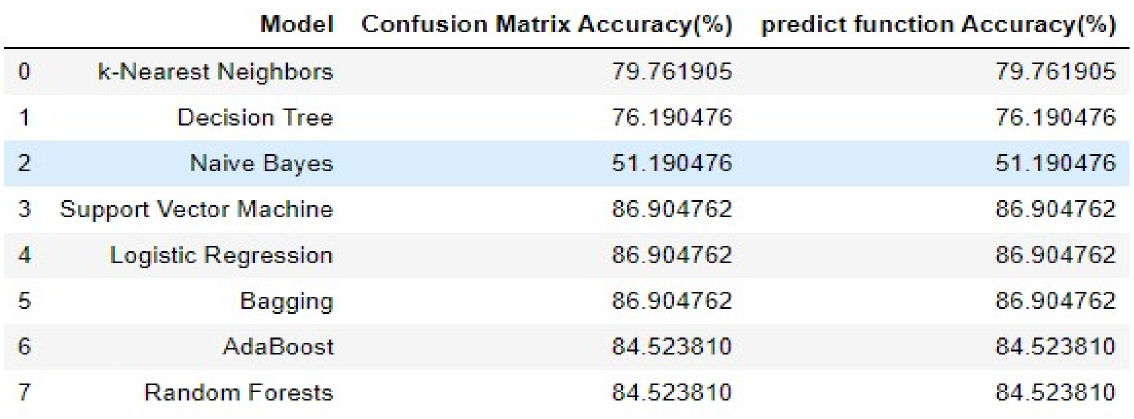
Problem Statement:

An invite for a Kaggle competition will be sent. For that dataset, we hold out the class labels for testing data. Apply various tricks on top of any classification algorithm discussed in class (including nearest neighbor, decision tree, Naïve Bayes, SVM, logistic regression, bagging, AdaBoost, random forests) and tune parameters using training data. You can call packages for these algorithms but need to implement any improvement on top of these algorithms.

Solution Implemented:

1. The training and test data is first read from the provided csv files.
2. The training data is split into training and test set to measure the performance of our model.

3. Random Forest Classifier is used first with default parameters and performance calculated.
4. The decision of choosing Random Forest Classifier stems from the fact that we tried training the various models listed with the training data and training label, and assessed which algorithm returned the optimum accuracy.



	Model	Confusion Matrix Accuracy(%)	predict function Accuracy(%)
0	k-Nearest Neighbors	79.761905	79.761905
1	Decision Tree	76.190476	76.190476
2	Naive Bayes	51.190476	51.190476
3	Support Vector Machine	86.904762	86.904762
4	Logistic Regression	86.904762	86.904762
5	Bagging	86.904762	86.904762
6	AdaBoost	84.523810	84.523810
7	Random Forests	84.523810	84.523810

We then chose to go ahead with **Random Forest Classifier**.

5. Then, Hyper Parameter Tuning is done using **Randomized Search CV** and **Grid Search CV**.
    - Randomized Search CV does a Randomized search on hyper parameters.
    - Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.
6. Multiple values for the parameters of the Random Forest function are taken like n_estimators, max_features and max_depth etc.
7. The Randomized search is done for the Random Forest Classifier using the defined values (trained for each parameter combination) and the best parameters are obtained.
8. Then, a model is created using the above obtained parameters and the model is trained on the training set.
9. The trained model is then used to predict the labels of the test set.
10. Same process of determining the best parameters is done using the Grid Search CV parameter tuning.
11. The best parameters are determined, model trained using these parameters and the performance evaluated on the test set.
12. The predicted labels are saved in a csv which is used for submission on the Kaggle website for evaluation.
13. Also, Gaussian Naïve Bias model is used to train on the train set and used to predict the labels for the test set and saved to be evaluated on Kaggle.

ACKNOWLEDGMENT

Heartfelt thanks to Professor Jing Gao and TAs who have helped us to ace this course.

REFERENCE

- https://www.ritchieng.com/machine-learning-cross-validation/
- https://machinelearningmastery.com/implement-random-forest-scratch-python/
- https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?#
- https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222
- https://medium.com/machine-learning-101/chapter-3-decision-trees-theory-e7398adac567
- https://github.com/arunshar/Academic-Projects/blob/master/Data%20Mining%20and%20Bioinformatics/Classification%20with%20Ensemble%20Learning/NaiveBayes.py
- https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn
- https://www.datacamp.com/community/tutorials/decision-tree-classification-python
- https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/
- https://www.saedsayad.com/k_nearest_neighbors.htm
- https://www.python-course.eu/k_nearest_neighbor_classifier.php