Data Mining
Project 3

Arvind Srinivass Ramanathan | **arvindsr** | **50205659**
Naveen Muralidhar Prakash | **naveenmu** | **50208032**
Senthil Kumar Laguduva Yadindra Kumar | **laguduva** | **50207553**

K-Nearest Neighbors Algorithm

K-Nearest Neighbors Algorithm:

K-NN Classification algorithm is a simple supervised algorithm which classifies the unknown test records sample based on the class which is most common among its K-Nearest Neighbors. The K-Nearest Neighbors for each test record are chosen based on the nearest distances to the training sample.

Implementation:

- The input file was read and the classification labels (ground truth labels) are found and saved into *'class_labels'*.
- The features are loaded into the matrix called *'matrix'*.
- If there are any **nominal features**, they are normalized using **LabelEncoder** and updated in the feature matrix, and now the overall feature matrix is normalized by finding the mean and standard deviation of each feature . Hence now all the features are normalized and are numeric.
- Now the normalized feature matrix is validated using 10-Fold Cross Validation method. Here the feature matrix is split into 10 parts, where one part is used as a training set and the other 9 parts are used as a training data set at each fold/iteration.
- For each data sample in the testing set, the Euclidean distance to each of the data samples in the training set is calculated.
- Then the K-Nearest Neighbors are chosen based on the minimum distances, which were calculated in the previous step.
- The testing record, whose classification label is unknown at this time, is assigned a classification label based on the majority votes of the classification label of the K-Neighbors.
- This process is repeated for each of the test records in this K-Fold testing set and the Accuracy, Precision, Recall and F-Measure are calculated.
- Now we pick another dataset from the remaining 9 datasets as the testing set and remaining parts are used for training and the above steps are repeated. We continue this process until each of the parts is used as a testing set once.
- The final performance measures are calculated, which is the mean of the values calculated at each fold.

Choosing K:

In KNN, the value of K determines how many neighbors have to be considered before assigning a class label for a test record. We ran the algorithm iteratively from K=1 to find out the value of K which gives the maximum accuracy and good precision and recall. For dataset 1, the K value chosen is 11 and for dataset2, the K value chosen is 13.

The optimum K in KNN always varies depending on the data set. If the dataset is big, any noise never affects the prediction. A small value of K means that the noise affects the result more. A large value of K makes the process computationally expensive and defeats the main philosophy behind KNN. Sometimes, the square root on N can be a good number, but generally the best approach is to try many K values and do cross validation in order to determine the optimum K.

Results for project3_dataset1.txt at K = 11:

Accuracy: 0.970144110276
Precision: 0.989769820972
Recall: 0.929113565638
F-Measure: 0.956865584764

Results for project3_dataset2.txt at K = 13:

Accuracy: 0.714292321924
Precision: 0.661022588523
Recall: 0.385946560749
F-Measure: 0.474015017406

Calculating Distance for Continuous and Categorical Features for K-NN:

The nominal or categorical features in the dataset are first encoded to numbers based on the unique values present in a particular feature. This is updated in the matrix and the feature matrix is normalized based on mean and standard deviation. **Euclidean Distance** is used to calculate the distance between each test record and the training samples to determine the neighbors.

Pros and Cons:

K-NN:

Pros:

- K-NN works well for datasets that have very small dimensions.
- Since K-NN doesn’t learn anything extensive in the training process, there is virtually no training process involved.
- Since K-NN keeps hold of all the training data, there is no loss in information that occurs from the training data.

Cons:

- In K-NN, we use the training data itself for classification. Hence there is no actual learning from the training data i.e., one does not learn anything from the training data and the data is directly used for classification.
- The value of K to be chosen has a large impact on the classification accuracy. Having a very large or a very small K value results in wrong classification. Hence it is difficult and tedious to determine the accurate K value to be used for choosing neighbors.
- The overhead in computing distances from a test sample to each of the training records in the data is computationally high. Also for every test sample, finding the minimum K values requires sorting to be performed across the distance matrix. Since we have to calculate the distances to choose the K-neighbors for each testing sample, the complexity of calculating distances is high, which is basically Number of Testing records * Number of Training Records.
- Since the distance measure can be calculated through a variety of methods (Example: Manhattan, Euclidean etc.), the algorithm is heavily reliant on the method chosen, and choosing a different distance measure can result in [a major difference in how] a test record is classified.

Naive Bayesian Classification Algorithm:

Naive Bayesian is a classification algorithm based on the Bayes Theorem. The objective of the algorithm is to classify unknown samples based on the probability of a test sample being in a class $\mathbf{H_i}$. For nominal features, probability is calculated as counts, while for continuous features, probability density function is used . The algorithm works on the principle that each feature is independent of the other.

Implementation:

- The input file was read and the classification labels (ground truth labels) are found and saved into *'class_labels'*.
- The features are loaded into the matrix called *'matrix'*.
- The matrix is validated using 10-Fold cross validation method. Here, the feature matrix is split into 10 parts, where one part is used as a testing set and the other 9 parts are used as a training dataset at each fold/iteration.
- The Class Prior Probability $\mathbf{P(H_i)}$ is calculated from the training data set as the number of training samples of class C_i divided by the total number of training samples.
- The descriptor posterior probability $\mathbf{P(X \mid H_i)}$ is calculated. For each training sample, say X, we have X=(X1, X2,…. Xd) , where each feature is independent of the other. Hence $P(X \mid H_i) = P(X_1 \mid H_1) * P(X_2 \mid H_2)$ .. and so on.
- **For nominal features, posterior probabilities were found by just counting through the features. But for continuous features, we cannot count the continuous values to get posterior probabilities. Hence, for continuous features, we used Gaussian Distribution to find posterior probability.**

- For each continuous feature, the mean and standard deviation are found and probability density function is used to calculate posterior probability for a continuous feature.
- The Class Posterior Probability **$P(H_i \mid X)$** is calculated by multiplying **P(Hi)** and **$P(X|H_i)$** **.** Since we are dealing with binary classification, we calculate the probability for the sample to occur in class label 0 or class label 1.
- We assign the sample 0 or 1 based on the probability calculated in the previous step. [P(H0 | X) and P(H1 | X)]
- This is continued for all samples in the testing set and the performance measure for this fold is calculated. The process is continued until each of the folds is used as a testing set at least once.

Dealing with Continuous and Categorical Features:

Since each feature is independent of the other, the continuous features are assumed to be probabilistic distributions, hence they follow Gaussian distribution. Hence P.D.F is used to calculate $P(X \mid H_i)$ for all continuous features . For categorical features, the probability is easily calculated by finding the counts of occurrences across the feature.

Results:

Naive Bayesian Results for project3_dataset1.txt at the end of cross validation

Accuracy: 0.934899749373
Precision: 0.917870936253
Recall: 0.904442635683
F-Measure: 0.910031838126

Naive Bayesian Results for project3_dataset2.txt at the end of cross validation

Accuracy: 0.703469010176
Precision: 0.570927564008
Recall: 0.61593964515
F-Measure: 0.586739645814

Pros:

- Unlike K-NN, irrelevant data features don’t affect Naive Bayesian, since the features are assumed to be independent of each other.
- The algorithm can handle data of all types: - Real, continuous and nominal types.
- The training and the testing process of Naive Bayesian algorithm are simple and fast.

Cons:

- Naive Bayes classification has a strong assumption that features are independent of each other. Hence, the classification can be wrong since it assumes that there is no dependence between features.
- It runs good only for large datasets and poorly for smaller datasets since we need to estimate probability of each class. Hence for a small data set, the precision and recall will be very low.

Zero-Probability in Naive Bayesian:

Calculating the posterior probability in Naive Bayesian has its own disadvantages. Since we are dealing with the assumption of independent features , a single value $P(\mathbf{x_j} \mid \mathbf{H_i})$ can result in zero probability making the entire product 0 , which in turn makes $\mathbf{P(H_i \mid X)} = 0$. This issue of zero probability can be handled using Laplacian Correction , where we simply add 1 to each case ($\mathbf{n_j}$) , and increase the total number of samples to n+ f , where f is the total number of values a attribute can take.

Cross Validation using K-Fold

We implemented Cross Fold Validation by using K-Fold method by shuffling through the dataset. The entire dataset is split into 10 parts, where 9 parts are used for training and 1 for testing. The performance measures such as Accuracy, Precision, Recall and F-Measure are calculated. This process is repeated until each of the parts generated using K-Fold are used as a testing set at least once. The final performance measure values are the mean of the values generated at the end of each fold.

Decision Tree

What is a decision tree?

A decision tree is a non-parametric learning A decision tree is a decision support tool which uses a graph or a tree like structure to map the decisions to their possible consequences and outcomes. They are able to fully include the advantages and disadvantages of different courses of actions based on the choices we have to make.

Advantages of decision tree:

- Visualization of a decision allows one to get a clear a picture of the flow of decisions that are being made. We will be able to trace the outcome of the event based on any decisions we make at any point.
- Pre-processing the data for building the decision tree is minimal as in most cases we require to handle empty fields and they should be filled with other values when found empty.

- The cost of traversing the tree is logarithmic to the number of records or rows or tuples used to build the decision tree.
- We have accommodations to handle both numerical and categorical data.
- Decision tree has the ability to handle different outputs based on the attributes we are testing for.
- This is a white box model of implementation. We have access to the sub systems internals, thus giving us a clear view about the working of the decision tree.
- We can generate the validity of some testing data based on different statistical models. This allows us to increase the validity and reliability of the model in ways we feel is best.
- Gives a fairly good result even when the initial assumptions are violated by the true model from which the data is generated.

Disadvantages of decision tree:

- Decision trees sometimes create over-complex trees that do not generalize accurately. This is called **over-fitting**. There are some methods like pruning, setting the minimum number of samples required at the leaf node or setting maximum depth of the tree are required as a work around in this case.
- Decision trees can be **unstable**. Small changes in classification measures or records generated will produce a different tree thus rendering them intolerant of small mistakes or errors in the training data.
- Learning an optimal decision tree is an NP–complete problem. Thus, we make locally optimal decision at each node. There is no guarantee that it will return an optimal algorithm.
- Decision trees are limited to classification. They cannot make predictions in continuous output like price.

Implementation:

1. Read the file into *recordsFromFile* make note of the categorical fields there are in the dataset in to a list *listCatIndex*.
2. Pass the *recordsFromFile* to the method *crossValidData()*
3. *crossValidData()* returns two lists with all the folds generated training data *trainRecordsList()* and testing records *testRecordsList()*.
4. Initialize lists to calculate the accuracy, precision, recall, f_measure for the K folds to be printed at the end.
5. Run for loop for the number folds generated.
6. Generate the lists of rows from the initial training set into *npUniqueTrainRec* and *npDuplicateTrainRec*, a union total of both the set of row indexes in *npTempTrainRecords*.
7. Populate the *npTrainRecords* with the rows from *npTempTrainRecords*.

8. Call *runTreeBuild* in recursion with initialization values for a constructor.
9. *runTreeBuild()* will run in recursion until the entire tree is built.
10. *runTreeBuild()* will run by calculating the measurement of the given attributes. We calculate the information gain values for the rows.
11. For Information gain we count the number of zero's and one's both greater than and lesser than the given value.
12. Based on this formula we can evaluate the entropy of an attribute from which we find the gain.
13. When maximizing this gain we will know the point of best split for the given data attributes.
14. First calculate the purity of the dataset being passed to the function using *calculatePurity()*. Note the same into *impurityStatus*.
15. If: the dataset is pure we declare a node instance and initialize the same with default values for a leaf node and declare a classification for the output using *classifyNode()* which will output the class variable that is present in the dataset.
16. Else: In the case of an impure subset we need to now run execution for nominal attributes and numerical attributes separately.
17. Now if the chosen best attribute is of nominal type we need to separate the dataset into two parts with values equal to the splitting value in the left and the unequal values in the right.
18. Call the function in recursion with the left and right subsets separately accepting left and right child nodes for the node at this iteration.
19. If there is no good split or if every attribute is appearing in the left we make the current node a leaf node.
20. If the attribute is a numerical attribute then we move the values less than the target value to the left and those with higher target value than that to the right.
21. If we have any records in the left we continue recursion with the records in the left and right.
22. Run loop for all rows in *testRecordsList()* we send the line to the decision tree and bring out the decision we need and multiply the same with a list of classifier weights.
23. We then generate the class for each row and define the row to hold a class value as the one which produces the highest value when multiplied with the weights of the classifiers.
24. Now extract the ground truth for the rows and compare them to the predicted results.
25. Calculate the accuracy, precision, recall, F-measure for the ith fold and mean once the loop is over.

Analysis:

project3_dataset1.txt	project3_dataset2.txt
Metrics for fold iteration: 1 Accuracy: 0.8771929824561403 Precision: 0.8333333333333334 Recall: 0.8695652173913043 F Measure: 0.851063829787234	Metrics for fold iteration: 1 Accuracy: 0.6382978723404256 Precision: 0.6 Recall: 0.5714285714285714 F Measure: 0.5853658536585366
Metrics for fold iteration: 2 Accuracy: 0.8947368421052632 Precision: 0.7894736842105263 Recall: 0.8823529411764706 F Measure: 0.8333333333333334	Metrics for fold iteration: 2 Accuracy: 0.6808510638297872 Precision: 0.4375 Recall: 0.5384615384615384 F Measure: 0.4827586206896552
Metrics for fold iteration: 3 Accuracy: 0.9649122807017544 Precision: 1.0 Recall: 0.8571428571428571 F Measure: 0.9230769230769231	Metrics for fold iteration: 3 Accuracy: 0.6304347826086957 Precision: 0.5714285714285714 Recall: 0.42105263157894735 F Measure: 0.48484848484848486
Metrics for fold iteration: 4 Accuracy: 0.9649122807017544 Precision: 0.9523809523809523 Recall: 0.9523809523809523 F Measure: 0.9523809523809523	Metrics for fold iteration: 4 Accuracy: 0.6304347826086957 Precision: 0.42857142857142855 Recall: 0.4 F Measure: 0.41379310344827586
Metrics for fold iteration: 5 Accuracy: 0.8421052631578947 Precision: 0.72 Recall: 0.9 F Measure: 0.8	Metrics for fold iteration: 5 Accuracy: 0.6086956521739131 Precision: 0.6666666666666666 Recall: 0.2 F Measure: 0.3076923076923077
Metrics for fold iteration: 6 Accuracy: 0.9649122807017544 Precision: 1.0 Recall: 0.9259259259259259 F Measure: 0.9615384615384616	Metrics for fold iteration: 6 Accuracy: 0.5 Precision: 0.3684210526315789 Recall: 0.3888888888888889 F Measure: 0.3783783783783784
Metrics for fold iteration: 7 Accuracy: 0.9473684210526315 Precision: 0.9130434782608695 Recall: 0.9545454545454546 F Measure: 0.9333333333333333	Metrics for fold iteration: 7 Accuracy: 0.6304347826086957 Precision: 0.35714285714285715 Recall: 0.38461538461538464 F Measure: 0.37037037037037035
Metrics for fold iteration: 8 Accuracy: 0.9649122807017544 Precision: 0.9333333333333333 Recall: 0.9333333333333333 F Measure: 0.9333333333333333	Metrics for fold iteration: 8 Accuracy: 0.7608695652173914 Precision: 0.5 Recall: 0.6363636363636364 F Measure: 0.56
Metrics for fold iteration: 9 Accuracy: 0.9122807017543859 Precision: 0.9333333333333333 Recall: 0.9032258064516129 F Measure: 0.9180327868852459	Metrics for fold iteration: 9 Accuracy: 0.5652173913043478 Precision: 0.35714285714285715 Recall: 0.3125 F Measure: 0.3333333333333333

Metrics for fold iteration: 10 Accuracy: 0.8928571428571429 Precision: 0.9 Recall: 0.8181818181818182 F Measure: 0.8571428571428571	Metrics for fold iteration: 10 Accuracy: 0.6304347826086957 Precision: 0.42105263157894735 Recall: 0.5714285714285714 F Measure: 0.48484848484848486
Average score for all 10 records: **Accuracy is: 0.922619047619** **Precision is: 0.897489811485** **Recall is: 0.899665430653** **F-Measure is: 0.896323581081**	**Average score for all 10 records:** **Accuracy is: 0.62756706753** **Precision is: 0.470792606516** **Recall is: 0.442473922277** **F-Measure is: 0.440138893727**

Random Forests

Random forests are an ensemble learning method for classification, regression that operate by constructing a multitude of decision trees at training time and outputting the class as a result of a statistical formulae. They correct the over fitting errors that can arise out of the training data.

Advantages of random forests:

- For multiple datasets, we get highly accurate results. This is because we implement the tree generation based on bagging results.
- It runs efficiently on large datasets.
- They require no input preparation.
- They perform implicit feature selection which can sometimes be randomized for generating different trees.
- Random forests are quick to train.
- It's really hard to build a bad random forest. Since it a collection of different trees, a cumulative result of all those results will generate the weighted prediction for the rows.
- Random forests are versatile, they can accommodate the results of different trees.
- If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups.
- It has methods for balancing error in class population unbalanced data sets.

Disadvantages of random forests:

- Unlike decision trees, the classifications made by random forests are difficult for humans to interpret.
- They are quite slow with both training and testing data.

Implementation:

26. Read the file into *recordsFromFile* make note of the categorical fields there are in the dataset in to a list *listCatIndex*.
27. Pass the *recordsFromFile* to the method *crossValidData()*
28. *crossValidData()* returns two lists with all the folds generated training data *trainRecordsList()* and testing records *testRecordsList()*.
29. Initialize lists to calculate the accuracy, precision, recall, f_measure for the K folds to be printed at the end.
30. Run for loop for the number folds generated.
31. Calculate 63% and 37% of the training rows being considered in a single iteration of K fold. Save the same to *uniqueNoOfRec* and *duplicateNoOfRec*.
32. Generate the lists of rows from the initial training set into *npUniqueTrainRec* and *npDuplicateTrainRec*, a union total of both the set of row indexes in *npTempTrainRecords*.
33. Populate the *npTrainRecords* with the rows from *npTempTrainRecords*.
34. Call *runTreeBuild* in recursion with initialization values for a constructor.
35. *runTreeBuild()* will run in recursion until the entire tree is built.
36. *runTreeBuild()* will run by calculating the measurement of the given attributes. We calculate the information gain values for the rows.
37. For Information gain we count the number of zero's and one's both greater than and lesser than the given value.
38. Based on this formula we can evaluate the entropy of an attribute from which we find the gain.
39. When maximizing this gain we will know the point of best split for the given data attributes.
40. First calculate the purity of the dataset being passed to the function using *calculatePurity()*. Note the same into *impurityStatus*.
41. If: the dataset is pure we declare a node instance and initialize the same with default values for a leaf node and declare a classification for the output using *classifyNode()* which will output the class variable that is present in the dataset.
42. Else: In the case of an impure subset we need to now run execution for nominal attributes and numerical attributes separately.
43. We will generate a random list of attributes.
44. We find the best separation point of the chosen attributes.
45. Now if the chosen best attribute is of nominal type we need to separate the dataset into two parts with values equal to the splitting value in the left and the unequal values in the right.
46. Call the function in recursion with the left and right subsets separately accepting left and right child nodes for the node at this iteration.
47. If there is no good split or if every attribute is appearing in the left we make the current node a leaf node.

48. If the attribute is a numerical attribute then we move the values less than the target value to the left and those with higher target value than that to the right.
49. If we have any records in the left we continue recursion with the records in the left and right.
50. Run loop for all rows in *testRecordsList()* we send the line to the decision tree and bring out the decision we need and populate a two dimensional array for collecting the votes.
51. We then generate the majority for each row and classify that row using that value.
52. Now extract the ground truth for the rows and compare them to the predicted results.
53. Calculate the accuracy, precision, recall, F-measure for the ith fold and mean once the loop is over.

Analysis:

project3_dataset1.txt	**project3_dataset2.txt**
Metrics for fold iteration: 1 Accuracy: 0.9473684210526315 Precision: 0.9166666666666666 Recall: 0.9565217391304348 F Measure: 0.9361702127659575	Metrics for fold iteration: 1 Accuracy: 0.7021276595744681 Precision: 0.6666666666666666 Recall: 0.6666666666666666 F Measure: 0.6666666666666666
Metrics for fold iteration: 2 Accuracy: 0.9473684210526315 Precision: 0.8888888888888888 Recall: 0.9411764705882353 F Measure: 0.9142857142857143	Metrics for fold iteration: 2 Accuracy: 0.8936170212765957 Precision: 0.7857142857142857 Recall: 0.8461538461538461 F Measure: 0.8148148148148148
Metrics for fold iteration: 3 Accuracy: 0.9824561403508771 Precision: 1.0 Recall: 0.9285714285714286 F Measure: 0.9629629629629629	Metrics for fold iteration: 3 Accuracy: 0.717391304347826 Precision: 0.7142857142857143 Recall: 0.5263157894736842 F Measure: 0.6060606060606061
Metrics for fold iteration: 4 Accuracy: 0.9824561403508771 Precision: 0.9545454545454546 Recall: 1.0 F Measure: 0.9767441860465116	Metrics for fold iteration: 4 Accuracy: 0.6304347826086957 Precision: 0.4166666666666667 Recall: 0.3333333333333333 F Measure: 0.37037037037037035
Metrics for fold iteration: 5 Accuracy: 0.9473684210526315 Precision: 0.9047619047619048 Recall: 0.95 F Measure: 0.926829268292683	Metrics for fold iteration: 5 Accuracy: 0.6521739130434783 Precision: 0.75 Recall: 0.3 F Measure: 0.42857142857142855
Metrics for fold iteration: 6 Accuracy: 0.9473684210526315 Precision: 0.9615384615384616	Metrics for fold iteration: 6 Accuracy: 0.5652173913043478 Precision: 0.4

Recall: 0.9259259259259259 F Measure: 0.9433962264150944	Recall: 0.2222222222222222 F Measure: 0.2857142857142857
Metrics for fold iteration: 7 Accuracy: 0.9824561403508771 Precision: 0.9565217391304348 Recall: 1.0 F Measure: 0.9777777777777777	Metrics for fold iteration: 7 Accuracy: 0.8043478260869565 Precision: 0.75 Recall: 0.46153846153846156 F Measure: 0.5714285714285714
Metrics for fold iteration: 8 Accuracy: 0.9824561403508771 Precision: 1.0 Recall: 0.9333333333333333 F Measure: 0.9655172413793104	Metrics for fold iteration: 8 Accuracy: 0.8043478260869565 Precision: 0.6 Recall: 0.5454545454545454 F Measure: 0.5714285714285714
Metrics for fold iteration: 9 Accuracy: 0.9298245614035088 Precision: 1.0 Recall: 0.8709677419354839 F Measure: 0.9310344827586207	Metrics for fold iteration: 9 Accuracy: 0.5434782608695652 Precision: 0.2222222222222222 Recall: 0.125 F Measure: 0.16
Metrics for fold iteration: 10 Accuracy: 0.9107142857142857 Precision: 0.8695652173913043 Recall: 0.9090909090909091 F Measure: 0.8888888888888888	Metrics for fold iteration: 10 Accuracy: 0.7391304347826086 Precision: 0.6 Recall: 0.42857142857142855 F Measure: 0.5
Average score for all 10 records: **Accuracy is: 0.955983709273** **Precision is: 0.945248833292** **Recall is: 0.941558754858** **F-Measure is: 0.942360696157**	**Average score for all 10 records:** **Accuracy is: 0.705226641998** **Precision is: 0.590555555556** **Recall is: 0.445525629341** **F-Measure is: 0.497505531506**

We note that here when we sample the dataset based on some repeating rows the classifier incurs the same data repeatedly thus is able to learn better. Its accuracy has improved. This is due to the random sampling of the rows and columns of the entire dataset.
This allows for the tree to be built with a variety of inputs. Upon gathering the vote we have a higher

Adaptive Boosting

The output of the different learning algorithms is combined into a weighted sum that represents the final output of the boosted classifier. The adaptive in the name comes from how the decision tree is able to tweak the weight of the weak learners by reducing that and increasing the ones that are correctly classified. AdaBoost is sensitive to noisy data and outliers. It doesn’t over fit. The individual learners are weak and as long as the performance of each of the tree is improved through some n iterations the weighted total result will improve.

Advantages of AdaBoost:

1. Weighted total result will consider only those outputs which come from genuine trees generating values that correspond to ground truth values.
2. This implementation will reduce the weights of the rows of data that are predicted correctly thus allowing the tree to learn quickly for the rows it has predicted wrongly for.
3. Not very prone to over fitting.
4. Suited for all classifier problems.

Disadvantages of AdaBoost:

1. They are sensitive to noisy data and outliers.

Implementation:

54. Read the file into *recordsFromFile* make note of the categorical fields there are in the dataset in to a list *listCatIndex*.
55. Pass the *recordsFromFile* to the method *crossValidData()*
56. *crossValidData()* returns two lists with all the folds generated training data *trainRecordsList()* and testing records *testRecordsList()*.
57. Initialize lists to calculate the accuracy, precision, recall, f_measure for the K folds to be printed at the end.
58. Run for loop for the number folds generated.
59. Calculate 63% and 37% of the training rows being considered in a single iteration of K fold. Save the same to *uniqueNoOfRec* and *duplicateNoOfRec* based on the probability distribution of the rows being sampled *weightOfTrainingRecords*.
60. Generate the lists of rows from the initial training set into *npUniqueTrainRec* and *npDuplicateTrainRec*, a union total of both the set of row indexes in *npTempTrainRecords*.
61. Populate the *npTrainRecords* with the rows from *npTempTrainRecords*.
62. Call *runTreeBuild* in recursion with initialization values for a constructor.
63. *runTreeBuild()* will run in recursion until the entire tree is built.
64. *runTreeBuild()* will run by calculating the measurement of the given attributes. We calculate the information gain values for the rows.
65. For Information gain we count the number of zero's and one's both greater than and lesser than the given value.
66. Based on this formula we can evaluate the entropy of an attribute from which we find the gain.
67. When maximizing this gain we will know the point of best split for the given data attributes.

68. First calculate the purity of the dataset being passed to the function using *calculatePurity()*. Note the same into *impurityStatus*.
69. If: the dataset is pure we declare a node instance and initialize the same with default values for a leaf node and declare a classification for the output using *classifyNode()* which will output the class variable that is present in the dataset.
70. Else: In the case of an impure subset we need to now run execution for nominal attributes and numerical attributes separately.
71. Now if the chosen best attribute is of nominal type we need to separate the dataset into two parts with values equal to the splitting value in the left and the unequal values in the right.
72. Call the function in recursion with the left and right subsets separately accepting left and right child nodes for the node at this iteration.
73. If there is no good split or if every attribute is appearing in the left we make the current node a leaf node.
74. If the attribute is a numerical attribute then we move the values less than the target value to the left and those with higher target value than that to the right.
75. If we have any records in the left we continue recursion with the records in the left and right.
76. Once we generate the tree based on the training records we can calculate the error of the prediction by testing the tree with training dataset itself and then we use that value to give the classifier an importance.
77. In the case of the classifier having less than 0.5 error, we use that tree; else we discard it.
78. Continue generating tree till we get the desired number of rows.
79. Run loop for all rows in *testRecordsList()* we send the line to the decision tree and bring out the decision we need and multiply the same with a list of classifier weights.
80. We then generate the class for each row and define the row to hold a class value as the one which produces the highest value when multiplied with the weights of the classifiers.
81. Now extract the ground truth for the rows and compare them to the predicted results.
82. Calculate the accuracy, precision, recall, F-measure for the ith fold and mean once the loop is over.

Analysis:

project3_dataset1.txt	**project3_dataset2.txt**
Metrics for fold iteration: 1 Accuracy: 0.9473684210526315 Precision: 1.0 Recall: 0.8695652173913043 F Measure: 0.9302325581395349	Metrics for fold iteration: 1 Accuracy: 0.5957446808510638 Precision: 0.55 Recall: 0.5238095238095238 F Measure: 0.5365853658536586
Metrics for fold iteration: 2	Metrics for fold iteration: 2

Accuracy: 0.9473684210526315 Precision: 0.8888888888888888 Recall: 0.9411764705882353 F Measure: 0.9142857142857143	Accuracy: 0.7446808510638298 Precision: 0.5294117647058824 Recall: 0.6923076923076923 F Measure: 0.6
Metrics for fold iteration: 3 Accuracy: 0.9824561403508771 Precision: 1.0 Recall: 0.9285714285714286 F Measure: 0.962962962962963 9	Metrics for fold iteration: 3 Accuracy: 0.717391304347826 Precision: 0.75 Recall: 0.47368421052631576 F Measure: 0.5806451612903226
Metrics for fold iteration: 4 Accuracy: 0.9649122807017544 Precision: 0.9523809523809523 Recall: 0.9523809523809523 F Measure: 0.9523809523809523	Metrics for fold iteration: 4 Accuracy: 0.6521739130434783 Precision: 0.46153846153846156 Recall: 0.4 F Measure: 0.42857142857142855
Metrics for fold iteration: 5 Accuracy: 0.9298245614035088 Precision: 0.9 Recall: 0.9 F Measure: 0.9	Metrics for fold iteration: 5 Accuracy: 0.6521739130434783 Precision: 0.8333333333333334 Recall: 0.25 F Measure: 0.38461538461538464
Metrics for fold iteration: 6 Accuracy: 0.9473684210526315 Precision: 0.9615384615384616 Recall: 0.9259259259259259 F Measure: 0.9433962264150944	Metrics for fold iteration: 6 Accuracy: 0.6304347826086957 Precision: 0.5294117647058824 Recall: 0.5 F Measure: 0.5142857142857142
Metrics for fold iteration: 7 Accuracy: 0.9824561403508771 Precision: 0.9565217391304348 Recall: 1.0 F Measure: 0.9777777777777777	Metrics for fold iteration: 7 Accuracy: 0.717391304347826 Precision: 0.5 Recall: 0.46153846153846156 F Measure: 0.48
Metrics for fold iteration: 8 Accuracy: 0.9649122807017544 Precision: 0.9333333333333333 Recall: 0.9333333333333333 F Measure: 0.9333333333333333	Metrics for fold iteration: 8 Accuracy: 0.8260869565217391 Precision: 0.7142857142857143 Recall: 0.45454545454545453 F Measure: 0.5555555555555556
Metrics for fold iteration: 9 Accuracy: 0.9473684210526315 Precision: 1.0 Recall: 0.9032258064516129 F Measure: 0.9491525423728814	Metrics for fold iteration: 9 Accuracy: 0.6086956521739131 Precision: 0.4166666666666667 Recall: 0.3125 F Measure: 0.35714285714285715
Metrics for fold iteration: 10 Accuracy: 0.9107142857142857 Precision: 0.9047619047619048 Recall: 0.8636363636363636 F Measure: 0.8837209302325582	Metrics for fold iteration: 10 Accuracy: 0.6739130434782609 Precision: 0.46153846153846156 Recall: 0.42857142857142855 F Measure: 0.4444444444444444
Average score for all 10 records:	**Average score for all 10 records:**

Accuracy is: 0.952474937343 **Precision is: 0.949742528003** **Recall is: 0.921781549828** **F-Measure is: 0.93472429979**	**Accuracy is: 0.681868640148** **Precision is: 0.574618616677** **Recall is: 0.44969567713** **F-Measure is: 0.488184591176**

The rows to sample are chosen based on a sampling randomness and the rows are weighted differently. This allows the true to push its accuracy based on the rows it doesn't learn well.

How do you handle continuous data?

The performance measure we use here is the information gain which we used to calculate the best point to split the data across all attributes. At this point we bring to the right node all values greater than and equal to the split value and all values less than the split value to the left.

How do you handle categorical data?

Nominal fields are first identified using pandas library and the attributes are categorically encoded. The information gain is calculated for all values of the nominal field and we move the equal values at the attribute of interest and the corresponding value of the attribute.

How to choice the best feature?

We choose the best feature by calculating the information gain across all attributes and all values of the dataset and this value is consistently maximized to generate the chosen best split.

What are the post processing methods?

We are stopping the execution of tree growth at the set where the classification is pure and this cannot be split. Cutting the tree or ending the growth earlier introduces some amount of randomness into the tree's classification thus generating wrong results.

Cross Validation implementation:

The cross validation implementation was done by generating the different K folds of data from the sampled data. Here we used nine parts for training and one part for testing because of the 10 - fold implementation. This execution is same as the KNN and Naïve Bayes implementations.

Result Analysis

Even though Naive Bayesian Classification trains better than K-NN in theory , the assumption of independent features in Naive Bayesian might affect the performance based on the dataset. Since we do not consider the prior knowledge of the data distribution and features , Naive Bayesian does not outperform K-NN in terms of performance for the two datasets given.

The decision trees being built on a block of data will not be the best to train the data. The sequence of rows does not allow the tree to build with any amount of variance or variety thus making the system weaker in terms of robustness. Introduction of the random forests allows all element of randomness in generating the data samples for the training set. This allows the trees generated as a part of the forests to be able to handle a wide variety of inputs making it more robust. As another alternative, we have adaptive boosting. This method will allow the system or the tree to build or train itself on the basis of its own wrong decisions this rectifying its own mistakes with increasing iterations of trees being initialized. Improvement on the decision tree such as random forests or adaptive boosting generates improvement in the accuracy percentages for both the project3_dataset1.txt and project3_dataset2.txt.