University at Buffalo
The State University of New York

Project 3: Classification

By:
Akshay Kumar (50169103, akumar34)
Priyanka Singh (50169994, psingh28)
Sahil Dureja (50168872, sahildur)

K Nearest Neighbor

KNN is a very simple to implement algorithm. To classify, we simply measure distance between the test data point and all the training data points (prior data points). We only consider the k nearest points and assign the class having majority votes within those points. For implementing this we need to decide some factors:

- **Value of k:** If value of k is too small it is sensitive to noise points, but if it is too large neighborhood may include points from other class. It depends a lot on the data and there is no fix rule. To select the optimal value of k we first take it as square root of number of

training data points and then tune it to get the best results. For *dataset1* value of k is 30 and for *dataset2* value of k is 25.

- **Distance function:** The data points to be included in the nearest neighbor depends on the distance function used to calculate the distance between data points. Various functions which can be used are Euclidean distance, Manhattan distance, Minkowski distance, etc. In the project we have used Euclidean distance function.

$$d(p,q) = \sqrt{\Sigma(p_i - q_i)^2}$$

- **Weighted vote:** To reduce the effect of noise data points we can take large value of k but to further reduce the effect of data points of other class label being included we can take weighted vote.

$$w = 1 \div d^2$$

- **Scaling data:** Every column in the dataset have different type of values. Some may vary from 0-10, some may vary from 50-1000. The effect of latter column will be more while calculating the distance which might lead to false neighbors. To correct this we can scale the data and make it all fall into a fixed range. In the project we took the range to be 0-100.

- **Handling categorical data:** For categorical data the distance is either 0 if same or 1 if different. To handle categorical data while inputting the data we assigned a number to each category so that it didn't cause any problem while scaling and then before calculating the distance we have a check to see if the values are same or different. Same values would not create problem while calculating distance but for different values we changed them to 0 (lower limit) and 100 (upper limit).

For each query the performance is O(kn).

We have also used 10-fold cross-validation. The data is divided into 10 parts, 9 parts are considered as training data and 1 part is considered as test data. This is done with 10 iterations so that each part get to be test data once. The results of 10 iterations are then averaged to get the final result.

Result of *dataset1*:

Using 10-fold cross validation and k = 30

final accuracy: 0.965769230769
final precision: 0.990277777778
final recall: 0.915970958498
final f-measure: 0.950419750268

Result of *dataset2*:

Using 10-fold cross-validation and k = 25

final accuracy: 0.716666666667
final precision: 0.66481962482
final recall: 0.382954259792
final f-measure: 0.475141263435

Advantages of KNN:

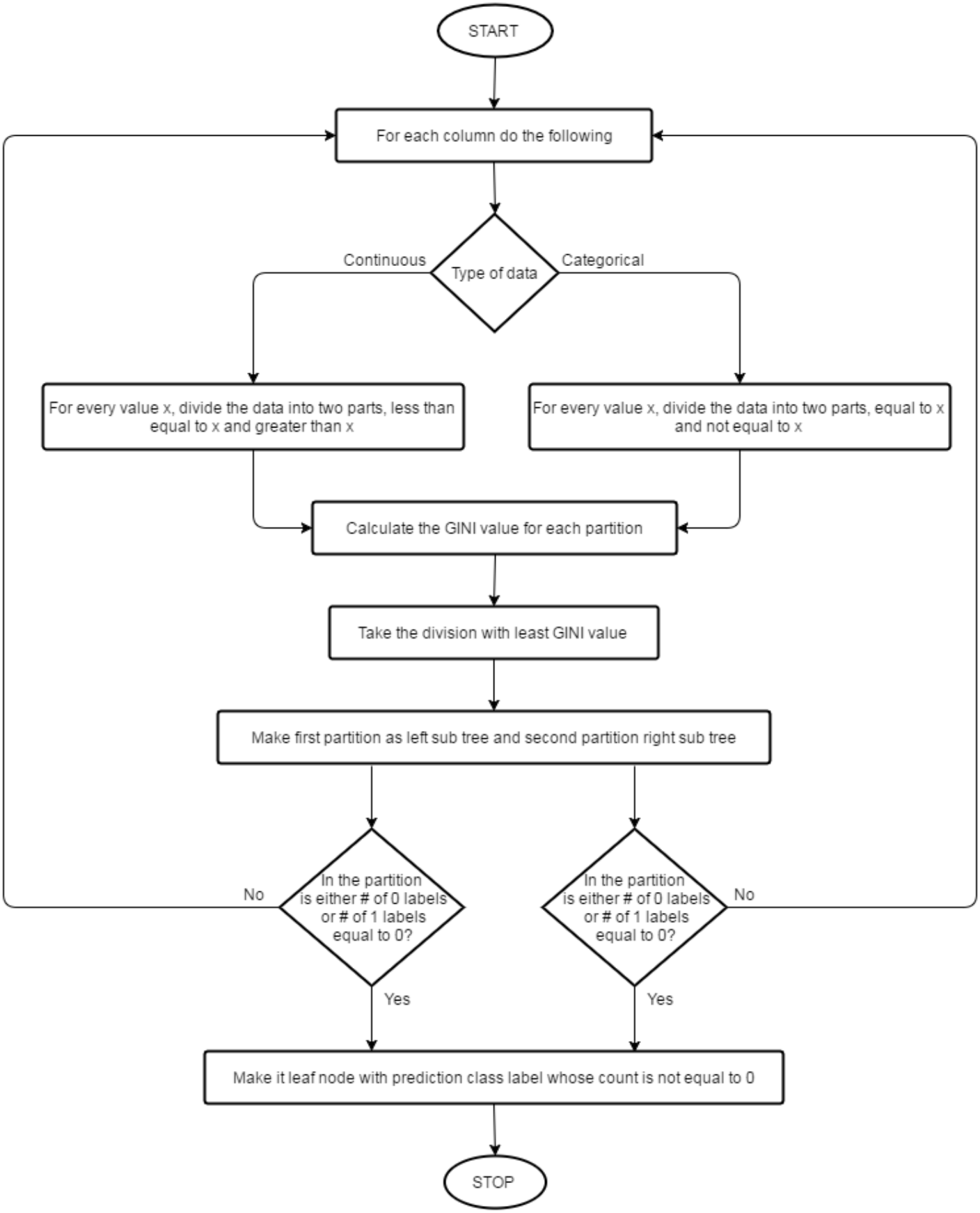
- Very simple to implement
- The cost of the learning process is zero.
- No assumptions about the characteristics of the concepts to learn have to be done.
- Robust to noisy training data especially if we use weighted votes.

Disadvantages of KNN:

- Computation cost is high as we need to compute the distance of each query instance to all training examples.
- Performance depends on the number of dimensions that we have.

Decision Tree

Decision tree builds classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.


START
For each column do the following
Type of data
Continuous
Categorical
For every value x, divide the data into two parts, less than equal to x and greater than x
For every value x, divide the data into two parts, equal to x and not equal to x
Calculate the GINI value for each partition
Take the division with least GINI value
Make first partition as left sub tree and second partition right sub tree
In the partition is either # of 0 labels or # of 1 labels equal to 0?
No
Yes
In the partition is either # of 0 labels or # of 1 labels equal to 0?
No
Yes
Make it leaf node with prediction class label whose count is not equal to 0
STOP

The performance of above algorithm is $O(n^2)$.

Above is the implementation of basic decision tree formation. 10-fold cross validation was also implemented around it. The data is divided into 10 parts, 9 parts are considered as training data and 1 part is considered as test data. This is done with 10 iterations so that each part get to be test data once. The results of 10 iterations are then averaged to get the final result.

Training a model with decision tree can lead to overfitting of data. This can be resolved by further using Random Forest or Boosting with Decision Tree. These are covered in next two sections.

Pruning experiments:
Pre- Pruning:
1) Prune-multiple : X means
If in a split a left tree or a right tree has true and false in the ratio more than X then no further decision tree is called on the branch instead it is made as a leaf node with prediction label as greater between # of true or # of false.
Example if a split is having
left tree [True : 30, False 50] **max-ratio is 50/30 = 1.66**
right tree [True : 15, False 85] **max-ratio is 75/15 = 5.66**
If X is 5 then as right tree X is > 5 this will pruned with label as False

2) Prune-minimum: Y means
If in a node in the recursion a decision tree algorithm is to applied to further split, but if the size of the data is below Y then mod/greater of the # of True or # of False if the final answer
Example if a split is having
left tree [True : 10, False 30] **size is 10+30=40**
right tree [True : 2, False 5] Y **size is 2+5=7**
If Y is 10 then as right tree size is 7 < 10 so we will assign only # of False> # of True = Label:False

Result of *dataset1*:

1) No Pruning

Average Accuracy: 92.3901098901
Average Precision: 0.875629463129
Average Recall: 0.918214933215
Average F-measure: 0.918214933215

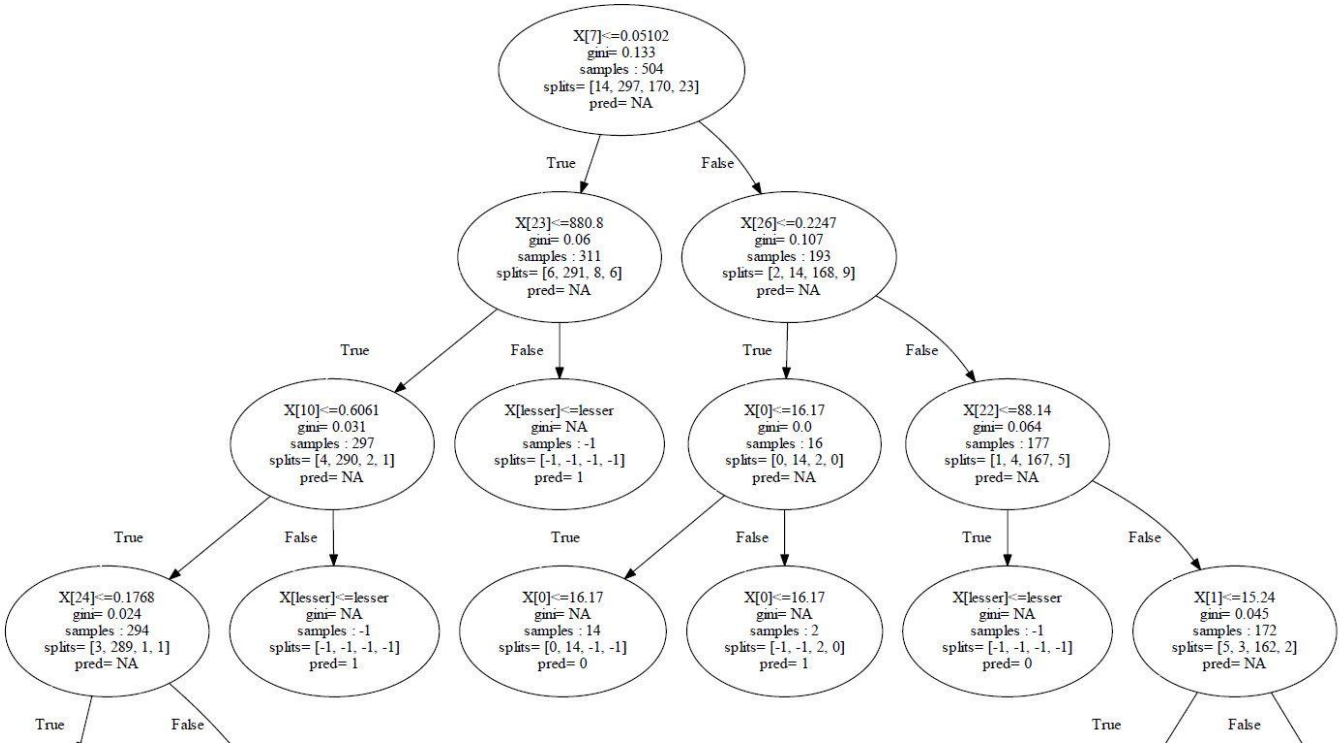
2) Using Prune-multiple 5, Prune-minimum 0
Due to only 1 child

Average Accuracy: 89.7362637363
Average Precision: 0.902223028033

Average Recall: 0.818849298849
Average F-measure: 0.818849298849

3) Using Prune-multiple 0, Prune-minimum 15

Average Accuracy: 92.5192307692
Average Precision: 0.898621930891
Average Recall: 0.907432197432
Average F-measure: 0.907432197432



Result of *dataset2*:

1) Using Prune-multiple 0, Prune minimum 0

Average Accuracy: 61.25
Average Precision: 0.444102579567
Average Recall: 0.481037931367
Average F-measure: 0.481037931367

2) Using Prune-multiple 6, Prune-minimum 0

Average Accuracy: 64.2934782609
Average Precision: 0.487244397759

Average Recall: 0.457778741726
Average F-measure: 0.457778741726

3) Using Prune multiple 0, Prune less 10

Average Accuracy: 62.7536231884
Average Precision: 0.458393532852
Average Recall: 0.439045289506
Average F-measure: 0.439045289506

4) Using Prune multiple 5, Prune less 10

Average Accuracy: 65.1630434783
Average Precision: 0.496432097314
Average Recall: 0.437923735621
Average F-measure: 0.437923735621

Improvements in Dataset 2

Average Accuracy

For No Pruning	**:**	61.25
Prune multiple 0, Prune less 10	**:**	62.7536231884
Prune-multiple 6, Prune-minimum 0	**:**	64.2934782609
Prune multiple 5, Prune less 10	**:**	65.1630434783

Advantages of Decision Tree:

- Extremely fast at classifying unknown records.
- Easy to interpret for small-sized trees.
- Able to handle both continuous and discrete attributes.
- Work well in the presence of redundant attributes.
- Robust to the effect of outliers.
- Provide a clear indication of which fields are most important for prediction.

Disadvantages of Decision Tree:

- Preparing decision trees, especially large ones with many branches, are complex and time-consuming affairs.
- Irrelevant attributes may affect badly the construction of a decision tree.
- Small variations in the data can imply that very different looking trees are generated.
- A sub-tree can be replicated several times.
- Error-prone with too many classes.
- Not good for predicting the value of a continuous class attribute.

Random Forest

Random Forest is a method to overfitting in a Decision Tree. Here instead of a single tree we generate several trees and then combine the result for prediction. Following are the things to consider while modeling trees:

- **Preparing training data:** There are various ways to generate training data randomly. In the project we have used Bootstrap sampling. Let's say the data from which to sample has N rows. While sampling we take N samples randomly but with replacement. It has been proven that this method of sampling generates 63.2% of unique sample. This process is repeated for every tree generation

- **Selecting the columns:** The reason why random forest helps in avoiding overfitting of data is because we don't use all the columns. Here we randomly select 20% of the columns at every node.

- **Combining the result:** There are many ways to combine the result of several trees to generate one prediction class. In the project we are using mode of the result set of all the trees. This method provides flexibility if we have more than two classes.

Result of *dataset1*:

1) Using # of Trees = 5, # of columns = 5 and 10-fold cross validation

Average Accuracy: 95.07
Average Precision: 0.92
Average Recall: 0.95
Average F-measure: 0.95

Result of *dataset2*:

1) Using # of Trees = 9, # of columns = 2 and 10-fold cross validation

Average Accuracy: 66.27
Average Precision: 0.52
Average Recall: 0.45
Average F-measure: 0.45

2) Using # of Trees = 5, # of columns = 2 and 10-fold cross validation

Average Accuracy: 64.54
Average Precision: 0.48
Average Recall: 0.47
Average F-measure: 0.47

3) Using # of Trees = 5, # of columns = 3 and 10-fold cross validation

Average Accuracy: 65.33
Average Precision: 0.51
Average Recall: 0.48
Average F-measure: 0.48

Advantages of random forest:

- It is fully parallelizable to go faster.
- Resistance to over training.
- Resistant to outliers.
- It runs efficiently on large databases.
- It gives estimates of what variables are important in the classification.

Disadvantages of random forest:

- It is slower than other methods like boosting.

Boosting

Boosting creates ensemble of weak learned decision trees given weights to each tree for the final prediction which helps in lowering down the overfitting.

Our approach:

1) Initially assign equal normalized weights to training set

2) Preparing training data:
    There are various ways to generate training data randomly. In the project we have used Bootstrap sampling. Let's say the data from which to sample has N rows. While sampling we take N samples randomly but with replacement. It has been proven that this method of sampling generates 63.2% of unique sample.
3) Create weak learned Decision tree: Apply the decision tree algorithm upto 1 level only

4) Now calculate the error as
    error=Sum of weights of misclassified on training data / sum of weights

5) Tree_weight=(ln(1-error/error))/2
    *Note: Tree_weight is different the weights of training data

6) Calculate new weights:
    weight[i]=weight[i]*e^Tree_weight*missclassification_flag[i]

For all the training data which is misclassified the weights will increase with respect to other correctly classified weights.

7) Normalize the weights

8) Create other trees using updated weights

9) Combining the result: For all the prediction done on testing data by weak learners combine the result using Tree_weight as weight for each learned tree.

10) After step 4 when we calculate error:

If the error goes above 0.5 or 50% we break and go to step one as the tree in that step is not counted because a random tree could have given error below 0.5 as there are only two labels

Result of *dataset1*:

Max week trees 10
Reset at error below 0.5

10 fold i'th loop result
Error changes in one of the iteration of 10 fold run
[0.081, 0.115, 0.183, 0.264, 0.253, 0.222, 0.284, 0.297, 0.302, 0.278]

Tree weight changes in one of the iteration of 10 fold run
[1.212, 1.0213, 0.748, 0.512, 0.541, 0.626, 0.463, 0.430, 0.418, 0.477]

Average Accuracy: 95.23
Average Precision: 0.95
Average Recall: 0.91
Average F-measure: 0.91

Result of *dataset2*:

Max weak trees 10

10 fold i'th loop result
Error changes in one of the iteration of 10 fold run
[0.338, 0.341, 0.410, 0.407, 0.394, 0.412, 0.363, 0.411, 0.390, 0.435]

Tree weight changes in one of the iteration of 10 fold run
[0.336, 0.329, 0.182, 0.188, 0.215, 0.178, 0.282, 0.181, 0.224, 0.131]

Average Accuracy: 72.55
Average Precision: 0.64
Average Recall: 0.45
Average F-measure: 0.45

Advantages of Boosting:

- It is very easy to program and adapt for different types of weak classifiers.
- Provide more flexibility by allowing user to stop at a reasonable resolution and thus avoid overfitting.
- Avoid use of strong learners in the beginning stage of modeling and progressively use them in the end.
- Require very few predefined parameters.

Disadvantages of Boosting:

- Boosting can be sensitive to noisy data and outliers.
- Time and computation expensive.
- Hard to implement in real time platform.
- Complexity of the classification increases.

Naive Bayes Classification

Naive bayes is a probabilistic classifier which uses the Bayes Theorem. Naive bayes assumes that the features of data are independent of each other. It is a very efficient algorithm which works well with large amounts of data.

Bayes Theorem:

$$P(H_i \mid X) = (P(H_i) * P(X \mid H_i)) \div P(X)$$

Implementation:

P(Hi), the class prior probability is calculated b - ni/n,
where ni is the number of training data points belonging to class i.
n is the total number of training data points.

P(X|Hi) the descriptor posterior probability is calculated by- probability of X in data belonging to Hi. X is a vector of independent(assumption of naive bayes) features. P(X|Hi) = P(x1|Hi)*P(x2|Hi)*...*P(xn|Hi)

P(X) is the descriptor prior probability which is calculated by- probability of X in al training data.

Features:
Naive bayes is usually used for discrete/categorical features. It can also be used on continuous data.

For discrete data- We have implemented the data considering it as strings. All comparisons are string comparisons.

For continuous data- There are 2 ways to deal with continuous data for naive bayes.

1. **Discretize the data**. The continuous data is preprocessed to make it discrete.
    a. For dataset 1 and dataset 2, the precision of the continuous features are reduced. For eg. a value of 1.3 is converted to 1.The amount of precision reduction is done independently for each feature. This reduces the continuous data to fewer categories.
    b. String comparisons are made between the test and preprocessed train feature values.
2. **Use a pdf-** A probability distribution function can be used such as Gaussian distribution.This ensures the data is not lost, and the continuity of the data is preserved.

Gaussian distribution is made for each class and each of its features. Each feature of the test sample is then compared to the distributions(mean and standard deviations) and the probability is computed.

Both the approaches above were tried and very similar performance metrics were observed. By following Occam Razor's principle, the simpler of the two approaches was used, which is the discretization of data, as some of the features are already discrete.

Zero Probability Problem:
When calculating the descriptor probabilities, we take a product of probability of each of the features belonging to a particular class. If there appears a feature value in the test sample, which does not exist in the training data, we will get a 0 probability. For example:

$P(X|Hi) = P(x1|Hi)*P(x2|Hi)*...*P(xn|Hi)$
if $P(x2|Hi) = 0$, i.e. value of x2 does not exist in the training data
$P(X|H1) = P(x1|Hi) * 0 * ...*P(xn|Hi)$
$P(X|H1) = 0$

To solve for this problem, we make **Laplacian Correction.**
In Laplacian correction, we add 1 to the numerator of all probability calculations of all features and add one for each of the features to the denominator. This in effect means that we introduce single feature data points in the training data with the value of test sample.

Cross Validation:
To calculate the performance metrics with more robustness, k-fold cross validation technique is used.
The dataset is randomly divided into k equal (or nearly equal) parts. One part is used as test data, and the rest k-1 are used as training data.This is performed for each part, i.e. each part is used as test data once.
For k = 10,
Data is randomly split into 10 equal parts. Each split is used once as test data and the rest of 9 sets are used as training data.
Performance metrics of accuracy, precision, recall and f-measure are calculated for each of the 10 iterations. They are then averaged.

Result and Analysis:

For final result, the pre-processing is- all continuous features are discretized by precision reduction.

<u>Dataset 1:</u>

Pre-processed:
Avg Accuracy: 0.93

Avg Precision: 0.90
Avg Recall: 0.91
Avg F-Measure: 0.90

Original:
Avg Accuracy: 0.39
Avg Precision: 0.38
Avg Recall: 1.0
Avg F-Measure: 0.55

Dataset 2:

Pre-processed:
Avg Accuracy: 0.64
Avg Precision: 0.49
Avg Recall: 0.60
Avg F-Measure: 0.54

Original:
Avg Accuracy: 0.53
Avg Precision: 0.41
Avg Recall: 0.90
Avg F-Measure: 0.57

Original data is largely continuous for dataset 2 and fully continuous for dataset 1. The result is in coherence with this fact, as we can see the quality metrics are extremely low for original dataset 1.
On discretization of dataset1, we see significant improvement of the performance measures. Final Dataset 1 result shows that the class labels are highly and directly influenced by the features.

Advantages and Disadvantages:

Naive bayes is a very simple algorithm to implement, which is known to give quality results for data with independent features.

It is a very efficient algorithm, runs fast for big data and is also scalable.
Classification of one data point takes O(nm),
where n = number of data points in training set
m = number of features

It also gives good results for data with partially dependent features, but not the best results.

Its biggest flaw lies in its assumption of independent features. The meaning of the interaction of features is lost in naive bayes.
This makes naive bayes only effective for simple data and perform poorly for complex data with interdependent features.

It takes a frequentist approach to probability calculation, but can also be made to work on distribution functions such as normal distribution.

Because of its frequency based probability calculation and assumption of independent variables, posterior probability calculation involves a product of probabilities of all features. This makes naive bayes prone to zero probability problem. i.e if a test data sample lies outside the data used in training, it will give a zero probability.

Various methods like Lagrange and laplacian correction is used to solve this issue.