PROJECT 3: Classification Algorithms

Tariq Siddiqui – 50208470

Kaushik Ramasubramanian – 50207352

Junaid Shaikh - 50208476

1 Classification

In classification, the idea is to predict the target class by analysis the training dataset. This could be done by finding proper boundaries for each target class. In a general way of saying, Use the training dataset to get better boundary conditions which could be used to determine each target class. Once the boundary conditions determined, the next task is to predict the target class as we have said earlier. The whole process is known as classification.

A test set is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Model then learns to make predictions based on past observations:

– Training data: Labels are available

– Test data: Labels are unavailable and need to be predicted based on learning from training data

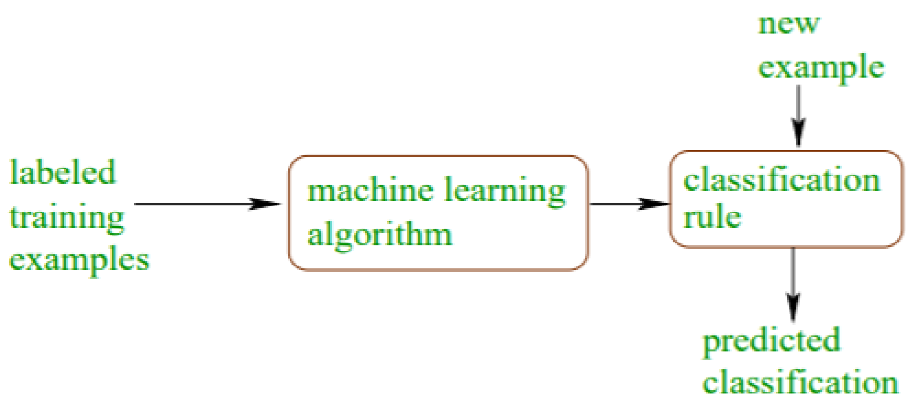


Figure 1.1-Overview Of Classification Algorithm

1.1 Examples of Classification Task

Classification learning model is used for number of predictive tasks as below:

1. Predicting tumor cells as benign or malignant
2. Classifying credit card transactions as legitimate or fraudulent
3. Classifying emails as spams or normal emails
4. Categorizing news stories as finance, weather, entertainment, sports, etc

1.2 k-fold cross validation

k-fold cross validation involves randomly dividing the training set into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k−1 folds. The misclassification rate is then computed on the observations in the held-out fold. This procedure is repeated k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error which are then averaged out.

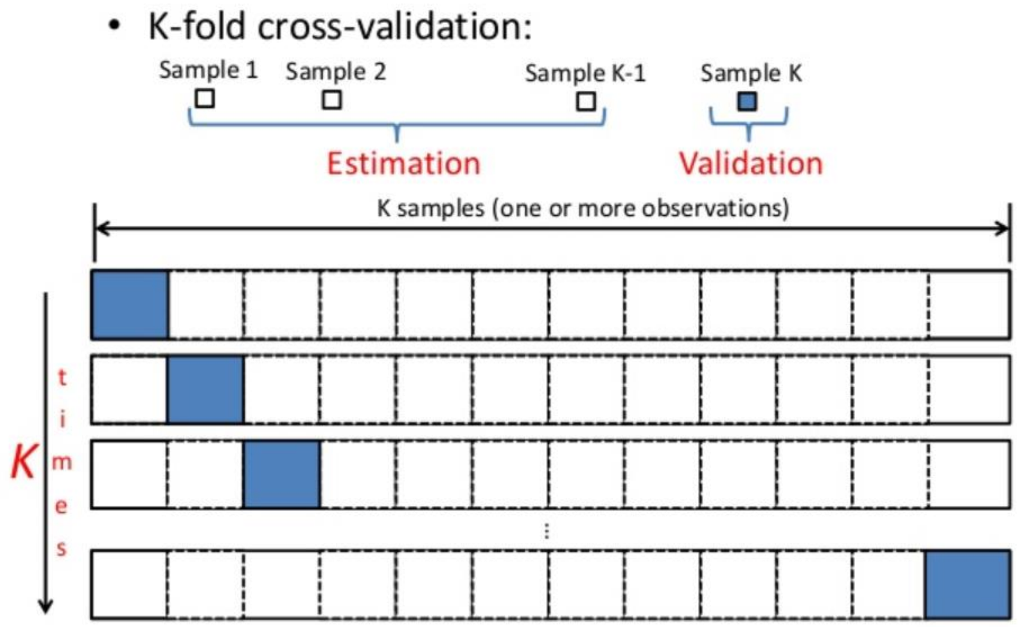
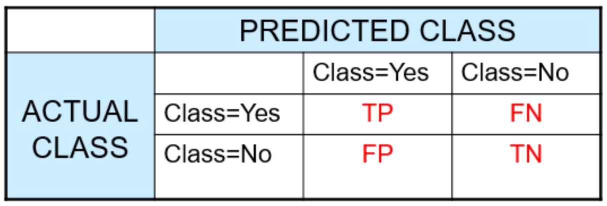


Figure 1.2-1 k-Fold Cross Validation

2 Performance Matrix



		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	TP	FN
	Class=No	FP	TN

Figure 1.2-1Confusion Matrix

True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.

True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.

False positives and false negatives, these values occur when your actual class contradicts with the predicted class.
False Positives (FP) – When actual class is no and predicted class is yes.

False Negatives (FN) – When actual class is yes but predicted class in no.

Accuracy - Accuracy is ratio of correctly predicted observation to the total observations. **accuracy is a great measure but only when you have symmetric datasets** where values of false positive and false negatives are almost same.

$$\text{Accuracy} = \frac{TruePositive + TrueNegative}{TruePositive + TrueNegative + FalseNegative + FalsePositive}$$

Precision - Precision is the ratio of **correctly predicted positive observations to the total predicted positive observations**. High precision relates to the low false positive rate.

$$\text{Precision} = \frac{TruePositive}{TruePositive + FalsePositive}$$

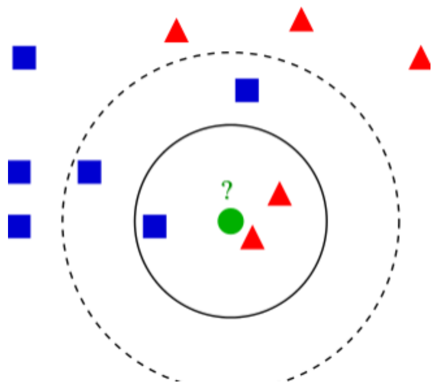
Recall - Recall is the ratio of **correctly predicted positive observations to the all observations in actual class – yes**

$$\text{Recall} = \frac{TruePositive}{TruePositive + FalseNegative}$$

F-Measure: F-Measure Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.

$$\text{F - Measure} = \frac{2 * TruePositive}{2 * TruePositive + FalseNegative + FalsePositive}$$

3 K-Nearest Neighbor Classification



k-nearest neighbor's algorithm (k-nn) is a non-parametric method used for classification and regression. However, it is extensively used to perform classification as compared to regression.

By non-parametric, it means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as in the real world, most of the practical data does not obey the typical theoretical assumptions made (eg Gaussian mixtures, linearly separable etc.).

It is also a lazy algorithm. What this means is that it does not use the training data points to do any *generalization*. In other words, there is *no explicit training phase* or it is very minimal. This means the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase(in the best case a subset of them). This is in contrast to other techniques like SVM where you can discard all non-support vectors without any problem. Most of the lazy algorithms – especially KNN – makes decision based on the entire training data set

3.1 Algorithm

A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

3.2 Preprocessing: Normalizing Attributes

One major drawback in calculating distance measures directly from the training set is in the case **where variables have different measurement scales** or there is a mixture of numerical and categorical variables. For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated.

Above causes certain attributes to dominate in their contribution to the distance measure. For these types of problems, you will want to rescale all data attributes into the **range 0-1 (called normalization)** before calculating similarity. Update the model to support data normalization.

3.3 Implementation Detail

Below is the approach which was used to implement the k-nn classification algorithm:

- The entire dataset was **split into training and testing records using 10-fold cross** validation i.e. 9 parts of the dataset were used as the training data and 1 part was the testing data.
- The above was performed for 10 iterations by changing the set of testing data and training data in every iteration.
- We check for any categorical and continuous attributes in the dataset.

- Then the normalization of training dataset is performed. In order to do this, **the mean and variance** are computed for the columns that have **continuous attributes and normalization** is done based on that for these attributes.
- In **case there are any categorical attributes we assign these with numbers in a particular range depending on the number of distinct attributes in a particular column**. Then the **mean** and **variance** is computed for the respective columns of containing these attributes and further **normalization** is done based on that for these attributes.
- Once we get the mean and variance for all the columns (except the last column containing labels) in the training data we use the same to **normalize all the columns in Training Data**.
- Then, we pick a testing record and Normalize it using Mean and Standard deviation we calculated from Training Data
- Now we compute the **Euclidean distances** for every testing record with every other training record, and **sort** them to get the k-nearest neighbors to a particular testing record.
- The **labels** are then **predicted** for each testing record based on the **majority** of the count of labels of different classes (0,1) from the k nearest neighbors
- The predicted labels of the testing data and the original labels are compared. The number of matching labels is divided by the total number of records to get the accuracy of the k-nn classifier.
- Finally, the average of all the accuracies obtained for 10-fold iterations is computed to get the average accuracy of the k-nn classifier.

3.4 Advantages

- The classifier is not much affected by noisy training data.
- It is very effective and useful when the classification is to be performed on a large training dataset.

3.5 Disadvantages

- The parameter **k has to be known** in order to use this classifier.
- There is ambiguity when it comes to choosing the right distance based learning to be used. When the **dataset has categorical attributes** the accuracy of the classifier reduces as the Euclidean distance measure is not an efficient technique.
- The **time complexity of the algorithm is relatively higher** as the distance of every test record has to be computed with all the training samples.

3.6 Results

Below table shows the results obtained for the ***project3_dataset1.txt*** and ***project3_dataset2.txt*** files for**10-fold cross validation.**

project3_dataset1.txt				
Value of K	***Average Accuracy***	***Average Precision***	***Average Recall***	***Average F-measure***
3	96.84%	99.06%	92.02%	95.28%
4	97.19%	97.02%	94.92%	95.86%
5	97.01%	97.95%	93.27%	95.47%
10	97.01%	97.64%	93.70%	95.46%
15	96.31%	98.65%	91.22%	94.63%
20	96.13%	99.11%	90.38%	94.22%
40	95.61%	99.06%	88.92%	93.56%
50	95.43%	99.06%	88.45%	93.30%
70	94.90%	99.03%	86.86%	92.37%
80	94.20%	99.01%	85.20%	91.32%
90	94.02%	99.54%	84.25%	91.14%
100	93.32%	98.99%	83.05%	90.16%

Figure 3.6-1 knn performance for 10-fold cross validation and project3_dataset1.txt

project3_dataset2.txt				
Value of K	***Average Accuracy***	***Average Precision***	***Average Recall***	***Average F-measure***
3	66.25%	53.11%	38.10%	43.33%
4	63.46%	46.98%	51.40%	48.70%
5	64.97%	49.02%	36.29%	41.26%
10	65.60%	49.31%	40.11%	43.68%
15	68.85%	58.57%	33.03%	41.12%
20	69.93%	58.84%	40.32%	47.04%
40	71.89%	67.28%	36.26%	46.68%
50	72.30%	70.76%	34.14%	45.43%
70	71.23%	69.66%	30.00%	41.13%
80	70.79%	74.39%	24.56%	35.86%
90	72.09%	82.14%	24.15%	36.61%
100	71.23%	81.30%	21.91%	33.74%

Figure 3.6-2 knn performance for 10-fold cross validation and project3_dataset1.txt

3.7 Estimating value of 'K'

Choosing the optimal value for k is best done by **first inspecting the data**. In general, a **large k value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation** is another way to retrospectively determine a good K value by using an independent dataset to validate the K value. Historically, **the optimal k for most datasets has been between 3-10**. That produces much better results than 1NN.

As we can see from the above results, the larger the k value the more smoothing takes place and eventually we get a model that is under-fitted rather than over-fitting. For **K=4 we get the highest average accuracy** and therefore it is the most optimal K for the k-NN model.

If the K value is too low i.e 1 or 2, then it leads to over-fitting. **<u>Thus, as we increase the K value the error rate reduces until it reaches a point after which it again increases.</u>** Thus, the k value for which this minima point of the error rate is achieved has to be taken as the optimal value.

3.8 Converting Categorical to Numerical Value:

In case there are any categorical attributes, we replace those values with a range of numerical values (example: 'absent' is replace with 1, 'present' is replace with 2 and so on).Then the mean and variance is computed for the respective columns of containing these attributes and further normalization is done based on that for these attributes.

3.9 Calculating Distance

Euclidean distance measure is the best method for distance calculation when the dataset contains only **continuous attributes**.

In case of dataset containing **categorical attributes, hamming distance** calculation is more preferable as it computes the binary vectors.

However, in this project we have used Euclidean distance for both types of data by converting Categorical data into numerical data.

3.10 Observations

Below were the observations made based on the results obtained above:

<u>OBSERVATION 1:</u> We observed from the results mentioned above that the error rate for the dataset containing continuous attributes is marginal as we use Euclidean distance as the measure. However, the same isn't the most appropriate way to measure the distances for categorical attributes. That leads to a higher error rate, **even though we Normalized the Categorical column.**

<u>OBSERVATION 2:</u> Since F-measure gives us an indication of how important are both the classes in binary labels, the dataset1.txt has higher F-measure values for all the K-values, thus indicating that the classifier gives equal importance to both the labels.

However, for dataset2.txt the F-measure has low values, **thus indicating a bias in giving more importance to one of the class labels.**

OBSERVATION 3: For dataset1.txt the average precision values are high thus indicating that there were lesser **instances classified as positive thus the lower recall.** On the other hand, for dataset2.txt the precision in very low.

4 Naïve Bayes Classifier

4.1 Algorithm:

- The Naïve Bayes Classifier is a probabilistic model for a classification algorithm with the underlying principle of assumption of independence between the various features.
- **Maximum Likelihood** (i.e. predicting the label of a testing data point based on the probability of a testing sample belonging to a class being higher than it belonging to any other class) is used to classify the testing data. The formula for calculating the posterior probability is as follows:

$$P(H_i|X) = \frac{P(Hi)*P(X|Hi)}{P(X)}$$

- Since the classification of a testing sample requires comparison of the different class posterior probabilities, the **descriptor prior probability P(X) can be ignored** in the equation since it is going to remain the same for different classes.
- The calculation of the descriptor posterior probabilities e.g.: $P(X|H_0)$ and $P(X|H_1)$, which is required to calculate the class posterior probability of a given testing record ($P(H_0|X)$ and $P(H_1|X)$) and predict its label, can be implemented in a number of ways like simple probability calculations, Gaussian pdf, Multinomial and Bernoulli.

4.2 Continuous and Nominal Attributes:

- For nominal or categorical attributes, these probabilities can be calculated as the number of instances of training records with a particular class label having the same value for the attribute divided by the number of training samples belonging to the class.
- However, for continuous attributes in the dataset, one of the probability distribution functions is required to be used to calculate the descriptor posterior and the class prior probabilities. We have implemented the Gaussian distribution for this project due to its suitability for continuous attributes and its normal distribution.

4.3 Implementation Details:

Pre-processing:

1. To implement 10 Fold Cross Validation for the calculation of the efficiency scores of the algorithm, the given dataset is bifurcated into 10 sets, 9 of them are used as training and 1 as testing for each of the 10 runs of the algorithm.
2. The nominal and continuous features are distinguished from the dataset.

3. The class prior probabilities for the two classes of labels $P(H_0)$ and $P(H_1)$ are calculated from the training dataset. This is done by calculating: number of records belonging to a class/total records in training

4. The calculation of descriptor posterior probabilities $P(X|H_0)$ and $P(X|H_1)$ is different for nominal and continuous features

5. For the continuous features, the mean and standard deviation of the values for that feature are calculated from the training data set. They are calculated separately for the different classes in the training dataset. These will be required for the probability calculations of the testing dataset features using Gaussian distribution

Classification of testing data:

6. The testing samples are used one by one to predict the labels for them.

7. For the nominal features, the probability is calculated as the number of training records containing the same value as the testing sample attribute belonging to a particular class/no of training examples belonging to that class.

8. For the continuous attributes in the dataset, the probabilities are calculated using the Gaussian Distribution formula:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$$

Where μ is the mean for the particular attribute for a given class label and σ its standard deviation

9. These two probabilities for the two types of attributes are then multiplied together to give the complete descriptor posterior probability.

10. Once we have the descriptor posterior probability, we can calculate the class posterior probabilities for the two classes 0 and 1 based on the formula:

$$P(H_i|X) = \frac{P(Hi)*P(X|Hi)}{P(X)}$$

11. As was discussed earlier, since the objective here is to compare the probabilities of the two different classes, there isn't a need to calculate the predictor prior probability P(X), since it will be same for both the classes.

12. The label for the training dataset is then predicted based on which class posterior probability is higher.

Post-processing:

13. The predicted labels are then compared with the true labels for the testing dataset provided and parameters Accuracy, Precision, Recall and F-Measure are calculated.

14. These calculations are then averaged for 10 runs based on the 10 Fold cross validation parameter setting

4.4 Results

For Dataset 1, the average efficiency parameters were found to be as follows:

Average Accuracy	**Average Precision**	**Average Recall**	**Average F-Measure**
93.57	92.41	90.53	91.32

For Dataset 2, the average efficiency parameters were found to be as follows:

Average Accuracy	**Average Precision**	**Average Recall**	**Average F-Measure**

70.21	57.07	61.74	58.42

4.5 Analysis/Observations

The disparity in the accuracies for the two datasets can be explained by the **high number of attributes** in dataset 1 (31) as compared to dataset 2 (10)

Since Naïve Bayes classification involves calculation of probabilities of the features, the calculation for both **continuous and categorical data is similar in nature unlike K-nn** which involves calculation of distances between the features of the data points (for which the categorical data has to be normalized). Hence, there is no discrepancy in the importance of a categorical feature as compared to a continuous feature.
We tried to check if the categorical feature is causing any imbalance in the prediction of labels by removing the feature from the 2nd dataset. However, there was no significant change in the accuracy and other efficiency parameters. This suggests that all the features are contributing equally to the class determination process.

4.6 Advantages

- The Naïve Bayes classifier is **easy to implement** and is preferred for large data sets since it requires minimal pre-processing. There are no major training steps involved apart from the calculation of mean and variances of the continuous features of the dataset and creating a dictionary of the categorical features and their probabilities.
- Even though the classification includes a very generous assumption of the independence of various features, Naïve Bayes performs pretty well for many practical problems. It can be seen that for a decent sized dataset with a lot of features, **the accuracy and reliability is pretty high** (dataset 1).
- Naïve Bayes also **does not suffer from overfitting**, as it depends on an absolute measure of the probabilities to label the test samples and does not involve any parameters that have to be fine-tuned.

4.7 DisAdvantages

- Naïve Bayes is a very simple probabilistic model which assumes the independence of various attributes. In the real world, datasets usually do contain features which are correlated.
- Also, Naïve Bayes is a good classifier, since it calculates the probabilities for different classes and compares them to predict the label. However, the **probability calculation is not an accurate measure** of the actual probability of the data point belonging to a particular class.
- Naïve Bayes suffers from **Zero-Probability** issue, which is explained further in detail

4.8 Improvements

- It would be beneficial if a field expert can do feature selection and assign appropriate weights to the important features which could affect the probability calculation for label prediction to increase the accuracy of the algorithm. Different probability distributions can also be used based on the nature of the attributes, e.g. multinomial distribution for discrete parameters or Bernoulli distribution for binary attributes.

- In the real world, it would be desirable to preserve the relation between highly correlated attributes and factor this co-relation into the classification.
- For example, given an object which is known to be a fruit and red in color, the co-relation between fruit and red suggests the object is probably an apple, but Naïve Bayes would calculate the individual probabilities of fruits and red objects to label the object.
- Or in case it is required to utilize the independence assumption of Naïve Bayes, it would be desirable to remove one of the two highly correlated attributes so that **redundancy and correlation between the attributes is minimized.**

4.9 Zero-Probability Issue:

Naïve Bayes suffers from zero-probability, i.e. if for a categorical attribute, the testing sample contains a feature which is not present in the training data, its probability would come out to be 0 which would bring the entire descriptor posterior probability to 0 without regard for the probabilities of the other features.
e.g. if the values for weather attribute present in the training data are {sunny, hot, cool} and a new attribute Cold is introduced in the testing data set later on, its probability will be 0.

The problem can be solved easily by using a smoothing approach like **Laplace Correction**. It involved addition of a constant to the numerator and adding the same constant times the no. of distinct values for the attribute to the denominator of the probability calculation
e.g. for constant = 1, the probability calculation for Cold weather will be done as

(0+1)/(no of training samples of the class + 1*3)

(since there are 3 distinct values for weather in the training dataset)

5 Classification and Regression Trees (CART)

Classification and Regression Trees or CART refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.

The CART algorithm provides a foundation for important algorithms like bagged decision trees, random forest and boosted decision trees.

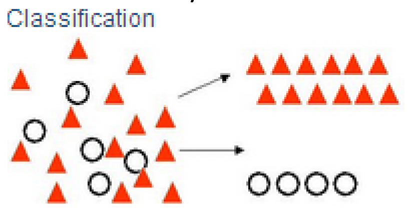
5.1 CART Model Representation

The representation for the CART model is a binary tree. Each root node represents a single input variable (x) and a split point on that variable (assuming the variable is numeric).

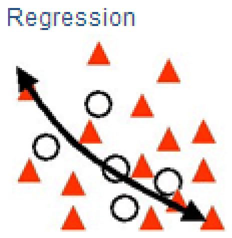
The **leaf nodes of the tree contain an output variable (y) which is used to make a prediction**.

Decision Trees are commonly used in data mining with the objective of creating a model that predicts the value of a target (or dependent variable) based on the values of several input (or independent variables).

- **Classification Trees**: where the target variable is categorical and the tree is used to identify the "class" within which a target variable would likely fall into.



- **Regression Trees**: where the target variable is continuous and tree is used to predict its value.



The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions.

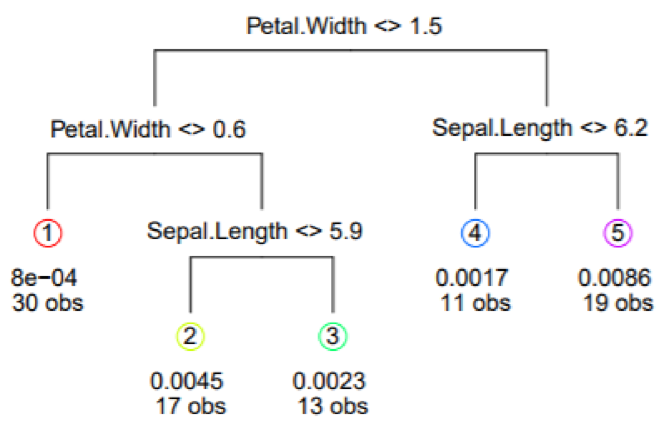


Figure 5.1-1Example tree from iris data

5.2 Learning CART Model

Creating a CART model involves selecting input variables and **split points** on those variables until a suitable tree is constructed.

The selection of which input variable to use and the specific **split or cut-point is chosen using a greedy algorithm to minimize a cost function**. Tree construction ends using a predefined stopping criterion, such as a minimum number of training instances assigned to each leaf node of the tree.

The main elements of CART (and any decision tree algorithm) are:

1. ***Rules for splitting data at a node based on the value of one variable;***
2. ***Stopping rules for deciding when a branch is terminal and can be split no more; and***
3. ***Finally, a prediction for the target variable in each terminal node.***

We discuss above elements below:

5.2.1 Greedy Splitting

Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.
This is a numerical procedure where all the values are lined up and **different split points are tried and tested using a cost function**. The split with the best cost (lowest cost because we minimize cost) is selected.
All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).

For classification, we have used the Gini index function which provides an indication of how “pure” the leaf nodes are (how mixed the training data assigned to each node is).

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

Where GINI is the Gini index over all classes, p(j | t) is the relative frequency of class j at node t. A node that has all classes of the same type (**perfect class purity**) will have GINI=0, where as a G that has a 50-50 split of classes for a binary classification problem (**worst purity**) will have a GINI=0.5.

For a binary classification (class 0 or class 1) problem, this can be re-written as:

$$G = 1 - (p1^2 + p2^2)$$

We want impurity function to look as shown in adjacent figure

5.2.2 Stopping Criterion

The recursive binary splitting procedure described above needs to know when to stop splitting as it works its way down the tree with the training data.

The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the count is less than some minimum, then the split is not accepted and the node is taken as a final leaf node.

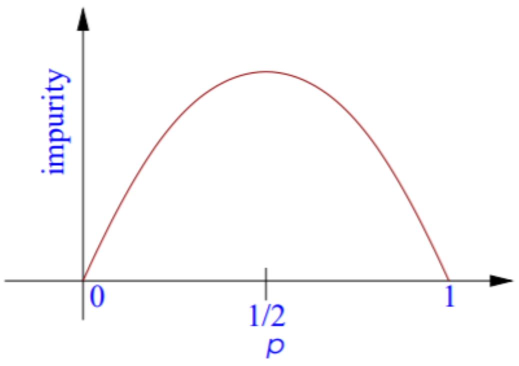
The count of training members is tuned to the dataset, e.g. 5 or 10. It defines how specific to the training data the tree will be. Too specific (e.g. a count of 1) and the tree will overfit the training data and likely have poor performance on the test set.

We however in **our implementation** have not put any stopping criterion and hence our implementation has **full-grown tree** implementation.

5.2.3 PostProcessing/Pruning The Tree

The stopping criterion is important as it strongly influences the performance of your tree. You can use pruning after learning your tree to further lift performance.

The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred. They are easy to understand (you can print them out and show them to subject matter experts), and they are less likely to overfit your data.



The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.

More sophisticated pruning methods can be used such as cost complexity pruning (also called weakest link pruning) where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

5.3 Advantage

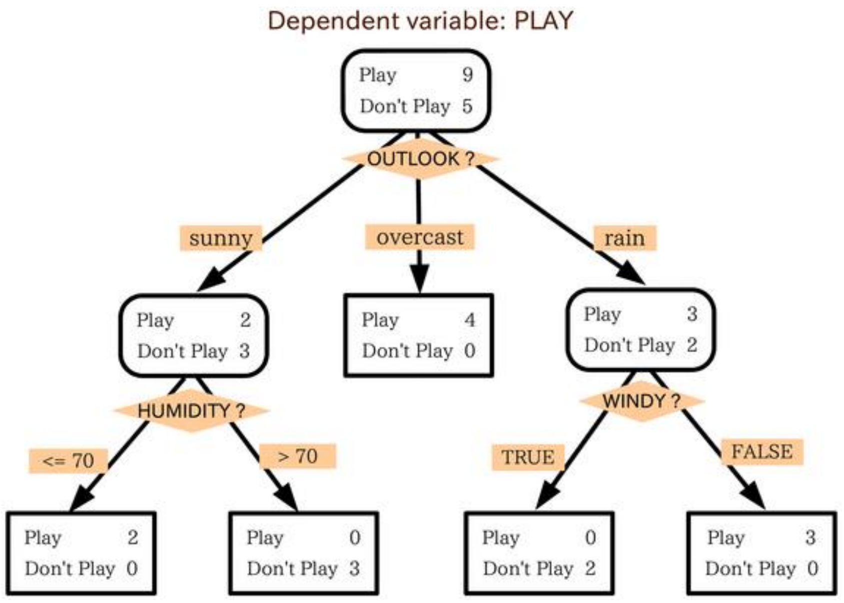
- CART based implementation is very fast to train and evaluate
- Its relatively easy to interpret
- but: accuracy often not state-of-the-art

6 Decision Trees

6.1 Predicting Labels

Given a new input, the tree is traversed by evaluating the specific input started at the root node of the tree.

New data is filtered through the tree and lands in one of the rectangles and the output value for that rectangle is the prediction made by the model. This gives you some feeling for the type of decisions **that a CART model is capable of making, e.g. boxy decision boundaries.**



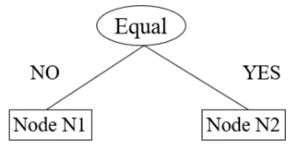
6.2 Advantages

- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Accuracy is comparable to other classification techniques for many simple data sets

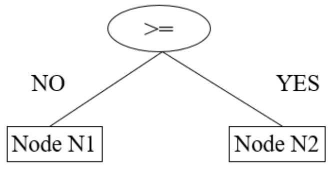
6.3 Implementation Detail

Below is the approach which was used to implement the k-nn classification algorithm:

- The entire dataset was **split into training and testing records using 10-fold cross** validation i.e. 9 parts of the dataset were used as the training data and 1 part was the testing data.
- The above was performed for 10 iterations by changing the set of testing data and training data in every iteration.
- Now, training record from above is used to construct decision tree.
- Implemented algorithm splits records based on all possible values of a Column (from Training Data) and this operation is performed for all the columns.
- Of all the combinations above, splitting is performed on the column and value inside column which **gives most reduction (maximizes GAIN). GINI is used here for impurity measurement.**
- We are essentially using binary Split here.
- Now, values of a column to split can be Categorical value or Continuous value.
- **If it's a Categorical value**, above steps are recursively called for left and right child of this splitted Node. NOTE: Right child contain all those records which have value **same as value** at Splitted Node and Left child contain all those records which value different than value at Splitted Node.



- **And, if its Continuous value**, above steps are recursively called for left and right child of this splitted Node. NOTE: Right child contain all those records which have value **Greater than** value at Splitted Node and Left child contain all those records which have value **Less than** value at Splitted Node.



- However, if at any stage all records belong to the same class, then t is **a leaf node labeled as having that particular class.**
- This record is recursively repeated for each subset until no more splitting is possible.
- Finally, the average of all the accuracies obtained for 10-fold iterations is computed to get the average accuracy of the Decision Tree.

6.4 Results

Below table shows the results obtained for the ***project3_dataset1.txt*** and ***project3_dataset2.txt*** files for **10-fold cross validation.**

Data	Average Accuracy	Average Precision	Average Recall	Average F-Measure
DataSet1	90.50	87.59	86.70	86.73
DataSet2	59.75	42.35	46.00	43.34

Table 6.4-1Decsison Tree: Obtained Result for DataSet1 and DateSet2

6.5 Post Processing

For pruning method, we can iterate througheach leaf node in the tree and evaluate the effect of removing it using a hold-out test set. **Leaf nodes can be removed only if it results in a drop in the overall cost function on the entire test set**. You stop removing nodes when no further improvements can be made.

7 Ensemble Learners

Ensemble is a Machine Learning concept in which the idea is to train **multiple models** using the same learning algorithm. The ensembles take part in a bigger group of methods, called **multiclassifiers,** where a set of hundreds or thousands of learners with a common objective are fused together to solve the problem

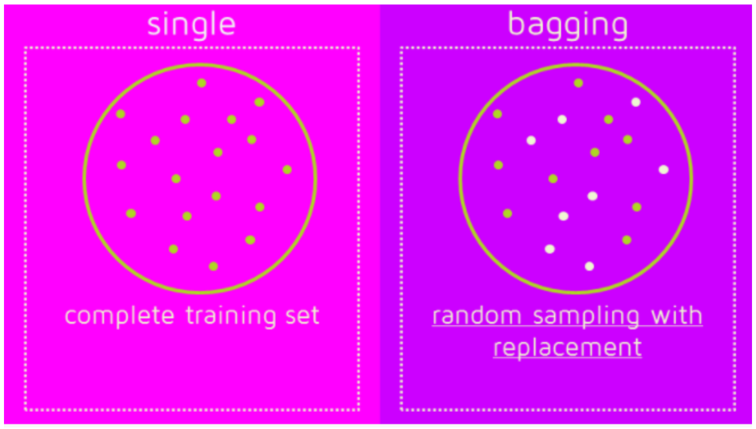
Given a data set $D=\{x_1,x_2,\ldots,x_n\}$ and their corresponding labels $L=\{l_1,l_2,\ldots,l_n\}$

An ensemble approach computes:

- A set of classifiers $\{f_1,f_2,\ldots,f_k\}$, each of which maps data to a class label: $f_j(x)=l$
- A combination of classifiers f^* which minimizes generalization error: $f^*(x)= w_1f_1(x)+ w_2f_2(x)+\ldots+ w_kf_k(x)$
- **Bootstrap**
  - Sampling with replacement
  - Contains around 63.2% original records in each sample
- **Bootstrap Aggregation**
  - Train a classifier on each bootstrap sample
  - Use majority voting to determine the class label of ensemble classifier

7.1 Bagging

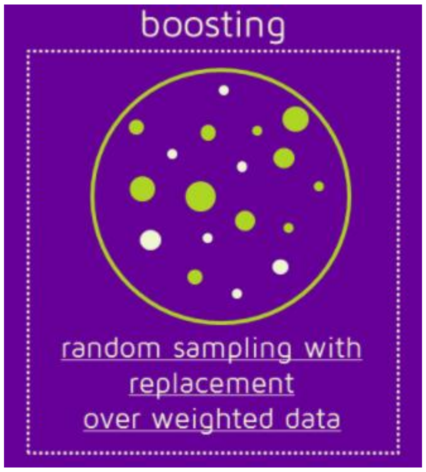
Bagging get N learners by generating additional data in the training stage. N new training data sets are produced by **random sampling with replacement** from the original set. By sampling with replacement some observations may be repeated in each new training data set.

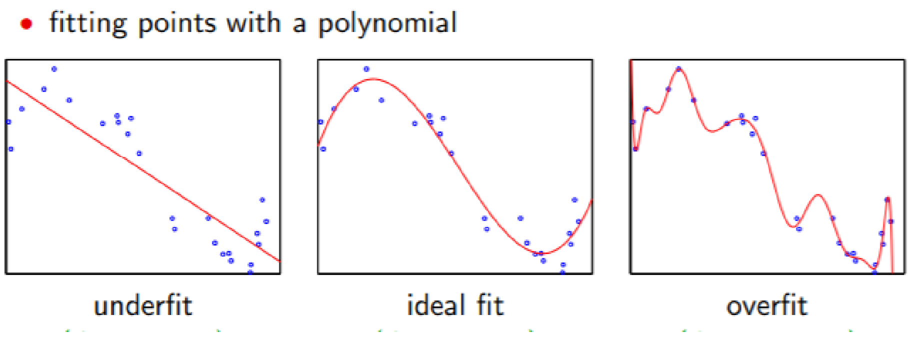


if the difficulty of the single model is **over-fitting**, then **Bagging** is the best option

7.2 Boosting

Boosting gets N learners by generating additional data in the training stage or Boosting the observations are weighted and therefore some of them will take part in the new sets more often.





In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success. After each training step, the weights are redistributed. M**isclassified data increases its weights** to emphasise the most difficult cases. In this way, subsequent learners will focus on them during their training.

Boosting reduces variance and provide higher stability. **Boosting tries to reduce bias**. On the other hand, Bagging may solve the over-fitting problem, **while Boosting can increase it.**

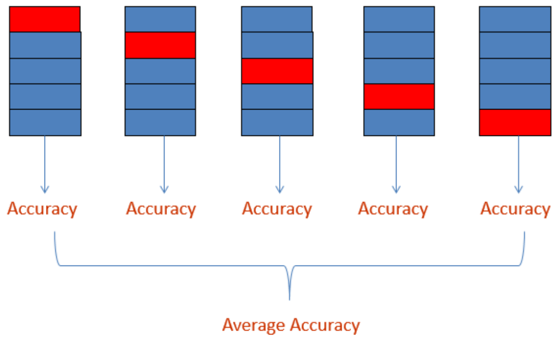
Bagging and Boosting decrease the variance of your single estimate as they combine several estimates from different models. So the result may be a model with **higher stability**.

8 Random Forest Trees

8.1 Random Forest Tree Algorithm

- The entire dataset was **split into training and testing records using 10-fold cross** validation i.e. 9 parts of the dataset were used as the training data and 1 part was the testing data.
- The above was performed for 10 iterations by changing the set of testing data and training data in every iteration.
- Choose *T*—number of trees to grow
- Choose *m<M* (M is the number of total features) —number of features used to calculate the best split at each node. We have selected 20% in our experiment.

- For each tree and using the algorithm discussed for Decision Tree above,
  - **Bagging:** Choose a training set by choosing *N* times (*N* is the number of training examples) with **replacement** from the training set.
  - For each node, **randomly choose *m* features** and calculate the best split as explained in decision Tree before
  - Tree is fully grown (not pruned)
- We use majority voting among all the trees to predict the Label for a record.
- Finally, the average of all the accuracies obtained for 10-fold iterations is computed to get the average accuracy of the Decision Tree.



8.2 Advantage

1. Random Forest Tree implements both Bagging and random features.
   a. Hence, Improve accuracy by Incorporate more diversity and reduce variances
2. Improve efficiency
   a. Searching among **subsets of features is much faster** than searching among the complete set
3. Training can be very efficient. Particularly true for very large datasets.
   a. No cross-validation based estimation of parameters for some parametric methods.
4. Natural multi-class probability.
5. Imposes very little about the structures of the model.

8.3 Results

Below table shows the results obtained for the ***project3_dataset1.txt*** and ***project3_dataset2.txt*** files for **10-fold cross validation.**

Size Of Forest	Average Accuracy	Average Precision	Average Recall	Average F-Measure
5	93.32	93.02	88.75	90.62
10	94.37	92.45	92.19	92.06
15	94.73	95.25	90.59	92.77

Table 8.3-10 Random Forest Tree: Obtained Result for different values of Trees in Random Forest for DateSet1

Size Of Forest	Average Accuracy	Average Precision	Average Recall	Average F-Measure
5	67.75	53.92	27.69	35.67
10	64.95	47.83	22.61	29.40
15	66.02	54	13.2	20.6

Table 8.3-21 Random Forest Tree: Obtained Result for different values of Trees in Random Forest for DateSet2

9 Boosting

Principle of Boosting is to make a set of weak learners to a strong learner.

Boosting algorithm train a classifier on each bootstrap sample and them makes records currently misclassified more important so that they are selected by other learner in next round.

Finally, weighted voting (based on importance of classifier) is used to determine the class label of the test data.

9.1 Boosting Algorithm

- The entire dataset was **split into training and testing records using 10-fold cross** validation i.e. 9 parts of the dataset were used as the training data and 1 part was the testing data.
- The above was performed for 10 iterations by changing the set of testing data and training data in every iteration.
- Choose *L*—number of Learners
- For each Tree/Learner and using the algorithm discussed for Decision Tree above,
    - Initially, set uniform weights on all the records
    - Repeat below rounds for **"number of Learners" times**.
        - At each round
            - Bagging: Create a bootstrap sample **based on the weights**
            - Train a classifier on the sample and apply it on the original training set
            - Calculate **Error based on predicted Labels** as below:

$$\varepsilon_i = \frac{\sum_{j=1}^{N} w_j \delta(C_i(x_j) \neq y_j)}{\sum_{j=1}^{N} w_j}$$

            - Calculate importance of Learner as below:

$$\alpha_i = \frac{1}{2} \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- Records that are wrongly classified will have their weights increased and records that are classified correctly will have their weights decreased using below formula

$$w_j^{(i+1)} = \frac{w_j^{(i)} \exp\left(-\alpha_i y_j C_i(x_j)\right)}{Z^{(i)}}$$

- If the error rate is higher than 50%, Cancel this iteration/Learner and Start over

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

$$C^*(x) = \sum_{i=1}^{K} \alpha_i \delta\left(C_i(x) = y\right) > 0.5$$

- Finally, the average of all the accuracies obtained for 10-fold iterations is computed to get the average accuracy of the Decision Tree.

9.2 Results

Below table shows the results obtained for the ***project3_dataset1.txt*** and ***project3_dataset2.txt*** files for **10-fold cross validation.**

# Learners	Average Accuracy	Average Precision	Average Recall	Average F-Measure
5	96.31	95.64	94.00	94.61
10	96.31	96.43	93.06	94.51
15	96.30	96.87	93.02	94.76

Table 9.2-10 Random Forest Tree: Obtained Result for different values of Trees in Boosting Forest for DateSet1

Size Of Forest	Average Accuracy	Average Precision	Average Recall	Average F-Measure
5	63.01	46.56	39.91	42.23
10	63.85	46.07	39.64	42.10
15	65.15	49.54	40.91	44.21

Table 9.2-21 Boosting Tree: Obtained Result for different values of Trees in Boosting Forest for DateSet2

10 References:

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

https://en.wikipedia.org/wiki/Decision_tree

https://en.wikipedia.org/wiki/Naive_Bayes_classifier