# **CSE 601 Data Mining and Bio Informatics**

# **Project 3: Classification Algorithms**

# TEAM 6

Submitted by:

Anurag Devulapalli (50208153)

Rimi Das (50206511)

Sindhura Uppu (50206730)

# Contents

1. K- Nearest Neighbors
   1. Algorithm flow
   2. Choice description
      1. K
      2. Distance calculation for Categorical and Continuous data
2. Decision Tree
   1. Dealing with categorical & continuous features
   2. How to choose best features
   3. Post-processing method used
3. Naïve Bayes
   1. Algorithm flow
   2. Dealing with continuous features
   3. Dealing with zero-probability issue
4. Random Forests
5. Boosting
6. Cross Validation
7. Result analysis
8. Pros and Cons of five classification algorithms

# **K- Nearest Neighbors**

1. Algorithm flow

For a given test data point

* 1. Calculate the distances to the available labeled data points.
  2. Sort the data points by the distance in increasing order.
  3. Choose the top k data points and find the most common label from them.
  4. Assign that label to the given test data point.

1. Choice description
   1. Choosing K

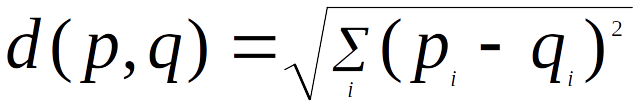
K is initially selected as the 'Square root of n' i.e., K = n^ (1/2), where n is the number of features.

* 1. Distance computation for categorical and continuous data

Categorical data is converted to numerical by label encoding.

Then, the features are normalized to avoid the dominance of any particular feature.

The distance between the data points is then computed based on the following Euclidean formula.



# **Decision Trees**

Decision Trees split the data into two or more sets based on the best split condition. They have nodes that map to an attribute in the data and split the data to whatever split condition the attribute’s value in a record maps to. Split criterion can be selected as per the decision tree algorithm which is followed. In our case, we have implemented C4.5 Decision Tree Algorithm in Java.

## Splitting Criterion:

In our implementation of the decision tree, we decided to use a univariate impurity-based splitting criterion called entropy.

Entropy is the measure of impurity or chaos in the data.  If all elements in a set of data belong to the same class, the entropy would be zero and if all the elements in a dataset were evenly mixed, the entropy would be one.  Entropy may be measured with the following equation

[http://blog.afewguyscoding.com/wp-content/uploads/2010/03/entropy-equation.png](http://blog.afewguyscoding.com/wp-content/uploads/2010/03/entropy-equation.png)

where http://blog.afewguyscoding.com/wp-content/uploads/2010/03/p-plus.pngis the number of positive training examples in T and http://blog.afewguyscoding.com/wp-content/uploads/2010/03/p-plus.pngis the number of negative training examples in T.

The C4.5 algorithm uses this measure of entropy of each attribute set of values to determine the best gain, or expected reduction in entropy due to splitting on A or the difference in entropies before splitting and after splitting on A.

[](http://blog.afewguyscoding.com/wp-content/uploads/2010/03/gain-equation.png)

At each level of the tree, the “best” attributes can be found to create the maximum reduction in entropy, called information gain.  These two calculations represent the preference to create the smallest tree possible for several different reasons – mainly that a short hypothesis that accurately describes the data is unlikely to be coincidence.

**Implementation**:

1. The dataset is loaded first from the path specified and each record is stored.
2. As part of **pre-processing**, while parsing through the dataset file, we store what kind of attribute values a record (or row) has and this also aides in identifying the missing values.
3. While creating the unique set of values for each attribute, we purposely leave out the missing values so that no entropy or gain is calculated of split on a missing/blank value.
4. We implemented 10-fold cross validation by taking the test data for each iteration as 1/10th of the total records and the rest of the data as training data.
5. For each fold:
   1. 9/10th (sometimes little more) of the overall records act as the training data, which is used to build a decision tree.
   2. All the features, except actual class value, are considered for the first node and the best split feature is obtained as follows.
   3. For continuous variables, we are creating binary split tree, based on greater than or less than conditions on the best split record value at that feature and for discrete variables, we are creating multi split tree.
   4. Supposing feature 1 is continuous, for 1st record’s value in the feature, we check how many records in that feature have lesser value than this and how many records have greater values than the first record’s value and store them in separate lists.
   5. These “left” and “right” subsets will be used to calculate entropies for that record’s value in that one particular feature.
   6. Similarly, we take each value of the feature and find the left and right splits based on the numeric comparison of data points and calculate the entropies and then their corresponding gain values.
   7. We select the value at which we obtained the minimum entropy and maximum gain and make that as the best split value for that feature/attribute and store it.
   8. If a feature is discrete, then we obtain the number of distinct values for that feature present in the training data and create a multi-split where in the records gets split into child nodes based on which value it equates to in the distinct values of the feature.
   9. For the rest of the continuous features/attributes, except the actual class attribute, we calculate and find out which value in each feature gives the best split and use that as the best split value for the corresponding feature.
   10. Once all the features have their best split values and their respective optimum split value with their gains, we identify the attribute which has the maximum gain and make that as the node and split the data based on the best split value for that attribute.
   11. If the entropy for any particular node is obtained as “0”, then it means that all the records under that split can be identified under a single class and hence, that node will be considered a leaf node. No further computation is made on leaf nodes and a class value is assigned to that node corresponding to the class to which all its records belong to.
   12. In case the entropy is not zero for a particular split value, then it would mean that the tree needs to be further constructed and for each child node, we perform the same operations starting from step 2 till the current step.
   13. Once the tree is constructed and has leaf nodes in every branch which are assigned a particular class value, then we need to traverse the tree using the test data.
   14. For each record in the test dataset, for every attribute value, we check whether it is discrete or continuous and compare the values of the attribute at each node with the values it will be split upon and continue to the child node.
   15. This way, each record will traverse through the tree until a particular leaf node is reached, after checking its values at each node based on the splitting condition and whichever value is at the leaf node, that class value will be assigned as the predicted class value of that test record.
   16. Once all the test records are classified, we compute the accuracy of the decision tree based on the following formula:



Where the numerator is the total number of correctly classified test records as compared to the total number of test records.

a – TP – True Positive (Both classes are, say “Yes”)

b – TN – True Negative (Actual class is “Yes”, predicted class is “No”)

c – FP – False Positive (Actual class if “No” while the predicted class is “Yes”)

d – FN – False Negative (Both actual and predicted classes are “No”)

We mapped ‘a’ and ‘d’ to be number of correctly classified (0’s and 1’s) records and ‘b’ and ‘c’ to be number of incorrectly classified records.

* 1. After constructing decision tree for each fold with respective training and testing sets, we obtain 10 decision trees with 10 accuracies. We then take the average of all these accuracies to find the accuracy, precision, recall and f-measure value of the decision tree classifier developed.
  2. **Decision Tree Accuracies of given datasets (in %):**

|  |  |  |
| --- | --- | --- |
|  | Dataset1 | Dataset2 |
| Accuracy | 91.80 | 62.71 |
| Precision | 88.23 | 46.31 |
| Recall | 90.68 | 47.86 |
| F-Measure | 89.38 | 46.60 |

* 1. **Parameter Setting:**
     1. For decision trees, the splitting depends on the type of variables (categorical / discrete, continuous) and number of ways to split (binary or multi-split).
     2. Continuous / Discrete Variables:
        + 1. We have identified many ways on how we can construct the decision tree, one of which is considering a multi way split for continuous variables based on some range of values in that variable.
          2. However, this did not ensure that the optimum split range is selected for each dataset as merely dividing the values of the attribute into 3 or 4 sub-intervals did not provide efficient results.
          3. Later, we considered doing a binary split by taking the average of the attribute values and splitting the tree into left and right depending on whether the value of the test record is less than the average value or greater than or equal to the average.
          4. This caused the surrogate values and the split points to be the same value. Additionally, it does not handle outliers in the data well and forces everything towards the center of the sample set.
          5. This also did not provide us with efficient results, after which we identified that considering each value in a feature/attribute and identifying the left and right splits based on comparison of the rest of the values in that feature set with the initial value is much more efficient and provided a good way to select the best split.
          6. This way, entropy (and gain) for split at each value is calculated and the value at which there is lowest entropy and highest gain is selected as the best split value for that particular attribute.
          7. This worked well for all continuous attributes.
          8. For discrete variables, we identified that there are two ways to handle the splitting – one way is to do a multi split and split the tree into n number of child nodes, where n is the number of distinct values in a variable, based on which value is given in the test record’s variable.
          9. Another way is to do a binary split again on the discrete values by grouping different categorical variables in various ways and identifying which split gave the best gain. If one or both of the splits has more than one discrete variable in the group, then continue splitting it until we arrive at the leaf nodes.
          10. We went ahead with the first method and found that it does not create much complications while building the tree and also provides good accuracy while building and traversing the tree using test data that had categorical variables.
     3. Overfitting:
        + 1. Overfitting happens when there are too many features in the dataset and the tree constructed is too complex. This kind of tree poses the problem of showing too high accuracy for the training data and less adaptable to other test data. This could be avoided if we stop the decision tree construction at a certain step.
          2. Although we haven’t incorporated these steps in our implementation, since in our implementation we have tried to identify leaf nodes as soon as possible and therefore, avoid over fitting, we believe that they could help avoid the overfitting problem in decision trees.
          3. Firstly, we can keep a limit on the number of features to be used so that even if the dataset has too many features, we get a selection from only limited number of features.
          4. We could also give a maximum depth number so that the tree stops constructing child nodes for a branch which has reached the maximum depth number.
          5. Also, nodes which are not important could be identified, starting from the leaf nodes and pruned/removed as long as the error rate of classification does not change much. This known as pruning of tree, reduces the complexity of the trees and help prevent overfitting.

# **Naïve Bayes**

1. Algorithm flow
   1. Calculate the prior probability for each class.
   2. We use the below Gaussian function to estimate the probability of a given attribute value (Likelihood), given the known mean and standard deviation for the attribute estimated from the training data.



* 1. The final classification is produced by combining both sources of information, i.e., the prior and the likelihood, to form a posterior probability using the Bayes' rule.



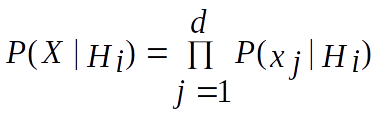
1. Dealing with continuous features

In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P (xi | y).

In this project, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**.

1. Dealing with zero probability issue

As mentioned in the slides, we can use Laplacian correction (or Laplacian estimator) to avoid the zero-probability issue in which, the descriptor posterior probability goes to 0 if any of probability is 0.



In this process, a value 1 is added to each feature count (not just to the feature having a count/frequency of 0).

# **Random Forests**

Decision trees help obtain predictions using best split terminology for each node. However, complex decision trees may lead to the problem of overfitting.

Random Forests help in preventing this problem. Typically, random forests algorithm constructs a bunch of trees and takes the majority voting of the class values for each test record. This ensures that probability of assigning correct class value to the user might increase as compared to taking the output of a single decision tree.

**Implementation:**

1. We have implemented random forest decision trees algorithm in Java.
2. The dataset is loaded first from the path specified and each record is stored.
3. As part of **pre-processing**, while parsing through the dataset file, we store what kind of attribute values a record (or row) has and this also aides in identifying the missing values.
4. While creating the unique set of values for each attribute, we purposely leave out the missing values so that no entropy or gain is calculated of split on a missing/blank value.
5. We implemented 10-fold cross validation by taking the test data for each iteration as 1/10th of the total records.
6. Choose the number of trees to be constructed in a forest in each fold (say, 10 here). This value is taken as user input as a program argument.
7. For each fold:
   * 1. We need to construct 10 trees for each fold.
     2. Choose the number of features to be considered at each node of a tree for finding the best split, say m < M (M is the total number of features in the data) as a random number between 1 and the number of features.
     3. For each tree:

Take a random bootstrap sample of training data which has the same number of records as does the training data, but is picked randomly and with replacement from the original training data.

* + 1. For each node, randomly choose m features and calculate the best split attribute and value among those m features as per the steps mentioned in decision tree implementation.
    2. This tree will be a fully grown tree and not a pruned tree.
    3. Once the tree is constructed, the test data is applied on the tree and corresponding class value predictions for each record in the test data are computed by traversing the tree.
    4. We obtain all the class values for test records for each decision tree, now a majority voting is taken out of all the class values for each record and assign that value as the predicted class value for that record in the test dataset.
    5. Calculate accuracy of the random forests algorithm, similar to decision tree, as follows:



Where the numerator is the total number of correctly classified test records as compared to the total number of test records.

* + 1. Compute the random forest algorithm with n decision trees for each fold and then calculate the average of the accuracies for each fold to obtain the accuracy of the random forest algorithm.
  1. Random Forest Accuracies for given datasets (in %):

|  |  |  |
| --- | --- | --- |
|  | Dataset1 | Dataset2 |
| No. of trees = 10 | | |
| Accuracy | 95.1 | 69.46 |
| Precision | 96.1 | 60.08 |
| Recall | 91.46 | 40.19 |
| F-Measure | 93.04 | 46.45 |
| No. of trees = 20 | | |
| Accuracy | 96.01 | 69.23 |
| Precision | 96.82 | 58.64 |
| Recall | 92.48 | 36.56 |
| F-Measure | 94.55 | 43.38 |
| No. of trees = 30 | | |
| Accuracy | 96.19 | 68.61 |
| Precision | 97.16 | 56.78 |
| Recall | 91.78 | 39.66 |
| F-Measure | 94.58 | 45.01 |

* 1. Parameter Setting
  + **No. of trees -** We need to choose the number of trees for which the algorithm gives maximum accuracy and we found that both are proportionate to each other, meaning that as the number of trees increase, the accuracy and precision value for a dataset increases, although over a small range.
  + Since the number of trees is already known, there is no requirement of any post-processing or pruning.

# **Boosting**

Boosting is used to boost the weak learners / weak classifiers to strong learners by using bootstrap samples of training data so that performance of the classifiers can be increased. It gives importance to records that have been incorrectly classified and helps the decision tree in making best splits as per these records in the training data.

**Implementation:**

* 1. The dataset is loaded first from the path specified and each record is stored.
  2. As part of pre-processing, while parsing through the dataset file, we store what kind of attribute values a record (or row) has and this also aides in identifying the missing values.
  3. While creating the unique set of values for each attribute, we purposely leave out the missing values so that no entropy or gain is calculated of split on a missing/blank value.
  4. We implemented 10-fold cross validation by taking the test data for each iteration as 1/10th of the total records.
  5. Choose the number of trees to be constructed in a forest in each fold (say, 10 here).
  6. For each fold:
     1. We need to construct 10 trees for each fold.
     2. Initially, uniform weight, (1.0) is assigned to each record in the training set.
     3. We then pick a bootstrap sample which contains as many records as the training set, but with replacement.
     4. The weights which are higher than others will be given more weightage while picking the random sample.
     5. Once the sample is attained, we build the decision tree using this sample and all the features in the sample.
     6. Traverse the tree for each record in the original training set (not the sample one) and identify which records are correctly classified and which are incorrectly classified.
     7. Update the weights based on the following formulae:

Error Rate: 

Where wj is the weight of ith record and Ci(xj) is the predicted class value for the ith record and yj is the actual class value. Here, we compute the fraction of sum of misclassified records’ weights from the total sum of weight for all the records. i denotes the current classifier/decision tree.

Importance of classifier: 

The importance of the classifier is found by taking the logarithm of (1-Error Rate) / Error Rate.

Update weight as: 

Here, is should be noted that this formula works when the class values are of the form -1 and 1, which is why, we take the class value 0 and convert it to -1 whenever we find that in either predicted or actual class value.

When both the actual and predicted class values are same, the record is correctly classified and its weight needs to be reduced, which is done using this expression as the product of yj and Ci(xj) will be positive, which implies that the expression will give a lesser value due to negative sign. When both the classes are of differing values, the expression will have a positive notation and hence, the weight gets increased.

Once we update all the weights using the numerator, take the total sum of updated weights across all the records and normalize the weight of each record using the total, which is Z(i) here.

Calculate the error rate again and if it greater than 50%, we discard that tree and construct a new one again.

If the error is less than 50%, we traverse the tree using the test set for that fold and obtain the predicted class values and store the alpha value (classifier importance) for that tree in the fold.

This procedure is followed for every tree in the fold.

Once all the n number of trees are constructed, after discarding the ones which have higher error rate than 50%, we now have n number of predicted class values for each record in the test set and corresponding alpha values for each tree.

We compute the weighted average using:



We take the sum of products of alpha and the predicted class value at each tree (1 if class value is 1 and -1 if class value is 0) and if this sum is greater than 0, we assign the class value as 1 for that record, else class 0 is assigned.

This way, all the records in the test set now have class values assigned by boosting and accuracy is computed using the correctly and incorrectly classified records.

After obtaining accuracies for each fold, we then take a weighted average of all the accuracies in the 10 folds and measure the performance of decision trees using boosting, or rather, using AdaBoost algorithm.

* 1. Boosting Accuracies for given datasets (in %):

|  |  |  |
| --- | --- | --- |
|  | Dataset1 | Dataset2 |
| No. of trees = 10 | | |
| Accuracy | 94.23 | 67.11 |
| Precision | 93.62 | 52.75 |
| Recall | 90.44 | 50.19 |
| F-Measure | 91.92 | 50.44 |
| No. of trees = 20 | | |
| Accuracy | 94.38 | 66.87 |
| Precision | 93.72 | 52.23 |
| Recall | 90.51 | 51.94 |
| F-Measure | 91.98 | 51.28 |
| No. of trees = 30 | | |
| Accuracy | 94.20 | 67.54 |
| Precision | 93.34 | 53.78 |
| Recall | 90.71 | 53.81 |
| F-Measure | 91.96 | 52.61 |

h. Parameter Setting

* + **No of trees**: We need to choose the number of trees for which the algorithm gives maximum accuracy and we found that both are proportionate to each other, meaning that as the number of trees increase, the accuracy and precision for a dataset increases, although over a small range.
  + Since the number of trees is already known, there is no requirement of any post-processing or pruning.
  + After certain number, the accuracy tends to remain constant as the number of trees increase, which might be because the best weights are already determined.

# **6. Cross Validation**

## Non-exhaustive cross validation methods

## K-Fold Cross Validation

As there is never enough data to train your model, *removing a part of it for validation poses a problem of underfitting.* ***By reducing the training data***, ***we risk losing important patterns/ trends in data set, which in turn increases error induced by bias.*** So, what we require is a method that provides ample data for training the model and also leaves ample data for validation. K Fold cross validation does exactly that.

In **K Fold cross validation**, the data is divided into k subsets. Now the holdout method is repeated k times, such that ***each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set***. The *error estimation is averaged over all k trials to get total effectiveness of our model*. As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set *k-1*times. ***This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set.*** Interchanging the training and test sets also adds to the effectiveness of this method.

## Stratified K-Fold Cross Validation

In some cases, there may be a large imbalance in the response variables. For example, in dataset concerning price of houses, there might be large number of houses having high price. Or in case of classification, there might be several times more negative samples than positive samples. For such problems, a slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds. This variation is also known as Stratified K Fold.

## Exhaustive cross validation methods

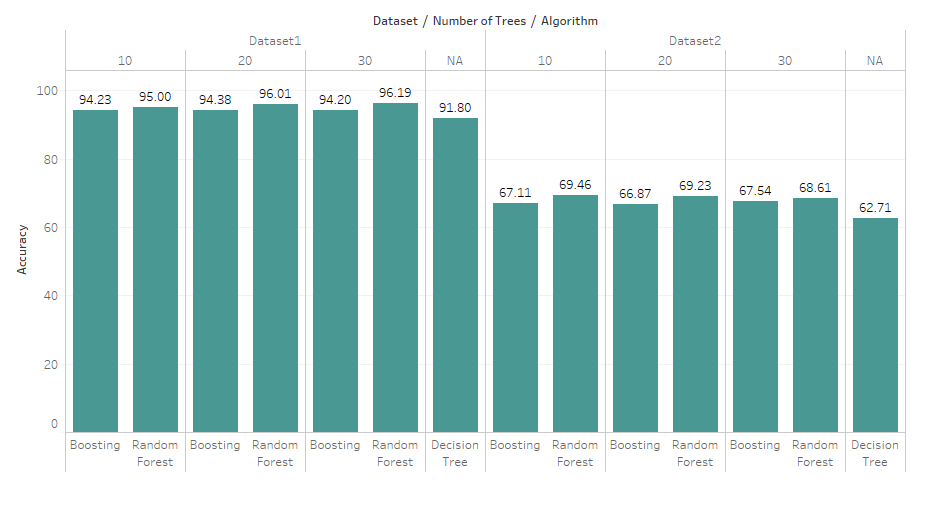
## Leave-P-Out Cross Validation

This approach leaves p data points out of training data, i.e. if there are n data points in the original sample then, n-p samples are used to train the model and p points are used as the validation set. This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness.

This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible [[ref]](https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f).

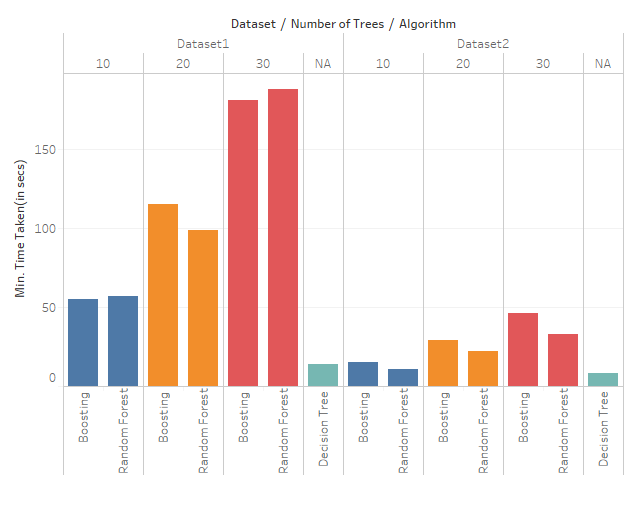
A particular case of this method is when p = 1. This is known as Leave one out cross validation. This method is generally preferred over the previous one because it does not suffer from the intensive computation, as number of possible combinations is equal to number of data points in original sample or n.

# **Result Analysis**



Here we see that consistently Random Forest marginally better accuracy than Boosting Algorithm . However, both of the bossting and random forest techniques give us a huge accuracy improvement as compared simple decision tree results.

Also, it is worth mentioning here that the accuracy of a certain dataset increases as the number of trees in it is increased.



Here, we also have to take into account the performance vs time trade off. Therefore, as seen in the chart above, we see that for less number of trees, both Boosting and Random Forest techniques take comparable time (in seconds) for tree construction, training and testing as per our C4.5 Algorithm implementation in Java.

However, as we increase the number of trees, on an average of 3 iterations, boosting takes slightly more time than random forest technique, perhaps because of the multiple constructions of tree as the ones with error rate greater than 0.5 are discarded and re-constructed again.

# **8. Pros and Cons of five classification algorithms**

## **K-Nearest Neighbors**

**Pros:**

1. Simple to implement
2. Flexible to feature / distance choices
3. Naturally handles multi-class cases
4. Can do well in practice with enough representative data

**Cons:**

1. Large search problem to find nearest neighbors
2. Storage of data
3. Must know we have a meaningful distance function

## **Naïve Bayes**

**Pros:**

1. It’s relatively simple to understand and build
2. It’s easily trained, even with a small dataset
3. It’s fast!
4. It’s not sensitive to irrelevant features

**Cons:**

1. It assumes every feature is independent, which isn’t always the case.

## **Decision Trees**

**Pros of Decision Tree Algorithms:**

* 1. Easy to interpret, visualize and trace output for smaller trees.
  2. It helps in giving faster results for unclassified records.
  3. Inexpensive to construct.
  4. Not much data preprocessing is required for implementing decision tree algorithms.
  5. Accuracy is high when run on simpler datasets and is at par with other classification techniques.
  6. It can handle both categorical and continuous variables well.

**Cons of Decision Tree Algorithms:**

1. Overfitting: The decision tree might be difficult to construct when there are too many features and the complex structure of the tree can lead to overfitting.
2. If small changes are made to the data, then we might get a complete different decision tree.
3. There can be many decision trees made on the same data, which will depend on the best split that we choose. It may be possible that the decision tree constructed is not the one with highest accuracy.

## **Random Forest**

**Pros of Random Forest Algorithm:**

* + 1. Random forests help in both classification and regression tasks.
    2. It can handle large datasets and perform dimensionality reduction to identify important features.
    3. It produces a highly accurate classifier for datasets with thousands of features.
    4. It is significantly faster to train and the output of a normal validation, without 10-fold cross validation gives a decent accuracy.

**Cons of Random Forest Algorithm**:

Overfitting problem appears here too as we are building complete trees, without pruning, and there is a chance that when random ‘m’ features are selected without looking at the relative importance of the features, the tree might have a complex structure.

## **Boosting**

**Pros of Boosting**:

1. Feature selection is done in an optimized way, based on the weights of the data, thereby resulting in a simple classifier.
2. Weighted average ensures that misclassification rate gets decreased as when compared to that of decision trees and random forest.
3. Simple to implement.

**Cons of Boosting:**

1. Sensitive to noise and outliers.

References:

[[ref]](https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f) : https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f