CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Ensemble Methods

Prepared by:
Pratik Tiwari (2019201023)
Kritika Singh (2019201075)
Neha (2019900080)

Contents

1 ENSEMBLE METHODS

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.The goal of any machine learning problem is to find a single model that will best predict our wanted outcome. Rather than making one model and hoping this model is the best/most accurate predictor we can make, ensemble methods take a myriad of models into account, and average those models to produce one final model. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error).
Types of ensemble classifier:

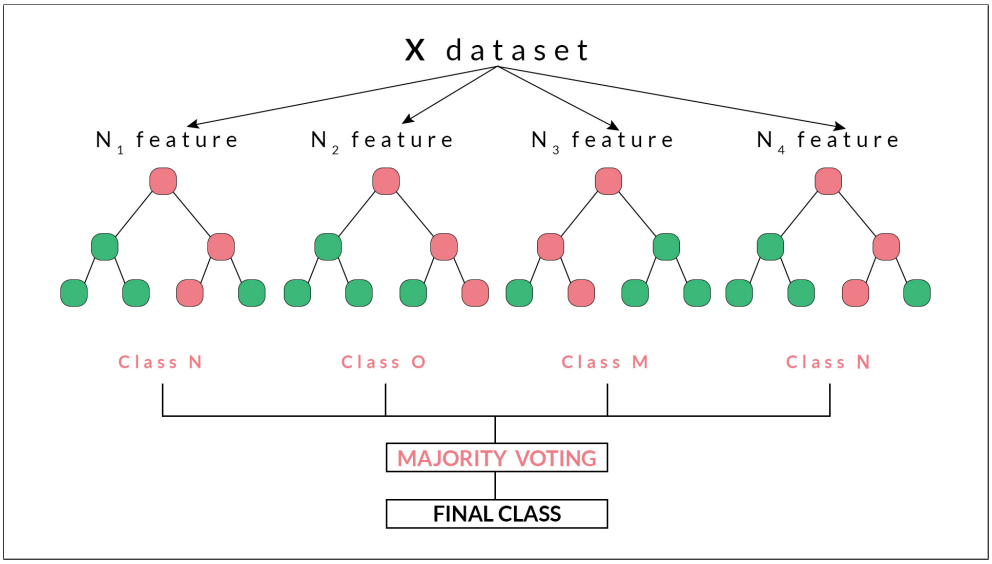
1. Random forest
2. Bagging

1.1 RANDOM FOREST

Random Forest algorithm is a supervised classification algorithm.The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a "forest"), this model uses two key concepts that gives it the name random: .

1. Random sampling of training data points when building trees.
2. Random subsets of features considered when splitting nodes.

In data science speak, the reason that the random forest model works so well is:A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.The low correlation between models is the key. The trees protect each other from their individual errors (as long as they don't constantly all produce error in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.With a random forest model, our chances of making correct predictions increase with the number of uncorrelated trees in our model.



1.2 BAGGING (BOOTSTRAP AGGREGATION)

Bagging (Bootstrap Aggregation) — Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.With bagging we are not sub setting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6]. Notice that both lists are of length six and that "2" and "6" are both repeated in the randomly selected training data we give to our tree (because we sample with replacement).Bagging (Bootstrap Aggregation) is used to reduce the variance of a decision tree.

ALGORITHM

1. Draw a bootstrap sample Z* of size N from the training data.
2. Grow a random-forest Tb to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree,until the minimum node size n of reached.
   (a) Select m variables at random from p variables.
   (b) Pick the best variable/split-point among the m
   (c) Split the node into two daughter nodes
3. Output the ensemble of trees.

EXAMPLE

1. Create a Bootstrapped dataset.

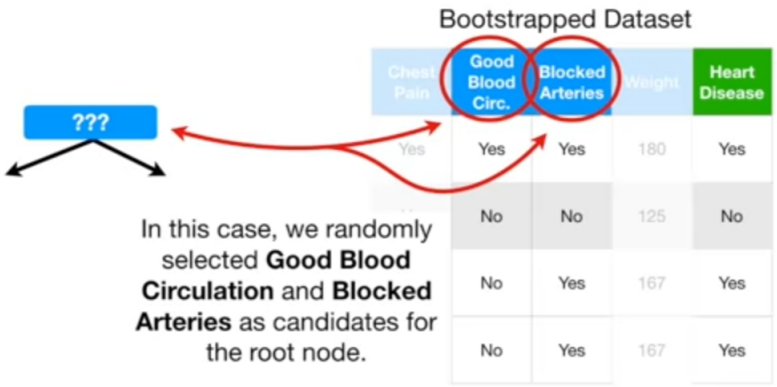
Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

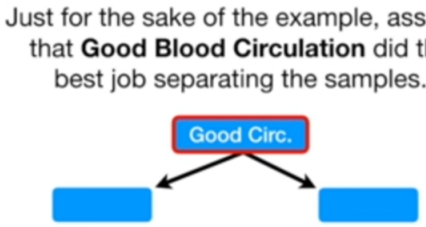
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

2. Create a decision tree using bootstrapped dataset,but only use a random subset of variables or columns at each step.



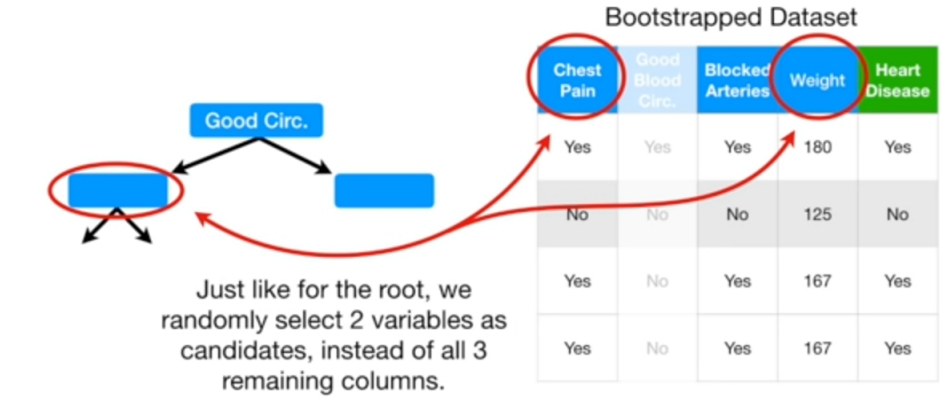
3. Lets assume Good blood circulation gives least Gini index .

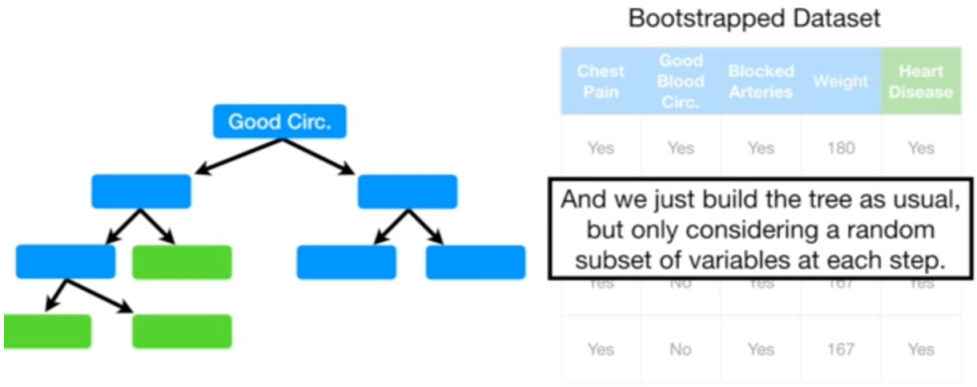


Bootstrapped Dataset

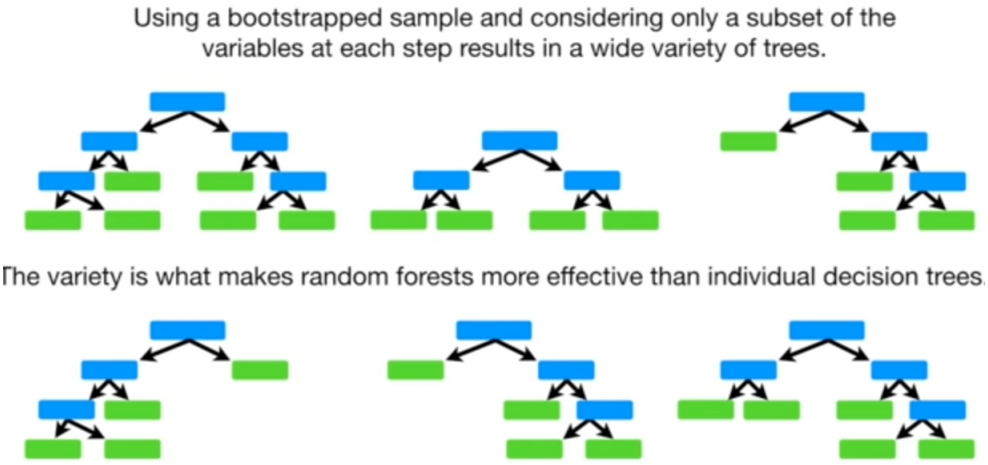
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

4. Select random subset of variables at each node.





5. Repeat these steps to obtain the total number of trees.



IDEA OF RANDOM SELECTION VARIABLE SELECTION
In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node. In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.So in our random forest, we end up with trees that are not only trained on different sets of data (thanks to bagging) but also use different features to make decisions.The size of the subset of random features at every node is generally square root of total number of features present.

1.3 OUT-OF-BAG(OOB) ERROR

Out of bag (OOB) error is a way of validating the Random forest model.In Random forest we do not split the data into training and validation set.In Random forest we bootstrap the dataset for building the trees.Now,nearly one third of the sample data do not end up in the Bootstrap dataset(due to replacement of data).This is called Out-Of-Bag dataset.This dataset can be used for testing of the trees.We can do this for all Out-Of-Bag samples for all trees.Proportion of Out-Of-Bag samples that were incorrectly classified is called Out-Of-Bag error. OOB leads to reducing the overall aggregation effect in bagging. In an ideal case, about 36.8 percent of the total training data forms the OOB sample.
let's assume there are five DTs in the random forest ensemble labeled from 1 to 5

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Let the first bootstrap sample be made of the first three rows of this data set as shown in the green box below.

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Bootstrap sample

Then the last row that is "left out" in the original data (see the red box in the image below) is known as the Out of Bag sample. This row will not be used as the training data

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Out of Bag sample

After the DTs models have been trained, this leftover row or the OOB sample will be given as unseen data to the DT 1. The DT 1 will predict the outcome of this row. Let DT 1 predict this row correctly as "YES". Similarly, this row will be passed through all the DTs that did not contain this row in their bootstrap training data.

Decision Tree	Prediction
1	YES
3	NO
5	YES
Majority vote : YES	

By a majority vote of 2 "YES" vs 1 "NO" the prediction of this row is "YES". It is noted that the final prediction of this row by majority vote is a correct prediction since originally in the "Play Tennis" column of this row is also a "YES".Similarly, each of the OOB sample rows is passed through every DT that did not contain the OOB sample row in its bootstrap training data and a majority prediction is noted for each row.And lastly, the OOB error is computed as the number of correctly predicted rows from the out of bag sample

1.4 ONE FEATURE SELECTION USING RANDOM FOREST

There are basically two methods through which random forest algorithm can be used for selection of important features from a large set of dimensions:

1. Mean decrease accuracy.

Directly measures the impact of each feature on accuracy of the model. The general idea is to permute the values of each feature and measure how much the permutation of value a feature decreases the accuracy of the model or increases the OOB error. Clearly, for unimportant variables, the permutation should have little to no effect on model accuracy, while permuting important variables should significantly decrease it.

2. Mean decrease impurity.

   Random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity. For classification, it is typically either Gini impurity or information gain/entropy and for regression trees it is variance. Thus when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.

1.5 TREES Vs RANDOM FOREST

Trees

1. Trees are very sensitive to data on which they are training and lead to overfitting of data thus producing high variance.
2. They are clear,simpler and easy to tune parameters.
3. They are fast.
4. They yield insights into the decision rules used to build the tree.

Random Forest

1. They have smaller prediction variance and therefore perform better than decision trees.
2. It is easier to tune parameters in case of random forest.
3. Since it involves building a large number of trees so they are slow.
4. Based on “Black Box” concept.It’s harder to get insights into decision rules used to build the trees.

APPLICATIONS

1. For the application in medicine, Random Forest algorithm can be used to both identify the correct combination of components in medicine, and to identify diseases by analyzing the patient’s medical records.
2. For the application in the stock market, Random Forest algorithm can be used to identify a stock’s behavior and the expected loss or profit.

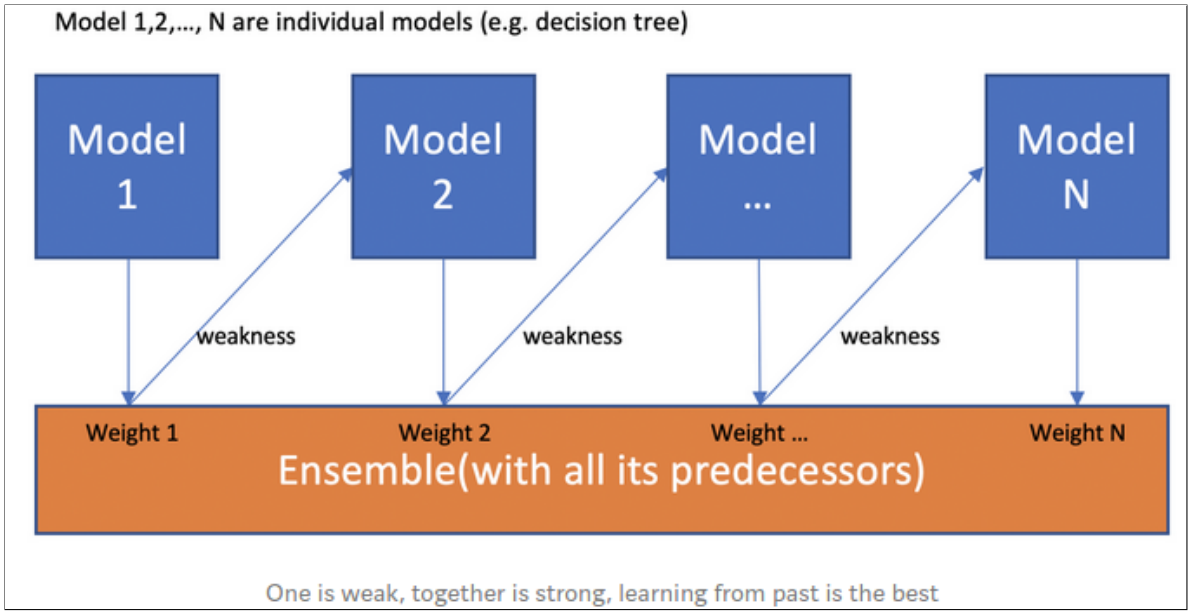
3. For the application in banking, Random Forest algorithm is used to find loyal customers, which means customers who can take out plenty of loans and pay interest to the bank properly, and fraud customers, which means customers who have bad records like failure to pay back a loan on time or have dangerous actions.

4. For the application in e-commerce, Random Forest algorithm can be used for predicting whether the customer will like the recommend products,based on the experience of similar customers.

2 Boosting Algorithms

Boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors.

Boosting is a generic algorithm rather than a specific model. Boosting needs you to specify a weak model (e.g. regression, shallow decision trees, etc) and then improves it.



2.1 AdaBoost Algorithm

AdaBoost is one of the first boosting algorithms to be adapted in solving practices. Adaboost helps you combine multiple "weak classifiers" into a single "strong classifier".

- The weak learners in AdaBoost are decision trees with a single split, called decision stumps.
- AdaBoost works by putting more weight on difficult to classify instances and less on those already handled well.
- AdaBoost algorithms can be used for both classification and regression problem.

$$\text{for a weak classifier } C$$
$$X: n \times d, Y: n \times k \text{ with sample weights } W: n \times 1$$
$$Error = \frac{\sum_{j=1}^{n} W_j I(C(X_j) \neq Y_j)}{\sum_{j=1}^{n} W_j}, I(x) = \begin{cases} 1, & \text{if } x \text{ is True} \\ 0, & \text{if } x \text{ is False} \end{cases}$$

subsubsection Understanding AdaBoost using Decision Stumps AdaBoost is a specific Boosting algorithm developed for classification problems (also called discrete AdaBoost). The weakness is identified by the weak estimator's error rate:

2.2 Psuedo-Code Adaboosting

$$\text{Initial weights } W: W_1, W_2, \ldots, W_n = \frac{1}{n}$$
$$\text{for } i \text{ in } [1, M] \text{// M weak classifiers}$$
$$\text{fit weak classifier } C^i \text{ with sample weights } W$$
$$Error^i = \frac{\sum_{j=1}^{n} W_j I(C^i(X_j) \neq Y_j)}{\sum_{j=1}^{n} W_j}$$
$$\alpha^i = \log\left(\frac{(1 - Error^i)}{Error^i}\right) + \log(K - 1) \text{//coefficient for } C^i$$
$$W_j = W_j * e^{\alpha^i * I(C^i(X_j) \neq Y_j)} \text{ for } W_j \in W \text{// if wrong, increase weights}$$
$$W = W - mean(W) \text{// normalize weights}$$
$$\text{//output of the model is done by weighted voting, find the class with highest vote}$$
$$\text{Prediction: } \hat{Y}_j = \max_k \left(\sum_{i=1}^{i} \alpha_i I(C^i(X_j) = k)\right)$$

2.3 Training Model using Adaboost

A weak classifier (decision stump) is prepared on the training data using the weighted samples. Only binary (two-class) classification problems are supported, so each decision stump makes one decision on one input variable and outputs a +1.0 or -1.0 value for the first or second class value.

The misclassification rate is calculated for the trained model. Traditionally, this is calculated as: **error = (correct − N) / N**

Where error is the misclassification rate, correct are the number of training instance predicted correctly by the model and N is the total number of training instances. For example, if the model predicted 78 of 100 training instances correctly the error or misclassification rate would be (78-100)/100 or 0.22.

This is modified to use the weighting of the training instances:

error = sum(w(i) * terror(i)) / sum(w)

Which is the weighted sum of the misclassification rate, where w is the weight for training instance i and terror is the prediction error for training instance i which is 1 if misclassified and 0 if correctly classified.

For example, if we had 3 training instances with the weights 0.01, 0.5 and 0.2. The predicted values were -1, -1 and -1, and the actual output variables in the instances were -1, 1 and -1, then the terrors would be 0, 1, and 0. The misclassification rate would be calculated as:

error = (0.01*0 + 0.5*1 + 0.2*0) / (0.01 + 0.5 + 0.2) or error = 0.704

A stage value is calculated for the trained model which provides a weighting for any predictions that the model makes. The stage value for a trained model is calculated as follows:

stage = ln((1-error) / error)

Where stage is the stage value used to weight predictions from the model, ln() is the natural logarithm and error is the misclassification error for the model. The effect of the stage weight is that more accurate models have more weight or contribution to the final prediction.

The training weights are updated giving more weight to incorrectly predicted instances, and less weight to correctly predicted instances.

For example, the weight of one training instance (w) is updated using:

w = w * exp(stage * terror)

Where w is the weight for a specific training instance, exp() is the numerical constant e or Euler's number raised to a power, stage is the misclassification rate for the weak classifier and terror is the error the weak classifier made predicting the output variable for the training instance, evaluated as:

terror = 0 if(y == p), otherwise 1

Where y is the output variable for the training instance and p is the prediction from the weak learner.

This has the effect of not changing the weight if the training instance was classified correctly and making the weight slightly larger if the weak learner misclassified the instance.

2.4 Making Prediction using AdaBoost

Predictions are made by calculating the weighted average of the weak classifiers.

For a new input instance, each weak learner calculates a predicted value as either +1.0 or -1.0. The predicted values are weighted by each weak learners stage value. The prediction for the ensemble model is taken as a the sum of the weighted predictions. If the sum is positive, then the first class is predicted, if negative the second class is predicted.

For example, 5 weak classifiers may predict the values 1.0, 1.0, -1.0, 1.0, -1.0. From a majority vote, it looks like the model will predict a value of 1.0 or the first class. These same 5 weak classifiers may have the stage values 0.2, 0.5, 0.8, 0.2 and 0.9 respectively. Calculating the weighted sum of these predictions results in an output of -0.8, which would be an ensemble prediction of -1.0 or the second class.

2.5 Example

1. Adaboost combines lots of weak learners for classification.
2. Some stumps get more say in classification then others.
3. Each stump is made up by taking previous stump mistake in account.

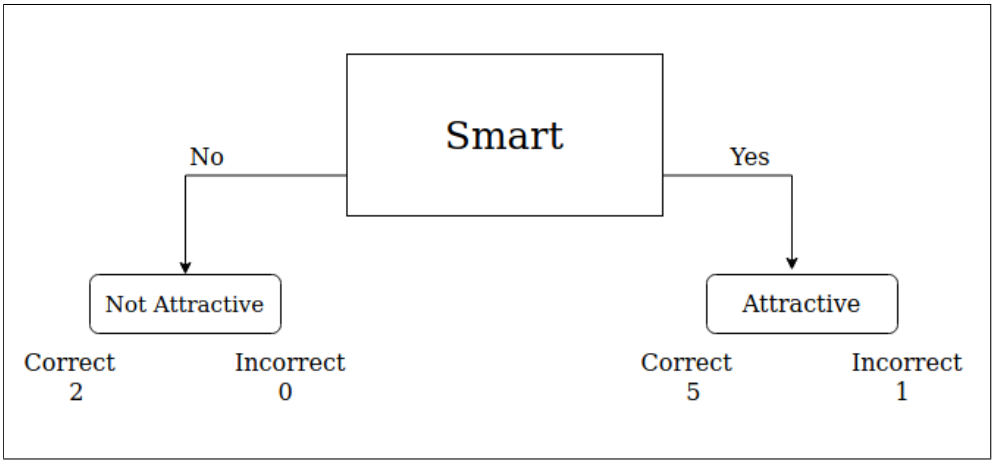
1. In the proceeding example, we'll be using a dataset that categories people as attractive or not based on certain features.

Weight	Smart	Polite	Fit	Attractive
180	No	No	No	No
150	Yes	Yes	No	No
175	No	Yes	Yes	Yes
165	Yes	Yes	Yes	Yes
190	No	Yes	No	No
201	Yes	Yes	Yes	Yes
185	Yes	Yes	No	Yes
168	Yes	No	Yes	Yes

2. Initialize the sample weights

Weight	Smart	Polite	Fit	Attractive	Sample Weights
180	No	No	No	No	**1/8**
150	Yes	Yes	No	No	**1/8**
175	No	Yes	Yes	Yes	**1/8**
165	Yes	Yes	Yes	Yes	**1/8**
190	No	Yes	No	No	**1/8**
201	Yes	Yes	Yes	Yes	**1/8**
185	Yes	Yes	No	Yes	**1/8**
168	Yes	No	Yes	Yes	**1/8**

3. Build a decision tree with each feature, classify the data and evaluate the result.For example, assume that we built a tree that classifies people as attractive if they're smart and unattractive if they're not.



4. Calculate the significance of the tree in the final classification.
5. **Significance = 1/2*log((1-total error)/total Error)**
6. **Total error = Sum of weights of incorrect samples**
7. **Significance = 1/2*log((1-(1/8)/(1/8))**
8. Update the sample weights so that the next decision tree will take the errors made by the preceding decision tree into account
9. We look at the samples that the current tree classified incorrectly and increase their associated weights using the following formula.
10. **New Weight Sample = Sample Weight * pow(e,significance)**

11. we look at the samples that the tree classified correctly and decrease their associated weights using the following formula.

12. **New Weight Sample = Sample Weight * pow(e,(-S))**

Weight	Smart	Polite	Fit	Attractive	Sample Weights	New Weight	Normalized Weight
180	No	No	No	No	**1/8**	0.05	0.07
150	Yes	Yes	No	No	**1/8**	0.33	0.49
175	No	Yes	Yes	Yes	**1/8**	0.05	0.07
165	Yes	Yes	Yes	Yes	**1/8**	0.05	0.07
190	No	Yes	No	No	**1/8**	0.05	0.07
201	Yes	Yes	Yes	Yes	**1/8**	0.05	0.07
185	Yes	Yes	No	Yes	**1/8**	0.05	0.07
168	Yes	No	Yes	Yes	**1/8**	0.05	0.07

13. Form a new dataset
    We start by making a new and empty dataset that is the same size as the original. Then, imagine a roulette table where each pocket corresponds to a sample weight. We select numbers between 0 and 1 at random. The location where each number falls determines which sample we place in the new dataset.
    Since the samples that were incorrectly classified have higher weights in relation to the others, the likelihood that the random number falls under their slice of the distribution is greater. Therefore, the new dataset will have a tendency to contain multiple copies of the samples that were misclassified by the previous tree. As a result, when we go back to the step where we evaluate the predictions made by each decision tree, the one with the highest score will have correctly classified the samples the previous tree misclassified.

14. Repeat steps 2 through 5 until the number of iterations equals the number specified by the hyperparameter (i.e. number of estimators)

15. Use the forest of decision trees to make predictions on data outside of the training set

16. The AdaBoost model makes predictions by having each tree in the forest classify the sample. Then, we split the trees into groups according to their decisions. For each group, we add up the significance of every tree inside the group. The final classification made by the forest as a whole is determined by the group with the largest sum.