Contents

[Ensemble model 1](#_Toc521329372)

[Bagging(Bootstrap Aggregating) 3](#_Toc521329373)

[Steps for Bagging 4](#_Toc521329374)

[To extend Bagging for classification problem – majority vote 4](#_Toc521329375)

[Random Forest 4](#_Toc521329376)

[Algorithm 5](#_Toc521329377)

[Why Random Forest Pros 6](#_Toc521329378)

[Cons 7](#_Toc521329379)

[Error Estimation - out-of-bag (OOB) observations 7](#_Toc521329380)

[Code 8](#_Toc521329381)

[Boosting 9](#_Toc521329382)

[Algorithm – Weighted Classifier are used 9](#_Toc521329383)

[AdaBoosting 9](#_Toc521329384)

[Gradient Boosting 10](#_Toc521329385)

[How Gradient Boosting Works 10](#_Toc521329386)

[Why with Trees 12](#_Toc521329387)

[Stacking 12](#_Toc521329388)

[Interview Questions 12](#_Toc521329389)

[What is the difference between AdaBoost, and Gradient Boosting? 12](#_Toc521329390)

[What is the difference between Random forest and Bagging 13](#_Toc521329391)

[What are the differences between Random Forest and Gradient Tree Boosting algorithms? 13](#_Toc521329392)

[How to avoid overfitting in random forest 13](#_Toc521329393)

[What is the difference between Bagging and Boosting 14](#_Toc521329394)

[Hoe do you interpret Random Forest 14](#_Toc521329395)

[What is the main idea behind ensemble learning? If I had many different models that predicted the same response variable, what might I want to do to incorporate all of the models? Would you expect this to perform better than an individual model or worse? 14](#_Toc521329396)

# Ensemble model

Imagine you are playing the game “Who wants to be millionaire?” and reached up to last question of 1 million dollars. You have no clue about the question, but you have audience poll and phone a friend life lines. Thank God. At this stage you don’t want to take any risk, so what will you do to get sure-shot right answer to become millionaire?

You will use both life lines, isn’t it? Let’s say 70% audience is saying right answer is D and your friend is also saying the right answer is D with 90% confidence because he is an expert in the area of the question. Use of both life lines gives you  an average 80% confidence that D is correct and gets you closer to becoming a millionaire.

This is the approach of [**ensemble methods**](https://www.kdnuggets.com/tag/ensemble-methods).

The famous [Netflix Prize](https://en.wikipedia.org/wiki/Netflix_Prize) competition took almost 3 years before the goal of 10% improvement [was reached](https://www.kdnuggets.com/news/2009/n14/1i.html).  The winners used gradient boosted decision trees to combine over [500 models](http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/).

In ensemble methods, more diverse the models used, more robust will be the ultimate result.

Different models used in ensemble improves overall variance from difference in population, difference in hypothesis generated, difference in algorithms used and difference in parametrization. There are main 3 widely used ensembles techniques:

Bagging

[Boosting](https://www.kdnuggets.com/tag/boosting)

Stacking

So if you have different models built for same data and same response variable, you can use one of the above methods to build ensemble model. As every model used in the ensemble has its own performance measures, some of the models may perform better than ultimate ensemble model and some of them may perform poorer than or equal to ensemble model. But overall the ensemble methods will improve overall accuracy and stability of the model, although at the expense of model understandability.

Ensembling is a technique of combining two or more algorithms of similar or dissimilar types called base learners.

# Bagging(Bootstrap Aggregating)

*(Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly*with replacement*. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.*

The decision tree has a problem of high variance. **Averaging** a set of observation reduces the variance .In this we First, we create random samples of the training data set (sub sets of training data set). Then, we build a classifier for each sample. Finally, results of these multiple classifiers are combined using average or majority voting. Bagging helps to reduce the variance error.

Assume that we need to learn a decision tree to predict the price of a house based on 100 inputs. Prediction accuracy of such a decision tree would be low, given the problem of variance it suffers from. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we may get could be quite different. What we really want is a result that has low variance if applied repeatedly to distinct data sets.

Create many (e.g. 100) random sub-samples of our dataset with replacement (meaning we can select the same value multiple times). Learn(train) a decision tree on each sample.

Given new dataset, Calculate the prediction for each sub-sample. Calculate the average of all of our collected predictions(also called bootstrap estimates) and use that as our estimated prediction for the data. In this approach, trees are grown deep and are not pruned . Thus each individual tree has high variance, but low bias. Averaging these trees reduces the

variance dramatically.

Bagging has a single parameter, which is the number of trees. All trees are fully grown binary tree (unpruned) and at each node in the tree one searches over all features to find the feature that best splits the data at that node.

The application of the Bootstrapping procedure to a high-variance machine learning algorithm, typically decision trees as shown in the above example, is known as Bagging (or bootstrap aggregating).

### Steps for Bagging

1. **Create Multiple DataSets:**

Sampling is done with replacement on the original data and new datasets are formed.

1. **Build Multiple Classifiers:**

Classifiers are built on each data set.

Generally the same classifier is modeled on each data set and predictions are made.

1. **Combine Classifiers:**

The predictions of all the classifiers are combined using a mean, median or mode value

depending on the problem at hand.

The combined values are generally more robust than a single model.

### To extend Bagging for classification problem – majority vote

1. **parallel** ensemble: each model is built independently
2. aim to **decrease variance**, not bias
3. suitable for high variance low bias models (complex models)
4. an example of a tree based method is **random forest**, which develop fully grown trees (note that RF modifies the grown procedure to reduce the correlation between trees)

## Random Forest

**How Random forest is different from Bagging**

Bagging has a single parameter, which is the number of trees. All trees are fully grown binary tree (unpruned) and at each node in the tree one searches over all features to find the feature that best splits the data at that node.

Random forests has 2 parameters:

The first parameter is the same as bagging (**the number of trees**)

The second parameter (unique to randomforests) is mtry which is **how many features** to search over to find the best feature. this parameter is usually 1/3\*D for regression and sqrt(D) for classification. thus during tree creation randomly mtry number of features are chosen from all available features and the best feature that splits the data is chosen.

Random forest is an extension of bagged decision trees. Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split.

The fundamental difference is that in Random forests, only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node.

**Random forests has 2 parameters:**

The first parameter is the same as bagging (the number of trees)

The second parameter (unique to randomforests) is mtry which is how many features to search over to find the best feature. this parameter is usually **1/3\*D for regression** and **sqrt(D)** for classification. thus during tree creation randomly mtry number of features are chosen from all available features and the best feature that splits the data is chosen.

Multiple CART fitted on bootstrap dataset drawn from the sample.

***Feature Importance***

One problem with computing fully grown trees is that we cannot easily

interpret the results. And it is no longer clear which variables are

important to the relationship. **Calculating drop in the error function for a**

**variable at each split point gives us an idea of feature importance**. It

means that we record the total amount that the error is decreased due to

splits over a given predictor, averaged over all bagged trees. A large value

then indicates an important predictor.

In regression problems this may be

the drop in residual sum of squares and in classification this might be the

Gini score.

### Algorithm

* Bootstrap sample
* At each step bootstrap variables
* Grow multiple tree and vote

1. Randomly select **“k”** features from total **“m”** features.
   1. Where **k << m**
2. Among the**“k”** features, calculate the node **“d”** using the best split point.
3. Split the node into **daughter nodes** using the **best split**.
4. Repeat **1 to 3** steps until “l” number of nodes has been reached.
5. Build forest by repeating steps **1 to 4** for “n” number times to create **“n” number of trees**.



### Why Random Forest Pros

It reduces variance of the model

One of benefits of Random forest which excites me most is, the power of handle large data

set with higher dimensionality. **It can handle thousands of input variables and identify most**

**significant variables** so it is considered as one of the dimensionality reduction methods.

Further, the model outputs Importance of variable, which can be a very handy feature (on

some random data set).

It has an effective method for estimating missing data and maintains accuracy when a large

proportion of the data are missing.

It has methods for balancing errors in data sets where classes are imbalanced.

The capabilities of the above can be extended to unlabeled data, leading to unsupervised

clustering, data views and outlier detection.

Random Forest involves sampling of the input data with replacement called as bootstrap

sampling. Here one third of the data is not used for training and can be used to testing.

These are called the out of bag samples. Error estimated on these out of bag samples is

known as out of bag error. Study of error estimates by Out of bag, gives evidence to show

that the out-of-bag estimate is as accurate as using a test set of the same size as the

training set. Therefore, using the out-of-bag error estimate removes the need for a set aside

test set.

### Cons

* Speed
* Interpretability
* Overfitting

It surely does a good job at classification but not as good as for **regression** problem as it

does not give precise continuous nature predictions. In case of regression, it doesn’t predict

beyond the range in the training data, and that they may over-fit data sets that are

particularly noisy.

Random Forest can feel like a black box approach for statistical modelers – you have very

little con

### Error Estimation - out-of-bag (OOB) observations

An easy way of estimating the test error of a bagged model, without the

need for cross-validation is Out-of-Bag Error Estimation. The

observations not used to fit a given bagged tree are referred to as the

out-of-bag (OOB) observations. We can simply predict the response for the

ith observation using each of the trees in which that observation was OOB.

We average those predicted responses, or take a majority vote, depending

on if the response is quantitative or qualitative. An overall OOB

MSE(mean squared error) or classification error rate can be computed.

This is an acceptable test error rate because the predictions are based on

only the trees that were not fit using that observation.

## Code

# Import the model we are using  
from sklearn.ensemble import RandomForestRegressor

# Instantiate model with 1000 decision trees  
rf = RandomForestRegressor(n\_estimators = 1000, random\_state = 42)

# Train the model on training data  
rf.fit(train\_features, train\_labels);

**n\_estimators** : integer, optional (default=10)

The number of trees in the forest.

**random\_state** : int, RandomState instance or None, optional (default=None)

If int, random\_state is the seed used by the random number generator; If RandomState instance, random\_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

# Boosting

*The idea is to used the weak learning method several times to get a succession of hypotheses, each one refocused on the examples that the previous ones found difficult and misclassified. … Note, however, it is not obvious at all how this can be done*

1. **sequential** ensemble: try to add new models that do well where previous models lack
2. aim to **decrease bias**, not variance
3. suitable for low variance high bias models
4. an example of a tree based method is **gradient boosting**

The prediction accuracy of decision trees can be further improved by using Boosting algorithms. The basic idea behind boosting is converting many weak learners to form a single strong learner

Weak learner is a learner that will always do better than chance, when it tries to label the data, no matter what the distribution over the training data is. Doing better than chance means we are always going to have an error rate which is less than 1/2. This means that the learner algorithm is always going to learn something, and will not always be completely accurate i.e., it is weak and poor when it comes to learning the relationships between inputs and target. It also means a rule formed using a single predictor/classifier is not powerful individually.

Another parameter boosting takes is the number of iterations or number of trees in this case.

Additionally, it assigns weights to the inputs based on whether they were correctly predicted/classified or not.

## Algorithm – Weighted Classifier are used

1.First, the inputs are initialized with equal weights.

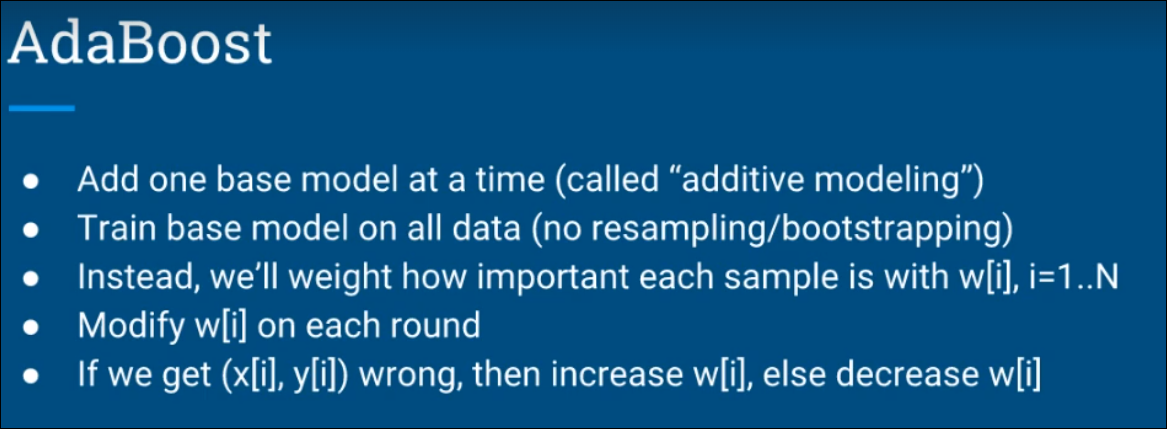
2.Update the weights of inputs based on previous run, and weights are higher for wrongly predicted/classified inputs

3.Make another rule(decision stump in this case) and fit it to a subsample of data. Note that this time rule will be formed by keeping the wrongly classified inputs(ones having higher weight) in mind.

4.After the iterations have been completed, we combine weak rules to form a single strong rule, which will then be used as our model

## AdaBoosting

The first realization of boosting that saw great success in application was Adaptive Boosting or AdaBoost for short. AdaBoost works by weighting the observations, putting more weight on difficult to classify instances and less on those already handled well. New weak learners are added sequentially that focus their training on the more difficult patterns.



In regression setting, the prediction error(usually calculated using least squares) is used to adjust weights of inputs, and consequent learners focus more on inputs with large error. This type of boosting approach is known as Adaptive Boosting or AdaBoost.

As with trees, boosting approach also minimizes a loss function. In case of Adaboost, it is the exponential loss function.

## Gradient Boosting

The basic concept remains the same, except here we don’t play with the weights, but fit the model on residuals (measurement of the difference in prediction and original outcome) rather than original outcomes. Adaboost is implemented using iteratively refined sample weights while Gradient Boosting uses an internal regression model trained iteratively on the residuals. This means that the new weak learners are formed keeping in mind the inputs that have high residuals.

### How Gradient Boosting Works

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

**Intution Behind Gradient Boosting**

**1.**Fit a simple linear regressor or decision tree on data (I have chosen decision tree in my code) [call x as input and y as output]

2. Calculate error residuals. Actual target value, minus predicted target value [e1= y - y\_predicted1 ]

3. Fit a new model on error residuals as target variable with same input variables [call it e1\_predicted]

4. Add the predicted residuals to the previous predictions

[y\_predicted2 = y\_predicted1 + e1\_predicted]

5. Fit another model on residuals that is still left. i.e. [e2 = y - y\_predicted2] and repeat steps 2 to 5 until it starts overfitting or the sum of residuals become constant. Overfitting can be controlled by consistently checking accuracy on validation data.

**1. Loss Function**

The loss function used depends on the type of problem being solved.

It must be differentiable, but many standard loss functions are supported and you can define your own.

For example, regression may use a squared error and classification may use logarithmic loss.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.

**2. Weak Learner**

Decision trees are used as the weak learner in gradient boosting.

Specifically regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and “correct” the residuals in the predictions.

Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.

Initially, such as in the case of AdaBoost, very short decision trees were used that only had a single split, called a decision stump. Larger trees can be used generally with 4-to-8 levels.

It is common to constrain the weak learners in specific ways, such as a maximum number of layers, nodes, splits or leaf nodes.

This is to ensure that the learners remain weak, but can still be constructed in a greedy manner.

3. **Additive Model**

Trees are added one at a time, and existing trees in the model are not changed.

A gradient descent procedure is used to minimize the loss when adding trees.

Traditionally, gradient descent is used to minimize a set of parameters, such as the coefficients in a regression equation or weights in a neural network. After calculating error or loss, the weights are updated to minimize that error.

Instead of parameters, we have weak learner sub-models or more specifically decision trees. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss.

## Why with Trees

1. Decision trees are non-linear. Boosting with linear models simply doesn’t work well.

2. The weak learner needs to be consistently better than random guessing. You don’t normally need to do any parameter tuning to a decision tree to get that behavior. Training an SVM, for instance, really does need a parameter search. Since the data is re-weighted on each iteration, you likely need to do another parameter search on each iteration. So you are increasing the amount of work you have to do by a large margin.

3. Decision trees are reasonably fast to train. Since we are going to be building 100's or 1000's of them, that’s a good property. They are also fast to classify, which is again important when you need 100's or 1000's to run before you can output your decision.

4. By changing the depth you have a simple and easy control over the bias/variance trade off, knowing that boosting can reduce bias but also significantly reduces variance The algorithm selection is also based on type of target variables

# Stacking

Stackingworks in two phases. First, we use multiple base classifiers to predict the class. Second, a new learner is used to combine their predictions with the aim of reducing the generalization error.

# Interview Questions

## What is the difference between AdaBoost, and Gradient Boosting?

AdaBoost is the shortcut for adaptive boosting. So what’s the differences between Adaptive boosting and Gradient boosting?

Both are boosting algorithms which means that they convert a set of weak learners into a single strong learner. They both initialize a strong learner (usually a decision tree) and iteratively create a weak learner that is added to the strong learner. They differ on **how they create the weak learners**during the iterative process.

At each iteration, adaptive boosting **changes the sample distribution** by modifying the weights attached to each of the instances. It increases the weights of the wrongly predicted instances and decreases the ones of the correctly predicted instances. The weak learner thus focuses more on the difficult instances. After being trained, the weak learner is added to the strong one **according to his performance** (so-called alpha weight). The higher it performs, the more it contributes to the strong learner.

On the other hand, gradient boosting doesn’t modify the sample distribution. Instead of training on a newly sample distribution, the weak learner **trains on the remaining errors**(so-called pseudo-residuals) of the strong learner. It is another way to give more importance to the difficult instances. At each iteration, the pseudo-residuals are computed and a weak learner is fitted to these pseudo-residuals. Then, the contribution of the weak learner (so-called multiplier) to the strong one isn’t computed according to his performance on the newly distribution sample but using a **gradient descent optimization process**. The computed contribution is the one minimizing the overall error of the strong learner.

## What is the difference between Random forest and Bagging

## What are the differences between Random Forest and Gradient Tree Boosting algorithms?

The main drawback of using decision tree directly with bagging is that the split choices at the top level of tree are statistically likely to remain approximately invariant to the bootstrapped sampling.Therefore the trees are more correlated which reduces the amount of error reduction obtained from bagging.

A Random Forest is using ensembling in its more intuitive fashion, essentially using the principle of “wisdom of the crowds”. A RF will build N decision trees and then average the predictions democratically. Each tree counts for one vote. Each tree uses a different sample from the original data thus introducing randomization.

A Gradient Boosting will take a different approach. It will start with a (usually) not very deep tree (sometimes a decision stump - a decision tree with only one split) and will model the original target. Then it takes the errors from the first round of predictions, and passes the errors as a new target to a second tree. The second tree will model the error from the first tree, record the new errors and pass that as a target to the third tree. And so forth. Essentially it focuses on modelling errors from previous trees. GB is one of the best algorithms available today and it’s almost always outperforming RF on most datasets I’ve tried.

## How to avoid overfitting in random forest

To avoid over-fitting in random forest, the main thing you need to do is optimize a tuning parameter that governs the number of features that are randomly chosen to grow each tree from the bootstrapped data.

Typically, you do this via k-fold cross-validation, where k∈{5,10}k∈{5,10}, and choose the tuning parameter that minimizes test sample prediction error. In addition, growing a larger forest will improve predictive accuracy, although there are usually diminishing returns once you get up to several hundreds of trees

## What is the difference between Bagging and Boosting

**Bagging** derives its name from **B**ootStrap Algorithm with **Agg**regation. First we will look into what actually a BootStrap Algorithm is.

Bootstrap allows us to draw random sample from our given dataset with replacement. Now assume **μ**is mean of our population.Now if I take too many random samples from the population , 95% of the confidence intervals on those sample would include **μ.**

Now idea is to use every bootstrap sample for training our model and we note down the variances of these model outputs. Now if we aggregate all these variances , we will get the aggregated model output which as very less variance.(this is our goal). In other words If we take the variance of the average of these outputs, it will be lower than the average of the variances by a factor of 1/N.

## Hoe do you interpret Random Forest

## What is the main idea behind ensemble learning? If I had many different models that predicted the same response variable, what might I want to do to incorporate all of the models? Would you expect this to perform better than an individual model or worse?

* The assumption is that a group of weak learners can be combined to form a strong learner.
* Hence the combined model is expected to perform better than an individual model.
* Assumptions:
  + average out biases
  + reduce variance
* Bagging works because some underlying learning algorithms are unstable: slightly different inputs leads to very different outputs. If you can take advantage of this instability by running multiple instances, it can be shown that the reduced instability leads to lower error. If you want to understand why, the original bagging paper( [http://www.springerlink.com/cont...](http://www.springerlink.com/content/l4780124w2874025/)) has a section called "why bagging works"
* Boosting works because of the focus on better defining the "decision edge". By reweighting examples near the margin (the positive and negative examples) you get a reduced error (see <http://citeseerx.ist.psu.edu/vie>...)
* Use the outputs of your models as inputs to a meta-model.

For example, if you're doing binary classification, you can use all the probability outputs of your individual models as inputs to a final logistic regression (or any model, really) that can combine the probability estimates.

One very important point is to make sure that the output of your models are out-of-sample predictions. This means that the predicted value for any row in your dataframe should NOT depend on the actual value for that row.

## When to use ensemble 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.

## When not to use ensemble model

I do not recommend using an ensemble classifier when your model needs to be interpretable and explainable. Sometimes you need predictions **and** explanations of the predictions.

## Do you think 50 small decision trees are better than a large one? Why?

## Q11. After spending several hours, you are now anxious to build a high accuracy model. As a result, you build 5 GBM models, thinking a boosting algorithm would do the magic. Unfortunately, neither of models could perform better than benchmark score. Finally, you decided to combine those models. Though, ensembled models are known to return high accuracy, but you are unfortunate. Where did you miss?

**Answer:** As we know, ensemble learners are based on the idea of combining weak learners to create strong learners. But, these learners provide superior result when the combined models are uncorrelated. Since, we have used 5 GBM models and got no accuracy improvement, suggests that the models are correlated. The problem with correlated models is, all the models provide same information.For example: If model 1 has classified User1122 as 1, there are high chances model 2 and model 3 would have done the same, even if its actual value is 0. Therefore, ensemble learners are built on the premise of combining weak uncorrelated models to obtain better predictions.

## Q21. Both being tree based algorithm, how is random forest different from Gradient boosting algorithm (GBM)?

**Answer:** The fundamental difference is, random forest uses bagging technique to make predictions. GBM uses boosting techniques to make predictions.

In bagging technique, a data set is divided into n samples using randomized sampling. Then, using a single learning algorithm a model is build on all samples. Later, the resultant predictions are combined using voting or averaging. Bagging is done is parallel. In boosting, after the first round of predictions, the algorithm weighs misclassified predictions higher, such that they can be corrected in the succeeding round. This sequential process of giving higher weights to misclassified predictions continue until a stopping criterion is reached.

Random forest improves model accuracy by reducing variance (mainly). The trees grown are uncorrelated to maximize the decrease in variance. On the other hand, GBM improves accuracy my reducing both bias and variance in a model.

## Q22. Running a binary classification tree algorithm is the easy part. Do you know how does a tree splitting takes place i.e. how does the tree decide which variable to split at the root node and succeeding nodes?

**Answer:** A classification trees makes decision based on Gini Index and Node Entropy. In simple words, the tree algorithm find the best possible feature which can divide the data set into purest possible children nodes.

Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure. We can calculate Gini as following:

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Entropy is the measure of impurity as given by (for binary class):

Entropy, Decision Tree

Here p and q is probability of success and failure respectively in that node. Entropy is zero when a node is homogeneous. It is maximum when a both the classes are present in a node at 50% – 50%.  Lower entropy is desirable.

## Q23. You’ve built a random forest model with 10000 trees. You got delighted after getting training error as 0.00. But, the validation error is 34.23. What is going on? Haven’t you trained your model perfectly?

**Answer:** The model has overfitted. Training error 0.00 means the classifier has mimiced the training data patterns to an extent, that they are not available in the unseen data. Hence, when this classifier was run on unseen sample, it couldn’t find those patterns and returned prediction with higher error. In random forest, it happens when we use larger number of trees than necessary. Hence, to avoid these situation, we should tune number of trees using cross validation.