ASSIGNMENT – 5

MACHINE LEARNING

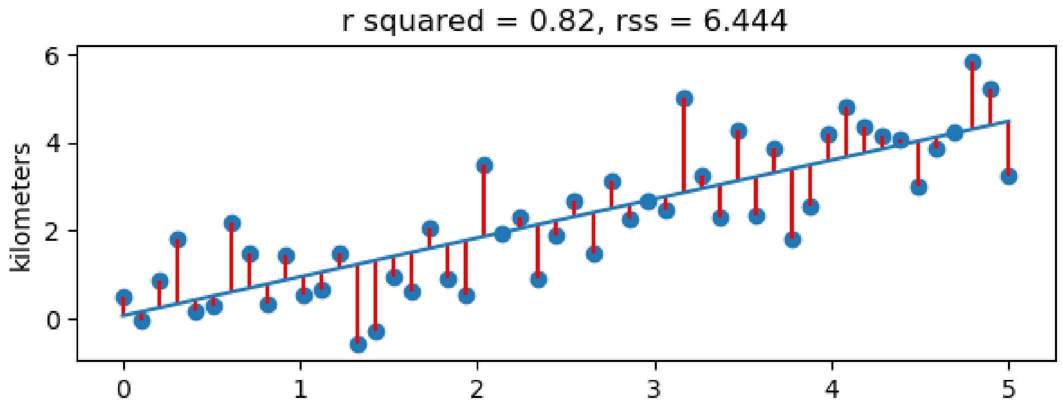
1. R-squared or Residual Sum of Squares (RSS) which one of these two is a better measure of goodness of fit model in regression and why?

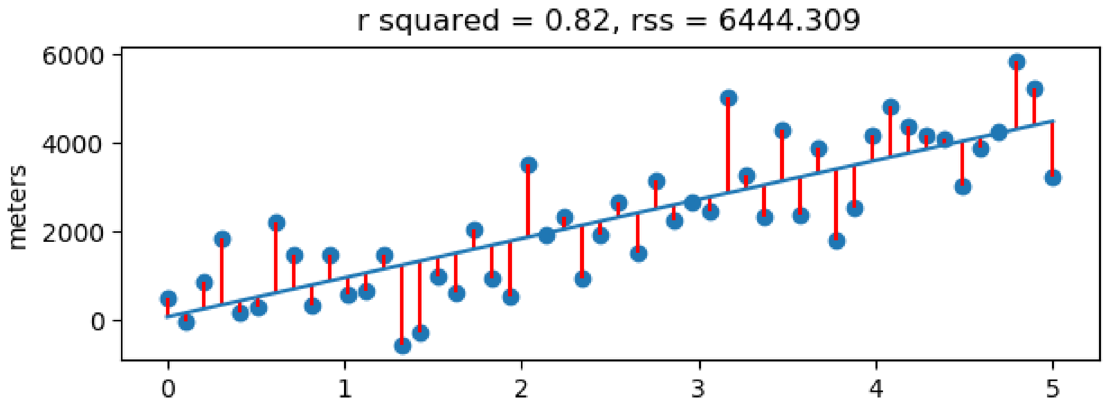
Sol-:

R-squared: is a goodness-of-fit measure for linear regression models. This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. R-squared measures the strength of the relationship between model and the dependent variable on a convenient 0 – 100% scale

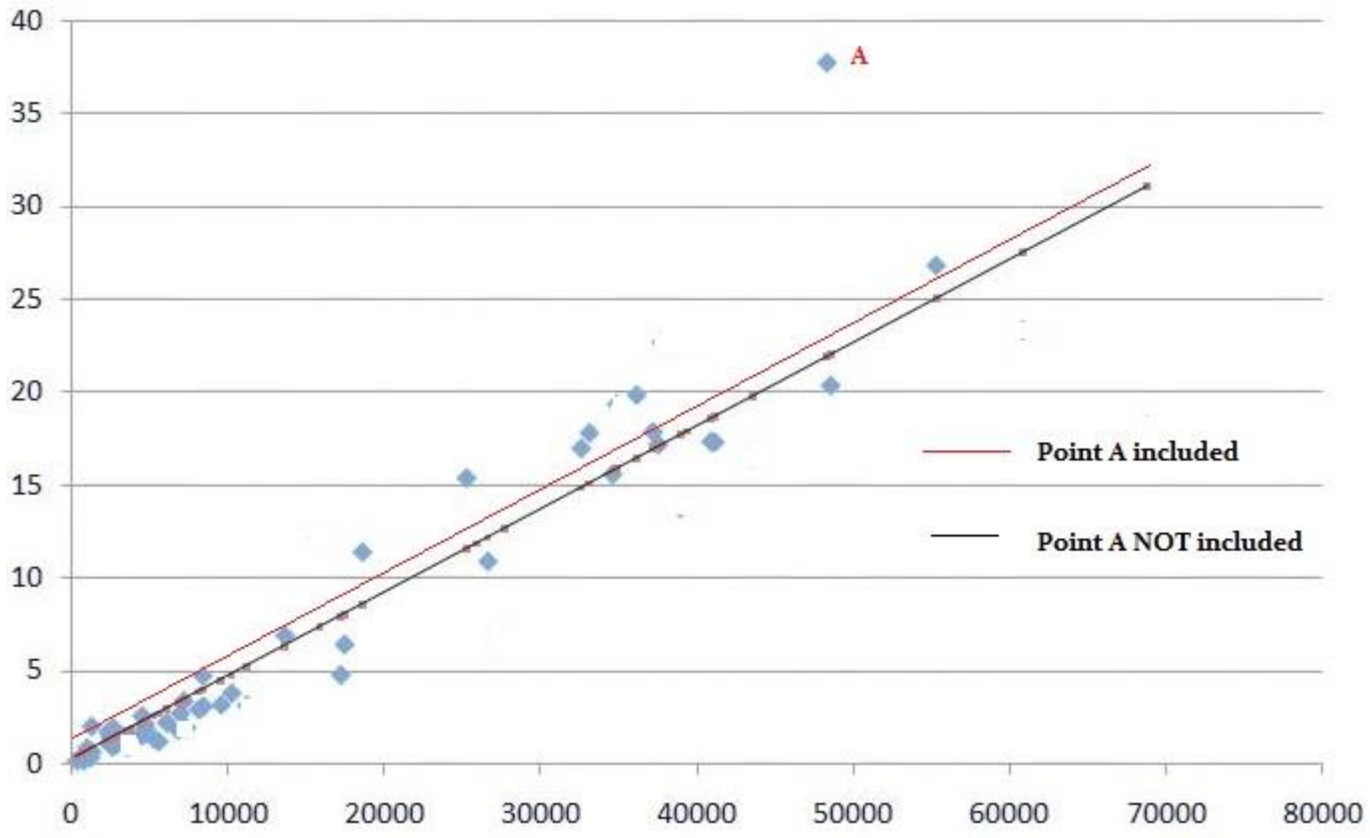
The residual sum of squares (RSS) is the sum of the squared distances between actual versus and predicted values.

RSS=∑i=1n(yi−y^i)2





residuals as a vertical line connecting actual values to our predicted value (red traces in the plot below). we can imagine that if y-axis is on a different scale, the number get will be very different
For instance, consider that your y-axis were kilometers, and a given point is about 0.5km away from your line of best fit. Then, the residual on that given datapoint is 0.5. However, if your scale is meters, then that same datapoint has a residual of 500. Your RSS will be much larger, but the fit does not change at all; in fact the data don't change at all either. But the RSS changes drastically.



it is possible to get a negative R-square for equations that do not contain a constant term. Because R-square is defined as the proportion of variance explained by the fit, if the fit is actually worse than just fitting a horizontal line then R-square is negative. In this case, R-square cannot be interpreted as the square of a correlation. Such situations indicate that a constant term should be added to the model.

2. What are TSS (Total Sum of Squares), ESS (Explained Sum of Squares) and RSS (Residual Sum of Squares) in regression. Also mention the equation relating these three metrics with each other.

Sol-:
TSS- Total Sum of Squares

The total sum of squares is a variation of the values of a dependent variable from the sample mean of the dependent variable. Essentially, the total sum of squares quantifies the total variation in a sample. It can be determined using the following formula:

$$TSS = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

Where:
yi – the value in a sample
ȳ – the mean value of a sample

SSE - Sum of Squares explained

The residual sum of squares essentially measures the variation of modeling errors. In other words, it depicts how the variation in the dependent variable in a regression model cannot be explained by the model. Generally, a lower residual sum of squares indicates that the regression model can better explain the data, while a higher residual sum of squares indicates that the model poorly explains the data.
The residual sum of squares can be found using the formula below:

$$SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:
yi – the observed value
ŷi – the value estimated by the regression line

SSR - Sum of Squares in regression

The regression sum of squares describes how well a regression model represents the modeled data. A higher regression sum of squares indicates that the model does not fit the data well.
The formula for calculating the regression sum of squares is:

$$\mathrm{SSR} = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$$

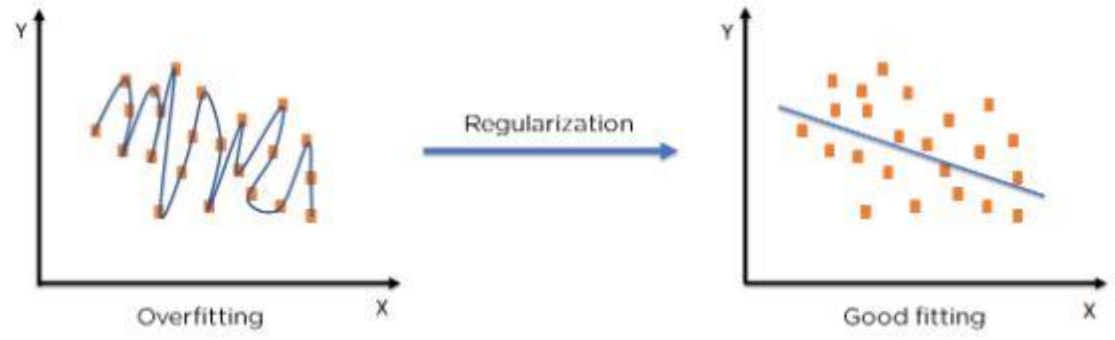
Where:
$\hat{y}_i$ – the value estimated by the regression line
$\bar{y}$ – the mean value of a sample
The relationship between the three types of sum of squares can be summarized by the following equation

$$\mathrm{TSS} = \mathrm{SSR} + \mathrm{SSE}$$

3. What is the need of regularization in machine learning

Sol-:
One of the major aspects of training machine learning model is avoiding overfitting. The model will have a low accuracy if it is overfitting. This happens because y model is trying too hard capture the noise in our training dataset. By noise we mean the data points that don't really represent the true properties of our data, but random chance. Learning such data points, makes our model more flexible, at the risk of overfitting.
A simple relation for linear regression looks like this. Here Y represents the learned relation and β represents the coefficient estimates for different variables or predictors(X)

$Y \approx \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta pXp$
Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.



Ridge Regression

Ridge regression, where the RSS is modified by adding the shrinkage quantity. Now, the coefficients are estimated by minimizing this function. Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \text{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

When $\lambda = 0$, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares. However, as $\lambda \to \infty$, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.

Lasso

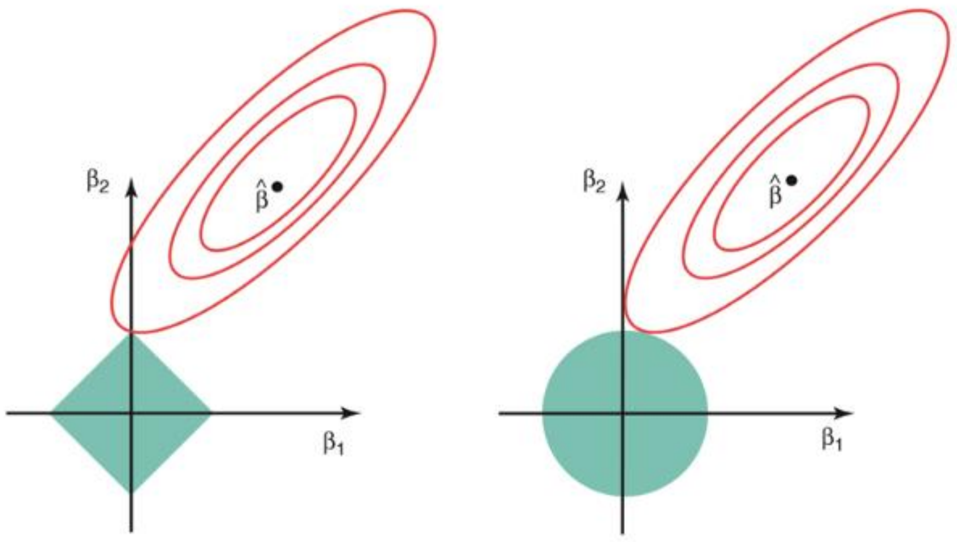
Lasso is another variation, in which the above function is minimized. Its clear that this variation differs from ridge regression only in penalizing the high coefficients. It uses $|\beta j|$(modulus)instead of squares of β, as its penalty. In statistics, this is known as the L1 norm.
Lets take a look at above methods with a different perspective. The ridge regression can be thought of as solving an equation, where summation of squares of coefficients is less than or equal to s. And the Lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to s. Here, s is a constant that exists for each value of

shrinkage factor λ. These equations are also referred to as constraint functions.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

The Lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to s. Here, s is a constant that exists for each value of shrinkage factor λ. These equations are also referred to as constraint functions.

The image below describes these equations.



4. What is Gini–impurity index?

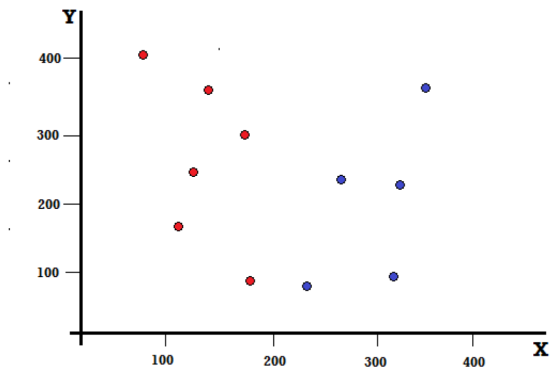
Gini Impurity is a measurement used to build Decision Trees to determine how the features of a dataset should split nodes to form the tree. More precisely, the Gini Impurity of a dataset is a number between 0-0.5, which indicates the likelihood of new, random data being misclassified if it were given a random class label according to the class distribution in the dataset.

The Gini Index or Gini Impurity is calculated by subtracting the sum of the squared probabilities of each class from one. It favours mostly the larger partitions and are very simple to implement. In simple terms, it calculates the probability of a certain randomly selected feature that was classified incorrectly.
The Gini Index varies between 0 and 1, where 0 represents purity of the classification and 1 denotes random distribution of elements among various classes. A Gini Index of 0.5 shows that there is equal distribution of elements across some classes.

The Gini Index is represented by

$$G = \sum_{i=1}^{C} p(i) * (1 - p(i))$$

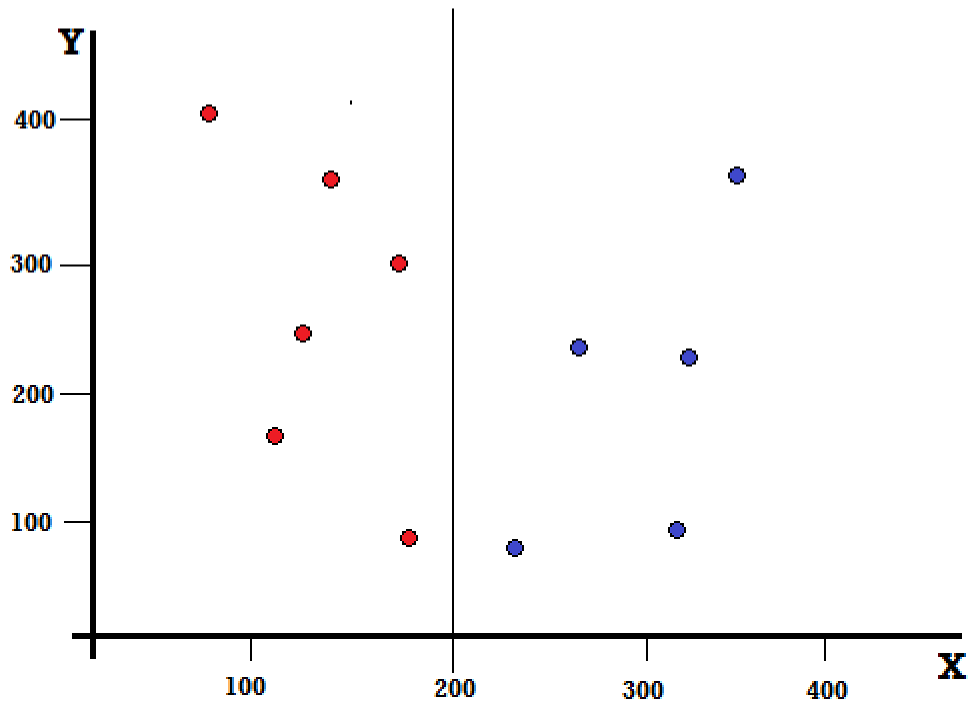
The calculation of the Gini Index with a simple example. In this, we have a total of 10 data points with two variables, the reds and the blues. The X and Y axes are numbered with spaces of 100 between each term. From the given example, we shall calculate the Gini Index and the Gini Gain.



Where, C is the total number of classes and p(i) is the probability of picking the data point with the class i.

For a decision tree, we need to split the dataset into two branches. Consider the following data points with 5 Reds and 5 Blues marked on the X-Y plane.

Suppose we make a binary split at X=200, then we will have a perfect split as shown below.



We are left with two branches, the left branch consisting of 5 reds and 1 blue, while the right branch consists of 4 blues. The following is referred to as an imperfect split. In training the Decision Tree model, to quantify the amount of imperfectness of the split, we can use the Gini Index.

calculate the Gini Impurity, let us first understand it's basic mechanism.
First, we shall randomly pick up any data point from the dataset Then, we will classify it randomly according to the class distribution in the given dataset. In our dataset, we shall give a data point chosen with a probability of 5/10 for red and 5/10 for blue as there are five data points of each colour and hence the probability.

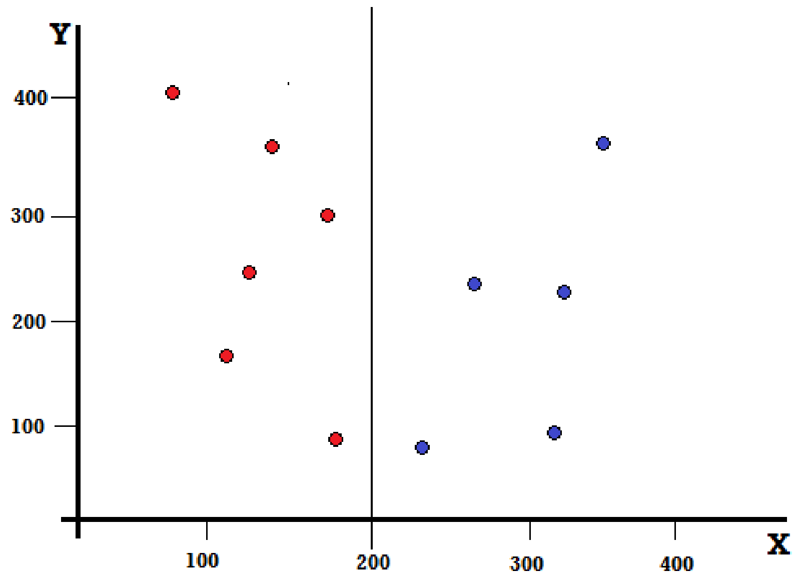
Now, in order to calculate the Gini Index, the formula is given by

Where, C is the total number of classes and p(i) is the probability of picking the data point with the class i.
In the above example, we have C=2 and p(1) = p(2) = 0.5, Hence the Gini Index can be calculated as,

$$G = p(1) * (1-p(1)) + p(2) * (1-p(2))$$
$$= 0.5 * (1-0.5) + 0.5 * (1-0.5)$$
$$= 0.5$$

Where 0.5 is the total probability of classifying a data point imperfectly and hence is exactly 50%.
Now, let us calculate the Gini Impurity for both the perfect and imperfect split that we performed earlier

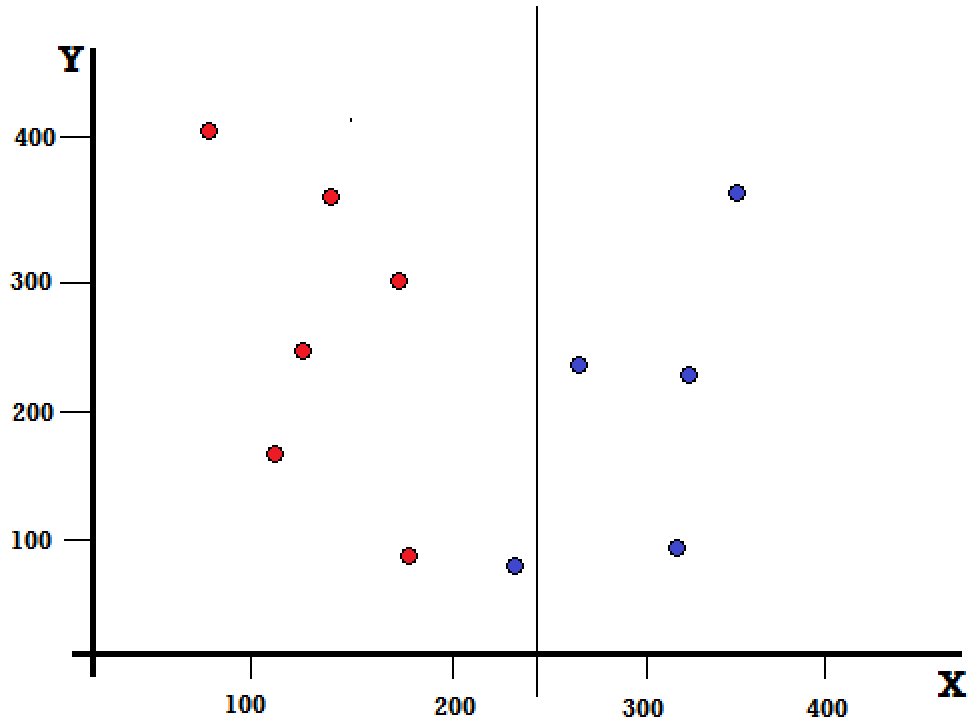
Perfect Split



The left branch has only reds and hence its Gini Impurity is,

G(left) =1 ∗ (1−1) + 0 ∗ (1−0) = 0
The right branch also has only blues and hence its Gini Impurity is also given by,
G(right) =1 ∗ (1−1) + 0 ∗ (1−0) = 0
From the quick calculation, we see that both the left and right branches of our perfect split have probabilities of 0 and hence is indeed perfect. A Gini Impurity of 0 is the lowest and the best possible impurity for any data set.

Imperfect Split



In this case, the left branch has 5 reds and 1 blue. Its Gini Impurity can be given by,
G(left) =1/6 ∗ (1−1/6) + 5/6 ∗ (1−5/6) = 0.278
The right branch has all blues and hence as calculated above its Gini Impurity is given by,

$G(\text{right}) = 1 * (1-1) + 0 * (1-0) = 0$

Now that we have the Gini Impurities of the imperfect split, in order to evaluate the quality or extent of the split, we will give a specific weight to the impurity of each branch with the number of elements it has.

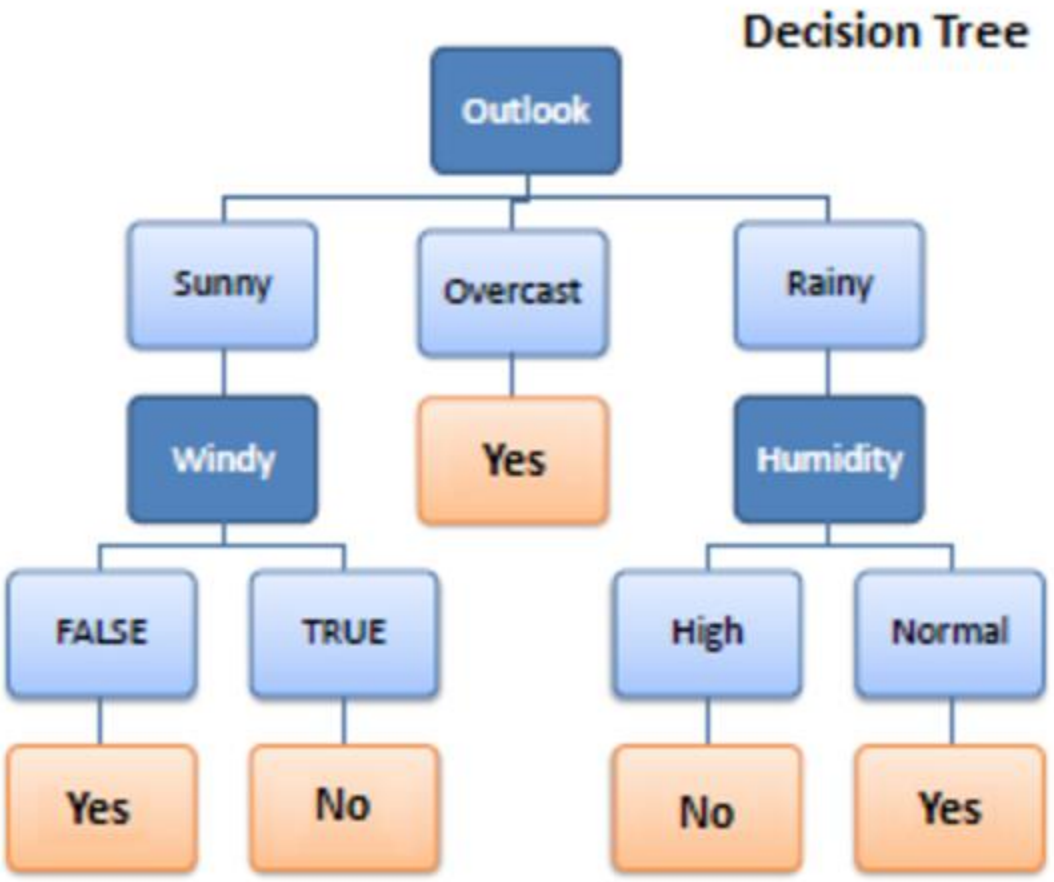
$(0.6*0.278) + (0.4*0) = 0.167$

Now that we have calculated the Gini Index, we shall calculate the value of another parameter, Gini Gain and analyse its application in Decision Trees. The amount of impurity removed with this split is calculated by deducting the above value with the Gini Index for the entire dataset (0.5).

5. Are unregularized decision-trees prone to overfitting? If yes, why?

Ans-

Decision trees are a type of model used for both classification and regression. Trees answer sequential questions which send us down a certain route of the tree given the answer. The model behaves with “if this than that” conditions ultimately yielding a specific result. This is easy to see with the image below which maps out whether or not to play golf.



The flow of this tree works downward beginning at the top with the outlook. The outlook has one of three options: sunny, overcast, or rainy. If sunny, we travel down to the next level. Will it be windy? True or false? If true, we choose not to play golf that day. If false we choose to play. If the outlook was changed to overcast, we would end there and decide to play. If the outlook was rainy, we would then look at the humidity. If the humidity was high we would not play, if the humidity is normal we would play.

Advantages to using decision trees:

1. Easy to interpret and make for straightforward visualizations.
2. The internal workings are capable of being observed and thus make it possible to reproduce work.
3. Can handle both numerical and categorical data.
4. Perform well on large datasets
5. Are extremely fast

Disadvantages of decision trees:

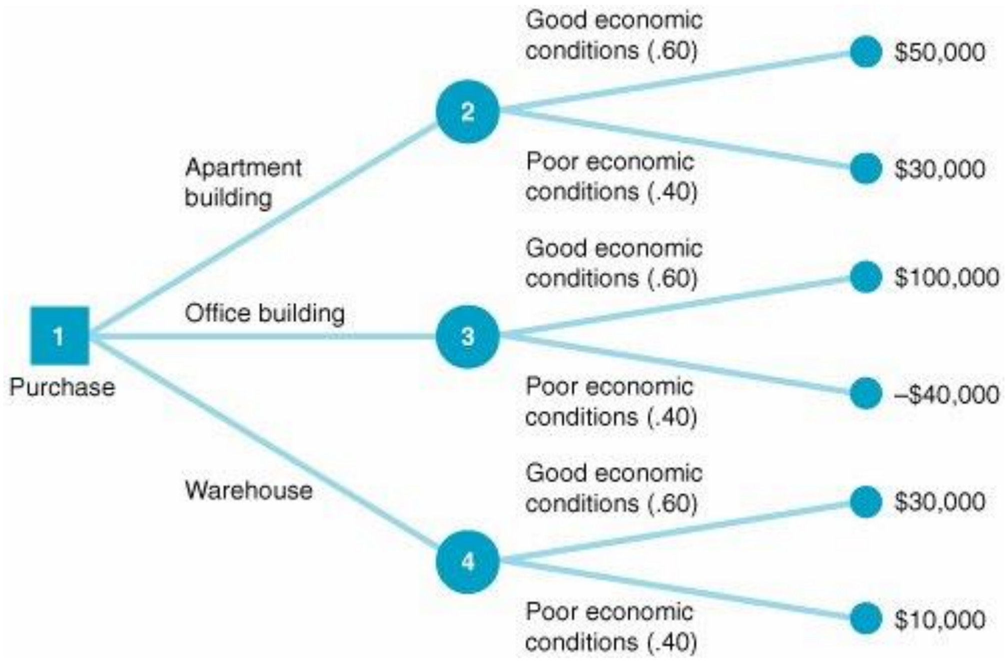
1. Building decision trees require algorithms capable of determining an optimal choice at each node. One popular algorithm is the Hunt's algorithm. This is a greedy model, meaning it makes the most optimal decision at each step, but does not take into account the global optimum. What does this mean? At each step the algorithm chooses the best result. However, choosing the best result at a given step does not ensure you will be headed down the route that will lead to the optimal decision when you make it to the final node of the tree, called the leaf node.

2. Decision trees are prone to overfitting, especially when a tree is particularly deep.
This is due to the amount of specificity we look at leading to smaller sample of events that meet the previous assumptions.
This small sample could lead to unsound conclusions. An example of this could be predicting if the Boston Celtics will beat the Miami Heat in tonight's basketball game.
The first level of the tree could ask if the Celtics are playing home or away.
The second level might ask if the Celtics have a higher win percentage than their opponent, in this case the Heat. The third level asks if the Celtic's leading scorer is playing? The fourth level asks if the Celtic's second leading scorer is playing. The fifth level asks if the Celtics are traveling back to the east coast from 3 or more consecutive road games on the west coast.
While all of these questions may be relevant, there may only be two previous games where the conditions of tonights game were met. Using only two games as the basis for our classification would not be adequate for an informed decision. One way to combat this issue is by setting a max depth.
This will limit our risk of overfitting; but as always, this will be at the expense of error due to bias. Thus if we set a max depth of three, we would only ask if the game is home or away, do the Celtics have a higher winning percentage than their opponent, and is their leading scorer playing. This is a simpler model with less variance sample to sample but ultimately will not be a strong predictive model.

Ideally, we would like to minimize both error due to bias and error due to variance. Enter random forests. Random forests mitigate this problem well. A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models.
One way Random Forests reduce variance is by training on different samples of the data. A second way is by using a random subset of features. This means if we have 30 features, random forests will only use a certain number of those features in each model, say five. Unfortunately, we have omitted 25 features that could be useful. But as stated, a random forest is a collection of decision trees. Thus, in each tree we can utilize five random features. If we use many trees in our forest, eventually many or all of our features will have been included. This inclusion of many features will help limit our error due to bias and error due to variance. If features weren't chosen randomly, base trees in our forest could become highly correlated. This is because a few features could be particularly predictive and thus, the same features would be chosen in many of the base trees. If many of these trees included the same features we would not be combating error due to variance.
With that said, random forests are a strong modeling technique and much more robust than a single decision tree. They aggregate many decision trees to limit overfitting as well as error due to bias and therefore yield useful results.

6. What is an ensemble technique in machine learning?
Ans-
Ensembling is nothing but the technique to combine several individual predictive models to come up with the final predictive model. And in this article, we're going to look at some of the ensembling techniques for both Classification and Regression problems such as Maximum voting, Averaging, Weighted Averaging, and Rank Averaging.

A Decision Tree determines the predictive value based on series of questions and conditions. For instance, this simple Decision Tree determining on whether an individual should play outside or not. The tree takes several weather factors into account, and given each factor either makes a decision or asks another question. In this example, every time it is overcast, we will play outside. However, if it is raining, we must ask if it is windy or not? If windy, we will not play. But given no wind, tie those shoelaces tight because were going outside to play.


1
Purchase
Apartment building
Office building
Warehouse
2
3
4
Good economic conditions (.60)
Poor economic conditions (.40)
Good economic conditions (.60)
Poor economic conditions (.40)
Good economic conditions (.60)
Poor economic conditions (.40)
$50,000
$30,000
$100,000
–$40,000
$30,000
$10,000

Types of Ensemble Methods

Bagging
Random Fores

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. It is important to note that Decision Trees are not the only form of ensemble methods, just the most popular and relevant in DataScience today.

7. What is the difference between Bagging and Boosting techniques?

Bagging

Bagging is an acronym for ‘Bootstrap Aggregation’ and is used to decrease the variance in the prediction model. Bagging is a parallel method that fits different, considered learners independently from each other, making it possible to train them simultaneously.
Bagging generates additional data for training from the dataset. This is achieved by random sampling with replacement from the original dataset. Sampling with replacement may repeat some observations in each new training data set. Every element in Bagging is equally probable for appearing in a new dataset.
These multi datasets are used to train multiple models in parallel. The average of all the predictions from different ensemble models is calculated. The majority vote gained from the voting mechanism is considered when classification is made.

Multiple subsets are prepared from the original data set with equal tuples. The observations with replacement are selected.
A base model is prepared on every subset.iii. Every model is learned in parallel with the training set. These models are independent of each other.
The final predictions are done by merging the predictions from all the models.

Example of Bagging:
The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

Consider X observations and Y features in the training data set.
Firstly, a model from the training data set is randomly chosen with substitution.
In this step, the tree is grown to the largest.
The above steps are repeated and the prediction is provided. The prediction depends on the collection of predictions from the ‘n’ number of trees.

Boosting

Boosting is a sequential ensemble method that iteratively adjusts the weight of observation as per the last classification. If an observation is incorrectly classified, it increases the weight of that observation. The term ‘Boosting’ in a layman language, refers to algorithms that convert a weak learner to a stronger one. It decreases the bias error and builds strong predictive models.
Data points mispredicted in each iteration are spotted, and their weights are increased. The Boosting algorithm allocates weights to each resulting model during training. A learner with good training data prediction results will be assigned a higher weight. When evaluating a new learner, Boosting keeps track of learner’s errors.
If a provided input is inappropriate, its weight is increased. The purpose behind this is that the forthcoming hypothesis is more likely to properly categorize it by combining the entire set, at last, to transform weak learners into superior performing models.
It involves several boosting algorithms. The original algorithms invented by Yoav Freund and Robert Schapire were not adaptive. They couldn’t make the most of the weak learners. These people then invented AdaBoost, which is an adaptive boosting algorithm. It received the esteemed Gödel Prize and was the first successful boosting algorithm created for binary classification. AdaBoost stands for Adaptive Boosting. It merges multiple “weak classifiers” into a “strong classifier”.
Gradient Boosting represents an extension of the boosting procedure. It equates to the combination of Gradient Descent and Boosting. It uses a gradient descent algorithm capable of optimizing any differentiable loss function. Its working involves the construction of an ensemble of trees, and individual trees are summed sequentially. The subsequent tree restores the loss (the difference between real and predicted values).

Bagging and Boosting: Differences

Bagging is a method of merging the same type of predictions. Boosting is a method of merging different types of predictions.

Bagging decreases variance, not bias, and solves over-fitting issues in a model. Boosting decreases bias, not variance.

In Bagging, each model receives an equal weight. In Boosting, models are weighed based on their performance.

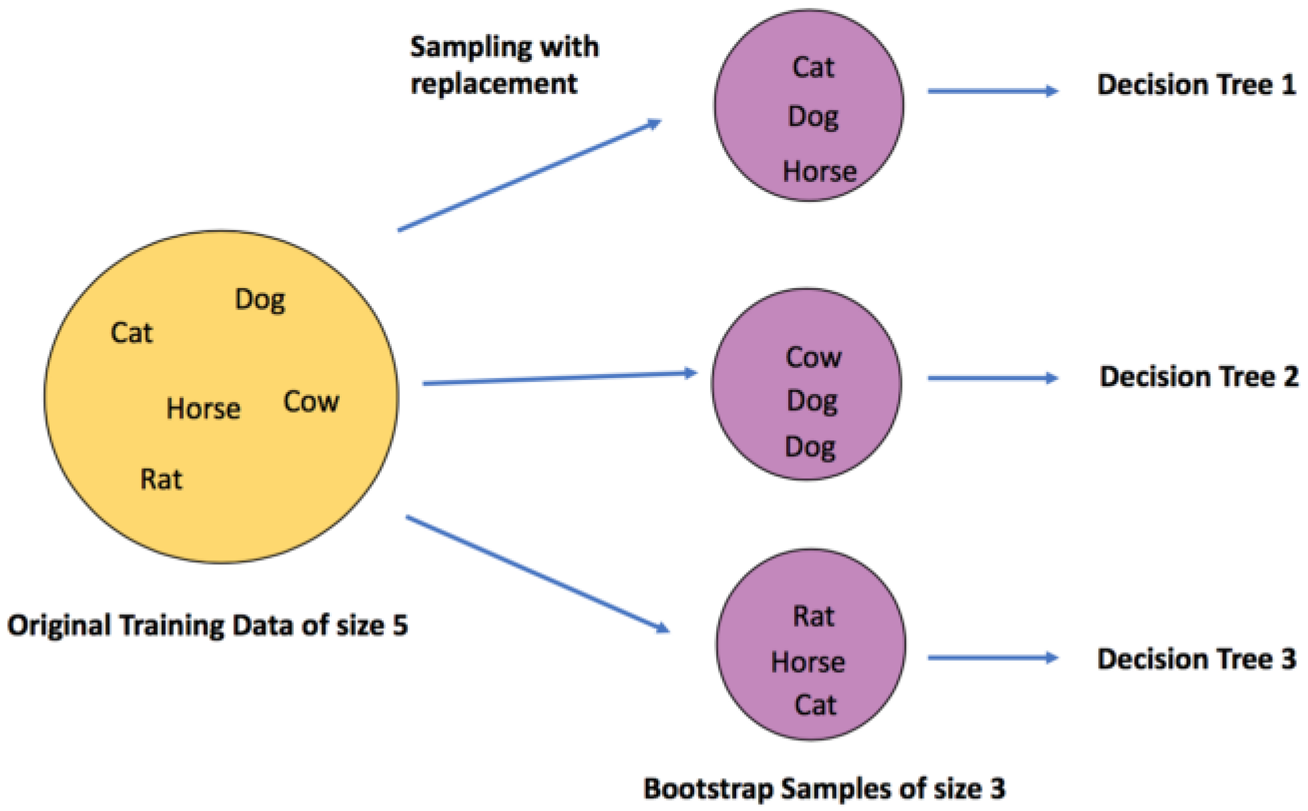
Models are built independently in Bagging. New models are affected by a previously built model's performance in Boosting.

In Bagging, training data subsets are drawn randomly with a replacement for the training dataset. In Boosting, every new subset comprises the elements that were misclassified by previous models.

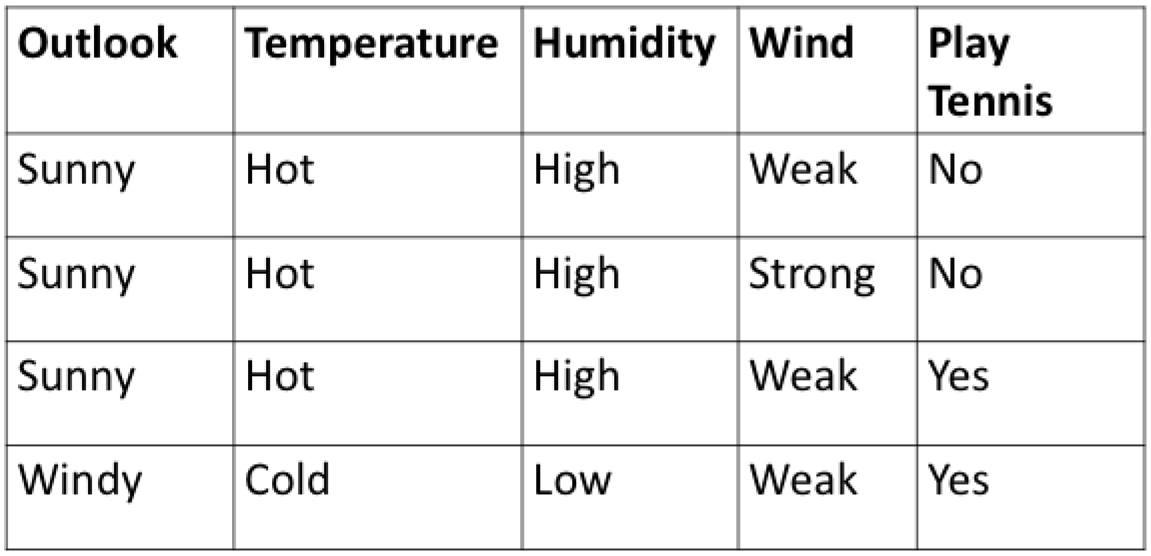
Bagging is usually applied where the classifier is unstable and has a high variance. Boosting is usually applied where the classifier is stable and simple and has high bias.

8 .What is out-of-bag error in random forests?

In the applications that require good interpretability of the model, DTs work very well especially if they are of small depth. However, DTs with real-world datasets can have large depths. Higher depth DTs are more prone to overfitting and thus lead to higher variance in the model. This shortcoming of DT is explored by the Random Forest model. In the Random Forest model, the original training data is randomly sampled-with-replacement generating small subsets of data (see the image below). These subsets are also known as bootstrap samples. These bootstrap samples are then fed as training data to many DTs of large depths. Each of these DTs is trained separately on these bootstrap samples. This aggregation of DTs is called the Random Forest ensemble. The concluding result of the ensemble model is determined by counting a majority vote from all the DTs. This concept is known as Bagging or Bootstrap Aggregation. Since each DT takes a different set of training data as input, the deviations in the original training dataset do not impact the final result obtained from the aggregation of DTs. Therefore, bagging as a concept reduces variance without changing the bias of the complete ensemble.

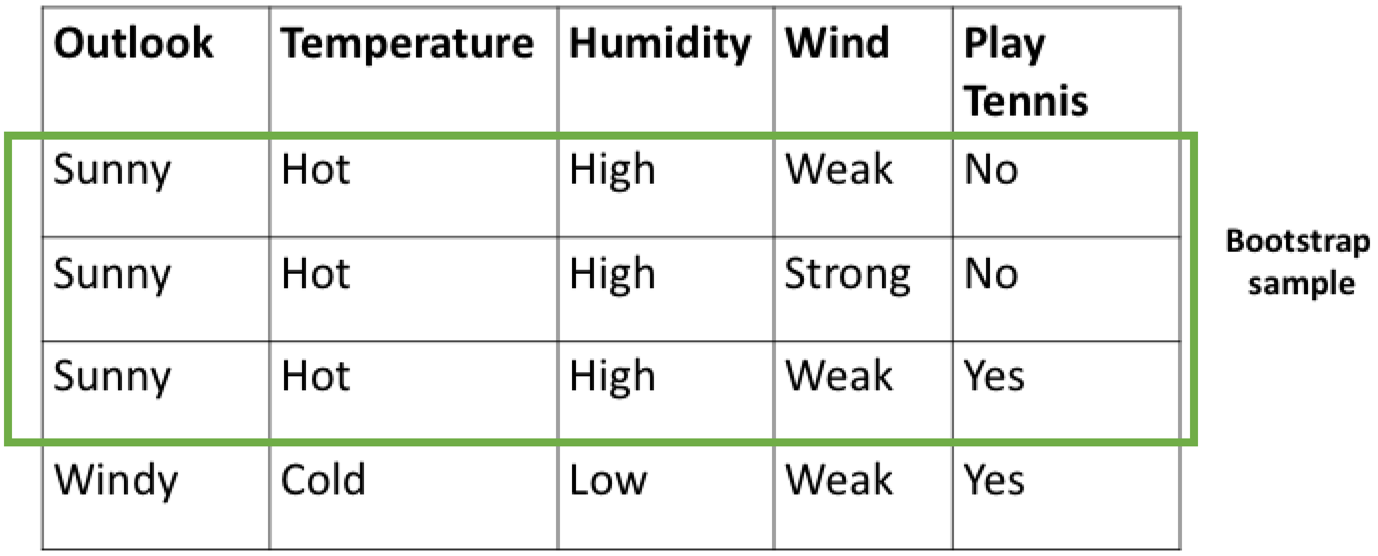


Out of bag (OOB) score is a way of validating the Random forest model. Below is a simple intuition of how is it calculated followed by a description of how it is different from validation score and where it is advantageous.
For the description of OOB score calculation, let’s assume there are five DTs in the random forest ensemble labeled from 1 to 5. For simplicity, suppose we have a simple original training data set as below.



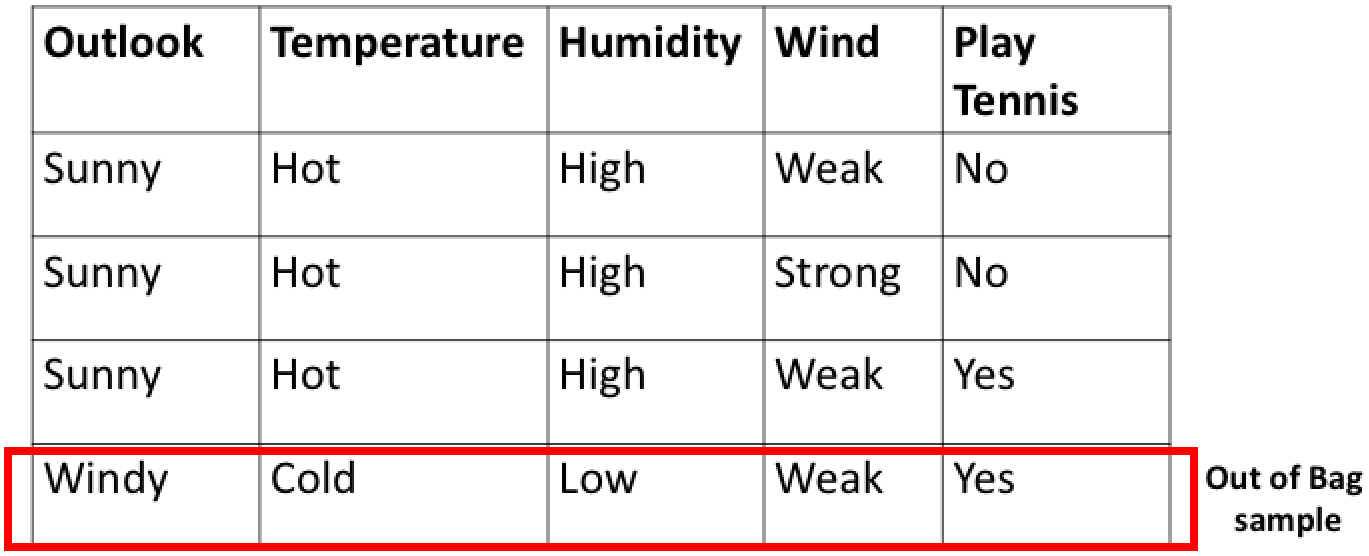
Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Hot	High	Weak	Yes
Windy	Cold	Low	Weak	Yes

Let the first bootstrap sample is made of the first three rows of this data set as shown in the green box below. This bootstrap sample will be used as the training data for the DT "1".

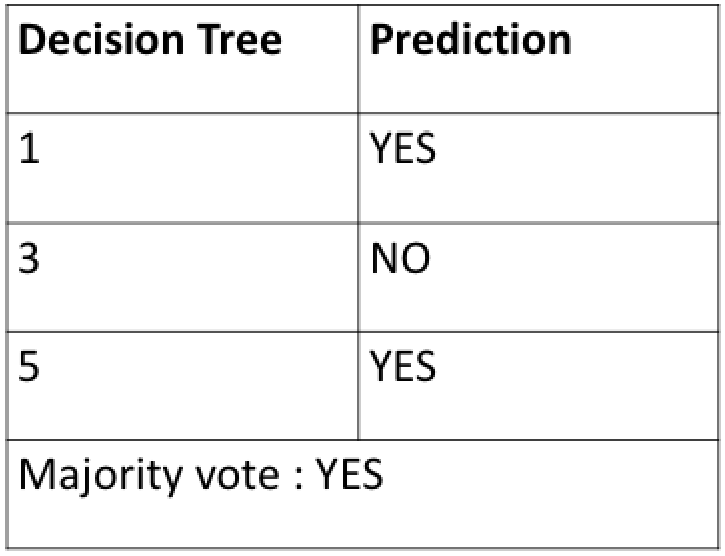

Bootstrap sample

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

Then the last row that is "left out" in the original data (see the red box in the image below) is known as Out of Bag sample. This row will not be used as the training data for DT 1. Please note that in reality there will be several such rows which are left out as Out of Bag, here for simplicity only one is shown.


Out of Bag sample

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



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

We see that 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 score is computed as the number of correctly predicted rows from the out of bag sample.

9 .What is K-fold cross-validation?

Ans-

Let’s say that you have trained a machine learning model. Now, you need to find out how well this model performs. Is it accurate enough to be used? How does it compare to another model? There are several evaluation methods to determine this. One such method is called K-fold cross validation.

Cross validation is an evaluation method used in machine learning to find out how well your machine learning model can predict the outcome of unseen data. It is a method that is easy to comprehend, works well for a limited data sample and also offers an evaluation that is less biased, making it a popular choice.

The data sample is split into ‘k’ number of smaller samples, hence the name: K-fold Cross Validation. You may also hear terms like four fold cross validation, or ten fold cross validation, which essentially means that the sample data is being split into four or ten smaller samples respectively.

k-fold cross validation performed

First, shuffle the dataset and split into k number of subsamples. (It is important to try to make the subsamples equal in size and ensure k is less than or equal to the number of elements in the dataset)
.
In the first iteration, the first subset is used as the test data while all the other subsets are considered as the training data.

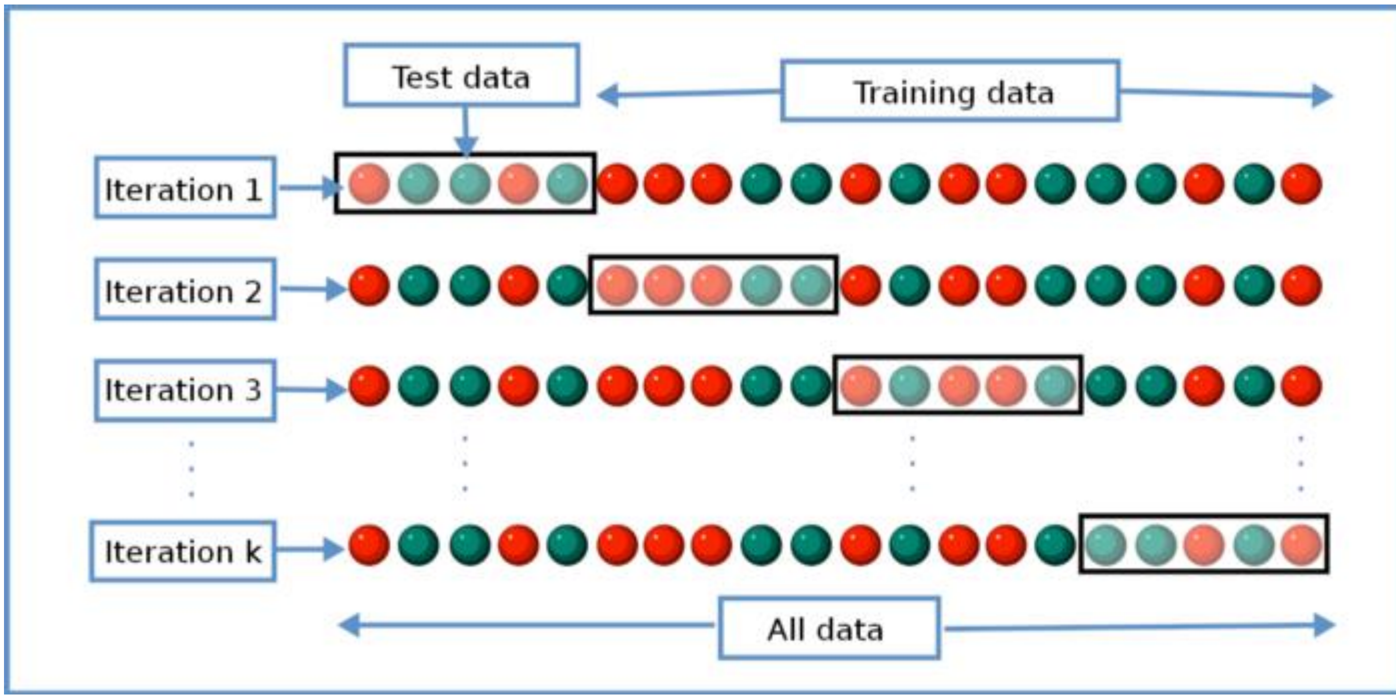
Train the model with the training data and evaluate it using the test subset. Keep the evaluation score or error rate, and get rid of the model.

Now, in the next iteration, select a different subset as the test data set, and make everything else (including the test set we used in the previous iteration) part of the training data

Re-train the model with the training data and test it using the new test data set, keep the evaluation score and discard the model.

Continue iterating the above k times. Each data subsamples will be used in each iteration until all data is considered. You will end up with a k number of evaluation scores.

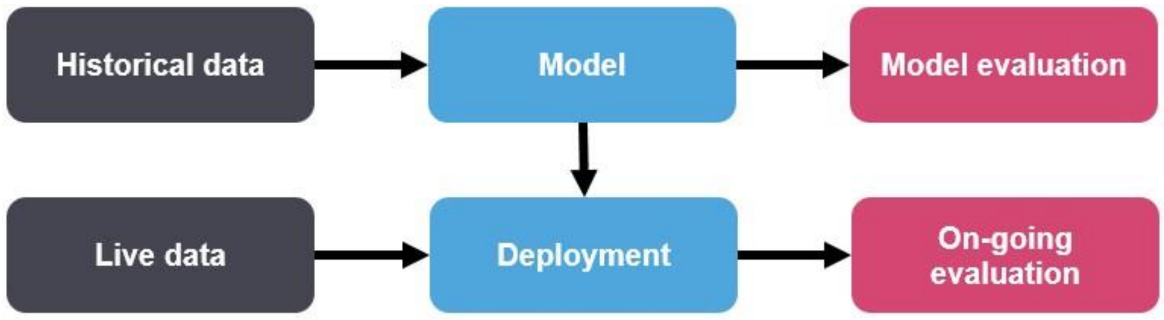
The total error rate is the average of all these individual evaluation scores.



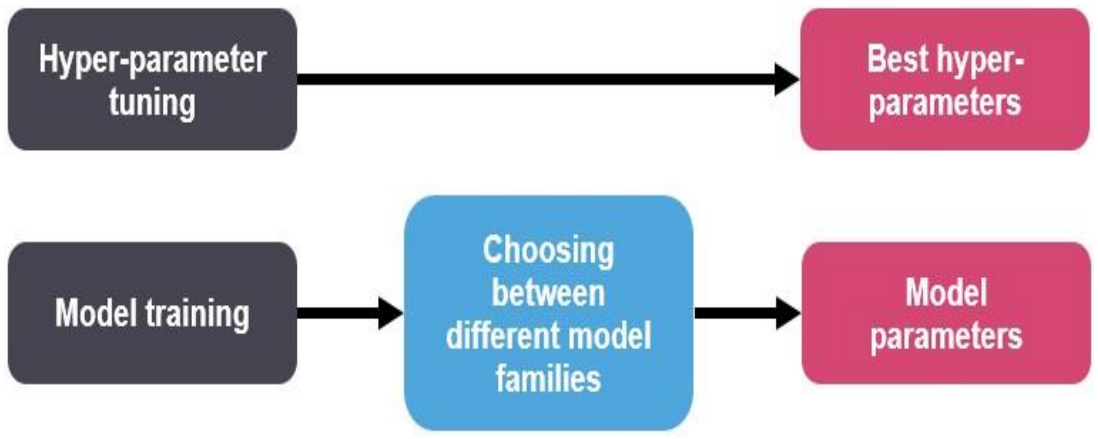
10. What is hyper parameter tuning in machine learning and why it is done?

Ans-

Hyperparameter tuning consists of finding a set of optimal hyperparameter values for a learning algorithm while applying this optimized algorithm to any data set. That combination of hyperparameters maximizes the model's performance, minimizing a predefined loss function to produce better results with fewer errors

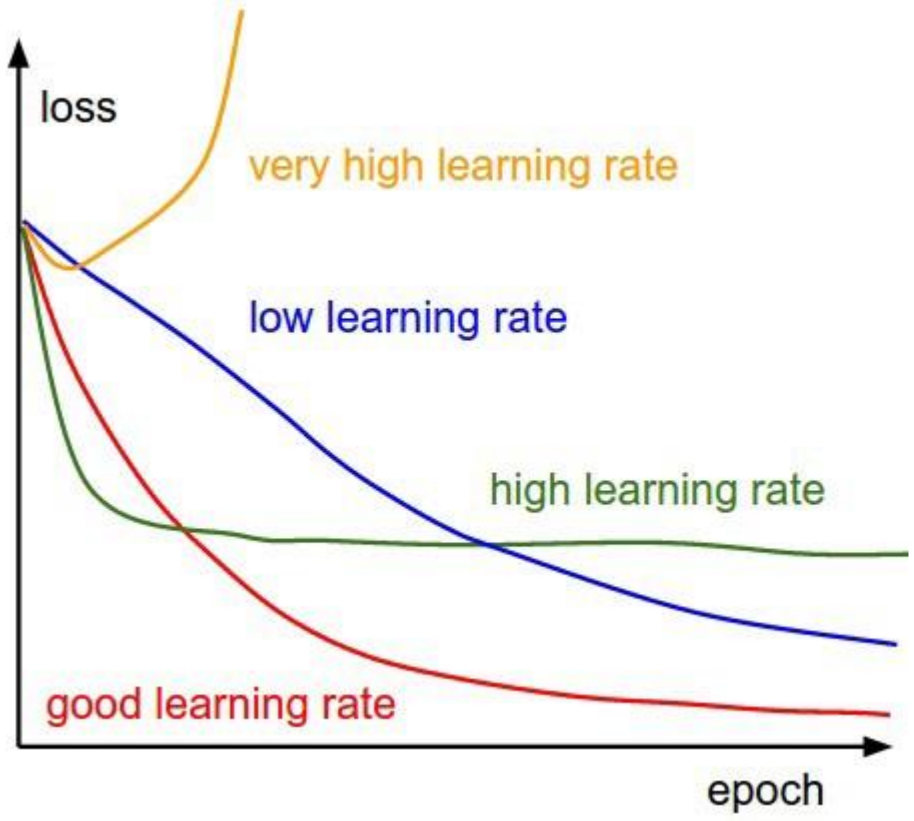


When creating a machine learning model, I 'll be presented with design choices as to how to define your model architecture. Often times, we don't immediately know what the optimal model architecture should be for a given model, and thus we'd like to be able to explore a range of possibilities. In true machine learning fashion, we'll ideally ask the machine to perform this exploration and select the optimal model architecture automatically. Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.



11.What issues can occur if we have a large learning rate in Gradient Descent?
Ans-

In order for Gradient Descent to work, we must set the learning rate to an appropriate value. This parameter determines how fast or slow we will move towards the optimal weights. If the learning rate is very large we will skip the optimal solution.



Learning rate (λ) is one such hyper-parameter that defines
the adjustment in the weights of our network with respect to the loss
gradient descent. It determines how fast or slow we will move
towards the optimal weights
The Gradient Descent Algorithm estimates the weights of the model
in many iterations by minimizing a cost function at every step.

In order for Gradient Descent to work, we must set the learning rate
to an appropriate value. This parameter determines how fast or slow
we will move towards the optimal weights. If the learning rate is very
large we will skip the optimal solution. If it is too small we will need
too many iterations to converge to the best values. So using a good
learning rate is crucial.
both low and high learning rates results in wasted time and resources

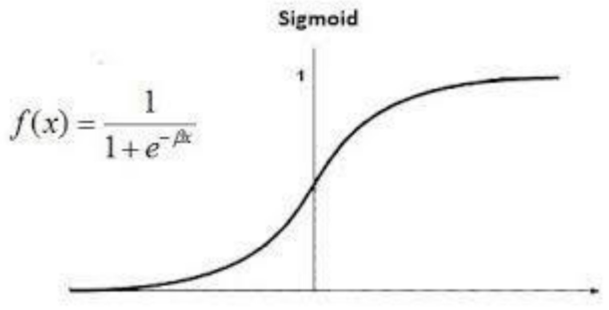
A lower learning rate means more training time

more time results in increased cloud GPU costs a higher rate could result in a model that might not be able to predict anything accurately.

12. Can we use Logistic Regression for classification of Non-Linear Data? If not, why?

Non-Linear Data- of classification

Logistic regression is a classification algorithm used to find the probability of event success and event failure. It is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature. It supports categorizing data into discrete classes by studying the relationship from a given set of labelled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function.



Logistic regression is easier to implement, interpret, and very efficient to train.
It makes no assumptions about distributions of classes in feature space.
It can easily extend to multiple classes(multinomial regression) and a natural probabilistic view of class predictions.
It not only provides a measure of how appropriate a predictor(coefficient size)is, but also its direction of association (positive or negative).
It is very fast at classifying unknown records.

Non linear data – why

It can only be used to predict discrete functions. Hence, the dependent variable of Logistic Regression is bound to the discrete number set.

Non-linear problems can’t be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Logistic Regression requires average or no multicollinearity between independent variables.

It is tough to obtain complex relationships using logistic regression. More powerful and compact algorithms such as Neural Networks can easily outperform this algorithm.
In Linear Regression independent and dependent variables are related linearly. But Logistic Regression needs that independent variables are linearly related to the log odds (log(p/(1-p)).

13. Differentiate between Adaboost and Gradient Boosting.

AdaBoost

AdaBoost or Adaptive Boosting is the first Boosting ensemble model. The method automatically adjusts its parameters to the data based on the actual performance in the current iteration. Meaning, both the weights for re-weighting the data and the weights for the final aggregation are re-computed iteratively.
In practice, this boosting technique is used with simple classification trees or stumps as base-learners, which resulted in improved performance compared to the classification by one tree or other single base-learner.

Gradient Boosting

Gradient Boost is a robust machine learning algorithm made up of Gradient descent and Boosting. The word ‘gradient’ implies that you can have two or more derivatives of the same function. Gradient Boosting has three main components: additive model, loss function and a weak learner.
The technique yields a direct interpretation of boosting methods from the perspective of numerical optimisation in a function space and

generalises them by allowing optimisation of an arbitrary loss function.

Features	Gradient boosting	Adaboost
Model	It identifies complex observations by huge residuals calculated in prior iterations	The shift is made by up-weighting the observations that are miscalculated prior
Trees	The trees with week learners are constructed using a greedy algorithm based on split points and purity scores. The trees are grown deeper with eight to thirty-two terminal nodes. The week learners should stay a week in terms of nodes, layers, leaf nodes, and splits	The trees are called decision stumps.
Classifier	The classifiers are weighted precisely and their prediction capacity is constrained to learning rate and increasing accuracy	Every classifier has different weight assumptions to its final prediction that depend on the performance.
Prediction	It develops a tree with help of previous classifier residuals by capturing variances in data. The final prediction depends on the maximum vote of the week learners and is weighted by its accuracy.	It gives values to classifiers by observing determined variance with data. Here all the week learners possess equal weight and it is usually fixed as the rate for learning which is too minimum in magnitude.
Short-comings	Here, the gradients themselves identify the shortcomings.	Maximum weighted data points are used to identify the shortcomings.

Loss value	Gradient boosting cut down the error components to provide clear explanations and its concepts are easier to adapt and understand	The exponential loss provides maximum weights for the samples which are fitted in worse conditions.
Applications	This method trains the learners and depends on reducing the loss functions of that week learner by training the residues of the model	Its focus on training the prior miscalculated observations and it alters the distribution of the dataset to enhance the weight on sample values which are hard for classification

14.What is bias-variance trade off in machine learning?

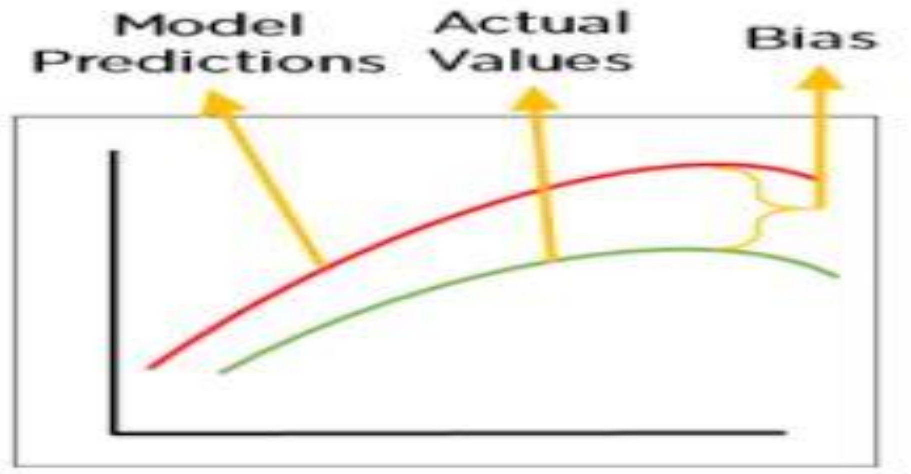
Ans-
In statistics and machine learning, the bias–variance tradeoff is the property of a model that the variance of the parameter estimated across samples can be reduced by increasing the bias in the estimated parameters.

While discussing model accuracy, we need to keep in mind the prediction errors, ie: Bias and Variance, that will always be associated with any machine learning model. There will always be a slight difference in what our model predicts and the actual predictions. These differences are called errors. The goal of an analyst is not to eliminate errors but to reduce them. There is always a tradeoff between how low you can get errors to be. In this article titled ‘Everything you need to know about Bias and Variance’, we will discuss what these errors are.

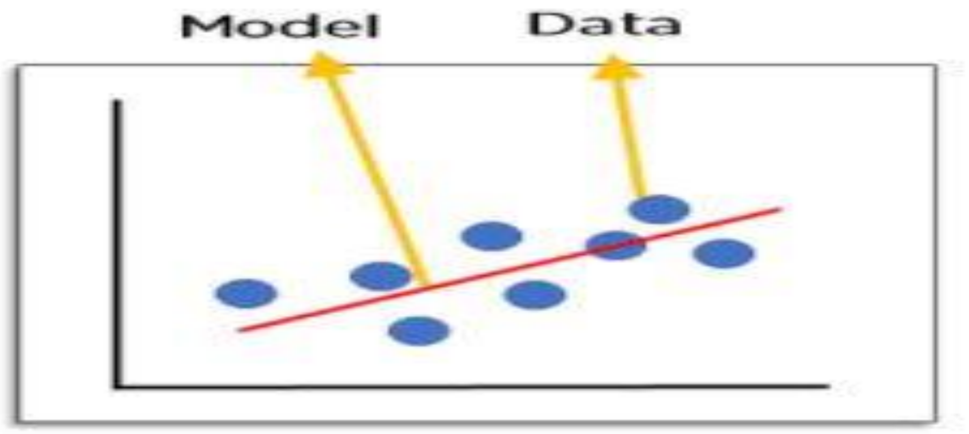
Bias

To make predictions, our model will analyze our data and find patterns in it. Using these patterns, we can make generalizations

about certain instances in our data. Our model after training learns these patterns and applies them to the test set to predict them.
Bias is the difference between our actual and predicted values. Bias is the simple assumptions that our model makes about our data to be able to predict new data.

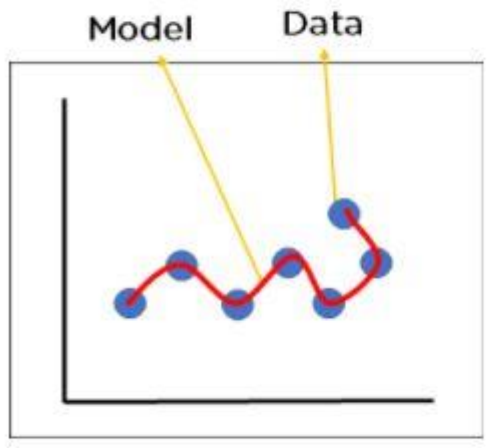


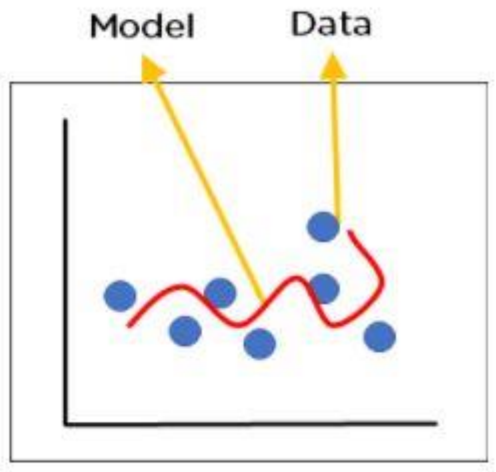
When the Bias is high, assumptions made by our model are too basic, the model can’t capture the important features of our data. This means that our model hasn’t captured patterns in the training data and hence cannot perform well on the testing data too. If this is the case, our model cannot perform on new data and cannot be sent into production.



Variance

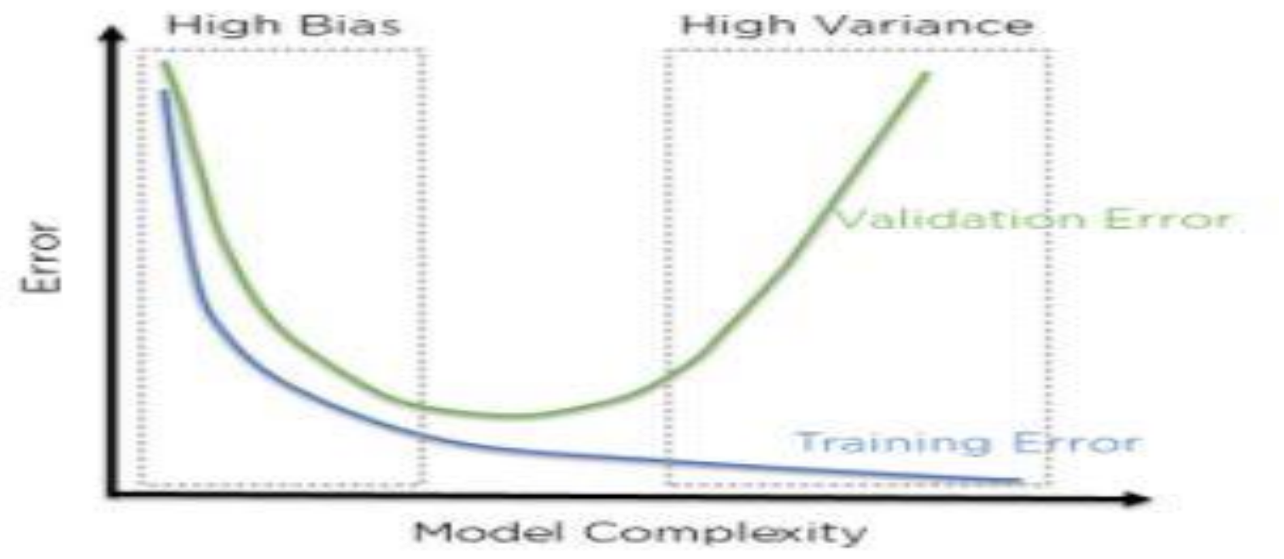
Variance is the very opposite of Bias. During training, it allows our model to ‘see’ the data a certain number of times to find patterns in it. If it does not work on the data for long enough, it will not find patterns and bias occurs. On the other hand, if our model is allowed to view the data too many times, it will learn very well for only that data. It will capture most patterns in the data, but it will also learn from the unnecessary data present, or from the noise.
We can define variance as the model’s sensitivity to fluctuations in the data. Our model may learn from noise. This will cause our model to consider trivial features as important.





Bias-Variance Tradeoff

For any model, we have to find the perfect balance between Bias and Variance. This just ensures that we capture the essential patterns in our model while ignoring the noise present it in. This is called Bias-Variance Tradeoff. It helps optimize the error in our model and keeps it as low as possible.
An optimized model will be sensitive to the patterns in our data, but at the same time will be able to generalize to new data. In this, both the bias and variance should be low so as to prevent overfitting and underfitting



15. Give short description each of Linear, RBF, Polynomial kernels used in SVM.

SVM Kernel

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid.
Introduce Kernel functions for sequence data, graphs, text, images, as well as vectors. The most used type of kernel function is RBF. Because it has localized and finite response along the entire x-axis.
The kernel functions return the inner product between two points in a suitable feature space. Thus by defining a notion of similarity, with little computational cost even in very high-dimensional spaces.

$$K\left(\overline{x}\right) = \begin{matrix} 1 & \text{if} \left\|\overline{x}\right\| \leq 1 \\ 0 & \text{otherwise} \end{matrix}$$

Linear Kernel

It is the most basic type of kernel, usually one dimensional in nature. It proves to be the best function when there are lots of features. The linear kernel is mostly preferred for text-classification problems as most of these kinds of classification problems can be linearly separated.
Linear kernel functions are faster than other functions.

$F(x, xj) = sum(x.xj)$

Polynomial Kernel

It is a more generalized representation of the linear kernel. It is not as preferred as other kernel functions as it is less efficient and accurate.

$F(x, xj) = (x.xj+1)^d$

Where, shows the dot product of both the values, and d denotes the degree.
F(x, xj) representing the decision boundary to separate the given classes.

Radial Basis Function (RBF)

It is one of the most preferred and used kernel functions in svm. It is usually chosen for non-linear data. It helps to make proper separation when there is no prior knowledge of data.

$F(x, xj) = exp(-gamma * ||x - xj||^2)$

The value of gamma varies from 0 to 1. You have to manually provide the value of gamma in the code. The most preferred value for gamma is 0.1.