ASSIGNMENT – 5

MACHINE LEARNING

Q1 to Q15 are subjective answer type questions, Answer them briefly.

A1. 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?

Both R-squared and Residual Sum of Squares (RSS) are measures of goodness of fit in regression analysis, but they capture different aspects of the model's performance.

R-squared (also known as the coefficient of determination) measures the proportion of variation in the dependent variable that is explained by the independent variables in the model. In other words, it indicates how well the model fits the data, with values ranging from 0 to 1. Higher R-squared values indicate a better fit, as they mean that a larger proportion of the variation in the dependent variable is explained by the independent variables in the model.

On the other hand, RSS measures the total sum of squared differences between the actual values of the dependent variable and the predicted values by the model. It represents the amount of unexplained variation in the data, and lower RSS values indicate a better fit, as they mean that the model is able to explain more of the variation in the data.

Therefore, both measures are useful in evaluating the goodness of fit of a model, but they serve different purposes. R-squared is a useful measure to assess the overall fit of the model and to compare different models, while RSS is useful to identify the degree of the error in the model's predictions.

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

A.2 In regression analysis, the three main types of sum of squares are the Total sum of squares(TSS), Explained sum of squares(ESS) and Residual sum of squares(RSS).

1. 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:

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

Where:

- y_i – the value in a sample
- $\bar{y}$ – the mean value of a sample

2. Explained sum of squares (also known as the sum of squares due to regression or explained sum of squares)

The Explained sum of squares describes how well a regression model represents the modeled data. A higher Explained/Regression sum of squares indicates that the model does not fit the data well.

The formula for calculating the Explained/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

3. Residual sum of squares (also known as the sum of squared errors of prediction)

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:

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

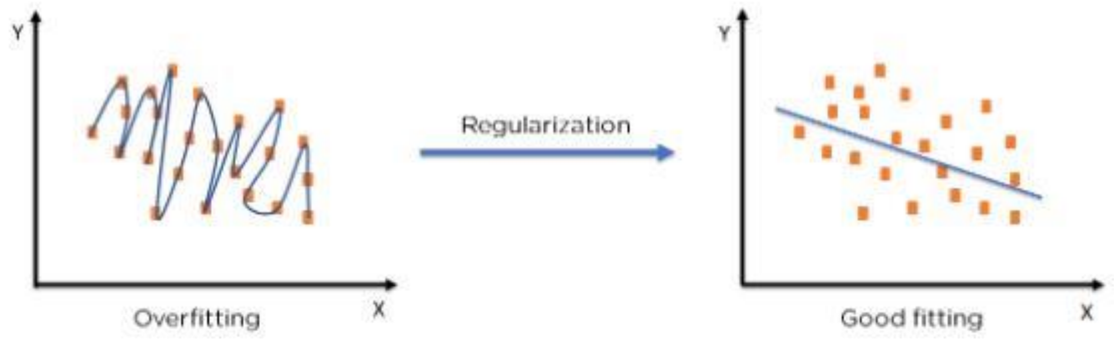
Where:

- y_i – the observed value
- $\hat{y}_i$ – the value estimated by the regression line
- 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?

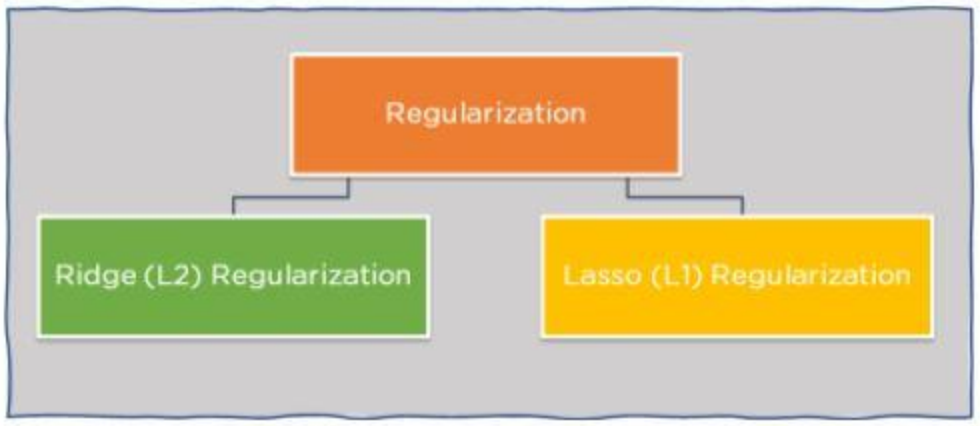
A.3 Regularization refers to techniques that are used to calibrate machine learning models in order to minimize the adjusted loss function and prevent overfitting or underfitting.



Regularization on an over-fitted model

Using Regularization, we can fit our machine learning model appropriately on a given test set and hence reduce the errors in it.

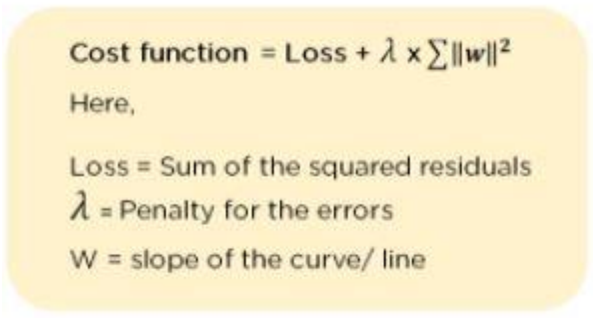
There are two main types of regularization techniques: Ridge Regularization and Lasso Regularization.



Ridge Regularization :

Also known as Ridge Regression, it modifies the over-fitted or under fitted models by adding the penalty equivalent to the sum of the squares of the magnitude of coefficients.

This means that the mathematical function representing our machine learning model is minimized and coefficients are calculated. The magnitude of coefficients is squared and added. Ridge Regression performs regularization by shrinking the coefficients present. The function depicted below shows the cost function of ridge regression :



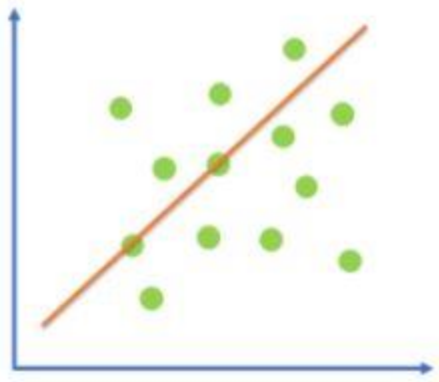
Cost function = **Loss** + λ **x** $\sum \|w\|^2$

Here,

Loss = Sum of the squared residuals

λ = Penalty for the errors

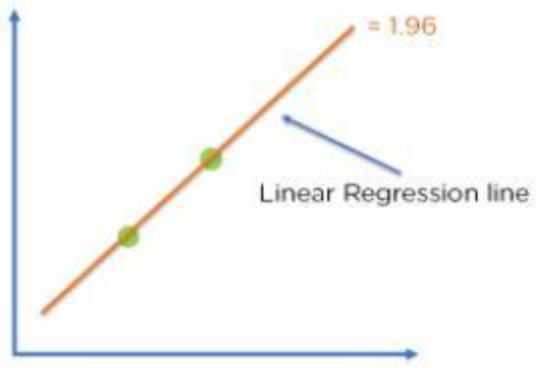
W = slope of the curve/ line



Cost Function of Ridge Regression

In the cost function, the penalty term is represented by Lambda λ. By changing the values of the penalty function, we are controlling the penalty term. The higher the penalty, it reduces the magnitude of coefficients. It shrinks the parameters. Therefore, it is used to prevent multicollinearity, and it reduces the model complexity by coefficient shrinkage.

Consider the graph illustrated below which represents Linear regression :



Linear regression model

Cost function = Loss + λ x∑‖w‖^2

For Linear Regression line, let’s consider two points that are on the line,

Loss = 0 (considering the two points on the line)

λ= 1

w = 1.4

Then, Cost function = 0 + 1 x 1.42

= 1.96

For Ridge Regression, let’s assume,

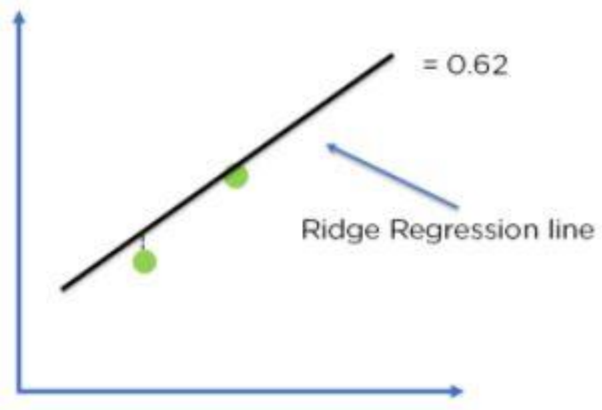
Loss = 0.32 + 0.22 = 0.13

λ = 1

w = 0.7

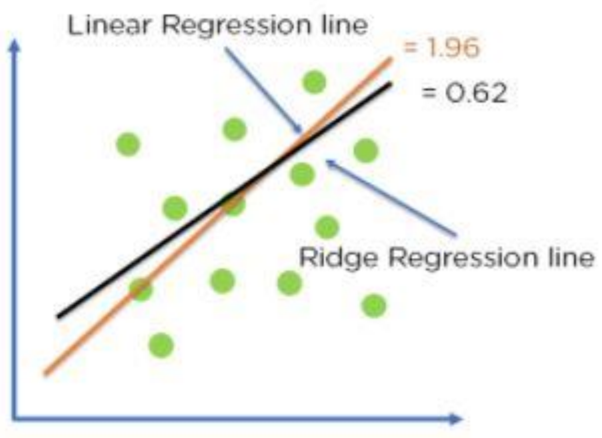
Then, Cost function = 0.13 + 1 x 0.72

= 0.62



Ridge regression model

Comparing the two models, with all data points, we can see that the Ridge regression line fits the model more accurately than the linear regression line.

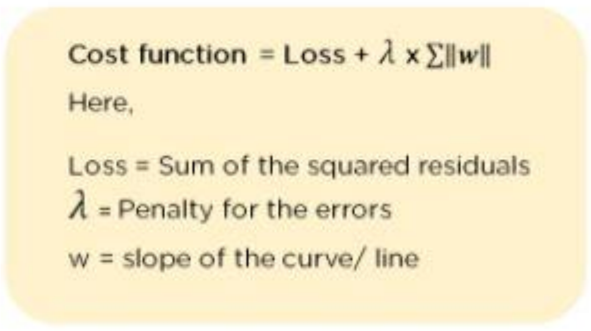


Optimization of model fit using Ridge Regression

Lasso Regression

It modifies the over-fitted or under-fitted models by adding the penalty equivalent to the sum of the absolute values of coefficients.

Lasso regression also performs coefficient minimization, but instead of squaring the magnitudes of the coefficients, it takes the true values of coefficients. This means that the coefficient sum can also be 0, because of the presence of negative coefficients. Consider the cost function for Lasso regression :



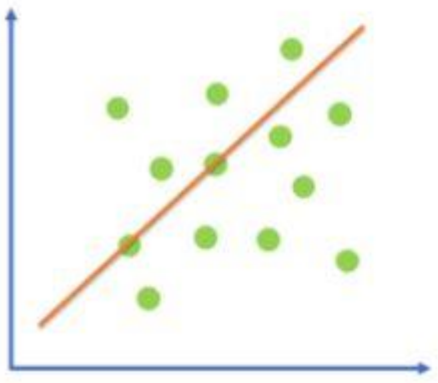
Cost function = **Loss** + λ **x** $\sum\|w\|$

Here,

Loss = Sum of the squared residuals

λ = Penalty for the errors

w = slope of the curve/ line



Cost function for Lasso Regression

We can control the coefficient values by controlling the penalty terms, just like we did in Ridge Regression. Again consider a Linear Regression model :

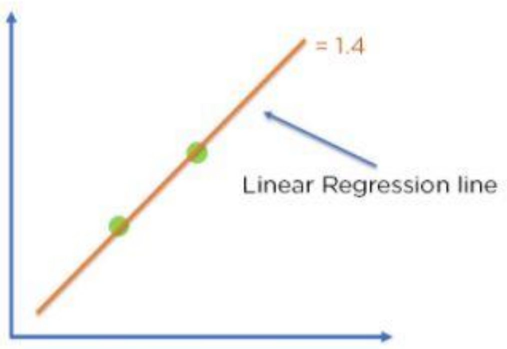


Figure 12: Linear Regression Model

Figure 12: Linear Regression Model

Cost function = Loss + λ x $\sum\|w\|$

For Linear Regression line, let’s assume,

Loss = 0 (considering the two points on the line)

$\lambda = 1$

w = 1.4

Then, Cost function = 0 + 1 x 1.4

= 1.4

For Ridge Regression, let’s assume,

Loss = 0.32 + 0.12 = 0.1

$\lambda = 1$

w = 0.7

Then, Cost function = 0.1 + 1 x 0.7

= 0.8

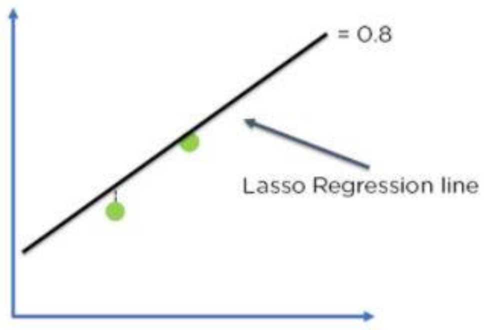


Figure 13: Lasso Regression

Comparing the two models, with all data points, we can see that the Lasso regression line fits the model more accurately than the linear regression line.

4. What is Gini–impurity index?

A.4 Gini impurity is a statistical measure used in Decision Trees to form a tree structure. While forming the tree structure, the algorithm (CART, ID3 etc.) must decide which feature is to be selected first. So in this post, we will take a close look at the main idea behind this selection.
Why do we need this statistical measure ?
The CART algorithm (Classification and Regression Trees) is a greedy algorithm which means that, the algorithm makes decisions with respect to the current state. It greedily searches for the "best" (optimum) split for the current level, and it repeats the same process for each subsequent levels. There is no guarantee that decisions made earlier will lead to optimum boundaries for the entire dataset. Now the question is that how does the CART algorithm approach to the splitting problem so that at the end of the process we have a reasonably good solution ?
To get the optimal split for the current level in the tree, we need a measure to distinguish the reasonably optimal one from the non-optimal ones. There are two common criterions which are used for this purpose:

1. **Gini Impurity (Gini Index)**
2. **Entropy**

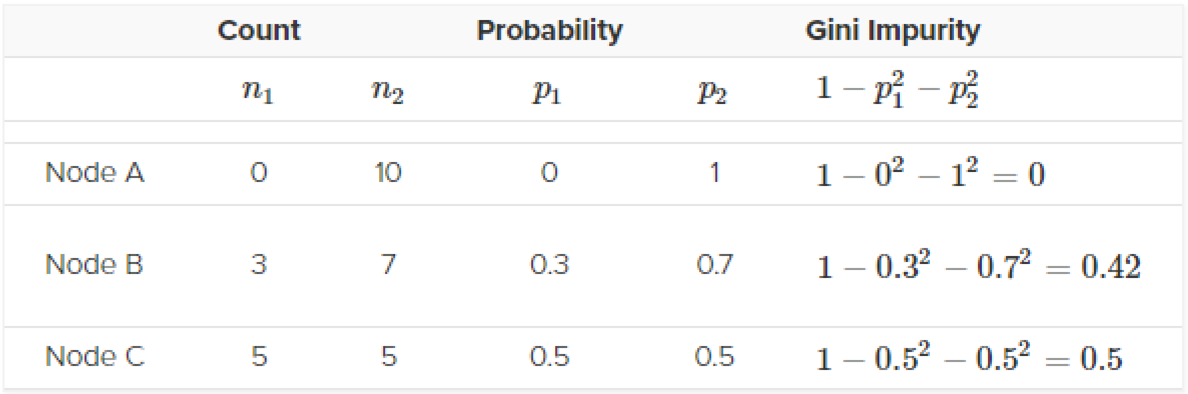
Definition of the Gini Impurity

For a single sample that is taken from the data set, the likelihood of being misclassified if it were given a random class label describes the Gini impurity. Now, we understand that this measure is all about the misclassification rate of the data with respect to all the samples in the given node (we will come this node concept later).

More formally, let **D** be the dataset containing the samples from **k** classes. The probability of the samples belonging to class **i** at a given node can be denoted as **p**ᵢ. Then the Gini impurity is calculated as:

$$Gini(D) = 1 - \sum_{i=1}^{k} p_i^2$$

Now, let us look at the implementation of the given formula. From the below table, we can infer that, uniformly distributed data leads to greater impurity score whereas the minimum impurity score is obtained if all the distribution belongs only one class.



	Count		Probability		Gini Impurity
	n_1	n_2	p_1	p_2	$1 - p_1^2 - p_2^2$
Node A	0	10	0	1	$1 - 0^2 - 1^2 = 0$
Node B	3	7	0.3	0.7	$1 - 0.3^2 - 0.7^2 = 0.42$
Node C	5	5	0.5	0.5	$1 - 0.5^2 - 0.5^2 = 0.5$

https://www.learndatasci.com/glossary/gini-impurity/

Feature Selection with Gini Impurity

Let **D** be the dataset. If **D** is split into two subsets, namely **D₁** and **D₂**, with the sizes of **n₁** and **n₂** with respect to an attribute **A**, Gini impurity associated with this attribute (feature) is defined as:

$$Gini_A(D) = \frac{n_1}{n} Gini(D_1) + \frac{n_2}{n} Gini(D_2)$$

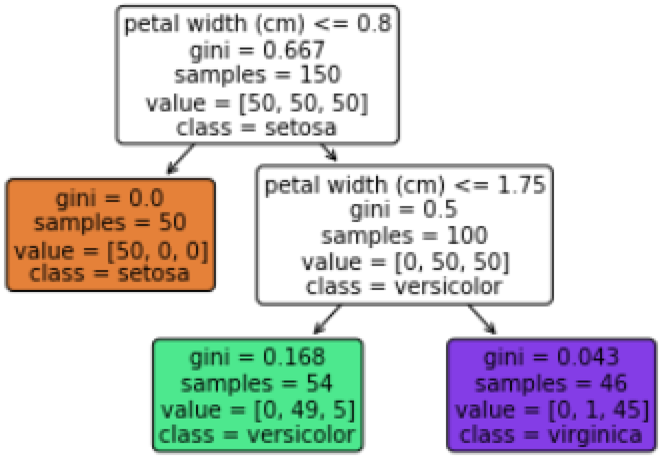
Where **n** is the total number of samples in **D**. It is the weighted sum of Gini impurities where weights are the ratio of the new sizes to the total size. While training a decision tree, feature with the lowest Gini impurity is selected by the CART.

Interpretation of the Value

Gini impurity takes values on the interval [0, 1]. If the value of gini close to 0, we can say that misclassification probability of that particular data is low, in other words this data point is likely to be classified correctly. If it is close to 1, the interpretation is just the opposite.

How Is It Used in the CART Algorithm ?

For the current node, CART algorithm should split the data optimally if the current node is not a leaf node. For this splitting, it uses the feature that has a minimum Gini impurity. But how CART algorithm decide what value of the bound will lead to optimum split for the current node ? Let's see this in the Iris dataset.



Decision Tree of Iris Dataset with max_depth = 2

For example, for the node belonging to depth 0, where did 0.8 boundary come from ? How did CART algorithm decide that the 0.8 for the sepal width feature will lead to optimum current split ?

We have just learned that feature selection — decision of which feature will be split at first — is done by the following formula:

$$Gini_A(D) = \frac{n_1}{n} Gini(D_1) + \frac{n_2}{n} Gini(D_2)$$

This formula is the cost function of the CART algorithm. It tries to minimize this value. Let $\mathbf{t_a}$ be a boundary for the attribute **A**. Taking different boundaries will lead to having different impurities. Out of all of them, CART will choose the minimum one.

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

A.5 Decision trees have a tendency to overfit to the training set because they can keep growing deeper and more complex until they perfectly classify the training data. This can lead to the tree capturing noise in the data, rather than the underlying relationships, and thus performing poorly on new, unseen data. Regularization techniques such as pruning, setting a minimum number of samples required to split a node, or limiting the maximum depth of the tree can help mitigate overfitting in decision trees. Regularization techniques aim to simplify the tree and prevent it from becoming overly complex, thus improving its ability to generalize to new data.

6. What is an ensemble technique in machine learning?

A.6 Ensemble Techniques

Here is a list of ensemble learning techniques, starting with basic ensemble methods and then moving on to more advanced approaches.

Simple Ensemble Methods

Mode: In statistical terminology, "mode" is the number or value that most often appears in a dataset of numbers or values. In this ensemble technique, machine learning professionals use a number of models for making predictions about each data point. The predictions made by different models are taken as separate votes. Subsequently, the prediction made by most models is treated as the ultimate prediction.

The Mean/Average: In the mean/average ensemble technique, data analysts take the average predictions made by all models into account when making the ultimate prediction.

Let's take, for instance, one hundred people rated the beta release of your travel and tourism app on a scale of 1 to 5, where 15 people gave a rating of 1, 28 people gave a rating of 2, 37 people gave a rating of 3, 12 people gave a rating of 4, and 8 people gave a rating of 5.

The average in this case is - (1 * 15) + (2 * 28) + (3 * 37) + (4 * 12) + (5 * 8) / 100 = 2.7

The Weighted Average: In the weighted average ensemble method, data scientists assign different weights to all the models in order to make a prediction, where the assigned weight defines the relevance of each model. As an example, let's assume that out of 100 people who gave feedback for your travel app, 70 are professional app developers, while the other 30 have no experience in app development. In this scenario, the weighted average ensemble technique will give more weight to the feedback of app developers compared to others.

Advanced Ensemble Methods

Bagging (Bootstrap Aggregating): The primary goal of "bagging" or "bootstrap aggregating" ensemble method is to minimize variance errors in decision trees. The objective here is to randomly create samples of training datasets with replacement (subsets of the training data). The subsets are then used for training decision trees or models. Consequently, there is a combination of multiple models, which reduces variance, as the average prediction generated from different models is much more reliable and robust than a single model or a decision tree.

Boosting: An iterative ensemble technique, "boosting," adjusts an observation's weight based on its last classification. In case observation is incorrectly classified, "boosting" increases the observation's weight, and vice versa. Boosting algorithms reduce bias errors and produce superior predictive models.

In the boosting ensemble method, data scientists train the first boosting algorithm on an entire dataset and then build subsequent algorithms by fitting residuals from the first boosting algorithm, thereby giving more weight to observations that the previous model predicted inaccurately.

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

A.7 Bagging

Bagging (or Bootstrap Aggregation), is a simple and very powerful ensemble method. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

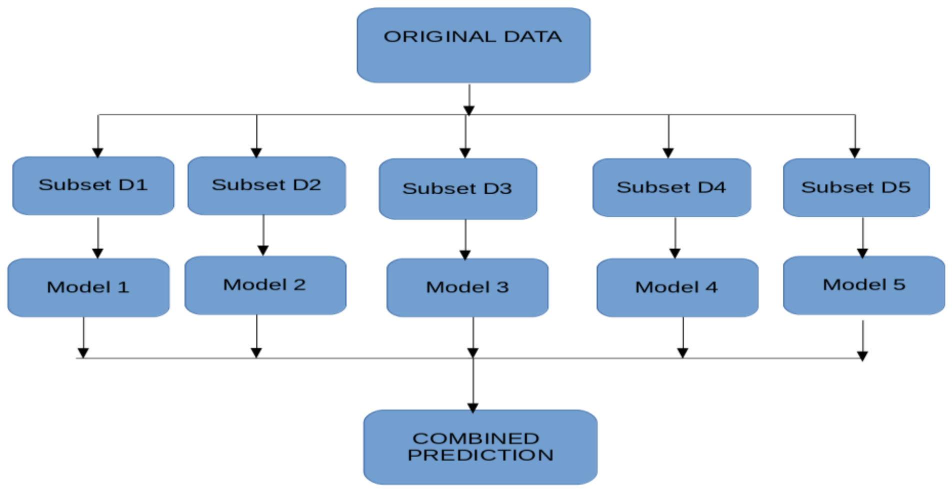
The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Now, bootstrapping comes into picture.

Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

Bagging works as follows:-

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

Now, bagging can be represented diagrammatically as follows:



Boosting

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model.

In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.

When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. By combining the whole set at the end converts weak learners into better performing model.

- Let's understand the way boosting works in the below steps.
    1. A subset is created from the original dataset.
    2. Initially, all data points are given equal weights.
    3. A base model is created on this subset.
    4. This model is used to make predictions on the whole dataset.

Errors are calculated using the actual values and predicted values.

The observations which are incorrectly predicted, are given higher weights.

Another model is created and predictions are made on the dataset.

In this approach multiple models are created, and each model correcting the errors of previous model.

Thus, the boosting algorithm combines a number of weak learners to form a strong learner.

Differences between Bagging and Boosting are as follows:-

1. **Bagging** is the simplest way of combining predictions that belong to the same type while **Boosting** is a way of combining predictions that belong to the different types.
2. **Bagging** aims to decrease variance, not bias while **Boosting** aims to decrease bias, not variance.
3. In **Baggiing** each model receives equal weight whereas in **Boosting** models are weighted according to their performance.
4. In **Bagging** each model is built independently whereas in **Boosting** new models are influenced by performance of previously built models.
5. In **Bagging** different training data subsets are randomly drawn with replacement from the entire training dataset. In **Boosting** every new subsets contains the elements that were misclassified by previous models.

6. **Bagging** tries to solve over-fitting problem while **Boosting** tries to reduce bias.
7. If the classifier is unstable (high variance), then we should apply **Bagging**. If the classifier is stable and simple (high bias) then we should apply **Boosting**.
8. **Bagging** is extended to Random forest model while **Boosting** is extended to **Gradient boosting**.

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

A.8 Generally, in machine learning and data science, it is crucial to create a trustful system that will work well with the new, unseen data. Overall, there are a lot of different approaches and methods to achieve this generalization. Out-of-bag error is one of these methods for validating the machine learning model.

Definition

This approach utilizes the usage of bootstrapping in the random forest. Since the bootstrapping samples the data with the possibility of selecting one sample multiple times, it is very likely that we won't select all the samples from the original data set. Therefore, one smart decision would be to exploit somehow these unselected samples, called out-of-bag samples.

Correspondingly, the error achieved on these samples is called out-of-bag error. What we can do is to use out-of-bag samples for each decision tree **to measure its performance. This strategy provides reliable results in** comparison to other validation techniques such as train-test split or cross-validation.

3.2. Probability of Out-of-bag Sample

Theoretically, with the quite big data set and the number of sampling, it is expected that out-of-bag error will be calculated on 36% of the training set. To prove this, consider that our training set has **n** samples. Then, the probability of selecting one particular sample from the training set is **1/n.**

Similarly, the probability of not selecting one particular sample is (1-1/n).Since we select the bootstrap samples with replacement, the probability of one particular sample not being selected n times is equal to (1-1/n)n. Now, if the number n is pretty big or if it tends to infinity.

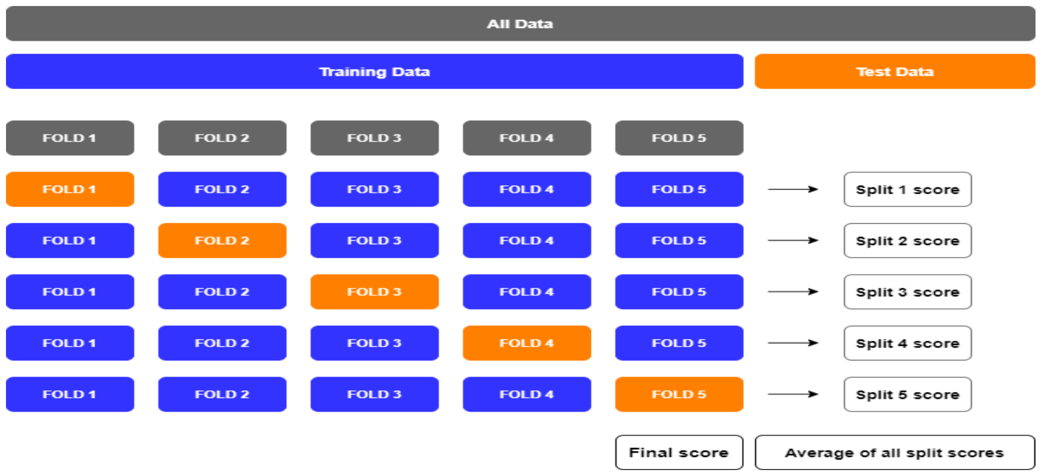
9. What is K-fold cross-validation?

A.9 K Fold Cross Validation ¶

In case of K Fold cross validation input data is divided into 'K' number of folds, hence the name K Fold. Suppose we have divided data into 5 folds i.e. K=5. Now we have 5 sets of data to train and test our model. So the model will get trained and tested 5 times, but for every iteration we will use one fold as test data and rest all as training data. Note that for every iteration, data in training and test fold changes which adds to the effectiveness of this method.

This significantly reduces underfitting as we are using most of the data for training(fitting), and also significantly reduces overfitting as most of the data is also being used in validation set. K Fold cross validation helps to generalize the machine learning model, which results in better predictions on unknown data.

For most of the cases 5 or 10 folds are sufficient but depending on problem you can split the data into any number of folds.



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

A.10 When you’re training machine learning models, each dataset and model needs a different set of hyperparameters, which are a kind of variable. The only way to determine these is through multiple experiments, where you pick a set of hyperparameters and run them through your model. This is called *hyperparameter tuning*. In essence, you're training your model sequentially with different sets of hyperparameters. This process can be manual, or you can pick one of several automated hyperparameter tuning methods.

Whichever method you use, you need to track the results of your experiments. You’ll have to apply some form of statistical analysis, such as the loss function, to determine which set of hyperparameters gives the best result. Hyperparameter tuning is an important and computationally intensive process.

What are hyperparameters?

Hyperparameters are external configuration variables that data scientists use to manage machine learning model training. Sometimes called *model hyperparameters*, the hyperparameters are manually set before training a model. They're different from parameters, which are internal parameters automatically derived during the learning process and not set by data scientists.

Examples of hyperparameters include the number of nodes and layers in a neural network and the number of branches in a decision tree. Hyperparameters determine key features such as model architecture, learning rate, and model complexity.

How do you identify hyperparameters?

Selecting the right set of hyperparameters is important in terms of model performance and accuracy. Unfortunately, there are no set rules on which hyperparameters work best nor their optimal or default values. You need to experiment to find the optimum hyperparameter set. This activity is known as *hyperparameter tuning* or *hyperparameter optimization*.

Why is hyperparameter tuning important?

Hyperparameters directly control model structure, function, and performance. Hyperparameter tuning allows data scientists to tweak model performance for optimal results. This process is an essential part of machine learning, and choosing appropriate hyperparameter values is crucial for success.

For example, assume you're using the learning rate of the model as a hyperparameter. If the value is too high, the model may converge too quickly with suboptimal results. Whereas if the rate is too low, training takes too long and results may not converge. A good and balanced choice of hyperparameters results in accurate models and excellent model performance.

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

A.11 The learning rate is an important hyperparameter that greatly affects the performance of gradient descent. It determines how quickly or slowly our model learns, and it plays an important role in controlling both convergence and divergence of the algorithm. When the learning rate is too large, gradient descent can suffer from divergence. This means that weights increase exponentially, resulting in exploding gradients which can cause problems such as instabilities and overly high loss values. On the other hand, if the learning rate is too small, then gradient descent can suffer from slow convergence or even stagnation—which means it may not reach a local minimum at all unless many iterations are performed on large datasets.

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

A.12 Logistic regression is a type of linear model that predicts the probability of a binary outcome, such as yes or no, true or false, or 0 or 1. It uses a logistic function, also known as a sigmoid function, to map the input features to a value between 0 and 1. The logistic function has an S-shaped curve that flattens at the extremes. Logistic regression then uses a threshold, usually 0.5, to classify the output as 0 or 1.
Logistic regression is simple and easy to implement, but it also has some drawbacks. One of them is that it assumes a linear relationship between the input features and the output. This

means that it cannot capture the complexity and non-linearity of the data. Another drawback is that it is sensitive to outliers and noise, which can affect the accuracy and stability of the model. Logistic regression also has a limited capacity to learn from multiple features, as it can only combine them linearly.

13. Differentiate between Adaboost and Gradient Boosting.

A.13 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.

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

A.14 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.

What is Bias?

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.

This instance, where the model cannot find patterns in our training set and hence fails for both seen and unseen data, is called Underfitting.

What is 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.

When Variance is high Model work very well on trained data, but when new data is given it cannot predict on that.

Variance model will perform really well on testing data and get high accuracy but will fail to perform on new, unseen data. New data may not have the exact same features and the model won’t be able to predict it very well. This is called Overfitting.

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.

A.15 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.

Linear Kernel Formula

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

Here, **x, xj** represents the data you're trying to classify.

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.

Polynomial Kernel Formula

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

Here '.' 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.

Gaussian 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.

Gaussian Radial Basis Formula

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**.