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?

Ans: R-squard (R^2) and Residual sum of squares (RSS) are both commonly used measures to assess the goodness of fit of regression model, but they capture different aspects of model performance, and the choice between them depends on the context and what you want to evaluate.

1. R-squared (R^2):- R-squared is a measure of the proportional of the variance in the dependent variable that is explained by the independent variables In your regression model.
   - R-squared range from 0 to 1, with higher values indicating a better fir. A value of 1 means that the model perfectly explains the variance in the dependent variable, while a value of 0 means that the model provides no explanatory power.
   - R-square is a relative measure, and it does not provide information about the absolute goodness of fit or the quality of the model's predictions.
   - R-square can be be useful for comparing different models or assessing how much of the variation in the dependent variable can be attributed to the example, it can artificially inflated by adding more predictors to a model, even if those predictors to a model, even if those predictors do not have a meaningful relationship with the dependent variable.
2. Residual sum of square (RSS):-RSS measures the total squared difference between the observed values of the dependent variable and predicted values from the regression model. It quantifies the overall error of residuals in the model's predictions.
   - A smaller RSS indicates a better fit because it means that the model's predictions are closer to the actual observed values.
   - RSS is an absolute measure of the goodness of fit. It tells you how well the model the model fits the data in terms of minimizing prediction errors. Unlike R-squared, RSS does not provide explained but gives you a direct measure of the model's prediction accuracy.

Which Measure to Use:-R-squared is often used when you want to understand the proportion of the variance explained by the model, especially in the context of comparing different models. It can provide insight into how well the predictors collectively contribute into how explaining the variation in the dependent variable.

- RSS is useful when you want to evaluate the absolute goodness of fit, focusing on the magnitude of the prediction errors. If minimizing prediction errors is a primary concern, RSS is a more appropriate choice.
- In practice, both measures can be valuable, R-squared provides a high-level summary of the explanatory power of your model while RSS gives you a detailed view of the models prediction accuracy. The choice between them should align with your specific goals and the questions you want to answer about your regression 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.

Ans:- TSS = ESS + RSS, where TSS is Total Sum of Squares, ESS is Explained Sum of Squares and RSS is Residual Sum of Suqares. The aim of Regression Analysis is explain the variation of dependent variable Y.

TSS (total sum of squares) is equal to **ESS** (explained sum of squares) plus **RSS** (residual sum of squares), we need to start with the definitions of these terms and then use some algebraic manipulations to arrive at the desired result.

Let us begin by defining the three terms:

TSS = $\sum(Y_i - \bar{Y})^2$, where Y_i is the actual value of the response variable for observation i, and $\bar{Y}$ is the mean of the response variable.

ESS = $\sum(\hat{Y}_i - \bar{Y})^2$, where $\hat{Y}_i$ is the predicted value of the response variable for observation i.

RSS = $\sum(Y_i - \hat{Y}_i)^2$, which is the sum of squared differences between the actual and predicted values of the response variable.

Now, we can expand the terms in the definition of TSS using the definition of $\hat{Y}_i$:

TSS = $\sum(Y_i - \bar{Y})^2 = \sum[(Y_i - \hat{Y}_i) + (\hat{Y}_i - \bar{Y})]^2 = \sum(Y_i - \hat{Y}_i)^2 + \sum(\hat{Y}_i - \bar{Y})^2 + 2\sum(Y_i - \hat{Y}_i)(\hat{Y}_i - \bar{Y})$

Next, we can use the definition of RSS to simplify the first term on the right-hand side:

$\sum(Y_i - \hat{Y}_i)^2 = RSS$

Similarly, we can use the definition of ESS to simplify the second term:

$\sum(\hat{Y}_i - \bar{Y})^2 = ESS$

Now, we need to simplify the third term using some algebraic manipulations. We can start by expanding the product:

$2\sum(Y_i - \hat{Y}_i)(\hat{Y}_i - \bar{Y}) = 2\sum(Y_i\hat{Y}_i - Y_i\bar{Y} - \hat{Y}_i\bar{Y} + \hat{Y}_i^2)$

Then, we can use the fact that the sum of the residuals ($Yi - \hat{Y}i$) is zero, which can be shown as follows:

$\sum(Yi - \hat{Y}i) = \sum Yi - \sum\hat{Y}i = n\bar{Y} - n\bar{Y} = 0$

Using this fact, we can simplify the third term:

$2\sum(Yi - \hat{Y}i)(\hat{Y}i - \bar{Y}) = 2\sum(Yi\hat{Y}i - \hat{Y}i\bar{Y}) = 2(\sum Yi\hat{Y}i - \sum\hat{Y}i\bar{Y}) = 2(\sum\hat{Y}iYi - n\bar{Y}^2) = 2(ESS - n(\bar{Y} - \hat{Y})^2)$

where $\hat{Y}$ is the sample mean of the predicted values, which is equal to $\bar{Y}$.

Substituting these results back into the equation for TSS, we get:

$TSS = RSS + ESS + 2(ESS - n(\bar{Y} - \hat{Y})^2) = RSS + 2ESS - 2n(\bar{Y} - \hat{Y})^2 = RSS + 2ESS - 2n(\bar{Y} - \bar{Y})^2 = RSS + 2ESS - 0 = RSS + ESS$

Therefore, we have shown that TSS is equal to ESS plus RSS.

3]What is the need of regularization in machine learning?

Ans:-In Machine Learning we often divide the dataset into training and test data, the algorithm while training the data can either

1) learn the data too well, even the noises which is called over fitting

2) do not learn from the data, cannot find the pattern from the data which is called under fitting.

Now, both over fitting and underfitting are problems one need to address while building models.

Regularization in Machine Learning is used to minimize the problem of overfitting, the result is that the model generalizes well on the unseen data

once overfitting is minimized.
To avoid overfitting, regularization discourages learning a more sophisticated or flexible model. Regularization will try to minimize a loss function by inducing penalty.
For Example
The residual sum of squares is our optimization function or loss function in simple linear regression (RSS).

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

Here ,

y is the dependent variable,
x_1, x_2, x_3,..... x_n are independent variables.
b_0, b_1 ,b_2......... b_n, are the coefficients estimates for different variables of x, these can also be called weights or magnitudes
Regularization will shrink these coefficients towards Zero,
Minimizing the loss means less error and model will be a good fit.

The way regularization can be done is by
1) RIDGE also know as L-2 Regularization
2) LASSO (Least Absolute and Selection Operator) also known as L-1 Regularization.

4] What is Gini–impurity index?

Ans:- The Gini Index (or Gini Impurity) is a widely employed metric for splitting a classification decision tree. In the following sections, you'll explore its definition, mathematical formula, its role in tree construction, and a step-by-step example demonstrating how it is computed.

What is Gini Index (aka Gini Impurity)?

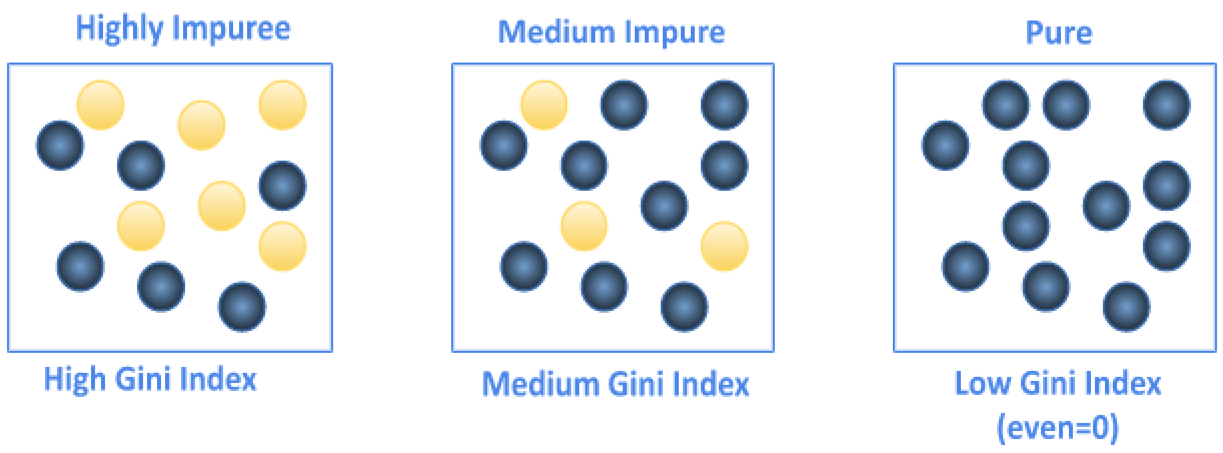
The ***Gini Index***, also known as ***Gini Impurity***, assists the CART algorithm in identifying the most suitable feature for node splitting during the construction of a decision tree classifier

It derives its name from the Italian mathematician Corrado Gini.

Gini Impurity is a method that measures the impurity of a dataset. The more impure the dataset, the higher is the Gini index.

The term “Impurity” indicates the number of classes present within a subset. The more distinct classes included in a subset, the higher the impurity. This concept is clearly illustrated in the following image.

Visualization



- The leftmost square contains an equal number of yellow and blue balls, indicating an equal probability. It’s diversified, impure. The Gini index will be high.

- The second squarecontains a greater number of blue balls compared to yellow ones. It's still diversified, but more pure than the first square. The Gini index will be medium.
- The last square, there are only blue balls, it's pure. The Gini index is equal to 0

In the following paragraph, you can use the provided mathematical formula, , to calculate the value of the Gini Index, as demonstrated in the preceding illustration.

Mathematical Formula

The computation of the Gini index is as follows:

$$I_G(p) = 1 - \sum_{i=1}^{C} p_i^2$$

Where:

- C: The number of classes included in the subset
- i: The class i, with i {1, 2, 3, ….C}
- pi : The proportion of the class *"i"* is in the subset.

To remember, ***the lowest the value of the Gini Index,*** the more pure is the dataset, and the lower is the entropy.

In the forthcoming sections, we will apply the Gini Index formula to split and construct a tree through practical exercises.

How is the Gini Index used in the Decision Trees model?

Gini index helps building a classification tree by following those steps:

- For each node:
    - For each feature in the dataset, and each threshold we compute the weighted average value of the Gini index

- The best feature to use to split the current node is the one having the lowest Gini Index
- Repeat these steps for each subtree
- Stop splitting the tree when stop criteria is met

Practical example

Let's take our favorite dataset as in the entropy article to compute the Gini Index and understand how to use it.

Savings	Assets	Salary ($1000)	Credit Note
High	High	20	Good
Low	High	25	Bad
High	Low	30	Good
Low	Low	35	Bad
Low	High	40	Good
Low	Low	50	Bad
High	Low	90	Good

General Gini Index

The Gini Index for this dataset is 0.49:

	#	p	p^2
Good	4	0,571	0,327
Bad	3	0,429	0,184
Gini Index			0,490

There are 4 Good credit notes and 3 Bad ones:

$$Gini\ Index = 1 - P_{Good}^{2} - P_{Bad}^{2} = 0.490$$

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

Ans:- Decision trees are prone to overfitting when they capture noise in the data. Pruning and setting appropriate stopping criteria are used to address this assumption

If a decision tree is fully grown, it may lose some generalization capability.

• This is a phenomenon known as overfitting. 1 Data Preprocessing Classification & Regression Definition of Overfitting Consider the error of hypothesis h.

We let error on the training data be error$train$ h and error over the entire distribution D of data be error$D$ h . Then a hypothesis h “overfits” the training data if there is an alternative hypothesis,(h'), such that: error$train$ h < error$train$(h') error$D$ h < error$D$(h')Data Preprocessing Classification & Regression

Model Overfitting Errors committed by classification models are generally divided into two types: 3 1 2 Training Errors The number of misclassification errors committed on training records; also called resubstitution error. Generalization Errors The expected error of the model on previously unseen records.

Avoiding Overfittingin:- Decision Trees • Stop growing the tree when the data split is not statistically significant • Grow the full tree, then prune – Do we really needs all the “small” leaves with perfect coverage.

6] What is an ensemble technique in machine learning?

Ans:- Ensemble methods fall into two broad categories, i.e., sequential ensemble techniques and parallel ensemble techniques. **Sequential ensemble techniques** generate base learners in a sequence, e.g., Adaptive Boosting (AdaBoost). The sequential generation of base learners promotes the dependence between the base learners. The performance of the model is then improved by assigning higher weights to previously misrepresented learners.

In **parallel ensemble techniques**, base learners are generated in a parallel format, e.g., random forest. Parallel methods utilize the parallel generation of base learners to encourage independence between the base learners. The independence of base learners significantly reduces the error due to the application of averages.

The majority of ensemble techniques apply a single algorithm in base learning, which results in homogeneity in all base learners. Homogenous base learners refer to base learners of the same type, with similar qualities. Other methods apply heterogeneous base learners, giving rise to heterogeneous ensembles. Heterogeneous base learners are learners of distinct types.

Main Types of Ensemble Methods

1. Bagging

Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.

Bagging is classified into two types, i.e., bootstrapping and aggregation. **Bootstrapping** is a sampling technique where samples are derived from the whole population (set) using the replacement procedure. The sampling with replacement method helps make the selection procedure randomized. The base learning algorithm is run on the samples to complete the procedure.

Aggregation in bagging is done to incorporate all possible outcomes of the prediction and randomize the outcome. Without aggregation, predictions will not be accurate because all outcomes are not put into consideration. Therefore, the aggregation is based on the probability bootstrapping procedures or on the basis of all outcomes of the predictive models.

Bagging is advantageous since weak base learners are combined to form a single strong learner that is more stable than single learners. It also eliminates any variance, thereby reducing the overfitting of models. One limitation of bagging is that it is computationally expensive. Thus, it can lead to more bias in models when the proper procedure of bagging is ignored.

2. Boosting

Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.

Boosting takes many forms, including gradient boosting, Adaptive Boosting (AdaBoost), and XGBoost (Extreme Gradient Boosting). AdaBoost uses weak learners in the form of decision trees, which mostly include one split that is popularly known as decision stumps. AdaBoost's main decision stump comprises observations carrying similar weights.

Gradient boosting adds predictors sequentially to the ensemble, where preceding predictors correct their successors, thereby increasing the model's accuracy. New predictors are fit to counter the effects of errors in the previous predictors. The gradient of descent helps the gradient booster identify problems in learners' predictions and counter them accordingly.

XGBoost makes use of decision trees with boosted gradient, providing improved speed and performance. It relies heavily on the computational speed and the performance of the target model. Model training should follow a sequence, thus making the implementation of gradient boosted machines slow.

3. Stacking

Stacking, another ensemble method, is often referred to as stacked generalization. This technique works by allowing a training algorithm to ensemble several other similar learning algorithm predictions. Stacking has been successfully implemented in regression, density estimations, distance learning, and classifications. It can also be used to measure the error rate involved during bagging.

Variance Reduction

Ensemble methods are ideal for reducing the variance in models, thereby increasing the accuracy of predictions. The variance is eliminated when multiple models are combined to form a single prediction that is chosen from all other possible predictions from the combined models. An ensemble of models combines various models to ensure that the resulting prediction is the best possible, based on the consideration of all predictions.

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

Ans:-

S.NO	Bagging	Boosting
1.	The simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by the performance of previously built models.
5.	Different training data subsets are selected using row sampling with	Every new subset contains the elements

S.NO	Bagging	Boosting
	replacement and random sampling methods from the entire training dataset.	that were misclassified by previous models.
6.	Bagging tries to solve the over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	In this base classifiers are trained parallelly.	In this base classifiers are trained sequentially.
9	Example: The Random forest model uses Bagging.	Example: The AdaBoost uses Boosting techniques

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

Ans:- What is Out-of-Bag Error?

Out-of-Bag Error, also known as OOB Error, is a concept used in ensemble **machine learning** algorithms such as **random forests**. When building a random forest model, each tree is trained using a subset of the original data, known as the bootstrap sample. During the training process, some observations are left out or "out-of-bag" (OOB) for each tree.

The OOB observations that were not used in the training of a particular tree can be considered as a **validation** set for that tree. The model's prediction accuracy on the OOB observations can then be calculated and averaged across all the trees to obtain the OOB Error.

How Out-of-Bag Error works

Out-of-Bag Error works by utilizing the OOB observations to estimate the model's performance on unseen data. For each observation, the OOB error is computed by comparing the model's prediction with the ground truth value. This process is repeated for all the OOB observations and averaged across all the trees in the ensemble.

The OOB Error provides an unbiased estimate of the model's performance without the need for a separate validation set. It serves as an internal validation mechanism within the random forest algorithm.

Why Out-of-Bag Error is important

Out-of-Bag Error is important for several reasons:

- **Model evaluation:** It provides a reliable estimate of the model's performance on unseen data, helping assess the quality of predictions and identifying potential overfitting.
- **Feature importance:** OOB Error can also be used to determine the relative importance of different features in the dataset. By comparing the performance metrics when using individual features, it is possible to identify the most influential predictors.
- **Reduced need for validation set:** The OOB Error allows for model evaluation without the need to set aside a separate validation dataset, making the training process more efficient.

The most important Out-of-Bag Error use cases

Out-of-Bag Error is commonly used in the following scenarios:

- **Model selection:** It helps in selecting the optimal number of trees in a random forest (or similar ensemble algorithms) by comparing the OOB Error across different model configurations.
- **Feature selection:** OOB Error can be used to identify the most relevant features in a dataset and guide **feature engineering** efforts.
- **Model tuning:** OOB Error can guide the optimization of hyperparameters related to tree growth and regularization, leading to improved model performance.

Related technologies or terms

Out-of-Bag Error is closely related to the following concepts and technologies:

- **Random Forest:** Out-of-Bag Error is specifically used in the context of random forest models, which are an ensemble learning method combining multiple **decision trees**.
- **Ensemble Learning:** Out-of-Bag Error is a technique used in ensemble learning, where multiple models are combined to make predictions.
- **Cross-validation:** While Out-of-Bag Error is an internal validation method, cross-validation is an external technique that involves splitting the dataset into multiple subsets for model evaluation.

Why Dremio users would be interested in Out-of-Bag Error

Dremio users who are involved in data processing and analytics can benefit from understanding Out-of-Bag Error for the following reasons:

- **Improved model evaluation:** Out-of-Bag Error offers a reliable way to estimate the performance of machine learning models, helping users assess their accuracy and make informed decisions.
- **Feature selection and engineering:** By leveraging Out-of-Bag Error, Dremio users can identify the most influential features in their datasets and optimize their feature engineering efforts.
- **Optimized model tuning:** Out-of-Bag Error can guide users in tuning the hyperparameters of ensemble models, leading to improved model performance in their **data processing** and analytics tasks.

9] What is K-fold cross-validation?

Ans:- What Is K-Fold Cross-Validation?

K-fold cross-validation is a procedure where a dataset is divided into multiple training and validation sets (folds), where k is the number of them, to help safeguard the model against random bias caused by the selection of only one training and validation set.

With this procedure, we run our modeling process on different subsets of the data to get multiple measures of model quality.

For example, we could begin by dividing the data into 5 pieces, each 20% of the full dataset. In this case, we say that we have broken the data into 5 "folds."

Then, we run one experiment for each fold:

- In Experiment 1, we use the first fold as a validation (or holdout) set, and use everything else as training data. This gives us a measure of model quality based on a 20% holdout set.
- In Experiment 2, we hold out data from the second fold (and use everything except the second fold for training the model). The holdout set is then used to get a second estimate of model quality.

- We repeat this process, using every fold once as the holdout set. Putting this together, 100% of the data is used as holdout at some point, and we end up with a measure of model quality that is based on all of the rows in the dataset (even if we don't use all rows simultaneously).

When Should You Use K-Fold Cross-Validation?

Cross-validation gives a more accurate measure of model quality, which is especially important if you are making a lot of modeling decisions. However, it can take longer to run because it estimates multiple models (one for each fold).

/So, Given These Tradeoffs, When Should you use each approach?

- For small datasets, where the extra computational burden isn't a big deal, you should run k-fold cross-validation.
- For larger datasets, a single validation set is sufficient. Your code will run faster, and you may have enough data that there's little need to reuse some of it for the holdout. There's no simple threshold for what constitutes a large vs. small dataset. But if your model takes a couple of minutes or less to run, it's probably worth switching to k-fold cross-validation.

Alternatively, you can run cross-validation and see if the scores for each experiment seem close. If each experiment yields the same results, a single validation set is probably sufficient.

An Example Of K-Fold Cross-Validation

We'll work with the same data as in the previous tutorial. We load the input data in X and the output data in y.

Then, we define a pipeline that uses an imputer to fill in missing values and a random forest model to make predictions.

While it's possible to do k-fold cross-validation without pipelines, it is quite difficult! Using a pipeline will make the code remarkably straightforward.

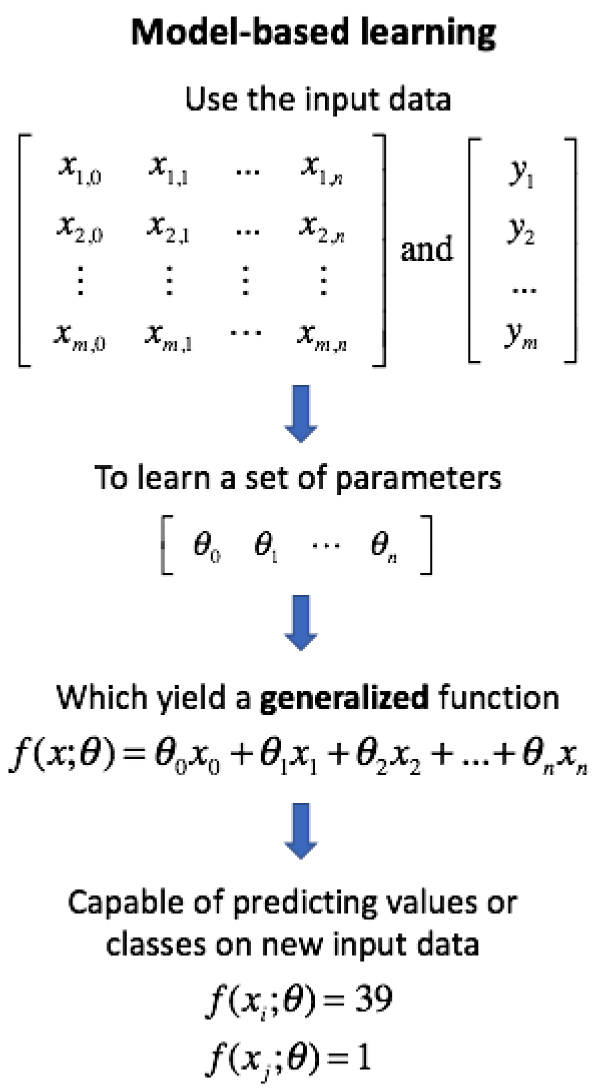
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

my_pipeline = Pipeline(steps=[('preprocessor', SimpleImputer()),
                              ('model',
RandomForestRegressor(n_estimators=50,

random_state=0))
```

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

Ans: When creating a machine learning model, you'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*.
These hyperparameters might address model design questions such as:

- What degree of polynomial features should I use for my linear model?
- What should be the maximum depth allowed for my decision tree?
- What should be the minimum number of samples required at a leaf node in my decision tree?
- How many trees should I include in my random forest?
- How many neurons should I have in my neural network layer?
- How many layers should I have in my neural network?
- What should I set my learning rate to for gradient descent?

I want to be absolutely clear, ***hyperparameters are not model parameters*** and they cannot be directly trained from the data. *Model parameters* are learned during training when we optimize a loss function using something like gradient descent.The process for learning parameter values is shown generally below.


Model-based learning
Use the input data
and
To learn a set of parameters
Which yield a generalized function
Capable of predicting values or classes on new input data

Whereas the model parameters specify how to transform the input data into the desired output, the hyperparameters define how our model is actually structured. Unfortunately, there's no way to calculate “which way should I update my hyperparameter to reduce the loss?” (ie. gradients) in order to find the optimal model architecture; thus, we generally resort to experimentation to figure out what works best.

In general, this process includes:

1. Define a model
2. Define the range of possible values for all hyperparameters

3. Define a method for sampling hyperparameter values
4. Define an evaluative criteria to judge the model
5. Define a cross-validation method

Specifically, the various hyperparameter tuning methods I'll discuss in this post offer various approaches to Step 3.

Model validation

Before we discuss these various tuning methods, I'd like to quickly revisit the purpose of splitting our data into training, validation, and test data. The ultimate goal for any machine learning model is to learn from examples in such a manner that the model is capable of generalizing the learning to new instances which it has not yet seen. At a very basic level, you should train on a subset of your total dataset, holding out the remaining data for evaluation to gauge the model's ability to generalize - in other words, "how well will my model do on data which it hasn't directly learned from during training?"

When you start exploring various model architectures (ie. different hyperparameter values), you also need a way to evaluate each model's ability to generalize to unseen data. However, if you use the testing data for this evaluation, you'll end up "fitting" the model architecture to the testing data - losing the ability to truely evaluate how the model performs on unseen data. This is sometimes referred to as "data leakage".

To mitigate this, we'll end up splitting the total dataset into three subsets: training data, validation data, and testing data. The introduction of a validation dataset allows us to evaluate the model on different data than it was trained on and select the best model architecture, while still holding out a subset of the data for the final evaluation at the end of our model development.

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

Ans:- Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. The goal of gradient descent is to find the set of parameters that minimizes this discrepancy and improves the model's performance.

radient descent can be applied to various machine learning algorithms, including linear regression, logistic regression, neural networks, and support vector machines. It provides a general framework for optimizing models by iteratively refining their parameters based on the cost function.

Example of Gradient Descent:-

Let's say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?

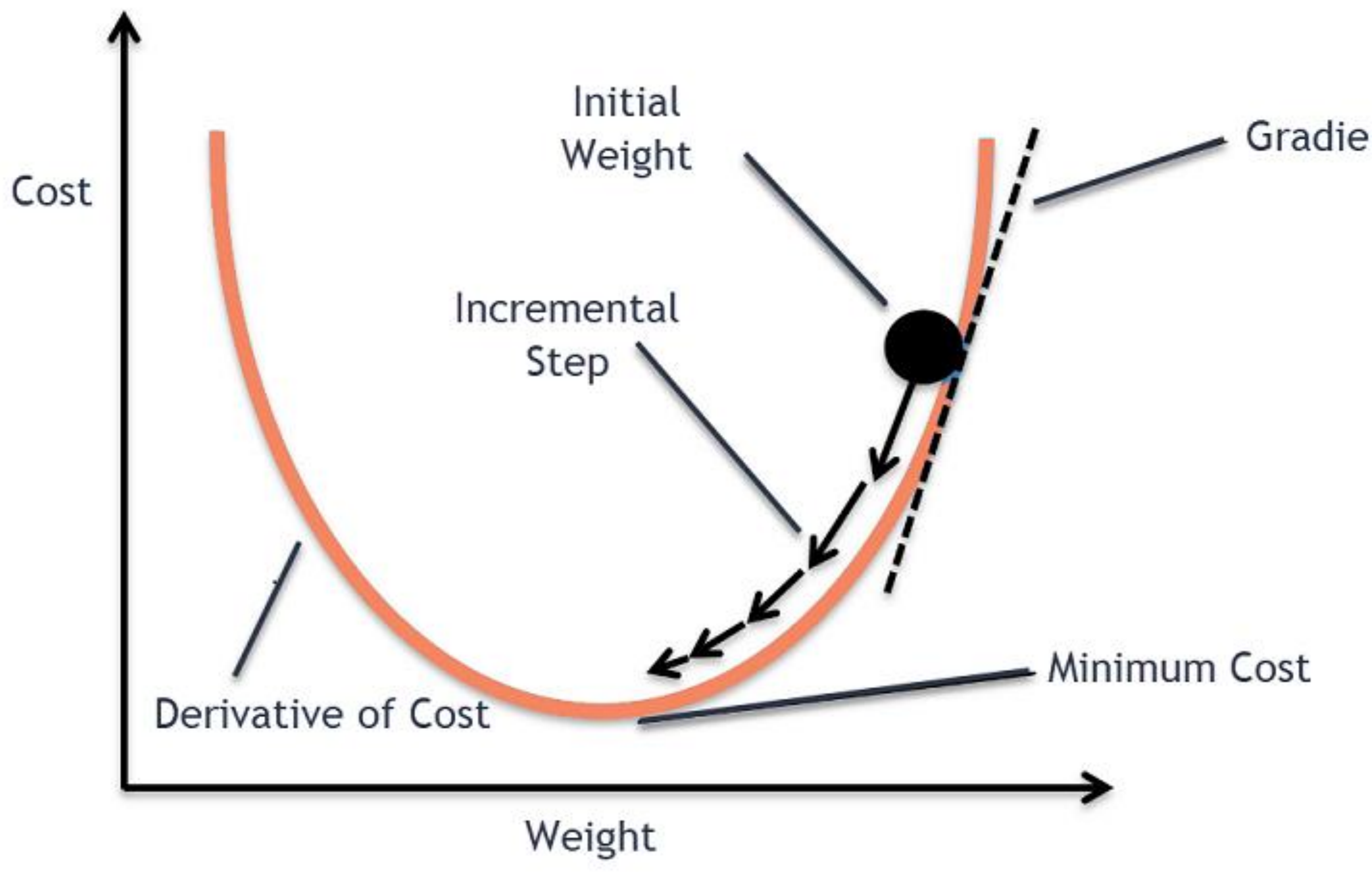
Take a moment to think about this before you read on.

The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point. If we take steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function, and the procedure is called **Gradient Ascent.**

Gradient descent was originally proposed by **CAUCHY** in 1847. It is also known as steepest descent.



Imagine you're lost in a dense forest with no map or compass. What do you do? You follow the path of steepest descent, taking steps in the direction that decreases the slope and brings you closer to your destination. Similarly, gradient descent is the go-to algorithm for navigating the complex landscape of machine learning. It helps models find the optimal set of parameters by iteratively adjusting them in the opposite direction of the gradient. In this article, we'll take a deep dive into the world of gradient descent, exploring its different flavors, applications, and challenges. Get ready to sharpen your optimization skills and join the ranks of the machine learning elite!

.It is a **function** that measures the performance of a model for any given data. **Cost Function** quantifies the error between predicted values and expected values and presents it in the form of a single real number.

After making a hypothesis with initial parameters, we calculate the Cost function. And with a goal to reduce the cost function, we modify the parameters by using the Gradient descent algorithm over the given data. Here's the mathematical representation for it:

What is Gradient Descent?

Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. The goal of gradient descent is to find the set of parameters that minimizes this discrepancy and improves the model's performance.

Example of Gradient Descent

Let's say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?

Take a moment to think about this before you read on.

The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point. If we take steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function, and the procedure is called **Gradient Ascent.**

Gradient descent was originally proposed by **CAUCHY** in 1847. It is also known as steepest descent.

The goal of the gradient descent algorithm is to minimize the given function (say cost function). To achieve this goal, it performs two steps iteratively:

1. **Compute the gradient** (slope), the first order derivative of the function at that point
2. **Make a step (move) in the direction opposite to the gradient**, opposite direction of slope increase from the current point by alpha times the gradient at that point

Alpha is called **Learning rate** – a tuning parameter in the optimization process. It decides the length of the steps.

How Does Gradient Descent Work?

1. Gradient descent is an optimization algorithm used to minimize the cost function of a model.
2. The cost function measures how well the model fits the training data and is defined based on the difference between the predicted and actual values.
3. The gradient of the cost function is the derivative with respect to the model's parameters and points in the direction of the steepest ascent.

4. The algorithm starts with an initial set of parameters and updates them in small steps to minimize the cost function.
5. In each iteration of the algorithm, the gradient of the cost function with respect to each parameter is computed.
6. The gradient tells us the direction of the steepest ascent, and by moving in the opposite direction, we can find the direction of the steepest descent.
7. The size of the step is controlled by the learning rate, which determines how quickly the algorithm moves towards the minimum.
8. The process is repeated until the cost function converges to a minimum, indicating that the model has reached the optimal set of parameters.
9. There are different variations of gradient descent, including batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, each with its own advantages and limitations.
10. Efficient implementation of gradient descent is essential for achieving good performance in machine learning tasks. The choice of the learning rate and the number of iterations can significantly impact the performance of the algorithm.

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

Ans:-Ev en though the idea behind the generation of this dataset is pretty simple, there were plenty of data points, and no noise at all, the logistic regression model performed poorly on this dataset. The reason is that the target label has no linear correlation with the features.

1. Logistic Regression

Logistic regression is an exercise in predicting (regressing to – one can say) discrete outcomes from a continuous and/or categorical set of observations. Each observation is independent and the probability *p* that an observation belongs to the class is some (& same!) function of the features describing that observation. Consider a set of *n* observations [*x_i, y_i; Z_i*] where *x_i, y_i* are the feature values for the *ith* observation. *Z_i* equals 1 if the *ith* observation belongs to the class, and equals 0 otherwise. The likelihood of having obtained *n* such observations is simply the product of the probability *p(x_i, y_i)* of obtaining each one of them separately.

(1)

The ratio *p/(1-p)* (known as the odds ratio) would be unity along the decision boundary as *p* = *1-p* = 0.5. If we define a function *f(x,y; c)* as:

(2)

then *f(x,y; c) =0* would be the decision boundary. *c* are the *m* parameters in the function *f*. And log-likelihood in terms of *f* would be:

(3)

All that is left to be done now is to find a set of values for the parameters *c* that maximize *log(L)* in Equation 3. We can either apply an optimization technique or solve the coupled set of *m* nonlinear equations *d log(L)/dc = 0* for *c*.

(4)

Either way, having *c* in hand completes the definition of *f* and allows us find out the class of any point in the feature space. That is logistic regression in a nut shell.

2. The function f(x,y; c)

Would any function for *f* work in Equation 2? Most certainly not. As *p* goes from 0 to 1, *log(p/(1-p))* goes from *-inf* to *+inf*. So we need *f* to be unbounded as well in both directions. The possibilities for *f* are endless so long as we make sure that *f* has a range from *-inf* to *+inf*.

2.1 A linear form for f(x,y; c)

Choosing a linear form such as

(5)

will work for sure and that leads to traditional logistic regression as available for use in scikit-learn and the reason logistic regression is known as a *linear* classifier.

2.2 A higher-order polynomial for f(x,y; c)

An easy extension Equation 5 would be to use a higher degree polynomial. A 2nd order one would simply be:

(6)

Note that the above formulation is identical to the linear case, *if* we treat *x^2, xy*, and *y^2* as three additional independent features of the problem. The impetus for doing so would be that we can then directly apply the API from scikit-learn to get an estimate for c. We see however in the results section that the *c* obtained this way is not as good as directly solving Equation 4 for *c*. It is a bit of a mystery as to why. But in any case solving Equation 4 is easy enough with modules from scipy. The derivatives we need for Equation 4 fall out simply as

2.3 Other generic forms for f(x,y; C)

A periodic form like *f(x,y; c) = sin(c_0x + c_2y) = 0* will not work as its range is limited. But the following will.

(7)

We can again evaluate the derivatives and solve for the roots of Equation 4 but sometimes it is simpler to just directly maximize *log(L)* in Equation 3, and that is what we will do in the simulations to follow.

3. Simulations

We generate equal number of points in the *x, y* feature space that belong to a class (*Z* = 0 when *f(x,y; c)* > small_value) and those that do not belong (*Z* = 1 when *f(x, y; c)* < small_value) as we know the explicit functional form of *f(x,y; c)*

13) . Differentiate between Adaboost and Gradient Boosting.

Ans:- **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.

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

Ans:- Bias-Variance Tradeoff is a fundamental concept in machine learning that deals with the balance between model bias and variance. In simpler terms, it refers to the tradeoff between a model's ability to accurately represent the underlying data patterns (low bias) and its susceptibility to fluctuations with changes in the training data (high variance).

How Bias-Variance Tradeoff Works:-

When building machine learning models, it's essential to understand that complex models can capture intricate patterns in the data but may also overfit to noise, resulting in high variance. On the other hand, simpler models may have high bias, leading to an oversimplified representation of the data.

The bias-variance tradeoff implies that as we increase the complexity of a model, its variance decreases, and its bias increases. Conversely, as we decrease the model's complexity, its variance increases, but its bias decreases. The goal is to find the right balance between these two aspects to create a model that performs well on new, unseen data.

Why Bias-Variance Tradeoff is Important:-

Bias-Variance Tradeoff is crucial in machine learning because it directly impacts a model's predictive performance. A model with high bias will consistently produce predictions that are far from the actual values, while a model with high variance will produce widely varying predictions for different training datasets. In both cases, the model's ability to generalize to new, unseen data is compromised.

By understanding and optimizing the bias-variance tradeoff, businesses can develop machine learning models that strike the right balance between simplicity and accuracy. This results in robust models that are less prone to overfitting and more likely to make accurate predictions on real-world data, leading to better decision-making and improved business outcomes.

The Most Important Bias-Variance Tradeoff Use Cases:-

Bias-Variance Tradeoff is applicable in various machine learning use cases, including:

- Financial Forecasting: Achieving accurate predictions for stock prices, market trends, and investment opportunities.
- Medical Diagnostics: Building reliable models for disease diagnosis and patient risk assessment.
- Customer Behavior Analysis: Understanding customer preferences and optimizing marketing strategies.
- Natural Language Processing: Developing language models for sentiment analysis, chatbots, and text generation.

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

Ans:- In our previous **Machine Learning** blog we have discussed about **SVM (Support Vector Machine)** in Machine Learning. Now we are going to provide you a detailed description of SVM Kernel and Different Kernel Functions and its examples such as linear, nonlinear, polynomial, Gaussian kernel, Radial basis function (RBF), sigmoid etc.

SVM Kernel Functions

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.

3. Kernel Rules

Define kernel or a window function as follows:

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

This value of this function is 1 inside the closed ball of radius 1 centered at the origin, and 0 otherwise . As shown in the figure below:

For a fixed xi, the function is K(z-xi)/h) = 1 inside the closed ball of radius h centered at xi, and 0 otherwise as shown in the figure below:

Examples of SVM Kernels

Let us see some common kernels used with SVMs and their uses:

4.1. Polynomial kernel

It is popular in image processing.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

where d is the degree of the polynomial.

4.2. Gaussian kernel

It is a general-purpose kernel; used when there is no prior knowledge about the data. Equation is:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

Gaussian kernel equation

4.3. Gaussian radial basis function (RBF)

It is a general-purpose kernel; used when there is no prior knowledge about the data.
Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Gaussian radial basis function (RBF)

statistics

1. Using a goodness of fit,we can assess whether a set of obtained frequencies differ from a set of frequencies

Ans:-Excepted

2. Chisquare is used to analyse

Ans:-All of these

3. What is the mean of a Chi Square distribution with 6 degrees of freedom?

Ans:6

4.Which of these distributions is used for a goodness of fit testing?

Ans:Chisquared distribution

5.Which of the following distributions is Continuou

Ans:-Binomial Distribution

6. A statement made about a population for testing purpose is called?

Ans:-Hypothesis

7. If the assumed hypothesis is tested for rejection considering it to be true is called?

Ans:-Null Hypothesis

8. If the Critical region is evenly distributed then the test is referred as?

Ans:-Two tailed

9. Alternative Hypothesis is also called as?

Ans:-research Hypothesis

10. . In a Binomial Distribution, if ‘n’ is the number of trials and ‘p’ is the probability of success, then the mean value is given by

Ans:-np