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?

   a) R-squared

   The percentage of variation in the target variable that the model explains is measured by R-squared. The values are between 0 and 1.
   • A model that exactly predicts the target variable has an R-squared of 1. When the R-squared is zero, it indicates that there is no variation in the model.
   • R-squared is simple to interpret; larger values suggest a better fit between the model and the data.

   R-squared, however, has some limitations. Good forecasts are not always ensured by a high R-squared, particularly in the case of complicated models.

   In conclusion, further analysis is necessary to ascertain the model's predictive power, even though R-squared shows how well the model matches the original data. Model efficacy for predictions is not validated by a high R-squared on its own. Additional examinations are also required.

   b) Residual Sum of Squares (RSS)

   - Total Mismatch between Response Values and Model Estimated Values is measured by Residual Sum of Squares (RSS).

   - The computation involves squaring the residuals, which signify the discrepancy between the actual and anticipated values of the dependent variable.

   - Because it reflects less error in the predictions, a lower RSS denotes a better fitting model. An RSS of 0 indicates an ideal match.
   - RSS measures the residual variability that the model is unable to account for. It illustrates the difference between the model's predicted and actual results.
   - RSS can be used to compare different models; a lower RSS indicates a better fit between the models because it indicates less uninterpreted variation in the data.
   - Variable additions and deletions affect RSS. A reduced RSS indicates that the inclusion of helpful factors has improved model fit

Thus, the choice of which is better relies on the part of model performance that you wish to highlight. R-squared is a better choice if you want to know how effectively your independent variables account for variations in the dependent variable. RSS might be a better option if you're worried about how effectively the model predicts the values of the dependent variable.

It’s frequently helpful to take into account both metrics simultaneously in order to obtain a thorough grasp of model performance.

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.

Total Sum of Squares (TSS) measures how far the answer variable deviates from its mean overall. The variation as a whole can be separated into two components: the percentage that the model explains (called the Explained Sum of Squares, or ESS) and the remaining variability that remains unaccounted for (called the Residual Sum of Squares).

TSS = ESS + RSS

Each of this represents:

a) Total Sum of Squares:

Around its mean, the dependent variable (Y)'s total variability is measured by TSS.
It measures how far the values of the observed dependent variable deviate overall from the dependent variable mean.
The total variability that the regression model aims to explain is denoted by TSS.

$$TSS=\sum i=1n(Yi-\bar{Y})2$$

b) How Sum of Squares (ESS) Are Explained

With the help of the independent variables (X) in the regression model, ESS calculates the amount of variation in the dependent variable (Y) that can be explained.
The divergence between the dependent variable's mean and its expected values is quantified.
A regression model's explanation of Y's variability is represented by ESS.

$$ESS=\sum i=1n(\hat{Y}i-\bar{Y})2$$

c) Summary of Residual Squares (RSS):

Regression model's independent variables (X) do not account for all of the variability in the dependent variable (Y), which is measured by RSS.
The difference between the dependent variable's observed values and its expected values is quantified.

In the context of regression models, RSS stands for the unexplained variability, or mistakes.

$$RSS=\sum i=1n(Yi-Y\hat{}i)2$$

In the formula

Total variability in the dependent variable is denoted by TSS=ESS+RSS, and it is divided into two components: ESS, or the variability that the model explains, and RSS, or the residuals, or the unexplained variability. This formula shows how the overall variability in the dependent variable can be divided into the variability that the model can account for and the variability that the model is unable to account for.

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

Regularization is a machine learning approach that helps make models more generalizable by preventing overfitting. A model performs poorly on unobserved data due to overfitting, which happens when the model learns to capture the noise in the training data rather than the underlying patterns. In order to encourage the model to learn simpler patterns that have a higher chance of generalizing effectively to new data, regularization adds a penalty term to the loss function.

The main requirement for regularization is :

- Complexity of Models: Machine learning models with a lot of parameters, like polynomial regression or neural networks, are better able to retain the training set. These models may perform poorly on unseen data if regularization is not applied, as they may get unduly complicated and fit the noise in the data.
- Features with Strong Correlations: The model may overfit when it learns redundant information from the dataset's highly associated features. The model is encouraged to choose a subset of pertinent features and the influence of multicollinearity is lessened using regularization techniques like L1 and L2 regularization (e.g., Lasso and Ridge regression).
- Noise in Data: Unimportant characteristics or noise in training data frequently prevent it from generalizing well to fresh data. Regularization improves generalization performance by helping the model concentrate on the most crucial features and lessening its sensitivity to noise.
- Limited Data: The amount of data that is readily available is often limited in real-world circumstances. Overfitting is more likely when there is little data available since the model could become accustomed to the training instances rather than discovering the underlying patterns. Regularization limits the complexity of the model, which helps avoid overfitting.
  To prevent overfitting, enhance model generalization, and guarantee robust performance on unknown data, regularization is crucial in machine learning, particularly in situations with sparse data or complex models.

4. What is Gini–impurity index?

The uncertainty or impurity in a collection of data points is measured by the Gini impurity index. It is frequently used in decision tree algorithms to assess the quality of a split in the data based on a categorical target variable, especially in binary classification jobs.

When a randomly selected element from the set is labeled based on the distribution of labels in the subset, the likelihood that it will be erroneously categorized is measured by the Gini impurity index. When a subset's members are entirely of the same class, then the Gini impurity score of 0 indicates perfect purity. When the score is 0.5, however, it represents maximum impurity, which denotes an equal distribution of elements across all classes.

From a mathematical perspective, the following formula yields the Gini impurity index for a given collection of data:

$$\text{Gini}=1-\sum_{i=1}^{C}(p_i)^2$$

Where

- C is the no of classes in target variable
- *pi is the probability of element being classified as class i.*

Lower values in the Gini impurity index indicate more purity and better splits in the data. The measure has a range of 0 to 0.5. Choosing splits that produce subgroups with a preponderance of one class label will reduce the Gini impurity index in decision tree algorithms, resulting in more precise and effective classification.

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

Certainly, when given unrestricted growth, unregularized decision trees are susceptible to overfitting. A model is said to be overfitted when it starts to learn from the training data how to represent noise or erratic oscillations instead of the underlying relationships or patterns. When applied to new data, unregularized decision trees perform poorly because they tend to become extremely complicated and capture the quirks of the training set.

- High Variance: The training data can be precisely separated by very complicated decision boundaries that decision trees are able to learn. With no limits, the tree can get deep enough to learn the entire training set—noise and outliers included—

resulting in a high variance. This implies that the learnt tree structure may alter significantly in response to even minor modifications in the training set, decreasing the model's resistance to data fluctuations.

- Restrictions on Sample Size: Unregularized decision trees are more prone to overfitting in datasets with restricted sample sizes since there isn't as much data available for the model to identify the underlying patterns. If the tree is overfit to the training set of data with fewer samples, it could capture noise instead of real information.
- Recursive Partitioning: To optimize the purity of the resulting subsets, decision trees divide the feature space recursively using the training data. The tree can divide the data indefinitely without facing growth limits, until every leaf node has only one training instance, thus forgetting the training set and associated noise.

Decision trees can be made less prone to overfitting by using regularization techniques like pruning, setting a minimum amount of samples needed to divide a node, or restricting the maximum depth of the tree. Better generalization performance on unobserved data results from these strategies, which limit the tree's complexity and encourage it to capture the most significant patterns in the data.

6. What is an ensemble technique in machine learning?

In order to increase the predictive model's overall performance and durability, machine learning ensemble techniques integrate the predictions of several base models. Using the diversity of numerous models instead of just one, ensemble methods are able to produce predictions that are often more accurate, stable, and generalized than single models.

Various kinds of ensemble approaches exist, such as:

- Bagging, also known as bootstrap aggregating, is the process of training multiple copies of the same base model on various training data subsets, usually sampled using replacement data (bootstrap samples). By averaging or selecting a majority vote from each model's projections, the final prediction is determined. Popular ensemble technique Random Forest is based on bagging; decision trees are trained on bootstrapped samples and aggregated by voting or averaging.
- Stacking (Stacked Generalization): Stacking, sometimes referred to as blended generalization or meta-learner, is the process of combining the predictions of several underlying models. Stacking trains a meta-model on the outputs of the base models, which act as input features, as opposed to just averaging or voting the predictions. Stacking is able to optimally integrate the predictions of various base models by capturing their complimentary capabilities.
- Boosting: Boosting techniques adaptively change the weights of training instances to concentrate on examples that are challenging to accurately categorize. They achieve this by successively training a number of weak learners (models that perform marginally better than random guessing). By highlighting the training examples that were incorrectly identified in earlier iterations, boosting techniques like Gradient Boosting Machine (GBM) and AdaBoost (Adaptive Boosting) repeatedly create a powerful learner.

- Voting: Using a simple majority vote for classification tasks or an average method for regression problems, voting ensembles aggregate the predictions of several base models. Voting ensembles can be classified as either soft (weighted average of estimated probability) or hard (simple majority vote).

  Because they frequently result in better prediction accuracy, less overfitting, and more stability when compared to individual models, ensemble approaches are commonly utilized in machine learning. When base models have complimentary and varied strengths and shortcomings, they work especially well.

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

In machine learning, bagging and boosting are two ensemble algorithms that combine the predictions of several base models to enhance the performance of predictive models. They take different methods, nevertheless, when it comes to training the base models and merging their predictions:

a) Method of Training:

   - Training multiple instances of the same base model (usually decision trees) on various subsets of the training data, usually sampled with replacement (bootstrap samples), is known as bagging (Bootstrap Aggregating). Every base model undergoes independent training from the others. The average of each base model's predictions (for regression) or a majority vote (for classification) yields the final forecast.

   - Boosting: To target examples that are hard to categorize correctly, adaptively alter the weights of training instances as a series of weak learners (models that perform marginally better than random guessing) are trained successively. With the intention of fixing the errors of the earlier models, each base model is trained again. By combining all of the base models' weighted predictions, usually with a weighted total, the final forecast is obtained.

b) Instance Weighting:

   - Bagging: Using replacement, each base model is trained on a random subset of the training set in bagging. Every training instance has the same weight, and every base model is trained separately from the others.
   - Boosting: In boosting, the weights of each training instance are initially set to be identical, but they are subsequently modified iteratively in response to the prior models' performance. In future iterations, instances with higher residuals or incorrect classifications are given greater weights, which enables the models to concentrate more on examples that are challenging to categorize.

To sum up, bagging and boosting differ primarily in how they aggregate the predictions of various base models and how they approach training. In contrast to boosting, which trains models sequentially and modifies training instance weights to concentrate on challenging

examples, bagging trains independent base models on random subsets of the data and aggregates their predictions through voting or averaging.

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

In Random Forests, out-of-bag (OOB) error, also called out-of-bag estimation, is a technique for estimating model performance without requiring an additional validation set. It uses the data points that are absent from the bootstrap samples used to train each individual tree to offer an unbiased estimate of the generalization error of the model.

Bootstrap Sampling: A bootstrap sample is taken from the original dataset with replacement after every decision tree in the Random Forest is trained. As a result, some data points might receive many samples, while others might receive none at all.

Prediction: Each tree is trained and then its predictions are derived from the out-of-bag data points. After then, all of the trees in the Random Forest have their forecasts combined.

Out-of-Bag Data: The data points that are not part of the bootstrap sample for each decision tree are known as the out-of-bag (OOB) data for that tree. These data items were not utilized in the tree's training process.

Error Estimation: The out-of-bag error is computed by comparing the aggregated predictions for the out-of-bag data with the true labels of those data points. The Random Forest's ability to generalize to previously undiscovered data is gauged by this error estimate.

Typically, the out-of-bag error estimate is used to adjust Random Forest hyperparameters, including the number of trees or the maximum tree depth, using methods like cross-validation. Conveniently and effectively, it offers a trustworthy approximation of the model's performance on unknown data without requiring an additional validation set.

9. What is K-fold cross-validation?

By dividing the original dataset into K equal-sized subsets, or folds, K-fold cross-validation is a technique used to evaluate the performance of a machine learning model. A separate fold is used as the validation set while the remaining folds are used as the training set during the K iterations of training and evaluating the model. Through the use of several data subsets for training and evaluation, the model's performance can be estimated with more accuracy.

The K-fold works in the following way:

- Partitioning the Dataset: K equal-sized subsets, or folds, are randomly selected from the original dataset.
- Training and Validation: Using a separate fold as the validation set and the remaining K-1 folds as the training set, the model is trained K times. For instance, the first fold is used as the training set while the remaining K-1 folds

are utilized as the validation set in the first iteration. The second fold serves as the validation set in the subsequent iteration, and so forth.

- Performance Evaluation: The validation set is used to assess the model's performance following each iteration. Depending on the kind of problem, this usually entails determining a performance indicator like accuracy, precision, recall, F1-score, or mean squared error (classification or regression).
- Aggregate Performance: A single estimate of the model's performance is obtained by averaging or combining the performance measures acquired from each iteration.

Though the amount of K can be changed depending on the size of the dataset and available computing power, common possibilities for K include 5- or 10-fold cross-validation.

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

The process of choosing the best values for a machine learning model's hyperparameters is referred to as hyperparameter tuning. Parameters known as hyperparameters are those that are predetermined and not determined by the training process based on data. Neural network learning rate, random forest tree count, and linear model regularization parameter are a few examples of hyperparameters.

By determining the combination of hyperparameters that yields the highest performance on a validation set or through cross-validation, hyperparameter tweaking is done to enhance the performance of a machine learning model. Finding hyperparameters that produce the most reliable and accurate predictions while still having good generalization to new data is the aim.

Other advantages due to hyperparameter are:

- Enhanced Performance: A machine learning model's performance can be greatly influenced by the selection of its hyperparameters. By adjusting the hyperparameters, one can choose values that improve performance measures like mean squared error, accuracy, precision, and recall.
- Generalization :

Hyperparameter tuning: By identifying hyperparameters that provide models that effectively generalize to new data, hyperparameter tuning helps avoid overfitting. Overfitting is when a model learns to represent the training data's noise or erratic fluctuations instead of its underlying relationships or patterns.

- Optimization: The process of finding the ideal set of hyperparameters inside a given search space is known as hyperparameter tuning. Using methods like grid search, random search, or Bayesian optimization, this procedure can be automated.

To optimize model performance and enable generalization to previously unknown data, hyperparameter tuning is, in general, an essential stage in the machine learning pipeline. Higher prediction accuracy and predictability can be attained by machine learning models by choosing the optimal hyperparameters.

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

An excessive learning rate in gradient descent can cause a number of problems that impede the algorithm's convergence and result in subpar performance. Among the primary concerns are:

- Overshooting the Minimum: When an algorithm approaches the loss function's minimum, it may take unduly large steps if the learning rate is high. The optimal solution may not converge as a result of overshooting the minimum and fluctuating around it.
- Divergence: When an algorithm experiences a high learning rate, it may diverge, causing the parameters to move away from the loss function's minimum rather than towards it. This can lead to erratic and unexpected behavior, which defeats the purpose of optimization.
- Updates from Gradient Descent: In this method, the parameter updates are based on the gradient of the loss function. The algorithm may overshoot or fluctuate about the loss function's minimum while learning at a high rate because the updates may grow excessively large.
- Instability: High learning rates can cause the training process to become unstable since the optimization process becomes extremely sensitive to even minute changes in the gradient or the data. As a result, the method may yield inconsistent results and have difficulty achieving convergence.
- Superfluous Computational Effort: High learning rates may lead to the algorithm taking big steps that aren't essential, which could need more iterations to reach the best answer. The model's training process may need more computing power and time as a result.

It's critical to carefully choose a suitable learning rate for gradient descent in order to minimize these problems. Methods like learning rate schedules, adaptive learning rates (like AdaGrad, RMSprop, and Adam), and validation-based tuning can assist in determining the ideal learning rate that strikes a compromise between stability and convergence speed. Finding and fixing any problems that may occur can also be aided by observing and experimenting with various learning rates during the training process.

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

A linear relationship between input feature and the log-odds of belonging to a specific class is assumed by the linear classification process known as logistic regression. This presumption, however, restricts its use to the classification of non-linear data, where the decision border between classes is non-linear. Underfitting, in which the model performs poorly and is unable to fully represent the complexity of the data, might result from the linear decision boundary of logistic regression.

On the other hand, models that can represent intricate relationships and patterns are necessary for non-linear data. For these kinds of tasks, decision trees, random forests, support vector machines (SVMs) with non-linear kernels, and neural networks are more appropriate.

Accurate classification of non-linear data is possible thanks to these models' ability to capture non-linear decision boundaries.

For example, take a look at a dataset where the classes aren't easily distinguished by a straight line. Subpar results would arise from logistic regression's inability to fully capture the complex relationships between features and classes. Non-linear models, on the other hand, can efficiently categorize non-linear data because of their capacity to learn intricate decision boundaries. Consequently, more expressive and flexible models are needed for non-linear data in order to obtain correct classification, even though logistic regression performs well in linear classification tasks.

13. Differentiate between Adaboost and Gradient Boosting.

While both ensemble learning techniques—Adaptive Boosting, or AdaBoost, and Gradient Boosting—are employed for classification and regression tasks, they take different approaches to turning weak learners into strong learners.

Gradient Boosting:

By gradually adding weak learners to fix the mistakes of the earlier models, gradient boosting creates a strong learner.
Weak learners are iteratively fitted to the residuals of the prior models using an optimization approach akin to gradient descent in order to minimize the loss function.
Gradient Boosting is a powerful technique that can handle both regression and classification tasks. Usually, it uses decision trees as weak learners.
It can capture complicated associations in the data and is less prone to overfitting than AdaBoost, although it could need more hyperparameter tuning.

AdaBoost:

AdaBoost is a framework for improving model performance through iterative training of weak learners, such as decision trees.
Incorrectly classified examples are given bigger weights in later iterations. Training instances are given weights based on how accurately they are classified.
A weighted sum is utilized to aggregate weak learners, whereby the performance of each weak learner throughout training establishes its respective weight.
Because it concentrates on fixing misclassifications in each iteration, AdaBoost is often more sensitive to noisy data and outliers.

In conclusion, while both Gradient Boosting and AdaBoost are ensemble techniques that integrate weak learners, they vary in terms of how they handle misclassifications that occur during training.

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

The ability of a model to accurately identify the underlying patterns in the data (bias) and to generalize to new data (variance) is known as the bias-variance trade-off, and it is a key idea in machine learning.

The term "bias" describes the inaccuracy that results from using a simplified model to approximate a real-world problem. Poor performance on both training and test data might result from a high bias model's tendency to underfit the training set and potential inability to identify underlying patterns.

Variance: The sensitivity of the model to variations in the training set of data is known as variance. When a high variance model is applied to training data, it has a tendency to overfit the data, misinterpreting noise and erratic fluctuations for real patterns. High variance models frequently perform badly when applied to new data, even though they may perform well on training data.

The trade-off between bias and variance occurs because decreasing bias usually results in higher variance and vice versa. Building models that effectively capture the underlying patterns in the training data and transfer well to new data requires striking the correct balance between variance and bias. Approaches including ensemble learning, regularization, and cross-validation are frequently employed to balance variance and bias and enhance model performance.

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

- The RBF kernel is a non-linear mapping of the input characteristics into a higher-dimensional space. It is also known as the Radial Basis Function kernel. The similarity of feature vectors is calculated using a Gaussian function, whose width is determined by the kernel parameter (gamma). A broad variety of classification problems can benefit from the flexibility and ability of the RBF kernel to capture complex non-linear decision boundaries.
- The most straightforward kernel utilized in SVM is the linear kernel. A linear decision boundary is essentially defined by computing the dot product between the feature vectors. It performs best with data that can be divided into classes by a straight line or plane in the feature space and is appropriate for data that can be divided linearly.
- Another type of non-linear kernel is the polynomial kernel, which calculates the dot product of feature vectors raised to a specific power (degree). Through the consideration of every potential polynomial combination of the input features, non-linearity is introduced. Higher degrees of the polynomial transformation enable the model to record more intricate decision boundaries. The degree parameter governs this transformation. Since it can handle relatively complex non-linear interactions between features, the polynomial kernel is appropriate for data that is not linearly separable.