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:- Both R-squared (R^2) and Residual Sum of Squares (RSS) serve as measures of model fit. R^2 quantifies the proportion of variance in the dependent variable explained by the regression equation, with higher values indicating a better fit. On the other hand, RSS measures the total squared differences between observed and predicted values, aiming to minimize these differences. While R^2 is intuitive, RSS provides an absolute measure but lacks comparability across models with different sample sizes or numbers of predictors. Hence, in multiple regression, Adjusted R-squared is preferred as it adjusts for the number of predictors, enabling fair comparison between models with varying numbers of independent variables without favoring more complex models.

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 represents the total variability in the dependent variable (Y) and is calculated as the sum of the squared differences between each observed value of Y and the mean of Y

ESS measures the variability in the dependent variable (Y) explained by the regression model. It is the sum of the squared differences between the predicted values of Y and the mean of Y

RSS quantifies the unexplained variability in the dependent variable (Y) that is not captured by the regression model. It is the sum of the squared differences between the observed values of Y and the predicted values of Y

The relationship between these metrics can be described by the following equation, known as the decomposition of variance TSS = ESS + RSS

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

Ans:- Regularization is a crucial technique in machine learning used to prevent overfitting and improve the generalization performance of models. Overfitting occurs when a model learns to capture noise and fluctuations in the training data, resulting in poor performance on unseen data. Regularization helps address this issue by adding a penalty term to the model's objective function, discouraging overly complex models that are sensitive to fluctuations in the training data.

Regularization is an essential tool in the machine learning toolbox, helping to improve the robustness, stability, and generalization performance of models, particularly in situations where the risk of overfitting is high.

4. What is Gini–impurity index?

Ans:- The Gini impurity index, pivotal in decision tree algorithms, quantifies the impurity or disorder of a set of elements in a classification task, reflecting the uncertainty in predicting class labels. Calculated based on the probabilities of each class within the set, it ranges from 0 (perfect purity) to 1 (maximum impurity). Decision trees utilize Gini impurity to evaluate potential splits on features, selecting splits that minimize impurity in resulting child nodes, thereby creating partitions that maximize homogeneity and facilitate accurate classification. Nodes with lower impurity values represent subsets with clearer separation between classes, while higher impurity values indicate subsets with more mixed distributions. Efficient and robust, the Gini impurity index aids in model interpretation and understanding, contributing to the creation of effective classification models.

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

Ans:- Yes, unregularized decision trees are prone to overfitting. Overfitting occurs when a model learns to capture noise and random fluctuations in the training data, rather than the underlying patterns that generalize well to unseen data. Decision trees, especially when left unregularized, have a tendency to become highly complex and deeply branched to perfectly fit the training data. This complexity allows them to capture even the smallest details and idiosyncrasies of the training data, including noise, outliers, and irrelevant features. As a result, the model may not generalize well to new data, leading to poor performance on unseen observations. Regularization techniques, such as pruning, limiting the maximum depth of the tree, or using ensemble methods like random forests, are employed to mitigate overfitting by constraining the complexity of the decision tree and promoting simpler models that generalize better.

6. What is an ensemble technique in machine learning?

Ans:- An ensemble technique in machine learning involves combining the predictions of multiple individual models to produce a more accurate and robust final prediction than any single model alone. These methods harness the collective intelligence of diverse models to overcome limitations and improve generalization performance. Techniques like bagging, boosting, stacking, and voting are commonly employed. Bagging creates multiple models trained on different subsets of the data and aggregates their predictions, while boosting iteratively trains models to focus on correcting errors of previous ones. Stacking combines predictions of diverse models using a meta-learner, while voting combines predictions through majority voting or averaging. Ensembles are effective in reducing overfitting, capturing complex relationships in data, and enhancing predictive performance across various domains and applications.

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

Ans:- The main difference between Bagging and Boosting techniques lies in their approach to creating ensemble models and how they handle the training of individual base models:

Bagging: Bagging involves training multiple base models independently on different subsets of the training data, which are randomly sampled with replacement (bootstrap sampling). Each base model is trained on a subset of the data, and their predictions are aggregated (e.g., averaged) to make the final prediction. Bagging aims to reduce variance by averaging out the predictions of multiple models, which helps to stabilize and improve overall prediction accuracy. Random Forest is a popular ensemble method based on bagging, where decision trees are trained on bootstrap samples of the data and combined through averaging or voting.

Boosting: Boosting is a sequential ensemble technique where base models are trained iteratively, with each subsequent model focusing on correcting the errors made by the previous ones. Each base model is trained on the full training dataset, but the training instances are weighted, with higher weights assigned to instances that were misclassified by previous models. Boosting algorithms aim to reduce bias and improve overall prediction accuracy by emphasizing difficult-to-classify instances during training. Examples of boosting algorithms include AdaBoost (Adaptive Boosting), Gradient Boosting Machines (GBM), and XGBoost.

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

Ans:- In Random Forests, the out-of-bag error is an estimate of the model's performance on unseen data, computed using the data points that were not included in the bootstrap samples used to train each decision tree in the forest. During the bootstrapping process, about one-third of the original dataset on average is left out (out-of-bag) and not used for training in each iteration. These out-of-bag data points can then be used to evaluate the performance of the model. For each data point, the Random Forest calculates the prediction using only the trees that did not include that data point in their bootstrap samples. The OOB error is then computed as the average error across all out-of-bag predictions. This approach allows for a robust estimate of the model's performance without the need for an additional validation dataset, making it particularly useful for assessing model performance in situations where data is limited.

9. What is K-fold cross-validation?

Ans:- K-fold cross-validation is a technique used to assess the performance of a machine learning model by partitioning the dataset into K subsets (or folds) of approximately equal size. The model is then trained and evaluated K times, each time using a different fold as the validation set and the remaining folds as the training set. This process allows every data point to be used for both training and validation, ensuring that the model is evaluated on a diverse set of data. The performance metrics (e.g., accuracy, error) obtained from each iteration are averaged to obtain a single performance estimate, which represents the overall performance of the model. K-fold cross-validation helps to provide a more reliable estimate of the model's performance compared to a single train-test split, as it reduces the variance in the performance metric caused by the randomness of the data split. It is commonly used for model selection, hyperparameter tuning, and assessing the generalization ability of machine learning models.

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

Ans:- Hyperparameter tuning in machine learning refers to the process of selecting the optimal values for the hyperparameters of a model. Hyperparameters are configuration settings that are external to the model and cannot be directly estimated from the data, unlike model parameters which are learned during the training process. Examples of hyperparameters include the learning rate in gradient descent algorithms, the number of hidden layers in a neural network, or the depth of a decision tree.

Hyperparameter tuning is done to optimize the performance of the model on unseen data. By selecting the best combination of hyperparameters, we aim to improve the model's ability to generalize to new, unseen data and enhance its predictive accuracy. This process involves searching through a predefined hyperparameter space, often using techniques such as grid search, random search, or more advanced optimization algorithms like Bayesian optimization. Hyperparameter tuning is crucial for maximizing the effectiveness of machine learning models and achieving better performance across various tasks and datasets.

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

Ans:- When using Gradient Descent optimization algorithms, setting a learning rate that is too large can lead to several issues.

Divergence: With a large learning rate, the updates to the model parameters can become excessively large, causing the optimization process to diverge. Instead of converging towards the optimal solution, the parameters may oscillate or diverge away from it, preventing the algorithm from finding the minimum of the loss function.

Instability: A large learning rate can result in unstable training dynamics, where the model parameters exhibit erratic behavior during optimization. This instability can make it challenging to interpret the training process and diagnose issues with the model's performance.

Overshooting the Minimum: High learning rates can cause the optimization algorithm to overshoot the minimum of the loss function. Instead of smoothly descending towards the minimum, the updates may jump over it, leading to oscillations or bouncing around the minimum without converging.

Slow Convergence: Surprisingly, setting the learning rate too high can also slow down the convergence of the optimization algorithm. While larger learning rates allow for faster updates, they can cause the optimization process to oscillate or diverge, requiring more iterations to reach convergence.

Difficulty in Fine-Tuning: Large learning rates can make it challenging to fine-tune model hyperparameters or to achieve optimal performance. Tuning other hyperparameters, such as the number of epochs or the regularization strength, may become less effective if the learning rate is too large, as it dominates the training dynamics.

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

Ans:- Logistic Regression is a linear classification algorithm, meaning it models the relationship between the input features and the output class using a linear decision boundary. While Logistic Regression is effective for linearly separable data, it may not perform well for non-linear data.

The main limitation of Logistic Regression in handling non-linear data is its inability to capture complex, non-linear relationships between the input features and the output class. Since the decision boundary of Logistic Regression is linear, it cannot adequately model non-linear decision boundaries that may exist in the data.

13. Differentiate between Adaboost and Gradient Boosting.

AdaBoost: AdaBoost, short for Adaptive Boosting, is an ensemble learning technique aimed at enhancing the performance of machine learning models, particularly in classification tasks. It operates by iteratively training a sequence of weak learners, typically shallow decision trees, on modified versions of the training data. In each iteration, AdaBoost adjusts the weights of misclassified instances, placing greater emphasis on those instances that are difficult to classify correctly. By sequentially focusing more on the misclassified instances in subsequent rounds, AdaBoost gradually improves the overall model's performance. This approach allows AdaBoost to effectively combine multiple weak learners into a strong learner that achieves better classification accuracy than any individual weak learner alone.

Gradient Boosting: Gradient Boosting is another powerful ensemble learning technique used for improving the predictive performance of machine learning models. Unlike AdaBoost, Gradient Boosting builds an ensemble of models by sequentially fitting weak learners to the residuals (errors) of the previous model. In each iteration, Gradient Boosting aims to minimize the loss function by updating the model's parameters in the direction that reduces the error of the ensemble. This iterative process allows Gradient Boosting to create a strong learner that gradually reduces the error of the ensemble, focusing on minimizing the loss function directly rather than adjusting instance weights. Gradient Boosting can use various base learners, including decision trees, shallow trees, or even linear models, providing flexibility in modeling complex relationships in the data. Overall, Gradient Boosting is effective for building robust models that generalize well to unseen data across different machine learning tasks.

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

Ans:- The bias-variance tradeoff is a fundamental concept in machine learning that describes the balance between the bias and variance of a model and its impact on predictive performance.

Bias refers to the error introduced by approximating a real-world problem with a simplified model. A high bias model tends to oversimplify the problem, leading to systematic errors and underfitting. In other words, the model is not able to capture the underlying patterns in the data, resulting in poor performance on both the training and test datasets.

Variance refers to the variability of a model's predictions across different training datasets. A high variance model is overly sensitive to fluctuations in the training data, capturing noise and idiosyncrasies specific to the training dataset. This can lead to overfitting, where the model performs well on the training data but generalizes poorly to unseen data.

The bias-variance tradeoff arises because decreasing bias typically increases variance, and vice versa. Finding the right balance between bias and variance is crucial for building models that generalize well to unseen data.

High Bias, Low Variance Models: These models are generally simple and have few parameters. They may underfit the training data, leading to high bias and low variance. Examples include linear models or models with few hidden layers in neural networks.

Low Bias, High Variance Models: These models tend to be more complex and have a large number of parameters. They may capture the noise and fluctuations in the training data, leading to low bias but high variance. Examples include decision trees with large depth or over-parameterized neural networks.

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

Ans:- Linear Kernel: The linear kernel is the simplest kernel used in SVM. It computes the dot product between two feature vectors, effectively measuring the similarity between them in the original feature space. The decision boundary created by the linear kernel is a straight line in the input space.

RBF Kernel: The RBF kernel is a popular non-linear kernel used in SVM. It maps the input features into a higher-dimensional space using a Gaussian (radial basis) function. The RBF kernel is capable of capturing complex, non-linear relationships between features, allowing SVM to create more flexible decision boundaries. It is often used when the relationship between features and the target variable is non-linear or when the data is not linearly separable in the original feature space.

Polynomial Kernel: The polynomial kernel is another non-linear kernel used in SVM. It computes the similarity between two feature vectors by raising the dot product of the vectors to a certain power (degree). The polynomial kernel allows SVM to model non-linear decision boundaries by projecting the data into a higher-dimensional space defined by polynomial functions. The degree parameter controls the complexity of the decision boundary: higher degrees allow for more complex decision boundaries, but may also lead to overfitting if not properly tuned.