<u>Ans</u>

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?**

   R-squared is a better measure of goodness of fit in regression models than the Residual Sum of Squares (RSS).

   R-squared measures the proportion of variance in the dependent variable that is explained by the independent variable(s) in the model. It provides a measure of how well the model fits the data and how much of the variation in the dependent variable can be explained by the independent variable(s). R-squared ranges from 0 to 1, with higher values indicating a better fit.

   On the other hand, RSS measures the total sum of squared differences between the predicted values and the actual values of the dependent variable in the model. It represents the amount of unexplained variation in the dependent variable that is left after the model has been fit to the data.

   While RSS is a useful measure for evaluating the fit of the model, it does not take into account the total variability of the dependent variable, and it does not provide a standardized measure of goodness of fit. In contrast, R-squared provides a standardized measure of the goodness of fit, and it can be used to compare the fit of different models with the same dependent variable.

   Therefore, R-squared is a better measure of goodness of fit in regression models than the Residual Sum of Squares (RSS) because it provides a standardized measure of the proportion of variance in the dependent variable that is explained by the independent variable(s) in the model.

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

   TSS (Total Sum of Squares), ESS (Explained Sum of Squares), and RSS (Residual Sum of Squares) are three important metrics used in regression analysis.

   TSS represents the total variation in the dependent variable (y) and is calculated as the sum of the squared differences between the actual values of y and the mean of y, represented by the equation:

$TSS = \Sigma(y - \bar{y})^2$

where y is the actual value of the dependent variable, ȳ is the mean of the dependent variable, and Σ represents the sum of the values over all observations.

ESS represents the amount of variation in the dependent variable that is explained by the independent variable(s) in the model and is calculated as the sum of the squared differences between the predicted values of y and the mean of y, represented by the equation:

$ESS = \Sigma(y\text{-hat} - \bar{y})^2$

where y-hat is the predicted value of the dependent variable, ȳ is the mean of the dependent variable, and Σ represents the sum of the values over all observations.

RSS represents the amount of variation in the dependent variable that is not explained by the independent variable(s) in the model and is calculated as the sum of the squared differences between the actual values of y and the predicted values of y, represented by the equation:

$RSS = \Sigma(y - y\text{-hat})^2$

where y is the actual value of the dependent variable, y-hat is the predicted value of the dependent variable, and Σ represents the sum of the values over all observations.

These three metrics are related to each other by the following equation:

TSS = ESS + RSS

This equation represents the fundamental principle of regression analysis, which states that the total variation in the dependent variable can be partitioned into the variation that is explained by the independent variable(s) in the model (ESS) and the variation that is not explained by the independent variable(s) (RSS).

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

Regularization is an important technique used in machine learning to prevent overfitting and improve the performance of a model. Overfitting occurs when a model fits the training data too well and as a result, it does not generalize well to new data. Regularization helps to reduce overfitting by adding a penalty term to the cost function of the model, which discourages the model from fitting the training data too closely.

Regularization can be achieved using different methods, such as L1 regularization, L2 regularization, and Elastic Net regularization. L1 regularization adds a penalty term to the cost function that is proportional to the absolute value of the model weights, while L2 regularization adds a penalty term that is proportional to the square of the model weights. Elastic Net regularization combines both L1 and L2 regularization.

Regularization helps to reduce the complexity of the model and prevents it from fitting the noise in the training data. It also improves the generalization performance of the model by reducing the variance and increasing the bias. Regularization is especially useful in situations where the amount of training data is limited or when the input features are highly correlated.

Overall, regularization is an important technique in machine learning that helps to improve the performance of a model and prevent overfitting.

4. What is Gini–impurity index?

Gini impurity is a measure of impurity or uncertainty used in decision tree algorithms for classification problems. The Gini impurity index measures the probability of incorrectly classifying a randomly chosen element in the dataset if it were labeled according to the distribution of labels in the subset.

The Gini impurity index is calculated by summing the probabilities of each class squared, subtracting the sum from one, and taking the result as the impurity index. A low Gini impurity index means that the subset is pure, with most elements belonging to the same class, while a high Gini impurity index indicates that the subset is impure, with elements belonging to different classes.

The Gini impurity index is used to evaluate the quality of a split in a decision tree algorithm. When splitting a node in a decision tree, the algorithm selects the feature and the threshold that results in the lowest Gini impurity index for the resulting subsets. The algorithm continues splitting the nodes until the subsets are pure or until a stopping criterion is met.

Overall, the Gini impurity index is a measure of impurity used in decision tree algorithms for classification problems to evaluate the quality of a split and determine the best feature and threshold for splitting a node.

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

Yes, unregularized decision trees are prone to overfitting. Decision trees are a non-parametric algorithm, meaning that they can capture complex relationships between input features and the target variable without making any assumptions about the underlying distribution of the data. However, this flexibility can also lead to overfitting, where the model becomes too complex and captures noise or idiosyncrasies in the training data that are not present in the test data.

Unregularized decision trees can easily overfit because they have a tendency to create nodes and branches to fit each individual training data point, resulting in a complex and deep tree that can fit the training data perfectly but generalize poorly to new data. This overfitting can be mitigated by pruning the tree, which involves removing some of the nodes and branches that do not improve the performance of the model on the test data.

Regularization techniques, such as restricting the depth of the tree, setting a minimum number of samples required to split a node, or adding a penalty term to the cost function, can also help to prevent overfitting in decision trees. Regularization techniques can reduce the complexity of the model and prevent it from capturing noise or idiosyncrasies in the training data, improving its ability to generalize to new data.

6. What is an ensemble technique in machine learning?

Ensemble techniques in machine learning involve combining the predictions of multiple individual models to create a more accurate and robust model. The basic idea behind ensemble methods is that by combining several models that are individually weak but have different strengths and weaknesses, the resulting model will be stronger and more accurate than any of the individual models.

There are several types of ensemble techniques, including:

a. Bagging (Bootstrap Aggregating): In bagging, multiple models are trained on different subsets of the training data using bootstrapping (sampling with replacement), and the final prediction is made by averaging the predictions of all the models.

b. Boosting: In boosting, multiple weak models are trained sequentially, with each model trying to correct the errors of the previous model. The final prediction is made by weighting the predictions of all the models based on their performance.

c. Stacking: In stacking, multiple models are trained on the training data, and their predictions are used as input to a meta-model, which learns to combine the predictions of the individual models.

Ensemble techniques can improve the accuracy and robustness of a model by reducing the variance and bias of the individual models, improving the ability of the model to generalize to new data and reducing the risk of overfitting. Ensemble methods have been used successfully in a wide range of machine learning applications, including classification, regression, and anomaly detection.

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

Bagging (Bootstrap Aggregating) and Boosting are both ensemble techniques in machine learning, but they differ in how they combine multiple models to improve the overall performance.

Bagging works by training multiple models independently on different subsets of the training data, using bootstrapping (sampling with replacement). The final prediction is made by aggregating the predictions of all the models, typically by averaging or taking the majority vote. Bagging is effective in reducing the variance of the individual models and improving the ability of the model to generalize to new data.

Boosting, on the other hand, works by sequentially training multiple weak models, with each model trying to correct the errors of the previous model. The final prediction is made by weighting the predictions of all the models based on their performance. Boosting is effective in reducing the bias of the individual models and improving the accuracy of the model on the training data.

In summary, the main difference between Bagging and Boosting is that Bagging uses multiple independent models trained on different subsets of the data, while Boosting uses a sequence of models trained on the same data, with each model trying to improve on the errors of the previous model. Bagging reduces variance and overfitting, while Boosting reduces bias and underfitting.

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

Out-of-bag (OOB) error in random forests is a method for estimating the performance of the model without the need for a separate validation set. In random forests, each decision tree in the ensemble is trained on a bootstrap sample of the training data, meaning that some data points are not included in the sample for each tree. The OOB samples are the data points that are not included in the bootstrap sample for a particular tree.

To estimate the OOB error, each data point in the training set is passed through all the decision trees in the ensemble, and the OOB prediction is calculated by averaging the predictions of the trees that did not use that data point in their training. The OOB error is then calculated as the average error of the OOB predictions over all the data points.

The OOB error is a useful measure of the performance of the random forest model because it provides an estimate of the generalization error without the need for a separate validation set. It is also efficient because it reuses the training data that was not included in the bootstrap sample for each tree. By comparing the OOB error with the validation error, we can check whether the model is overfitting or underfitting the data.

9. What is K-fold cross-validation?

K-fold cross-validation is a technique for evaluating the performance of a machine learning model by dividing the training data into K equally sized folds or partitions. The model is trained on K-1 folds and validated on the remaining fold, which is held out as a validation set. This process is repeated K times, with each of the K folds used once as the validation set.

The K-fold cross-validation technique provides a more reliable estimate of the model's performance than a single train-test split because it uses the entire dataset for both training and validation. It also reduces the variance of the estimate by averaging the results over K iterations.

The steps involved in K-fold cross-validation are as follows:

a. Shuffle the dataset randomly.
b. Split the dataset into K groups or folds of equal size.
c. For each fold, train a model on K-1 folds and validate it on the remaining fold.
d. Repeat this process K times, using each of the K folds once as the validation set.
e. Calculate the average performance score of the K models to obtain the final estimate of the model's performance.

K-fold cross-validation is a useful technique for evaluating the performance of a model and tuning its hyperparameters. It can also be used to compare the performance of different models and to detect overfitting.

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

Hyperparameter tuning is the process of selecting the best combination of hyperparameters for a machine learning model to optimize its performance on a given dataset.

Hyperparameters are parameters that are not learned from the training data but are set by the user before the training process begins. These parameters control the behavior of the model and can significantly impact its performance. Examples of hyperparameters include learning rate, regularization parameter, number of hidden layers in a neural network, etc.

Hyperparameter tuning is done to find the optimal values of these hyperparameters that can maximize the performance of the model on the given dataset. If the hyperparameters are not tuned properly, the model may overfit or underfit the data, resulting in poor performance on the test set.

There are several methods for hyperparameter tuning, including grid search, random search, and Bayesian optimization. In grid search, a set of values for each hyperparameter is defined, and the model is trained and evaluated on each combination of hyperparameters. In random search, a random set of hyperparameters is selected for each iteration. Bayesian optimization uses a probabilistic model to choose the next set of hyperparameters based on the results of previous iterations.

Hyperparameter tuning is an important step in the machine learning workflow as it can significantly improve the performance of the model and make it more reliable for real-world applications.

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

In Gradient Descent, the learning rate is a hyperparameter that controls the step size taken in each iteration towards minimizing the cost function. If the learning rate is set too high, it can lead to several issues such as:

a. Overshooting the optimal solution: A large learning rate can cause the algorithm to overshoot the minimum of the cost function and keep oscillating around it. This can result in slower convergence or even prevent the algorithm from converging altogether.

b. Divergence: If the learning rate is too high, the cost function may not decrease in each iteration but instead increase, leading to the algorithm diverging and failing to find the optimal solution.

c. Instability: A large learning rate can make the algorithm unstable and sensitive to small changes in the data. This can result in a model that is not robust and may not generalize well to new data.

d. Slow convergence: Even if the algorithm does not diverge or oscillate, a large learning rate can slow down the convergence rate and take longer to reach the optimal solution.

Therefore, it is important to choose an appropriate learning rate that balances the convergence rate and stability of the algorithm. A common approach is to start with a small learning rate and gradually increase it until convergence is achieved.

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

Logistic Regression is a linear classification algorithm that assumes a linear relationship between the input features and the output variable. Therefore, it is not suitable for classifying non-linear data, as it cannot model complex non-linear relationships between the features and the target variable.

However, it is possible to use Logistic Regression for non-linear data by transforming the input features using non-linear transformations such as polynomial transformations or basis function expansions. This transforms the original input features into a higher-dimensional space where a linear decision boundary can be used to classify the data.

Alternatively, non-linear classification algorithms such as decision trees, random forests, support vector machines, or neural networks can be used for classifying non-linear data. These algorithms can model complex non-linear relationships between the features and the target variable and can achieve better performance than Logistic Regression on non-linear data.

13. Differentiate between Adaboost and Gradient Boosting.

Adaboost and Gradient Boosting are both ensemble learning techniques used for improving the performance of machine learning models. However, they differ in some key aspects:

a. Algorithm: Adaboost is an iterative algorithm that sequentially trains weak learners on different subsets of the training data, with each subsequent weak learner being trained on the misclassified samples of the previous learner. Gradient Boosting, on the other hand, is also an iterative algorithm that trains a sequence of weak learners, but at each iteration, it fits a weak learner to the negative gradient of the loss function with respect to the current model.

b. Weights: Adaboost assigns higher weights to the misclassified samples, while Gradient Boosting assigns higher weights to the samples with larger residuals.

c. Base estimator: Adaboost can use any weak learner as a base estimator, such as decision trees, stumps, or even linear models. Gradient Boosting typically uses decision trees as the base estimator.

d. Training: Adaboost trains weak learners sequentially, while Gradient Boosting trains weak learners in parallel.

e. Robustness: Adaboost can be sensitive to noisy data and outliers, while Gradient Boosting is more robust to noisy data and can handle outliers better.

In summary, both Adaboost and Gradient Boosting are powerful techniques for improving the performance of machine learning models, but they differ in the algorithm, weights, base estimator, training, and robustness.

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

Bias-variance tradeoff is a fundamental concept in machine learning that refers to the problem of balancing the complexity of a model with its ability to generalize well to new data.

Bias refers to the error that is introduced by approximating a real-world problem with a simplified model. It represents the difference between the expected value of the predictions made by the model and the true values of the data. High bias models are too simple and may underfit the data, leading to poor performance on both the training and test sets.

Variance, on the other hand, refers to the error that is introduced by the model being too sensitive to the noise in the training data. It represents the variability of the model's predictions for different training sets. High variance models are too complex and may overfit the data, leading to good performance on the training set but poor performance on the test set.

The goal of machine learning is to find a model that achieves low bias and low variance, i.e., a model that is both accurate and generalizes well to new data. This is the bias-variance tradeoff. A good model is one that finds the right balance between bias and variance.

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

Support Vector Machines (SVM) is a popular machine learning algorithm that is used for classification and regression analysis. It can handle linear as well as non-linear data using different types of kernels. Some common kernels used in SVM are:

a. Linear Kernel: The linear kernel is a simple kernel that works well when the data is linearly separable. It creates a linear decision boundary that separates the classes.

b. RBF Kernel: The Radial Basis Function (RBF) kernel is a popular kernel used in SVM for non-linear data. It creates non-linear decision boundaries by projecting the data into a higher-dimensional space.

c. Polynomial Kernel: The polynomial kernel is another popular kernel used in SVM for non-linear data. It creates non-linear decision boundaries by mapping the data into a higher-dimensional space using a polynomial function. The degree of the polynomial can be adjusted to control the complexity of the decision boundary.

All these kernels have their own strengths and weaknesses and can be used based on the nature of the data and the problem at hand.