MACHINE LEARNING
ASSIGNMENT - 5

1. R-squared or Residual Sum of Squares (RSS) Which of these two is a better measure of the goodness of fit model in regression and why?

R-squared is generally considered a better measure of the goodness of fit for a regression model because it provides a more intuitive interpretation of the fit. R-squared is a value between 0 and 1 that represents the proportion of the variance in the dependent variable explained by the model's independent variables. A value of 1 indicates a perfect fit, while a value of 0 indicates that the model explains none of the variances in the dependent variable. On the other hand, the Residual Sum of Squares (RSS) is simply the sum of the squared differences between the predicted and actual values. While RSS can be used to compare different models, it is not as intuitive or easy to interpret as R-squared.

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.

In a linear regression model, Total Sum of Squares (TSS) is a measure of the total variance in the dependent variable. It is calculated by summing the squared differences between the actual values of the dependent variable and the mean of the dependent variable.

Explained Sum of Squares (ESS) is a measure of the variance in the dependent variable that is explained by the independent variables in the model. It is calculated by summing the squared differences between the predicted values of the dependent variable and the mean of the dependent variable.

Residual Sum of Squares (RSS) is a measure of the variance in the dependent variable that is not explained by the independent variables in the model. It is calculated by summing the squared differences between the actual values of the dependent variable and the predicted values of the dependent variable.

The relationship between TSS, ESS, and RSS can be represented by the following equation:

TSS = ESS + RSS

This equation states that the total variance in the dependent variable can be broken down into two components: the variance explained by the independent variables (ESS) and the variance not explained by the independent variables (RSS).

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

Regularization is a technique used in machine learning to prevent overfitting of models. Overfitting occurs when a model becomes too complex and starts to fit the noise or random variations in the training data rather than the underlying pattern. This leads to poor generalization performance on new, unseen data.

Regularization adds a penalty term to the cost function that the model is trying to optimize. This penalty term discourages large weights, which can help to reduce the complexity of the model and prevent overfitting. The most commonly used regularization techniques are L1 and L2 regularization, also known as Lasso and Ridge regularization respectively. L1 regularization adds a penalty term that is proportional to the absolute value of the weights, while L2 regularization adds a penalty term that is proportional to the square of the weights.

Regularization is particularly useful when there are a large number of features and/or a small amount of data. It helps to improve the generalization performance of the model by keeping the model simple and preventing overfitting.

4. What is Gini–impurity index?

Gini impurity is a measure of the impurity or disorder of a set of data. It is commonly used in decision tree-based machine learning algorithms to determine the best split point for a dataset when growing the tree. The Gini impurity index is calculated as the probability of misclassifying a randomly chosen element from the dataset, if it were randomly labeled according to the class distribution in the dataset.

The Gini impurity index ranges from 0 to 1, where 0 represents a pure dataset (all elements belong to the same class) and 1 represents an impure dataset (elements belong to different classes). A lower Gini impurity index indicates a more homogeneous split, where the elements in the split are more likely to belong to the same class. A higher Gini impurity index indicates a more heterogeneous split, where the elements in the split are more likely to belong to different classes.

When building a decision tree, the goal is to find the split point that minimizes the Gini impurity of the resulting subsets. The split point that results in the lowest Gini impurity is chosen as the current node, and the process is repeated recursively for each child node until a stopping criterion is met.

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

Unregularized decision trees are prone to overfitting because they can create complex models that fit the noise or random variations in the training data rather than the underlying pattern. In decision tree, as we keep on splitting the tree based on the feature with highest information gain or gini impurity, it can lead to creating many branches with small number of samples in each leaf. This can lead to creating a tree that fits the training data very well but performs poorly on new, unseen data.

An unregularized decision tree will not have any mechanism to limit the complexity of the tree, and it will continue to split the tree until it perfectly separates the training data. This can lead to a tree with many branches and leaves, each representing a small subset of the training data. This results in a model that is highly sensitive to small changes in the training data and is not able to generalize well to new data.

In summary, unregularized decision trees can easily overfit the training data due to its ability to create complex models that fit the noise in the training data and not the underlying pattern, which can lead to poor generalization performance on new, unseen data.

6. What is an ensemble technique in machine learning?

An ensemble technique in machine learning is a method that combines multiple models to improve the overall performance of the predictions. The idea behind ensemble techniques is that by combining several models, the strengths of each model can be harnessed while minimizing their individual weaknesses. Ensemble techniques can be used for both classification and regression problems.

There are several types of ensemble techniques, some of the most popular are:

Bagging (Bootstrap Aggregating) : which trains multiple models independently on different subsets of the training data and then combines their predictions by averaging or majority voting.
Boosting: which trains multiple models sequentially, each model focusing on the mistakes made by the previous model.
Random Forest: which is an extension of bagging to decision trees, where a random subset of features is considered for each split.
Stacking : which combines the predictions of multiple models by training a meta-model to make the final predictions.
Ensemble techniques are known to be powerful methods for improving the performance of models, as they have been shown to achieve state-of-the-art performance on many machine learning problems.

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

Bagging (Bootstrap Aggregating) and Boosting are both ensemble techniques that combine multiple models to improve the overall performance of predictions. However, they differ in the way they combine the models:

Bagging: Bagging is a parallel ensemble technique that trains multiple models independently on different subsets of the training data. These subsets are created by randomly sampling the data with replacement, a process known as bootstrapping. The predictions of the individual models are then combined by

averaging or majority voting. Bagging aims to reduce the variance of the final predictions by averaging the predictions of multiple models. Bagging is effective for reducing overfitting in decision trees, by averaging the predictions of multiple trees that were trained on different subsets of the training data.

Boosting: Boosting is a sequential ensemble technique that trains multiple models sequentially, where each model focuses on the mistakes made by the previous model. During training, the algorithm assigns higher weights to the misclassified samples and retrains the model. This process is repeated multiple times, with each model focusing on the mistakes made by the previous model. The final prediction is made by combining the predictions of all the models by weighting each model's prediction according to its performance. Boosting aims to reduce the bias of the final predictions by iteratively improving the performance of the models. Boosting is effective for reducing bias in weak learners, by iteratively improving the performance of the models.

In summary, Bagging and Boosting are both ensemble techniques, but they differ in the way they combine the models and the way they improve the performance of the predictions. Bagging focuses on reducing the variance of the final predictions by averaging the predictions of multiple models, while Boosting focuses on reducing the bias of the final predictions by iteratively improving the performance of the models.

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

In a random forest, each decision tree is built using a random subset of the training data, known as a bootstrap sample. The out-of-bag (OOB) error is a measure of the performance of a random forest that is calculated using the samples that were not included in the bootstrap sample for a particular tree. In other words, it is a way of estimating the error of the model without the need for a separate validation dataset.

The OOB error can be calculated by:

For each tree in the forest, predict the response for the samples not included in the bootstrap sample for that tree.
Compare these predictions with the actual response for the samples.
Average the misclassification rate over all the trees in the forest to get the OOB error.
OOB error is a useful measure of the performance of a random forest because it provides an estimate of the model's generalization performance without the need for a separate validation set. Additionally, since each tree in the random forest is

trained on a different subset of the data, the OOB samples for each tree are different and can be considered as a form of cross-validation.

It is important to note that OOB error is not a replacement for a proper validation set, but it can be a good estimate of the performance of the model and also it can be used to compare the performance of different models or tuning the hyperparameters of the random forest.

9. What is K-fold cross-validation?
K-fold cross-validation is a technique for evaluating the performance of machine learning models by splitting the data into k subsets or "folds", where k-1 folds are used for training and the remaining fold is used for testing. This process is repeated k times, with each fold being used as the test set once. The performance of the model is then averaged across all k iterations.

The main advantage of K-fold cross-validation is that it allows for a more robust evaluation of the model's performance by averaging the results across multiple train-test splits of the data. This can help to reduce the variance in the model's performance and provide a more reliable estimate of its generalization performance on new, unseen data.

The choice of k depends on the size of the dataset. A common choice is k = 10, which is known as 10-fold cross-validation. However, larger datasets may require larger k to ensure that each fold contains a sufficient number of samples.

It's also important to note that when the data is ordered or time-series data, it's better to use a technique such as Time-series cross-validation, where the data is split in a way that preserves the chronological order of the samples.

In summary, K-fold cross-validation is a powerful technique for evaluating the performance of machine learning models by averaging the results across multiple train-test splits of the data. It helps to provide a more robust estimate of the model's generalization performance on new, unseen data.

10. What is hyper parameter tuning in machine learning and why it is done?
Hyperparameter tuning, also known as hyperparameter optimization, is the process of systematically searching for the best combination of hyperparameters for a machine learning model. Hyperparameters are the parameters of a model that are set before training and are not learned from the data during training. Examples of hyperparameters include the learning rate and regularization strength in neural

networks, the number of estimators and maximum depth in random forests, and the regularization parameter in linear regression.

Hyperparameter tuning is done because the performance of a model can depend heavily on the choice of hyperparameters. Choosing the right combination of hyperparameters can greatly improve the performance of a model, while choosing the wrong combination can lead to poor performance.

There are different methods for hyperparameter tuning, such as grid search and random search. Grid search is an exhaustive search technique where all possible combinations of the hyperparameters are trained and evaluated. Random search is a more efficient method where random combinations of the hyperparameters are sampled and evaluated.

More recently, there are sophisticated algorithms like Bayesian Optimization and Genetic Algorithm, which are more efficient than the grid search and random search.

In summary, hyperparameter tuning is the process of systematically searching for the best combination of hyperparameters for a machine learning model. It is done to improve the performance of a model by choosing the right combination of hyperparameters. Different methods like grid search, random search, Bayesian Optimization and Genetic Algorithm can be used for hyperparameter tuning.

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

A large learning rate in Gradient Descent can lead to several issues:

Oscillations or Divergence: A large learning rate can cause the cost function to oscillate or diverge rather than converge to a minimum. This is because a large learning rate can cause the parameter updates to be too large, causing the model to overshoot the optimal solution.

Slow convergence: A large learning rate can cause the model to converge too quickly to a suboptimal solution. This is because a large learning rate can cause the parameter updates to be too large, causing the model to jump over the optimal solution.

Noise sensitivity: A large learning rate can make the model highly sensitive to noise in the data. This is because a large learning rate can cause the model to overreact to small fluctuations in the data, causing the model to converge to a suboptimal solution.

Stuck in a local minimum: A large learning rate can cause the model to get stuck in a local minimum, because it can cause the parameter updates to be too large and the model will jump over the global minimum.

High Variance: A high learning rate can cause a high variance in the model, which is undesirable because it can lead to overfitting.

In summary, a large learning rate in Gradient Descent can cause the model to converge too quickly to a suboptimal solution, oscillate or diverge, be highly sensitive to noise in the data, get stuck in a local minimum and lead to high variance and overfitting. To avoid these issues, it's important to choose a learning rate that is not too large and monitor the progress of the optimization during training.

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

Logistic regression is a linear model, which means that it is based on the assumption that the relationship between the features and the target variable is linear. Therefore, it is not suitable for classification of non-linear data as it cannot capture complex non-linear relationships between the features and the target variable.

Logistic regression uses a linear decision boundary to separate the classes, which is a straight line or a hyperplane. This means that it can only separate the classes if they can be separated by a linear boundary. For non-linear data, this is not the case. For example, if the data is distributed in a circular or a spiral shape, a linear boundary will not be able to separate the classes.

To classify non-linear data, other models such as decision trees, random forests, k-NN, SVM with a non-linear kernel, or neural networks should be used. These models can handle non-linear decision boundaries and can capture more complex relationships between the features and the target variable.

In summary, Logistic Regression is a linear model and it is not suitable for classification of non-linear data because it cannot capture the complex non-linear relationships between the features and the target variable and it uses a linear decision boundary to separate the classes. Other models such as decision trees, random forests, k-NN, SVM with a non-linear kernel, or neural networks should be used for classifying non-linear data.

13. Differentiate between Adaboost and Gradient Boosting.

AdaBoost and Gradient Boosting are both ensemble techniques that combine multiple models to improve the overall performance of predictions. However, they differ in the way they combine the models:

AdaBoost (Adaptive Boosting) is a boosting algorithm that works by iteratively training weak learners (e.g. decision trees with a small depth) and weighting the observations, giving more weight to the observations that are misclassified by the previous weak learners. The final prediction is made by combining the predictions of all the weak learners by weighting each learner according to its performance. AdaBoost is sensitive to noisy data and outliers, but it's computationally efficient and easy to implement.

Gradient Boosting, on the other hand, is a generalization of boosting to arbitrary differentiable loss functions. It works by iteratively training weak learners (e.g. decision trees) and updating the predictions by taking into account the negative gradient of the loss function with respect to the predictions. The final prediction is made by combining the predictions of all the weak learners. Gradient Boosting is less sensitive to noisy data and outliers, but it's computationally more expensive and requires more fine-tuning of the parameters.

In summary, Adaboost and Gradient Boosting are both ensemble techniques that combine multiple models to improve the overall performance of predictions. The main difference is that Adaboost focuses on weighting the observations, giving more weight to the observations that are misclassified by the previous weak learners, while Gradient Boosting focuses on updating the predictions by taking into account the negative gradient of the loss function with respect to the predictions. Gradient Boosting is less sensitive to noisy data and outliers and it's more suitable for complex data with non-linear relationships, but it's computationally more expensive and requires more fine-tuning of the parameters.

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

The bias-variance trade-off is a fundamental concept in machine learning that refers to the trade-off between how well a model can fit the training data (bias) and how well it can generalize to new, unseen data (variance).

Bias refers to the difference between the model's predictions and the true values for the training data. A model with high bias is said to underfit the data, meaning that it cannot capture the complexity of the underlying relationship between the features and the target variable. This leads to a high training error.

Variance refers to the variability of the model's predictions for different training sets. A model with high variance is said to overfit the data, meaning that it fits the

noise or random variations in the training data rather than the underlying pattern. This leads to a high generalization error.

The goal of machine learning is to find a balance between bias and variance, such that the model can fit the training data well and also generalize to new, unseen data. One way to achieve this balance is to use regularization techniques, which add a penalty term to the cost function that the model is trying to optimize. This penalty term discourages large weights, which can help to reduce the complexity of the model and prevent overfitting.

In summary, the bias-variance trade-off is a fundamental concept in machine learning that refers to the trade-off between how well a model can fit the training data (bias) and how well it can generalize to new, unseen data (variance). The goal is to find a balance between bias and variance to achieve a good generalization performance.

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

In Support Vector Machines (SVMs), the kernel trick is used to transform the input data into a higher-dimensional space, where it becomes possible to separate the data with a linear boundary. Different types of kernels can be used to perform this transformation, such as:

Linear kernel: The linear kernel is the simplest kernel, which simply performs the dot product between the input features. It's used when the data is linearly separable and it's computationally efficient.

RBF (Radial Basis Function) kernel: The RBF kernel is a non-linear kernel that maps the input data into a higher-dimensional space where it becomes possible to separate it with a non-linear boundary. The RBF kernel is defined as the exponential of the negative Euclidean distance between the input features. This kernel is widely used in practice as it can capture complex non-linear relationships in the data.

Polynomial kernel: The polynomial kernel is another non-linear kernel that maps the input data into a higher-dimensional space where it becomes possible to separate it with a non-linear boundary. The polynomial kernel is defined as the dot product of the input features raised to a certain power. This kernel can be used when the data has polynomial relationships.

In summary, the linear, RBF, and polynomial kernels are used in Support Vector Machines (SVMs) to transform the input data into a higher-dimensional space where it becomes possible to separate the data with a linear or non-linear boundary. The

linear kernel is the simplest kernel, which simply performs the dot product between the input features. The RBF kernel is a non-linear kernel that maps the input data into a higher-dimensional space where it becomes possible to separate it with a non-linear boundary. The polynomial kernel is another non-linear kernel that maps the input data into a higher-dimensional space where it becomes possible to separate it with a non-linear boundary.