MACHINE LEARNING ASSIGNMENT-5

Q1). **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-squared (R^2) and Residual Sum of Squares (RSS) are both commonly used measures to assess the goodness of fit of a 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 proportion of the variance in the dependent variable (the response variable) that is explained by the independent variables (predictors) in your regression model.

- R-squared ranges from 0 to 1, with higher values indicating a better fit. 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-squared 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-squared can be useful for comparing different models or assessing how much of the variation in the dependent variable can be attributed to the predictors. However, it has limitations. For example, it can be artificially inflated by adding more predictors to a model, even if those predictors do not have a meaningful relationship with the dependent variable.

2. Residual Sum of Squares (RSS):

- RSS measures the total squared difference between the observed values of the dependent variable and the predicted values from the regression model. It quantifies the overall error or "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 fits the data in terms of minimizing prediction errors. Unlike R-squared, RSS does not provide information about the proportion of variance 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 variance explained by the model, especially in the context of comparing different models. It can provide insights into how well the predictors collectively contribute to 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.

Q2). 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: In regression analysis, TSS (Total Sum of Squares), ESS (Explained Sum of Squares), and RSS (Residual Sum of Squares) are terms used to measure the variance in the dependent variable (response variable) and the relationship between the independent variables (predictors) and the dependent variable.

1. **Total Sum of Squares (TSS)**: TSS measures the total variance in the dependent variable (Y). It represents the total deviation of the dependent variable from its mean. Mathematically, TSS is calculated as the sum of the squared differences between each observed dependent variable value and the mean of the dependent variable:

$TSS = \sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2$ Where:

- $\hat{y}$ is the ith observed value of the dependent variable.
- $\bar{y}$ is the mean of the dependent variable.
- *n* is the total number of observations.

2. **Explained Sum of Squares (ESS)**: ESS measures the variance in the dependent variable that is explained by the independent variables (predictors) in the regression model. It quantifies how much of the total variability in the dependent variable is accounted for by the regression model. Mathematically, ESS is calculated as the sum of the squared differences between the predicted values of the dependent variable ($\hat{y}_i$) and the mean of the dependent variable ($\bar{y}$): $ESS = \sum_{i=1}^{n} \left(\hat{y}_i - \bar{y}\right)^2$

Where:

- $\hat{y}_i i$ is the predicted value of the dependent variable for the ith observation.
- $\bar{y}$ is the mean of the dependent variable.
- *n* is the total number of observations.

3. **Residual Sum of Squares (RSS)**: RSS measures the unexplained variance in the dependent variable after fitting the regression model. It represents the sum of squared differences between the observed values of the dependent variable and the predicted values from the regression model:

$$RSS = \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

Where:

- *yi* is the ith observed value of the dependent variable.
- $\hat{y}_i$ is the predicted value of the dependent variable for the ith observation.
- *n* is the total number of observations.

The relationship between these three metrics can be summarized by the following equation, known as the "Fundamental Regression Equation": *TSS=ESS+RSS* This equation states that the total variance in the dependent variable (*TSS*) can be decomposed into the variance explained by the regression model (*ESS*) and the unexplained variance (*RSS*).

Q3). What is the need of regularization in machine learning?

Ans: Regularization is a technique used in machine learning to prevent overfitting and improve the generalization of models. Overfitting occurs when a model learns to capture noise in the training data rather than the underlying patterns, leading to poor performance on unseen data. Regularization addresses this issue by adding a penalty term to the model's objective function, encouraging simpler models that are less sensitive to noise in the training data.

The need for regularization arises due to several reasons:

1. **Preventing Overfitting**: Overfitting occurs when a model learns to fit the training data too closely, capturing noise and random fluctuations rather than the underlying patterns. Regularization techniques, such as L1 (Lasso) or L2 (Ridge) regularization, help to prevent overfitting by penalizing large parameter values, thereby promoting simpler models that generalize better to unseen data.
2. **Handling High-Dimensional Data**: In high-dimensional feature spaces, such as those encountered in text data or gene expression data, overfitting becomes a significant concern. Regularization helps to constrain the model's complexity and prevent it from learning spurious relationships between features and the target variable.
3. **Dealing with Multicollinearity**: Multicollinearity occurs when two or more predictor variables are highly correlated with each other. In such cases, the estimated coefficients may become unstable, leading to overfitting. Regularization techniques, particularly Ridge regression, mitigate multicollinearity by shrinking the coefficients towards zero, reducing the impact of correlated predictors.
4. **Improving Model Interpretability**: Regularization techniques encourage sparsity in the model, i.e., they tend to force some of the coefficients to be exactly zero. This leads to a simpler model with fewer features, which can be easier to interpret and understand.
5. **Balancing Bias and Variance**: Regularization helps in finding the right balance between bias and variance. A model with high bias (e.g., linear regression) may underfit the data, while a model with high variance (e.g., decision trees) may overfit. Regularization techniques allow us to control the trade-off between bias and variance, leading to models that generalize well to new data.

Overall, regularization is a crucial tool in the machine learning toolbox, helping to improve the robustness and performance of models, particularly in scenarios where overfitting is a concern.

Q4). What is Gini–impurity index?

Ans: Gini-impurity index is a measure of the impurity or disorder in a set of data. It is commonly used in decision tree algorithms, particularly for binary classification problems, to evaluate the quality of a split in the data.

In the context of decision trees, the Gini impurity of a node is calculated based on the probability of each class occurring in the data subset associated with that node. The Gini impurity index ranges from 0 to 0.5, where a value of 0 indicates that the node contains data entirely from one class (perfectly pure), and a value of 0.5 indicates that the node contains an equal proportion of samples from each class (maximum impurity).

Mathematically, the Gini impurity index (I_G) for a node with K classes can be calculated as:

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

Where:

$p(i)$ is the probability of class i occurring in the data subset associated with the node.

To calculate the Gini impurity index for a split in a decision tree, the weighted sum of the impurities of the resulting child nodes is typically computed. The split that minimizes the Gini impurity after the split is chosen as the optimal split.

In summary, the Gini impurity index provides a measure of how well a split separates the classes in a dataset. It is commonly used in decision tree algorithms, such as CART (Classification and Regression Trees), to guide the construction of the tree by selecting the splits that minimize impurity and improve classification performance.

Q5). Are unregularized decision-trees prone to overfitting? If yes, why?

Ans: Yes, unregularized decision trees are indeed prone to overfitting. There are several reasons why this is the case:

1. **High Variance**: Decision trees are capable of capturing intricate details and patterns in the training data, including noise. Without any constraints or regularization, decision trees can grow to be extremely deep and complex, which allows them to fit the training data very closely. However, this often leads to high variance, meaning that the model's performance may be overly sensitive to small fluctuations or noise in the training data.
2. **Memorization of Training Data**: Unregularized decision trees have the capacity to memorize the training data, effectively encoding it within the structure of the tree. This memorization can result in poor generalization to unseen data, as the model may fail to capture the underlying patterns and instead merely replicate the training set.
3. **Overly Complex Models**: Decision trees can split the feature space in numerous ways to perfectly fit the training data. Without any constraints, the tree may continue to split until each leaf node contains only a few instances, resulting in a highly complex model. Such complexity increases the risk of overfitting, as the model becomes overly specialized to the training data and may not generalize well to new, unseen data.
4. **Sensitive to Outliers and Noise**: Unregularized decision trees can be highly sensitive to outliers and noise in the training data. Since they strive to minimize impurity at each split, they may create branches to accommodate individual data points that do not represent true patterns in the data. This sensitivity to outliers and noise further exacerbates the risk of overfitting.

To address these issues and mitigate overfitting, various techniques can be applied to regularize decision trees, such as limiting the maximum depth of the tree, imposing minimum sample size constraints for splitting, and using pruning methods to simplify the tree structure. Additionally, ensemble methods like random forests and gradient boosting combine multiple decision trees to reduce overfitting and improve predictive performance.

Q6). What is an ensemble technique in machine learning?

Ans: Ensemble techniques in machine learning involve combining multiple individual models to create a more powerful and robust predictive model. The basic idea behind ensemble methods is to leverage the wisdom of crowds, where multiple weak learners are combined to create a strong learner that performs better than any individual model.

There are several popular ensemble techniques, including:

1. **Bagging (Bootstrap Aggregating)**: Bagging involves training multiple instances of the same base learning algorithm on different subsets of the training data, sampled with replacement (bootstrap samples). Each model learns independently, and predictions are typically aggregated using averaging (for regression) or voting (for classification) to obtain the final prediction. Random forests are a popular example of a bagging ensemble method, where decision trees are trained on bootstrap samples of the data and combined to make predictions.
2. **Boosting**: Boosting is an iterative ensemble technique where base learners are trained sequentially, with each subsequent model focusing on the instances that were misclassified by the previous models. Boosting algorithms such as AdaBoost (Adaptive Boosting) and Gradient Boosting Machine (GBM) build a sequence of weak learners that are combined to create a strong learner. Each model in the sequence is trained to correct the errors made by the previous models, leading to improved performance.
3. **Stacking (Stacked Generalization)**: Stacking involves training multiple diverse base learners and combining their predictions using a meta-learner, often a simple linear model. Unlike bagging and boosting, stacking leverages the diversity of the base learners by training them on the entire training set rather than on subsets. The meta-learner learns to combine the predictions of the base learners to produce the final prediction.
4. **Voting**: Voting, also known as majority voting or ensemble averaging, combines predictions from multiple independent models and selects the most common prediction as the final output (for classification). It can be applied to both classification and regression problems, where predictions are aggregated using averaging (for regression) or voting (for classification).

Ensemble techniques are widely used in machine learning because they often result in improved predictive performance, increased robustness, and better generalization to unseen data compared to individual models. By leveraging the diversity of multiple models, ensemble methods can compensate for the weaknesses of individual models and exploit complementary patterns in the data.

Q7). What is the difference between Bagging and Boosting techniques?

Ans: Bagging (Bootstrap Aggregating) and Boosting are both ensemble techniques used in machine learning to improve predictive performance by combining multiple models. While they share the goal of ensemble learning, they differ in their approach and the way they leverage multiple models.

Here are the key differences between Bagging and Boosting:

1. **Training Approach**:
    - **Bagging**: In bagging, multiple instances of the same base learning algorithm are trained independently on different subsets of the training data, sampled with replacement (bootstrap

samples). Each model learns independently of the others, and predictions are typically aggregated using averaging (for regression) or voting (for classification) to obtain the final prediction.
- **Boosting**: In boosting, base learners are trained sequentially, with each subsequent model focusing on the instances that were misclassified by the previous models. Boosting algorithms such as AdaBoost (Adaptive Boosting) and Gradient Boosting Machine (GBM) build a sequence of weak learners that are combined to create a strong learner. Each model in the sequence is trained to correct the errors made by the previous models, leading to improved performance.

2. **Model Complexity**:
   - **Bagging**: Bagging typically involves training multiple models of the same complexity, often using the same base learning algorithm. Each model in the bagging ensemble learns independently, and there is no interdependence between the models.
   - **Boosting**: Boosting focuses on training a sequence of weak learners, where each individual model is relatively simple or weak. The complexity of the final boosted model increases with the number of iterations (base learners), as each subsequent model learns to correct the errors of the previous ones.
3. **Parallelism**:
   - **Bagging**: Since bagging involves training multiple models independently, the training process can be parallelized easily, making it suitable for distributed computing environments.
   - **Boosting**: Boosting trains models sequentially, where each subsequent model depends on the performance of the previous models. As a result, boosting algorithms are typically harder to parallelize than bagging algorithms.
4. **Handling of Outliers and Noise**:
   - **Bagging**: Bagging is effective at reducing the variance of the model and improving its robustness to outliers and noise in the training data. By aggregating predictions from multiple models, bagging can mitigate the impact of individual outliers or noisy instances.
   - **Boosting**: Boosting focuses on reducing both bias and variance by iteratively refining the model to correct its mistakes. While boosting algorithms can handle outliers and noise to some extent, they may be more susceptible to overfitting if the training data contains substantial noise or outliers.

In summary, bagging and boosting are both powerful ensemble techniques that improve predictive performance by combining multiple models. Bagging trains multiple models independently and aggregates their predictions, while boosting builds a sequence of weak learners that are combined to create a strong learner. The choice between bagging and boosting depends on factors such as the complexity of the data, the presence of outliers or noise, and the computational resources available.

Q8). What is out-of-bag error in random forests?

Ans: In random forests, each decision tree is trained using a bootstrap sample of the original dataset. A bootstrap sample is created by randomly sampling the original dataset with replacement, which means that some instances from the original dataset may be left out of each bootstrap sample.

The out-of-bag (OOB) error in random forests is an estimate of the model's performance on unseen data, calculated using the instances that are not included in the bootstrap sample used to train each

individual tree. Specifically, for each instance in the original dataset, the out-of-bag error is computed by averaging the predictions made by the trees that did not use that instance in their bootstrap sample.

Here's how the out-of-bag error estimation works:

1. During the training of each decision tree in the random forest, about one-third of the instances in the original dataset are not included in the bootstrap sample used to train that tree. These instances are referred to as out-of-bag instances.
2. After training all the trees in the random forest, each out-of-bag instance is passed through all the trees that did not include it in their bootstrap samples. The predictions made by these trees are aggregated, typically using averaging for regression tasks or voting for classification tasks.
3. The out-of-bag error is then computed by comparing the aggregated predictions for each out-of-bag instance to the true labels in the original dataset. For regression tasks, this could involve calculating the mean squared error (MSE), while for classification tasks, it could involve computing the misclassification rate or the Gini impurity.

The out-of-bag error provides a robust estimate of the model's performance on unseen data without the need for a separate validation set or cross-validation. It is particularly useful in scenarios where the dataset is small or when cross-validation may be computationally expensive. Additionally, the out-of-bag error can be used to tune hyperparameters of the random forest, such as the number of trees or the maximum depth of the trees.

Q9). 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 dividing the dataset into k subsets, or folds, and performing multiple rounds of training and evaluation. It is a widely used method for estimating the performance of a model on unseen data and for selecting model hyperparameters.

Here's how K-fold cross-validation works:

1. **Dataset Splitting**: The original dataset is randomly partitioned into k equal-sized subsets, or folds. Each fold contains approximately the same proportion of instances and maintains the distribution of the target variable, if it's a classification problem.
2. **Training and Validation**: The cross-validation process consists of k iterations. In each iteration, one fold is held out as the validation set, and the remaining k-1 folds are used as the training set. The model is trained on the training set and evaluated on the validation set.
3. **Performance Metrics Calculation**: After each iteration, a performance metric (e.g., accuracy, mean squared error, etc.) is computed based on the model's predictions on the validation set.
4. **Average Performance**: Once all iterations are completed, the performance metrics from each fold are averaged to obtain a single estimate of the model's performance.

K-fold cross-validation provides several advantages:

- **More Reliable Performance Estimates**: By averaging the performance across multiple folds, K-fold cross-validation provides a more reliable estimate of the model's performance compared to a single train-test split.
- **Better Utilization of Data**: Each instance in the dataset is used for both training and validation exactly once. This ensures that the model is evaluated on all instances of the dataset, which can lead to a more representative performance estimate.
- **Reduced Variance in Performance Estimates**: Since the performance estimate is based on multiple rounds of training and evaluation, K-fold cross-validation tends to have lower variance compared to a single train-test split.

Common choices for the value of k are 5 or 10, but the value can vary depending on the size of the dataset and computational constraints. In some cases, stratified K-fold cross-validation may be used to ensure that each fold maintains the same class distribution as the original dataset, particularly for imbalanced classification problems.

Q10). What is hyper parameter tuning in machine learning and why it is done?

Hyperparameter tuning, also known as hyperparameter optimization, is the process of selecting the optimal values for the hyperparameters of a machine learning model. Hyperparameters are configuration settings that are external to the model and cannot be directly estimated from the training data. They control aspects of the model's behavior and complexity, such as the learning rate in neural networks, the depth of a decision tree, or the regularization strength in linear models.

Hyperparameter tuning is performed to improve the performance of a machine learning model by finding the set of hyperparameters that minimizes a chosen evaluation metric, such as accuracy, mean squared error, or area under the ROC curve. The goal is to find hyperparameters that generalize well to unseen data and result in a model that performs optimally on the task at hand.

Here are several reasons why hyperparameter tuning is essential in machine learning:

1. **Optimizing Performance**: Hyperparameters significantly impact the performance of a machine learning model. By tuning hyperparameters, we can optimize the model's performance, leading to better accuracy, lower error rates, or improved predictive power.
2. **Preventing Overfitting**: Many hyperparameters control the complexity of the model and its ability to generalize to unseen data. Tuning these hyperparameters can help prevent overfitting, where the model learns to fit the training data too closely and performs poorly on new data.
3. **Improving Robustness**: Hyperparameter tuning can help create models that are more robust and resilient to variations in the data. By finding hyperparameters that generalize well across different datasets or data distributions, we can build more reliable models.
4. **Adapting to Different Datasets**: Hyperparameter tuning allows us to adapt the model to different datasets and problem domains. By adjusting the hyperparameters based on the characteristics of the data, we can build models that are tailored to specific tasks and datasets.
5. **Enhancing Interpretability**: In some cases, hyperparameters can affect the interpretability of the model. By tuning hyperparameters, we can create models that strike a balance between performance and interpretability, making them easier to understand and interpret.

Hyperparameter tuning is typically performed using techniques such as grid search, random search, Bayesian optimization, or evolutionary algorithms. These techniques systematically explore the hyperparameter space to find the combination of values that optimizes the model's performance on a validation set or through cross-validation.

Q11). What issues can occur if we have a large learning rate in Gradient Descent?

Ans: Having a large learning rate in gradient descent optimization algorithms, such as stochastic gradient descent (SGD), can lead to several issues that hinder the convergence of the algorithm and the quality of the learned model. Some of the prominent issues include:

1. **Divergence**: One of the most significant issues with a large learning rate is that it can cause the optimization process to diverge, rather than converging to the optimal solution. With a large learning rate, the updates to the model parameters can become too large, causing the algorithm to overshoot the minimum of the loss function and bounce around, preventing convergence.
2. **Overshooting the Minima**: Even if the optimization process does not diverge outright, a large learning rate can cause the algorithm to overshoot the minima of the loss function. This overshooting can lead to oscillations around the minima, slowing down the convergence process and preventing the algorithm from reaching the optimal solution.
3. **Instability**: A large learning rate can make the optimization process highly sensitive to the initial values of the model parameters and the choice of the learning rate schedule. Small perturbations in the learning rate or initial parameter values can lead to drastically different trajectories of the optimization process, making it challenging to find a stable solution.
4. **Poor Generalization**: Optimization algorithms with a large learning rate tend to converge quickly to the training data, but they may generalize poorly to unseen data. This is because the model parameters may become too specialized to the training data, capturing noise and outliers rather than the underlying patterns in the data.
5. **Difficulty in Finding an Appropriate Learning Rate**: Choosing an appropriate learning rate can be challenging, especially when using a large learning rate. It may require extensive tuning and experimentation to find a learning rate that strikes the right balance between convergence speed and stability.

To mitigate these issues, it is essential to carefully select the learning rate and regularly monitor the optimization process. Techniques such as learning rate decay, adaptive learning rate methods (e.g., AdaGrad, RMSProp, Adam), and gradient clipping can also help stabilize the optimization process and improve convergence when dealing with large learning rates. Additionally, starting with a smaller learning rate and gradually increasing it can be an effective strategy to prevent divergence and instability.

Q12). Can we use Logistic Regression for classification of Non-Linear Data? If not, why?

Ans: Logistic Regression is a linear classification algorithm, which means it assumes a linear relationship between the features and the log-odds of the target variable. Consequently, Logistic Regression is not inherently capable of modeling complex, non-linear relationships between the features and the target variable.

However, Logistic Regression can still be used for classification tasks involving non-linear data to some extent, but it may not capture complex non-linear relationships as effectively as other non-linear classifiers like decision trees, support vector machines (SVMs), or neural networks.

Here's why Logistic Regression may not perform well on non-linear data:

1. **Limited Flexibility**: Logistic Regression models the decision boundary as a linear function of the input features. This restricts its ability to capture non-linear decision boundaries that may exist in the data. As a result, Logistic Regression may underperform when the relationship between the features and the target variable is highly non-linear.
2. **Assumption of Linearity**: Logistic Regression assumes that the log-odds of the target variable are a linear combination of the input features. If this assumption does not hold true (i.e., if the relationship between the features and the target variable is non-linear), Logistic Regression may yield biased or inaccurate predictions.
3. **Risk of Underfitting**: In cases where the data contains non-linear relationships, Logistic Regression may struggle to capture the underlying patterns, leading to underfitting. Underfitting occurs when the model is too simple to capture the complexity of the data, resulting in poor performance on both training and test data.

While Logistic Regression may not be the best choice for handling non-linear data, there are techniques that can help improve its performance in such scenarios:

- **Feature Engineering**: Transforming the input features or creating new features based on non-linear transformations (e.g., polynomial features, interaction terms) can help Logistic Regression capture non-linear relationships in the data.
- **Ensemble Methods**: Using ensemble methods such as Random Forests or Gradient Boosting Machines (which inherently capture non-linear relationships) in conjunction with Logistic Regression can improve predictive performance on non-linear data.
- **Kernel Tricks**: Transforming the input features using kernel methods (e.g., kernel Logistic Regression) can project the data into a higher-dimensional space where non-linear relationships may become linear, making Logistic Regression more effective.

In summary, while Logistic Regression can be used for classification tasks involving non-linear data, its performance may be limited compared to other non-linear classifiers. It is important to consider the nature of the data and explore alternative models that are better suited for capturing non-linear relationships when dealing with non-linear classification problems.

Q13). Differentiate between Adaboost and Gradient Boosting.

Ans: AdaBoost (Adaptive Boosting) and Gradient Boosting are both popular ensemble learning techniques used for classification and regression tasks. While they share some similarities, they have distinct differences in their approaches to training and combining weak learners.

Here's a differentiation between AdaBoost and Gradient Boosting:

1. **Training Approach**:

- **AdaBoost**: AdaBoost is an iterative ensemble method that builds a sequence of weak learners, typically decision trees, where each subsequent model focuses on the instances that were misclassified by the previous models. In each iteration, AdaBoost assigns higher weights to the misclassified instances, making them more influential in the training of subsequent models. The final prediction is obtained by combining the predictions of all weak learners, weighted by their individual performance.
- **Gradient Boosting**: Gradient Boosting, on the other hand, builds a sequence of weak learners, typically decision trees, in a stage-wise manner, where each subsequent model learns to correct the errors made by the previous models. Unlike AdaBoost, which assigns higher weights to misclassified instances, Gradient Boosting minimizes a loss function (e.g., mean squared error for regression, log loss for classification) by fitting the residuals of the previous model in each iteration. The final prediction is obtained by summing the predictions of all weak learners.

2. **Objective Function**:
   - **AdaBoost**: AdaBoost minimizes an exponential loss function, which penalizes misclassifications more heavily as the number of iterations increases. The weights of the instances are updated in each iteration to focus on the misclassified instances.
   - **Gradient Boosting**: Gradient Boosting minimizes a specified loss function, such as mean squared error (MSE) for regression or log loss for classification, by fitting the negative gradient of the loss function in each iteration. This allows Gradient Boosting to optimize any differentiable loss function, making it more flexible than AdaBoost.
3. **Weak Learner Weighting**:
   - **AdaBoost**: In AdaBoost, each weak learner is assigned a weight based on its performance in classifying the instances. The weights of the weak learners are used to determine their influence on the final prediction.
   - **Gradient Boosting**: In Gradient Boosting, the weak learners are combined in a stage-wise manner, where each subsequent model is fitted to the residuals of the previous model. The weights of the weak learners are determined by the learning rate, which controls the contribution of each model to the final prediction.
4. **Model Complexity**:
   - **AdaBoost**: AdaBoost typically uses shallow decision trees (weak learners) as base models. The final prediction is obtained by combining the predictions of multiple weak learners.
   - **Gradient Boosting**: Gradient Boosting can use a variety of weak learners, including decision trees, with varying depths and complexities. The final prediction is obtained by summing the predictions of multiple weak learners.

In summary, while both AdaBoost and Gradient Boosting are ensemble learning techniques that build a sequence of weak learners to create a strong learner, they differ in their training approach, objective function, weak learner weighting, and model complexity. Gradient Boosting, with its ability to optimize any differentiable loss function and flexibility in weak learner choice, is often preferred for its superior performance and robustness on a wide range of tasks.

Q14). What is bias-variance trade off in machine learning?

Ans: The bias-variance tradeoff is a fundamental concept in machine learning that refers to the balance between bias and variance in the performance of a predictive model. It highlights the tradeoff that occurs when attempting to minimize two sources of error: bias and variance.

1. Bias:
    - Bias measures the systematic error in the model's predictions. It represents the difference between the average prediction of the model and the true value being predicted. A high bias indicates that the model is too simple and unable to capture the underlying patterns in the data, leading to underfitting. Models with high bias tend to perform consistently poorly across different datasets.
    - Example: A linear regression model attempting to fit a non-linear relationship between features and target variable will have high bias.
2. Variance:
    - Variance measures the variability of the model's predictions across different training sets. It represents the model's sensitivity to fluctuations in the training data. A high variance indicates that the model is too complex and captures noise in the training data, leading to overfitting. Models with high variance tend to perform well on the training data but poorly on unseen data.
    - Example: A decision tree with unlimited depth may capture noise in the training data and result in high variance.

The bias-variance tradeoff arises because reducing one source of error often leads to an increase in the other:

- High Bias, Low Variance: Simple models with high bias and low variance tend to generalize poorly to new data but are less sensitive to fluctuations in the training data. They underfit the training data.
- Low Bias, High Variance: Complex models with low bias and high variance tend to fit the training data well but generalize poorly to new data. They overfit the training data.

The goal in machine learning is to find the right balance between bias and variance to achieve optimal model performance. This can involve selecting an appropriate model complexity, tuning hyperparameters, using regularization techniques, and employing ensemble methods. Techniques such as cross-validation can help assess the bias-variance tradeoff and choose the model that strikes the best balance between bias and variance for a given problem. Ultimately, the aim is to build models that generalize well to unseen data while accurately capturing the underlying patterns in the data.

Q15). Give short description each of Linear, RBF, Polynomial kernels used in SVM.

Ans:

1. **Linear Kernel**:
    - The linear kernel is the simplest kernel used in SVMs.
    - It represents a linear decision boundary, which separates classes in the input feature space using a straight line.
    - The linear kernel is computationally efficient and is often used when the data is linearly separable or when the number of features is very high compared to the number of samples.
    - Mathematically, the linear kernel computes the dot product between feature vectors in the input space.
2. **RBF (Radial Basis Function) Kernel**:
    - The RBF kernel is a popular choice for SVMs and is suitable for non-linear classification tasks.
    - It is capable of capturing complex, non-linear decision boundaries in the input feature space.

- The RBF kernel defines similarity between two samples based on their Euclidean distance in the feature space.
- It is defined by a single hyperparameter, γ, which controls the smoothness of the decision boundary. A smaller γ value results in a smoother decision boundary, while a larger γ value can lead to a more complex and flexible decision boundary.
- The RBF kernel can handle data that is not linearly separable by transforming the input space into a higher-dimensional space, where classes may become separable.

3. **Polynomial Kernel**:
    - The polynomial kernel is used to capture non-linear relationships between features in SVMs.
    - It maps the input features into a higher-dimensional space using polynomial functions.
    - The polynomial kernel is defined by two hyperparameters: the degree of the polynomial (d) and an optional coefficient (c).
    - The degree (d) controls the complexity of the decision boundary. A higher degree polynomial can capture more complex relationships in the data but may also increase the risk of overfitting.
    - The coefficient (c) scales the influence of higher-degree terms in the polynomial.
    - Like the RBF kernel, the polynomial kernel can handle non-linear decision boundaries and is suitable for data that is not linearly separable.

In summary, Linear, RBF, and Polynomial kernels are commonly used in SVMs to handle linear and non-linear classification tasks by defining different types of decision boundaries in the input feature space. The choice of kernel depends on the nature of the data and the complexity of the underlying relationships between features.

STATISTICS WORKSHEET-5

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

a) Mean

b) Actual

c) Predicted

d) Expected

Ans. d) Expected

2. Chisquare is used to analyse

a) Score

b) Rank

c) Frequencies

d) All of these

Ans. c) Frequencies

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

a) 4

b) 12

c) 6

d) 8

Ans: c) 6

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

a) Normal distribution

b) Chisqared distribution

c) Gamma distribution

d) Poission distribution

Ans: b) Chisqared distribution

5. Which of the following distributions is Continuous

a) Binomial Distribution

b) Hypergeometric Distribution

c) F Distribution

d) Poisson Distribution

Ans: c) F Distribution

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

a) Statistic

b) Hypothesis

c) Level of Significance

d) TestStatistic

Ans: b) Hypothesis

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

a) Null Hypothesis

b) Statistical Hypothesis

c) Simple Hypothesis

d) Composite Hypothesis

Ans: a) Null Hypothesis

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

a) Two tailed

b) One tailed

c) Three tailed

d) Zero tailed

Ans: a) Two tailed

9. Alternative Hypothesis is also called as?

a) Composite hypothesis

b) Research Hypothesis

c) Simple Hypothesis

d) Null Hypothesis

Ans: b) 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

a) np

b) n

Ans: a) np