# MACHINE LEARNING

**Q.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?**

**Answer**: $R^2$ is typically more useful for understanding and comparing the explanatory power of regression models, while RSS provides a raw measure of the prediction error. Comparing models or understand the proportion of variance explained by the model, $R^2$ is generally more informative.

$R^2$ is a standardized measure, making it easier to compare the goodness of fit between different models, even if they have different scales or units. R-squared is a normalized measure of the goodness of fit that explains the proportion of variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with higher values indicating a better fit.

**Q.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.**

**Answer:** In regression analysis, the Total Sum of Squares (TSS), Explained Sum of Squares (ESS), and Residual Sum of Squares (RSS) are important metrics used to evaluate the fit of a regression model.

**Total Sum of Squares (TSS):** TSS measures the total variation in the dependent variable y. It is the sum of the squared differences between the observed values and the mean of the observed values. TSS represents the total variability to be explained by the model.

**Explained Sum of Squares (ESS):** RSS measures the variation not explained by the regression model. It is the sum of the squared differences between the observed values and the predicted values. RSS represents the residual or error variance.

**Residual Sum of Squares (RSS**): RSS measures the variation not explained by the regression model. It is the sum of the squared differences between the observed values and the predicted values. RSS represents the residual or error variance.

The three metrics are related by the following equation:

TSS = ESS + RSS

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

**Answer:** Regularization in machine learning is crucial for managing model complexity and improving generalization to unseen data. Here's why regularization is needed:

**1. Preventing Overfitting: Overfitting** occurs when a model learns the noise or random fluctuations in the training data rather than the underlying patterns. This leads to poor performance on new, unseen data.

- Regularization techniques add a penalty to the model's complexity, discouraging it from fitting the noise and thus helping to prevent overfitting.

**2. Improving Generalization:** By penalizing complex models, regularization encourages the development of simpler models that are more likely to generalize well to new data. This can lead to better performance on validation and test sets.

**3. Reducing Model Complexity:** Regularization controls the size of the model's parameters or coefficients. This is especially useful in high-dimensional data where having too many features can lead to overly complex models. Techniques like L1 regularization (Lasso) can even drive some coefficients to zero, effectively performing feature selection and simplifying the model.

**4. Enhancing Model Stability:** Models with regularization are less sensitive to small changes in the training data, which makes them more stable and reliable. This stability can be particularly important in real-world applications where data might be noisy or incomplete.

**5. Improving Interpretability:** Regularization can make models more interpretable by reducing the number of features used or by shrinking coefficients, which can highlight the most important variables and simplify the model's decision-making process.

**Types of Regularization**

1. **L1 Regularization (Lasso)**

   Adds the absolute value of the coefficients to the loss function.

Encourages sparsity in the model (drives some coefficients to zero).

2. **L2 Regularization (Ridge)**

   Adds the squared value of the coefficients to the loss function.

   Shrinks coefficients towards zero but does not necessarily make them exactly zero.

3. **Elastic Net**

   Combines L1 and L2 regularization.

   Balances between sparsity and coefficient shrinkage.

4. **Dropout (for Neural Networks)**

   Randomly drops units (and their connections) during training to prevent co-adaptation of features.

**Q 4. What is Gini–impurity index?**

**Answer:** The Gini impurity index is a metric used to measure the impurity or diversity of a dataset. It is commonly used in decision tree algorithms, such as CART (Classification and Regression Trees), to evaluate the quality of splits. The Gini impurity gives an indication of how mixed the classes are in the dataset.

The Gini impurity index is a key metric for evaluating the quality of splits in decision tree algorithms. It measures the impurity of a node, with lower values indicating purer nodes. By selecting splits that minimize the Gini impurity, decision trees aim to create nodes that are as homogeneous as possible, leading to more accurate and interpretable models.

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

**Answer:** Yes, unregularized decision trees are prone to overfitting. This is because decision trees have the ability to model very complex relationships by creating many branches, potentially capturing noise and small fluctuations in the training data. Here are some reasons why unregularized decision trees tend to overfit.

**Reasons for Overfitting**

1. **High Flexibility**: Decision trees can create very deep trees with many branches, which allows them to capture intricate patterns in the training

data. However, these patterns may include noise and outliers, which do not generalize well to unseen data.

2. **Perfectly Fitting the Training Data**: Unregularized decision tree will keep splitting the data until all leaves are pure or until it reaches the maximum depth. This can result in a tree that perfectly fits the training data but performs poorly on new data because it has learned the noise and specific details of the training set.
3. **Small Sample Sizes in Leaves**: As the tree grows deeper, the number of samples in each leaf becomes smaller. Small sample sizes can lead to high variance and poor generalization because the model's predictions are based on limited data points.

**Q. 6 What is an ensemble technique in machine learning?**

**Answer:** An ensemble technique in machine learning involves combining multiple models to improve overall performance, reduce variance, increase robustness, and achieve better generalization on unseen data. Ensemble methods leverage the strengths of individual models and mitigate their weaknesses by aggregating their predictions. There are several types of ensemble techniques, each with its own approach to combining models**.**

**Common Ensemble Techniques**

1. **Bagging (Bootstrap Aggregating)**: Bagging involves training multiple instances of the same model on different subsets of the training data, created using bootstrapping (random sampling with replacement).
2. **Boosting**: Boosting trains multiple weak learners sequentially, with each learner focusing on the mistakes made by its predecessors. The final prediction is a weighted sum of the predictions from all learners.
3. **Stacking (Stacked Generalization)**: Stacking involves training multiple base models and then using another model (meta-learner) to combine their predictions. The base models' predictions serve as input features for the meta-learner.
4. **Voting**: Voting combines the predictions of multiple models by taking a majority vote (for classification) or averaging (for regression) their outputs.

5. **Bagging vs. Boost:** Focuses on reducing variance by averaging over multiple models trained independently. Each model is trained on a different subset of the data.

   **Boosting**: Focuses on reducing bias by sequentially training models, where each model tries to correct the errors of the previous one.

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

**Answer:** Bagging (Bootstrap Aggregating) and Boosting are both ensemble techniques in machine learning that combine multiple models to improve performance. However, they differ significantly in how they construct and combine these models.

Bagging: Bagging aims to reduce variance by averaging the predictions of multiple models.

It generates multiple versions of a training dataset using bootstrapping (random sampling with replacement) and trains a separate model on each dataset.

Process:

Data Sampling: Create multiple bootstrapped datasets from the original training set.

Model Training: Train a separate model (often the same type) on each bootstrapped dataset.

Aggregation: For classification, aggregate the predictions by majority vote (hard voting). For regression, average the predictions.

**Boosting:** Boosting aims to reduce both bias and variance by sequentially training models. Each model is trained to correct the errors of the previous models, focusing more on the instances that were previously misclassified.

**Process**:

1. **Model Initialization**: Start with an initial weak model.
2. **Sequential Training**: Train subsequent models sequentially, each one focusing on the errors (residuals) made by the previous models.
3. **Weighting**: Adjust the weights of training instances or the predictions of the models to place more emphasis on difficult cases.

4. **Aggregation**: Combine the models' predictions, often through a weighted sum or a weighted majority vote.

Bagging focuses on reducing variance through parallel training of multiple models on different subsets of data and aggregating their predictions.

Boosting focuses on reducing both bias and variance by sequentially training models, each one correcting the errors of the previous ones.

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

**Answer:** Out-of-bag (OOB) error is a method of evaluating the performance of a Random Forest model without the need for a separate validation set. It leverages the nature of the bootstrap sampling process used to create the individual trees in the forest.

Out-of-bag error is an effective and efficient way to estimate the performance of a Random Forest model. It takes advantage of the bootstrap sampling process to provide an unbiased estimate of the model's generalization error, making it a valuable tool for model evaluation when data is limited.

**Q.9 What is K-fold cross-validation?**

**Answer:** K-fold cross-validation is a widely used technique for assessing the performance and generalizability of a machine learning model. It helps in evaluating how the model will perform on an independent dataset (i.e., unseen data) and reduces the likelihood of overfitting.

K-fold cross-validation is a robust technique for evaluating the performance of machine learning models. By partitioning the data into k folds and ensuring each fold is used as a validation set exactly once, it provides a comprehensive assessment of the model's ability to generalize to new, unseen data. The choice of k is important and typically balances the need for accurate performance estimation with computational efficiency.

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

**Answer:** Hyperparameter tuning in machine learning involves the process of selecting the optimal set of hyperparameters for a learning algorithm to improve its performance on a given task. Hyperparameters are parameters that

are not learned from the data but are set before the training process begins. Examples of hyperparameters include the learning rate for training a neural network, the depth of a decision tree, or the number of neighbours in a k-nearest neighbours' algorithm.

**Hyperparameter Tuning is Done to**

**Improve Model Performance**: Proper tuning of hyperparameters can significantly enhance the performance of a machine learning model. The right set of hyperparameters can lead to better accuracy, precision, recall, F1-score, or any other performance metric relevant to the task.

**Avoid Overfitting and Underfitting**: Tuning helps in finding a balance between underfitting and overfitting. For example, in decision trees, a very deep tree may overfit the training data, while a shallow tree may underfit. Proper tuning helps in finding the right depth that generalizes well to unseen data.

**Optimize Training Time and Resource Usage**: Efficient hyperparameter values can also optimize the training time and computational resources required. For instance, an appropriate batch size or learning rate can speed up convergence in neural networks.

**Ensure Model Robustness**: Well-tuned hyperparameters can make the model more robust to different data variations and help in achieving consistent performance across different datasets.

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

**Answer:** If the learning rate in Gradient Descent is set too high, several issues can occur that negatively affect the training process and the model's performance:

**1. Overshooting the Minimum:** A high learning rate can cause the algorithm to take large steps in the direction of the gradient. Instead of gradually approaching the minimum of the loss function, the algorithm may overshoot it repeatedly.

**2. Divergence:** When the learning rate is excessively large, the updates to the model parameters can be so large that the algorithm moves further away from the minimum with each iteration.

**3. Loss Function Instability:** Large learning rates can cause significant fluctuations in the value of the loss function between iterations.

**4. Poor Convergence:** With a high learning rate, the algorithm might converge to a suboptimal solution rather than finding the global or a good local minimum. Resulting model may have higher error rates and lower performance compared to what could be achieved with an appropriate learning rate.

**Visual Example**

Imagine a ball rolling down a hill towards the lowest point (representing the minimum of the loss function):

- **Proper Learning Rate**: The ball gradually rolls down the hill, slowing down as it approaches the bottom, and eventually settles at the minimum.
- **High Learning Rate**: The ball bounces wildly around the hill, frequently overshooting the lowest point and possibly even bouncing higher and higher away from the minimum.

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

**Answer:** Logistic Regression, in its basic form, is a linear classifier, which means it can only effectively classify data that is linearly separable. This means it works well when there is a clear linear boundary between the classes. However, it struggles with non-linear data because it cannot capture the complex relationships and boundaries needed to separate classes in such cases.

**Why Logistic Regression Struggles with Non-Linear Data**

1. **Linear Decision Boundary**: Logistic Regression finds a linear decision boundary (a hyperplane) to separate classes. For non-linear data, this boundary is insufficient as the classes are not separated by a straight line or a flat plane.
2. **Model Assumptions**: Logistic Regression assumes a linear relationship between the input features and the log-odds of the outcome. When this assumption is violated (as in non-linear data), the model performs poorly.

**Q 13. Differentiate between Adaboost and Gradient Boosting.**

**Answer:** AdaBoost (Adaptive Boosting) and Gradient Boosting are both ensemble learning techniques that combine multiple weak learners to form a strong predictive model. However, they differ in their approach and methodology. AdaBoost aims to improve the performance of weak learners by focusing on the samples that are misclassified. It adjusts the weights of the training samples so that subsequent classifiers focus more on the difficult cases.

Gradient Boosting builds the model in a stage-wise fashion by optimizing a loss function. It aims to minimize the residual errors (differences between the observed and predicted values) of the model by fitting new learners to these residuals.

**AdaBoost**: Focuses on misclassified samples by adjusting their weights.

Changes the weight of samples.

Uses a weighted majority vote or weighted sum of the weak learners.

**Gradient Boosting**: Focuses on the residuals (errors) of the model by fitting new learners to these residuals.

Fits new learners to the residuals.

Uses an additive model where each learner corrects the errors of the previous ones.

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

**Answer:** The bias-variance trade-off is a fundamental concept in machine learning that describes the trade-off between two sources of error that affect the performance of predictive models:

Bias is the error introduced by approximating a real-world problem, which may be complex, by a much simpler model.

High bias typically results from models that are too simple and unable to capture the underlying patterns in the data. This leads to **underfitting**.

Variance is the error introduced by the model's sensitivity to small fluctuations in the training dataset.

High variance typically results from models that are too complex and fit the training data too closely, capturing noise as if it were a genuine signal. This leads to **overfitting**.

The bias-variance trade-off is about finding the sweet spot between a model that is too simple (high bias, underfitting) and one that is too complex (high variance, overfitting). Understanding and managing this trade-off is crucial for building models that generalize well to new, unseen data.

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

**Answer**: Support Vector Machines (SVM) are powerful supervised learning models used for classification and regression. They can efficiently handle non-linear data by using kernel functions to transform the input space into a higher-dimensional space where a linear separation is possible. Here are short descriptions of three commonly used kernel functions: Linear, Radial Basis Function (RBF), and Polynomial kernels.

**Linear Kernel-** The linear kernel is the simplest kernel function. It is used when the data is linearly separable in the original feature space.

**Radial Basis Function (RBF) Kernel-** The RBF kernel, also known as the Gaussian kernel, is a popular choice for non-linear data. It maps the input data into an infinite-dimensional space.

**Polynomial Kernel-** The polynomial kernel represents the similarity of vectors in a feature space over polynomials of the original variables, allowing the learning of non-linear models.

Each kernel function transforms the input data into a higher-dimensional space, enabling the SVM to find a linear boundary that can separate the classes in this new space. The choice of kernel depends on the nature of the data and the problem at hand.