FLIP ROBO

ASSIGNMENT - 5 MACHINE LEARNING

Q1 to Q15 are subjective answer type questions, Answer them briefly.

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

Ans:

R-squared shows how much of the variation in the data is explained by the model. However, adding more variables can make R-squared go up, which might lead to overfitting. RSS (Residual Sum of Squares) directly measures how well the model fits by summing the squared differences between observed and predicted values. A lower RSS means a better fit.
Choosing between R-squared and RSS depends on the specifics of the dataset. It's often best to use both along with other metrics, like adjusted R-squared, which adjusts for the number of variables to prevent overfitting.
R-squared measures how much of the variance in the dependent variable is explained by the independent variables, compared to a model with only the mean of the dependent variable. An R-squared close to 1.0 suggests a better fit.
RSS measures how close the model's predictions are to the actual data by summing the squared differences between observed and predicted values. A lower RSS means the model fits the data better.
R-squared is popular because it's easy to understand as a percentage of variance explained. However, it can be misleading because it always increases when more variables are added, even if they don't improve the model. Adjusted R-squared accounts for the number of variables, helping to avoid overfitting.
For non-linear models, R-squared can be harder to interpret since the mean might not be a feature. Therefore, the best measure depends on the dataset and the model used.

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

Ans:

Total Sum of Squares (TSS): This measures the total variation in your data. It's how much the data points vary from the average value.

$\text{TSS}=\sum(\text{actual value}-\text{mean value})^2$

Explained Sum of Squares (ESS): This measures how much of the total variation is explained by your regression model. It shows how much the predicted values from your model differ from the average value.

$\text{ESS}=\sum(\text{predicted value}-\text{mean value})^2$

Residual Sum of Squares (RSS): This measures the variation that your model fails to explain. It's the difference between the actual values and the predicted values from your model.

$\text{RSS}=\sum(\text{actual value}-\text{predicted value})^2$

The equation that links these three measures is:

$\text{TSS}=\text{ESS}+\text{RSS}$

This means the total variation in the data (TSS) is equal to the variation explained by the model (ESS) plus the variation not explained by the model (RSS)

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

Ans:

Regularization is needed in machine learning to prevent overfitting. Overfitting happens when a model learns the details and noise in the training data, making it perform well on the training data but poorly on new, unseen data.

Here's why regularization is important:

1. **Overfitting Problem**: When a model is too complex, it can memorize the training data, including the noise, instead of learning the actual patterns. This results in poor performance on new data.
2. **Regularization Solution**: Regularization techniques add a penalty for the complexity of the model. This encourages the model to be simpler and focus on the main patterns in the data rather than the noise.
3. **Types of Regularization**:
   - **L1 Regularization (Lasso):** Adds a penalty based on the absolute value of the coefficients. It can reduce some coefficients to zero, effectively selecting important features.
   - **L2 Regularization (Ridge):** Adds a penalty based on the square of the coefficients. It reduces the size of the coefficients but does not set them to zero.
4. **Effect of Regularization**: By penalizing larger coefficients, regularization helps create a simpler model that generalizes better to new data, avoiding overfitting.

In short, regularization makes models better at predicting new data by preventing them from becoming too complex and overfitting the training data.

4) **What is Gini–impurity index?**

Ans:

The Gini impurity index is a measure used to determine how often a randomly chosen element would be incorrectly classified. It is commonly used in decision tree algorithms to decide the best splits at each node. Here's a simple explanation:

1. **Purpose**: The Gini impurity measures the "impurity" or "messiness" of a set of items. In other words, it tells us how mixed the classes are in a node.
2. **Values**: The Gini impurity ranges from 0 to 1.
   - A Gini impurity of 0 means that all elements are of the same class, so there's no impurity (perfectly pure).
   - A Gini impurity close to 1 means that the elements are evenly distributed among different classes, indicating high impurity (very mixed).
3. **Usage in Decision Trees**: During the construction of a decision tree, the Gini impurity is used to choose which feature to split on at each step. The goal is to select the feature that results in the largest reduction of impurity (best split).

   In simple words, the Gini impurity index helps to find the best way to split data into classes by measuring how mixed up the classes are. A lower Gini impurity means a better split, where items are more purely classified into their respective groups.

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

Ans:

Yes, unregularized decision trees are likely to overfit in machine learning. This happens because they tend to become very complex, splitting the data into many small groups to perfectly match the training data. As a result, they learn the noise and specifics of the training data rather than the general patterns. This makes them perform well on the training data but poorly on new, unseen data because they haven't learned the overall trends. Regularization helps by simplifying the tree, making it focus on the important patterns and improving its performance on new data.

6) **What is an ensemble technique in machine learning?**

Ans:

An ensemble technique in machine learning combines multiple models to create a stronger, more accurate model. Instead of relying on a single model, it uses the predictions from several models and combines them to improve overall performance. This helps reduce errors and increase accuracy because different models can correct each other's mistakes.

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

Ans:

Bagging and Boosting are both ensemble techniques in machine learning, but they work in different ways:

1. **Bagging (Bootstrap Aggregating)**:
   - **How it works**: It creates multiple independent models by training them on different random

subsets of the training data. These subsets are created by sampling with replacement, meaning some data points can be repeated in each subset.

   - **Combining models**: The final prediction is made by averaging the predictions of all models (for regression) or by majority vote (for classification).
   - **Goal**: Bagging aims to reduce variance and prevent overfitting by combining multiple models to smooth out their predictions.

2. **Boosting**:
   - **How it works**: It creates multiple models sequentially, where each new model focuses on correcting the errors made by the previous models. Each model is trained on the entire dataset but pays more attention to the data points that were incorrectly predicted by earlier models.
   - **Combining models**: The final prediction is made by combining the predictions of all models, usually by weighted voting or weighted averaging.
   - **Goal**: Boosting aims to reduce bias and improve accuracy by focusing on the hard-to-predict data points, thereby creating a strong model from weak models.

In simple words, Bagging builds multiple models independently and combines their results to reduce errors, while Boosting builds models one after another, each learning from the mistakes of the previous ones to improve accuracy.

8) What is out-of-bag error in random forests?
Ans:
The out-of-bag (OOB) error in Random Forests is a way to estimate how well the model will predict new data it hasn't seen before. It works on:

1. **Bootstrap Sampling**: Each tree in the Random Forest is trained on a random subset of the original data.
2. **Out-of-Bag Data**: Some data points are left out (not used) when creating each tree because of this random sampling.
3. **Estimating Error**: These left-out data points (out-of-bag data) are used to estimate how well each tree predicts unseen data. This estimation is averaged across all trees to give the overall out-of-bag error.

Why it's Useful:
- It gives a good idea of how accurate the Random Forest will be when making predictions on new, unseen data.
- It allows us to evaluate the model without needing a separate validation dataset.
- It helps in adjusting the model's parameters to improve its performance.
  In essence, the out-of-bag error helps ensure that the Random Forest model is reliable and performs well on data it hasn't been trained on.

9) What is K-fold cross-validation?
Ans:
K-fold cross-validation is a technique used to assess the performance of a machine learning model. Here's how it works in simple terms:
1. **Dividing the Data**: The original dataset is split into K equal-sized subsets, or folds.
2. **Training and Validation**: The model is trained K times. Each time, a different fold is used as the validation set, and the remaining K-1 folds are used as the training set.
3. **Evaluation**: After training on each fold, the model's performance is evaluated by calculating a metric such as accuracy or mean squared error on the validation set (the fold that was not used for training).
4. **Average Performance**: Finally, the performance metrics from all K folds are averaged to obtain a single performance estimation for the model.

Key Points:
- **Cross-validation** helps to assess how well a model generalizes to new data.
- **K-fold** ensures that each data point is used for both training and validation, reducing bias in the evaluation.

- **Common Values**: K is typically chosen as 5 or 10, but it can vary depending on the dataset size and specific requirements.

Benefits:
- It provides a more reliable estimate of model performance compared to a single train-test split.
- It maximizes the use of data for both training and validation, which is crucial especially when data is limited.

In summary, K-fold cross-validation is a robust method to evaluate and select models, ensuring they perform well on unseen data and are not overly sensitive to the particular way the data is split.

10) What is hyper parameter tuning in machine learning and why it is done?
Ans:
Hyperparameter tuning in machine learning refers to the process of finding the best set of hyperparameters for a model. **Hyperparameters** are settings or configurations external to the model itself, usually set before the learning process begins. They affect the learning process and the structure of the model, but they are not directly learned from the data. Examples include the number of trees in a Random Forest, the learning rate of a neural network, or the depth of a decision tree.

Why Hyperparameter Tuning is Done:
1. **Optimizing Model Performance**: Different values of hyperparameters can significantly affect how well the model learns from the data and generalizes to new data. Tuning helps find the set of hyperparameters that results in the best performance metrics (like accuracy, precision, recall, etc.).
2. **Avoiding Overfitting and Underfitting**: Properly tuned hyperparameters can help strike the right balance between overfitting (model performs well on training data but poorly on test data) and underfitting (model is too simple to capture patterns in the data).
3. **Improving Efficiency**: Tuning can lead to models that converge faster during training and require fewer computational resources, making them more efficient and scalable.
4. **Model Robustness**: Hyperparameter tuning can make models more robust by ensuring they perform consistently across different datasets and scenarios.

Methods for Hyperparameter Tuning:
- **Grid Search**: Exhaustively searches a predefined set of hyperparameter combinations.
- **Random Search**: Randomly selects hyperparameter combinations to explore.
- **Bayesian Optimization**: Uses probabilistic models to predict which hyperparameters are likely to yield the best results.
- **Automated Hyperparameter Tuning**: Using algorithms and libraries that automate the process of hyperparameter search and optimization.

In essence, hyperparameter tuning is crucial in machine learning to fine-tune models for optimal performance, ensuring they generalize well and perform effectively on new data beyond the training set.

11) What issues can occur if we have a large learning rate in Gradient Descent?
Ans:
The issues that can happen if the learning rate (a parameter that controls how much to change the model in response to the estimated error each time it updates) is too large in Gradient Descent:
1. **Divergence**: Instead of settling down to find the best solution, the model's parameters might start jumping around erratically, making it impossible to converge to a good solution.
2. **Instability**: The updates to the model's parameters could be so big that they swing wildly, making the learning process unstable and unpredictable.
3. **Overfitting**: The model might become too specialized to the training data, losing its ability to generalize to new, unseen data.
4. **Poor Performance**: With a large learning rate, the model may not learn the underlying patterns in the data effectively, leading to worse performance.

To avoid these issues, it's important to choose a moderate learning rate and monitor how the model performs during training to ensure it's improving steadily without these problems.

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

Ans:

Logistic Regression is a linear classification model, which means it assumes a linear relationship between the input variables (features) and the output (class label). Therefore, it works well when the decision boundary that separates classes can be approximated by a linear function.

Non-linear Data and Logistic Regression:

1. **Limitation of Linearity**: Logistic Regression cannot capture non-linear relationships between features and the target variable. If the data is inherently non-linear, meaning the boundary between classes is not linear but curved or complex, Logistic Regression will struggle to model and classify such data accurately.
2. **Decision Boundary**: Logistic Regression models a decision boundary as a straight line (or hyperplane in higher dimensions). If the true boundary between classes is non-linear (e.g., curved or irregular), Logistic Regression will produce a decision boundary that is too simple and may result in poor classification performance.
3. **Performance**: While Logistic Regression can sometimes approximate non-linear relationships to some extent, its performance on non-linear data is generally inferior compared to non-linear classifiers such as Decision Trees, Support Vector Machines with non-linear kernels, or Neural Networks.

Alternative Approaches for Non-linear Data:

- **Decision Trees and Random Forests**: These methods can handle non-linear relationships by partitioning the feature space into regions based on the data distribution.
- **Support Vector Machines (SVMs)**: SVMs with non-linear kernels (e.g., polynomial kernel, Gaussian RBF kernel) can map data into higher-dimensional spaces where non-linear relationships can be captured by linear decision boundaries.
- **Neural Networks**: Particularly deep neural networks with non-linear activation functions and multiple layers can learn complex non-linear mappings from data.

Conclusion:

In summary, while Logistic Regression is effective for linearly separable data, it is not suitable for handling non-linear data directly due to its inherent linearity assumption. For non-linear classification tasks, it's better to use algorithms and models designed to capture and learn complex, non-linear relationships in the data effectively.

13) Differentiate between Adaboost and Gradient Boosting.

Ans:

Adaboost (Adaptive Boosting):

- **Approach**: Adaboost trains a sequence of weak models (like shallow trees) where each new model corrects errors made by the previous ones. It adjusts how much each data point influences the model based on whether it was predicted correctly or not.
- **Focus**: It focuses on getting difficult examples right by giving them more weight in each round of training.
- **Final Prediction**: Combines the predictions of all weak models with weights based on their performance.

Gradient Boosting:

- **Approach**: Gradient Boosting also builds a sequence of models, typically decision trees, but each new model corrects the errors of the previous ones by minimizing a loss function (like error or log loss) using gradients (directions to move to reduce errors).
- **Training**: It adjusts the parameters of each new model to reduce the mistakes made by the previous ones.
- **Flexibility**: It can handle more complex relationships in data and includes methods to prevent overfitting.

Key Differences:

- **Learning Style**: Adaboost adjusts how much each example influences the model's training. Gradient Boosting adjusts the model itself to improve its predictions.
- **Model Complexity**: Adaboost often uses simple models (like shallow trees). Gradient Boosting can use more complex models and is more flexible.
- **Final Output**: Adaboost combines models based on their accuracy. Gradient Boosting combines models by minimizing overall errors sequentially.

Summary:
Adaboost focuses on difficult examples and combines simple models, while Gradient Boosting adjusts complex models to minimize errors, making it more flexible but potentially more powerful for handling various types of data.

14) What is bias-variance trade off in machine learning?
Ans:
The bias-variance trade-off in machine learning is about finding the right balance between a model's ability to accurately capture patterns in the data (bias) and its ability to generalize well to new, unseen data (variance).

- **Bias**: Indicates how much a model's predictions differ from the true values. High bias means the model is too simple to capture complex patterns.
- **Variance**: Shows how much a model's predictions vary across different datasets. High variance means the model is too sensitive to the training data and may not generalize well.

When you reduce bias, you often increase variance, and vice versa. The goal is to find a model that minimizes both bias and variance to make accurate predictions on new data. It's a balancing act in choosing models that are neither too simple (underfitting) nor too complex (overfitting) for the problem at hand.

15) Give short description each of Linear, RBF, Polynomial kernels used in SVM.
Ans:
Here are short descriptions of each type of kernel used in Support Vector Machines (SVMs):

1. **Linear Kernel**:
   - **Description**: The linear kernel computes the dot product between two vectors. It is suitable when the data is linearly separable, meaning a straight line (or hyperplane in higher dimensions) can separate the classes effectively.
   - **Use Case**: It's used when there's a clear linear boundary between classes in the feature space.
2. **RBF Kernel (Radial Basis Function)**:
   - **Description**: The RBF kernel measures the similarity between data points based on the Euclidean distance. It maps data into higher-dimensional space where it becomes easier to separate classes that are not linearly separable in the original space.
   - **Use Case**: It's effective for non-linear separation and works well when the decision boundary is complex or not easily defined by a straight line.
3. **Polynomial Kernel**:
   - **Description**: The polynomial kernel computes the similarity between data points as the polynomial of the original variables. It allows SVMs to handle non-linear relationships by transforming the original features into higher dimensions.
   - **Use Case**: It's useful when the decision boundary between classes is polynomial (e.g., quadratic, cubic) rather than linear.

Summary:

- **Linear Kernel**: Straightforward for linearly separable data.
- **RBF Kernel**: Versatile for complex, non-linear boundaries.
- **Polynomial Kernel**: Suitable for polynomial decision boundaries.

Each kernel offers a different approach to mapping data into a higher-dimensional space where it's easier to separate classes, making SVMs flexible for various types of classification problems.