ASSIGNMENT 3 = MACHINE LEARNING (SUBJECTIVE)

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-squared (R^2) and Residual Sum of Squares (RSS) are both used to evaluate the goodness of fit of a regression model, but they serve different purposes. Let's explore each measure:

1. **R-squared (R^2)**:
   - **Definition**: R-squared represents the proportion of the variance in the dependent variable (response) that is explained by the independent variables (predictors) in the model.
   - **Interpretation**: R-squared ranges from 0 to 1. A higher R-squared indicates that a larger proportion of the variability in the response variable is accounted for by the model. It measures how well the model fits the data.
   - **Advantages**:
     - Provides an overall assessment of model fit.
     - Easy to interpret: Higher R-squared values are desirable.
     - Widely used in practice.
   - **Limitations**:
     - R-squared tends to increase with the addition of more predictors, even if they are not truly relevant.
     - Does not indicate whether the model assumptions (such as linearity, homoscedasticity, and normality of residuals) are met.
     - Can be misleading when applied to complex models.
   - **Use Case**: R-squared is commonly reported in regression summaries, but it should not be the sole criterion for model evaluation.
2. **Residual Sum of Squares (RSS)**:
   - **Definition**: RSS represents the sum of the squared differences between the observed values and the predicted values (residuals) in the regression model.
   - **Interpretation**: Lower RSS values indicate better model fit because they imply smaller discrepancies between observed and predicted values.
   - **Advantages**:
     - Focuses directly on the quality of predictions.
     - Sensitive to deviations from the true values.
     - Useful for comparing different models.
   - **Limitations**:
     - RSS alone does not provide a standardized measure of fit.
     - It does not account for the total variability in the response variable.
     - Does not address the number of predictors.
   - **Use Case**: RSS is often used during model selection (e.g., comparing nested models) or when assessing the impact of specific predictors.

Conclusion:

- **R-squared** is valuable for understanding the overall fit of the model and explaining the proportion of variance explained.
- **RSS** is useful for assessing prediction accuracy and comparing models.

- Neither measure is inherently better; they serve different purposes. Consider using both in conjunction to gain a comprehensive understanding of model performance.

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 eachother.

Answer: Let's delve into some useful regression analysis concepts like TSS, ESS and RSS:

1. **Total Sum of Squares (TSS)**:
   - **Definition**: TSS represents the total variability in the observed dependent variable (response) around its overall mean.
   - **Formula**: Mathematically, TSS is calculated as the sum of squared differences between each observed value and the overall mean: [TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2] where:
     - (y_i) represents the observed dependent variable.
     - (\bar{y}) denotes the mean of the dependent variable.
2. **Explained Sum of Squares (ESS)**:
   - **Definition**: ESS quantifies the variability explained by the regression model. It measures how well the model fits the data.
   - **Formula**: ESS is computed as the sum of squared differences between the predicted values and the overall mean: [ESS = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2] where:
     - (\hat{y}_i) represents the predicted value of the dependent variable.
3. **Residual Sum of Squares (RSS)**:
   - **Definition**: RSS captures the unexplained variability or the discrepancies between the observed values and the predicted values (residuals).
   - **Formula**: RSS is calculated as the sum of squared residuals: [RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2] where:
     - (y_i) represents the observed value of the dependent variable.
     - (\hat{y}_i) denotes the predicted value of the dependent variable.
4. **Relationship Between TSS, ESS, and RSS**:
   - The relationship can be expressed as follows: [TSS = ESS + RSS]
     - TSS represents the total variability.
     - ESS accounts for the variability explained by the model.
     - RSS captures the unexplained variability.

Conclusion: TSS decomposes into ESS (explained by the model) and RSS (unexplained by the model), providing insights into the goodness of fit and the effectiveness of the regression model.

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

Answer: Regularization plays a crucial role in enhancing the generalization ability of machine learning models. Let's explore why it is needed:

<u>1. Overfitting Prevention:</u>

- **Overfitting** occurs when a model becomes too complex and fits the training data too closely. As a result, it performs poorly on unseen data.

- **Regularization** helps prevent overfitting by adding a penalty term to the model's objective function during training. This discourages the model from fitting the training data too precisely and promotes simpler models that generalize better to unseen data.

2. Balancing Bias and Variance:

- **Bias** refers to errors when a statistical model doesn't fit real-world data perfectly. High bias occurs when the model fails to learn patterns in the data, leading to poor performance.

- **Variance** arises when the model learns noise present in the data. High variance results in overfitting.

- **Regularization** helps strike a balance between bias and variance, known as the 'Bias-Variance Trade-off'. It prunes the model from getting overfitted to the training data.

3. Common Regularization Techniques:

- **Lasso Regularization (L1):** Adds the absolute value of the coefficient as a penalty term to the loss function. It encourages sparsity by shrinking some coefficients to zero, leading to feature selection.

- **Ridge Regularization (L2):** Adds the squared magnitude of the coefficient as a penalty term. It reduces the impact of large coefficients and encourages smoother models.

- **Elastic Net Regularization:** Combines L1 and L2 regularization, providing a balance between feature selection and coefficient shrinkage.

Conclusion: Regularization ensures that machine learning models don't succumb to overfitting or underfitting, making them more robust and less complex.

4. What is Gini–impurity index?

Answer: Gini impurity index (GII) is a measure used in decision tree algorithms to quantify a dataset's impurity level or disorder. Let's explore its key aspects:

1. GII: It calculates the likelihood of an incorrect classification when a randomly selected data point is assigned a class label based on the distribution of classes in a particular node. If all elements in a node belong to a single class, the Gini impurity is low (indicating purity). Conversely, if the elements are distributed across multiple classes, the impurity is higher.

2. Mathematical. Calculations: Gini impurity for a node with \(K\) classes is given by:
\[\text{Gini Impurity} = 1 - \sum_{i=1}^{K} p_i^2 \]
where:\(p_i\) represents the proportion of instances belonging to class \(i\) within the node.The Gini impurity ranges from 0 (pure node) to 0.5 (maximum impurity).

3. Role in Decision Trees:
- Decision trees use the Gini impurity to determine the best feature for splitting nodes during tree construction.
- When evaluating potential splits, the algorithm calculates the Gini impurity for each feature and selects the split that minimizes impurity (maximizes purity).

<u>4. Comparison with Other Metrics:</u>
- The Gini impurity is closely related to other metrics like entropy and information gain.
- While entropy measures the disorder or uncertainty in a system, the Gini impurity focuses on misclassification probabilities.
- Both metrics guide decision tree splits, but the choice between them depends on the specific problem and context.

In summary, the Gini impurity index helps decision trees find optimal splits by assessing the impurity of data at each node, leading to effective classification models.

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

Answer: 'Unregularized decision trees' can indeed be susceptible to **overfitting**, which occurs when a model becomes too complex and captures noise or irrelevant patterns in the training data. Let's delve into the reasons behind this:

<u>1. Complexity and Noise:</u>
- Decision trees aim to fit the training data well, but if left unrestricted, they can become overly complex.
- When a tree grows too large, it may end up with very few instances in each leaf node. Consequently, the estimated mean value at each leaf node might not accurately represent the underlying pattern.
- Adding nodes based on features that don't genuinely contribute information can lead to overfitting. For instance, a feature chosen randomly might degrade the model's generalization to new data.

<u>2. Specific Training Observations vs. General Rules:</u>
- Unpruned trees can learn rules specific to training observations rather than general rules applicable to unseen data.
- The objective of decision trees is to classify training points effectively, not necessarily predict well on new, unseen data.
- Overfitting occurs when the model fits the training data too closely, resulting in poor performance on unseen data.

<u>3. Pruning and Generalization:</u>
- Pruning is a technique to prevent overfitting by simplifying the tree.
- It involves removing nodes that do not significantly improve the model's performance.
- Pruned trees are more generalized and less prone to overfitting.

In summary, unregularized decision trees should be pruned or constrained to prevent overfitting. By doing so, we create simpler and more robust models that generalize better to new data.

6. What is an ensemble technique in machine learning?
Answer: "Ensemble learning" is a powerful machine learning technique that enhances accuracy and resilience by combining predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

Here's a concise overview of ensemble techniques:

<u>1. Concept:</u>
- Ensemble methods create an ensemble (group) of models and then combine their

predictions to produce improved results.
- Instead of relying on a single model, ensemble methods aggregate the insights from diverse models to enhance overall performance.

<u>2. Advantages of Ensemble Techniques:</u>
- **Increased Accuracy:** Ensemble models often outperform individual models by reducing bias and variance.
- **Robustness:** Combining multiple models provides resilience against uncertainties in the data.
- **Generalization:** Ensemble methods generalize well to unseen data.

<u>3. Common Ensemble Techniques:</u>

- **Bagging (Bootstrap Aggregating):**
  - Bagging builds multiple models (usually decision trees) on bootstrapped subsets of the training data.
  - It averages their predictions to reduce variance and improve stability.
  - Example: Random Forest.

- **Boosting:**
  - Boosting sequentially trains weak models (often decision trees) by emphasizing misclassified instances.
  - It combines their predictions to create a strong model.
  - Example: AdaBoost & Gradient Boosting.

- **Stacking:**
  - Stacking combines predictions from different models using a meta-model (another model).
  - It learns to weigh the individual models' outputs optimally.
  - Example: Stacking a linear regression model with a random forest model.

<u>4. Real-Life Analogy:</u>
- Imagine you're a movie director creating a short film. Instead of relying solely on your own judgment, you seek opinions from various people (friends, colleagues, critics) to make a better decision.
- Similarly, ensemble techniques combine diverse models to create a more precise prediction by considering multiple perspectives.

In summary, ensemble learning harnesses the collective wisdom of multiple models, resulting in robust and accurate predictions across various domains.

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

Answer: The major differences between **Bagging** and **Boosting** techniques in machine learning:

<u>1. Bagging (Bootstrap Aggregating):</u>
- **Objective:** Bagging aims to **reduce variance** and **increase stability** by training independent models on different subsets of data.
- **Process:**
  - Multiple subsets (bootstrap samples) are created from the original dataset by selecting observations with replacement.
  - A base model (usually decision trees) is created on each of these subsets independently.
  - Each model learns in parallel, and their predictions are combined to form the final ensemble prediction.

- Use Case: Bagging is effective for high variance and low bias models.
- Example: Random Forest model uses bagging, where multiple decision trees with higher variance are combined.

<u>2. Boosting:</u>
- **Objective:** Boosting aims to build a **strong classifier** by iteratively correcting errors made by previous models.
- **Process:**
  - Models are built sequentially, with each new model focusing on the errors made by the previous one.
  - The second model corrects the errors of the first, the third corrects the errors of the first two, and so on.
  - The final ensemble prediction is a weighted average of the individual model predictions.
- **Use Case:** Boosting is suitable when the model must be **adaptive to errors** and improve accuracy.
- Example: Gradient Boosting is a popular boosting technique that iteratively adjusts models to minimize bias and improve accuracy.

In summary, Bagging reduces variance and focuses on stability, while **Boosting** reduces bias and aims for accuracy by iteratively adjusting models. Both techniques enhance predictive performance by combining weak learners into a stronger ensemble model.

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

Answer: In ‘Random Forests’, the ‘out-of-bag (OOB) error’ is a valuable performance metric. Let's discuss:

<u>1. OOB:</u> This error provides an estimate of how well the ‘Random Forest’ model generalizes to unseen data. It is calculated using the samples that were **not included** in the training of individual trees within the forest.

<u>2. Process:</u>
- During the creation of each decision tree in the Random Forest:
- A bootstrap sample (subset of the original data with replacement) is used for training.
- Some data points are left out (out-of-bag samples) from this bootstrap sample.
- These out-of-bag samples are used to evaluate the tree's performance.
- The OOB error is computed by aggregating the errors made by each tree on its corresponding out-of-bag samples.

<u>3. Significance:</u>
- The OOB error serves as an ‘unbiased estimate’ of the model's performance on unseen data.
- It helps assess how well the ensemble model will perform when faced with new observations.

<u>4. Calculation:</u>
- In Python's scikit-learn library, you can obtain the OOB error using the `oob_score_` attribute of the Random Forest classifier or regressor.

In summary, the OOB error provides insight into how well the Random Forest model will generalize to unseen data, making it a crucial evaluation metric during model training.

9. What is K-fold cross-validation?

Answer: Let's explore 'k-fold cross-validation', a valuable technique for evaluating machine learning models:

1. Phenomenon:
- 'K-fold cross-validation' is a resampling procedure used to evaluate machine learning models on a limited data sample.
- It addresses the need to assess model performance more robustly than a single train/test split.

2. Procedure:
- The dataset is divided into **k subsets** (or folds).
- The model is trained and evaluated **k times**, using a different fold as the validation set each time.
- Performance metrics from each fold are **averaged** to estimate the model's generalization performance.

3. Significance:
- **Generalization:** K-fold CV provides an estimate of how well the model will perform on unseen data.
- **Bias Reduction:** It results in a less biased or optimistic estimate of model skill compared to simple train/test splits.
- **Robustness:** By repeatedly evaluating the model, it accounts for variations in data splits.

4. Choosing k:
- Common values for k are '5' or '10'.
- Larger k values lead to more robust estimates but require more computation.

5. Variations:
- **Stratified k-fold:** Ensures class distribution balance in each fold.
- **Repeated k-fold:** Repeats k-fold CV multiple times for better reliability.

In summary, k-fold cross-validation helps us assess model performance, choose hyperparameters, and understand how well our model generalizes to new data.

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

Answer: In the realm of machine learning, 'Hyperparameter Tuning' plays a pivotal role. Let's delve into its significance and explore why it is essential:

1. Define:
- 'Hyperparameters' are configuration variables that **control the learning process** of a machine learning model.
- Unlike model parameters (such as weights and biases), hyperparameters are **not learned from the data** but are set before the training process begins.
- They express critical properties of the model, such as its complexity, learning rate, or capacity.

2. Hyperparameter Tuning:
- **Optimal Performance:** The goal of hyperparameter tuning is to find the **best values** for these hyperparameters.
- **Impact on Model Metrics:** Properly tuned hyperparameters can significantly affect the model's accuracy, generalization and other performance metrics.
- **Avoiding Overfitting:** Tuning helps prevent overfitting or underfitting by adjusting hyperparameters appropriately.

<u>3. Different Types of Hyperparameters:</u>

- **Learning Rate:** Controls the step size taken by the optimizer during each iteration of training. Too small or too large rates can lead to convergence issues.
- **Epochs:** Represents the number of times the entire training dataset is passed through the model during training. More epochs can improve performance but may risk overfitting.
- **Number of Layers:** Determines the depth of neural networks, impacting model complexity.
- **Number of Nodes per Layer:** Influences the model's capacity to capture complex relationships.
- **Architecture:** Overall structure of neural networks, including layer count and connections.

<u>4. Tuning Strategies:</u>

- **Grid Search:** Exhaustively explores a predefined set of hyperparameter values.
- **Random Search:** Randomly samples hyperparameter values within specified ranges.
- **Bayesian Optimization:** Uses probabilistic models to guide the search efficiently.

In summary, hyperparameter tuning ensures that our machine learning models perform optimally by selecting the right configuration settings for learning processes.

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

Answer: Several issues can arise while using "Gradient Descent" with a large learning rate:

<u>1. Overshooting the Minima:</u>

- The learning rate acts as the step size (denoted as η) during each iteration.
- If the learning rate is too large, Gradient Descent may jump over the minima it's trying to reach.
- This results in overshooting, leading to oscillations around the minimum or even outright divergence.

<u>2. Unstable Training:</u>

- A large learning rate can cause the loss function to jump erratically in the direction of steepest descent.
- The training process becomes **unstable**, making it difficult to converge to an optimal solution.

<u>3. Divergence:</u>

- In extreme cases, a very high learning rate can cause the optimization process to diverge.
- The model's weights may grow uncontrollably, leading to poor performance and failure to learn.

<u>4. Slow Convergence:</u>

- Conversely, if the learning rate is too small, training becomes slow.
- The model may get **stuck** with high training error, preventing effective learning.

<u>5. Balancing Act:</u>

- Choosing an appropriate learning rate is crucial.
- It's essential to strike a balance between fast convergence and stable learning.

In summary, a large learning rate can lead to overshooting, instability, divergence, and slow convergence. Properly tuning the learning rate is essential for effective training of neural networks.

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

Answer: 'Logistic Regression' is traditionally used as a linear classifier, meaning it is used to come up with a hyperplane in feature space to separate observations that belong to a class from all the other observations that do not belong to that class. The decision boundary is thus linear.

However, Logistic Regression can be adapted to handle non-linear data through techniques like feature engineering. For instance, non-linear transformations can be applied to map the original feature space into a higher dimension space, where the linear model can separate the data more easily. This process involves creating new features from the original features in such a way that the new feature space can be separated linearly.

So, while Logistic Regression itself is a linear method, it can be used for classification of non-linear data by leveraging non-linear feature engineering. This allows the model to capture more complex relationships between the features and the target variable.

13. Differentiate between Ada-boost and Gradient Boosting.

Answer: Key differences between AdaBoost and Gradient Boosting:

1. AdaBoost:

- AdaBoost is an ensemble technique where a number of weak learners are combined together to form a strong learner.
- Each weak learner is developed as decision stumps (a stump is a tree with just a single split and two terminal nodes) that are used to classify the observations.
- AdaBoost tweaks the instance weights at every interaction.
- It assigns weights to both observations at the end of every tree and weights (scores) to every classifier.
- In AdaBoost, every classifier has a different weightage on the final prediction.

2. Gradient Boosting:

- Gradient Boosting is a generic algorithm to find approximate solutions to the additive modeling problem.
- It is much more flexible than AdaBoost.
- The difference between Gradient Boost and Ada Boost lies in what it does with the underfitted values of its predecessor.
- Contrary to AdaBoost, Gradient Boosting tries to fit the new predictor to the residual errors made by the previous predictor.
- Gradient Boosting can be used for both Classification and Regression.
- It works on the principle of the stagewise addition method, where multiple weak learning algorithms are trained and a strong learner algorithm is used as a final model from the addition of multiple weak learning algorithms trained on the same dataset.

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

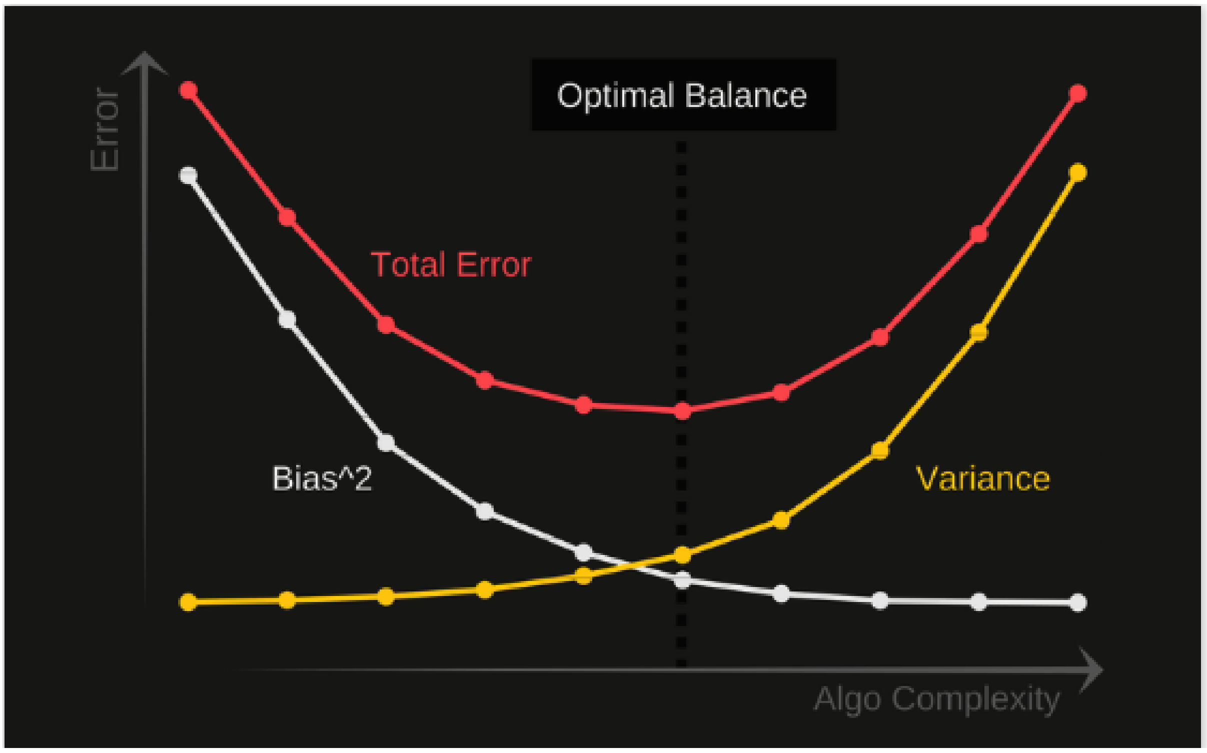
Answer: Bias-variance tradeoff is a fundamental concept in machine learning that describes the relationship between a model's complexity, the accuracy of its predictions, and how well it can make predictions on previously unseen data.

1. Bias: Bias is the inability of a machine learning model to capture the true relationship between the data variables. It is caused by the erroneous assumptions that are inherent to the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).

2. Variance: Variance is the amount by which our model would change if we estimated it using a different training dataset. Models with high variance pay a lot of attention to fitting the noise in the training data, and hence the model's performance can be poor on unseen data (overfitting).

Trade-off refers to the balance between a model's ability to minimize bias and variance. Striking the right balance is crucial for achieving optimal model performance. If the model is too simple, it may have high bias and low variance. If the model is too complex, it may have low bias and high variance. The goal is to find the right level of model complexity that achieves low bias and low variance.

Diagram representation of bias-variance tradeoff



In the graph above, as the complexity of the model increases, the bias decreases and the variance increases. The total error is the sum of the bias squared, the variance, and the irreducible error. The goal is to choose a model complexity at the bottom of the U-shaped curve, which gives us the lowest total error.

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

Answer: Linear, RBF, and Polynomial kernels used in Support Vector Machines (SVM) can be described as follows:

1. Linear Kernel: Linear Kernel is used when the data is linearly separable, that is, it can be separated using a single line. It is one of the most common kernels to be used. It is mostly used when there are a large number of features in a particular dataset. The Linear Kernel uses the original feature space without any transformation.

2. Radial Basis Function (RBF) Kernel: The RBF kernel is a type of kernel function that can be used with the SVM classifier to transform the data into a higher-dimensional space, where it is easier to find a separation boundary. The RBF kernel is defined by a single parameter, gamma, which determines the width of the kernel and therefore the complexity of the model. It measures similarity between two data points in infinite dimensions and then approaches classification by majority vote.

3. Polynomial Kernel: A polynomial kernel is a type of SVM kernel that uses a polynomial function to map the data into a higher-dimensional space. It does this by taking the dot product of the data points in the original space and the polynomial function in the new space. It represents the similarity of vectors in the training set of data in a feature space over polynomials of the original variables used in the kernel.

These kernels help in transforming the input data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces.