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 is generally considered a better measure of goodness of fit in regression models compared to the Residual Sum of Squares (RSS). Here's why:

1. **Interpretability**: R-squared provides a normalized value (between 0 and 1) that represents the proportion of variance in the dependent variable that is explained by the independent variables. This makes it easier to interpret compared to RSS, which can take any value from 0 to infinity and is dependent on the scale of the data.
2. **Comparison**: R-squared allows for easy comparison between models. You can compare the goodness of fit across different models or datasets directly. RSS, on the other hand, is more difficult to compare since it depends on the total number of observations and the scale of the data.
3. **Sensitivity to Model Complexity**: While RSS can decrease with added predictors, it does not account for the complexity of the model. R-squared can be adjusted (Adjusted R-squared) to penalize for the addition of unnecessary predictors, providing a more reliable measure of goodness of fit when comparing models with different numbers of predictors.

R-squared is often preferred due to its interpretability, comparability, and ability to adjust for model complexity. However, it's important to consider both metrics alongside other diagnostic tools for a comprehensive assessment 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 each other.

Ans- In regression analysis, TSS, ESS, and RSS are key metrics that help evaluate the model's performance. Here's a breakdown of each:

1. **Total Sum of Squares (TSS)**:

- TSS measures the total variance in the dependent variable. It represents the total variation from the mean of the observed values.
- Formula**:** **TSS** = $\sum(y_i - \bar{y})^2$
  where y_i is the observed value and $\bar{y}$ is the mean of the observed values.

2. **Explained Sum of Squares (ESS)**:

- ESS measures the variation explained by the regression model. It quantifies how much of the total variance is captured by the model's predictions.
- Formula**:** **ESS**=$\sum(\hat{y}_i - \bar{y})^2$

  where $\hat{y}_i$ is the predicted value from the regression model.

3. **Residual Sum of Squares (RSS)**:

- RSS measures the variation that is not explained by the model. It reflects the sum of the squared differences between the observed values and the predicted values.
- Formula**:** **RSS**= $\sum(y_i - \hat{y}_i)^2$

Relationship between TSS, ESS, and RSS

The relationship among these three metrics is given by the equation:

$$TSS=ESS+RSS$$

This equation indicates that the total variance in the data (TSS) can be decomposed into the variance explained by the model (ESS) and the variance that is not explained (RSS). This decomposition is fundamental to understanding the effectiveness of a regression model.

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

Ans- Regularization is a crucial technique in machine learning for several reasons:

1. **Prevent Overfitting**: One of the primary purposes of regularization is to prevent overfitting, which occurs when a model learns the training data too well, including its noise and outliers. Regularization helps constrain the model, allowing it to generalize better to unseen data.
2. **Simplify Models**: Regularization encourages simpler models by penalizing complexity. This is especially important in high-dimensional spaces, where complex models can fit the training data closely but perform poorly on new data.
3. **Improve Model Stability**: Regularization can enhance the stability of the model by reducing sensitivity to fluctuations in the training data. This leads to more consistent predictions when the model encounters new, unseen data.
4. **Feature Selection**: Some regularization techniques, like Lasso (L1 regularization), can effectively perform feature selection by driving some coefficients to zero. This helps in identifying the most important features in a dataset and can lead to more interpretable models.
5. **Robustness to Noise**: By imposing constraints on the model parameters, regularization makes the model more robust to noise in the data, leading to better performance in real-world scenarios where data may be imperfect.
6. **Control Complexity**: Regularization techniques (like L1 and L2) provide a way to control the complexity of the model explicitly, allowing practitioners to strike a balance between fitting the training data and maintaining generalizability.

Common Regularization Techniques

- **L1 Regularization (Lasso)**: Adds the absolute value of the coefficients as a penalty term to the loss function. It can lead to sparse models (many coefficients are zero).
- **L2 Regularization (Ridge)**: Adds the square of the coefficients as a penalty term, which helps to shrink the coefficients but does not force them to zero.
- **Elastic Net**: Combines both L1 and L2 regularization, providing a balance between the two.

4. What is Gini–impurity index?

Ans- Definition

The Gini impurity index is a metric used to measure the impurity or purity of a dataset in the context of classification tasks, particularly in decision tree algorithms. It quantifies how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

The Gini impurity is calculated using the following formula:

$$\boldsymbol{Gini} = \mathbf{1} - \sum (\boldsymbol{p_i})^{\mathbf{2}}$$

where:

- p_i is the proportion of instances in class i in the dataset.

Interpretation

- **Range**: The Gini impurity index ranges from 0 to 1.
  - **0**: Indicates perfect purity, meaning all elements belong to a single class (i.e., the dataset is homogeneous).
  - **1**: Indicates maximum impurity, meaning the classes are evenly distributed.

Example

Suppose we have a dataset with three classes: A, B, and C. If the distribution of classes is as follows:

- Class A: 4 instances
- Class B: 3 instances
- Class C: 2 instances

The total number of instances is 9. The proportions are:

- $P_A = \frac{4}{9}$
- $P_B = \frac{3}{9}$
- $P_C = \frac{2}{9}$

Calculating the Gini impurity:

$$Gini = 1 - \left(\left(\frac{4}{9}\right)^2 + \left(\frac{3}{9}\right)^2 + \left(\frac{2}{9}\right)^2\right)$$

$$= 1 - \left(\frac{16}{81} + \frac{9}{81} + \frac{4}{81}\right) = 1 - \frac{29}{81} = \frac{52}{81} \approx 0.64$$

Usage

In decision trees, the Gini impurity is used to evaluate the quality of splits. The algorithm will choose splits that minimize the Gini impurity, aiming for child nodes that are as pure as possible (i.e., containing instances predominantly from a single class). Lower Gini impurity indicates a better split in terms of class separation.

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

Ans- Yes, unregularized decision trees are prone to overfitting. Here are the reasons why:

1. **High Complexity**: Decision trees can create very complex models that perfectly fit the training data by making many splits. They can capture intricate patterns and noise in the data, which leads to a model that performs well on the training set but poorly on unseen data.
2. **Sensitive to Data Variations**: Decision trees are highly sensitive to variations in the data. A small change in the training data can result in a completely different tree structure, which can cause instability and overfitting.
3. **Deep Trees**: When decision trees are allowed to grow without constraints, they tend to become very deep. Each additional split can lead to more branches that fit the training data closely, including any noise or outliers, making the model less generalizable.
4. **Lack of Pruning**: Without regularization techniques such as pruning (removing branches that have little importance), a decision tree can retain many irrelevant splits, further contributing to overfitting.

5. **Bias-Variance Tradeoff**: Unregularized decision trees typically have low bias (they can model complex relationships) but high variance (they can vary significantly with different training data). This combination leads to overfitting.

Mitigation Strategies

To reduce the risk of overfitting in decision trees, several regularization techniques can be applied, including:

- **Pruning**: Removing branches that have little significance after the tree has been fully grown.
- **Setting Maximum Depth**: Limiting the maximum depth of the tree to prevent it from becoming too complex.
- **Minimum Samples Split**: Requiring a minimum number of samples to split a node, which can prevent small, potentially noisy splits.
- **Minimum Samples Leaf**: Setting a minimum number of samples that must be present in a leaf node, which helps ensure that leaves contain enough data for reliable predictions.

By applying these techniques, decision trees can be made more robust and less prone to overfitting.

6. What is an ensemble technique in machine learning?

Ans- Ensemble techniques in machine learning refer to methods that combine multiple models to improve overall performance, stability, and accuracy compared to individual models. The idea is that by aggregating the predictions from a group of models, the ensemble can leverage the strengths of each model while mitigating their weaknesses.

Key Concepts

1. **Diversity**: The models in an ensemble should be diverse, meaning they should make different errors on the data. This diversity helps in reducing variance and improving generalization.
2. **Aggregation Methods**: The predictions from individual models can be combined using various aggregation methods, including:
    - **Voting**: For classification tasks, where the majority vote among models is taken as the final prediction.
    - **Averaging**: For regression tasks, where the average of predictions from different models is calculated.

Common Ensemble Techniques

1. **Bagging (Bootstrap Aggregating)**:
    - Involves training multiple instances of the same model on different subsets of the training data, typically created through bootstrapping (random sampling with replacement).
    - Example: **Random Forest** is an ensemble method based on bagging that combines multiple decision trees.
2. **Boosting**:
    - Involves training models sequentially, where each new model attempts to correct the errors made by the previous ones. Weights are adjusted for misclassified instances, giving them more emphasis in subsequent models.
    - Example: **AdaBoost** and **Gradient Boosting** are popular boosting algorithms.
3. **Stacking**:
    - Involves training multiple models (base learners) and then using another model (meta-learner) to combine their predictions. The meta-learner is trained on the outputs of the base learners.
    - This technique allows for a more flexible combination of models, leveraging their strengths.

Advantages of Ensemble Techniques

- **Improved Accuracy**: Ensembles often achieve higher accuracy than individual models, especially on complex datasets.
- **Robustness**: They are generally more robust to noise and overfitting, as the aggregation helps smooth out individual errors.

- **Versatility**: Ensembles can be applied to a wide range of algorithms and can improve performance across various types of problems.

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

Ans-

Key differences	Bagging	Boosting
Approach	Involves training multiple instances of the same model on different subsets of the training data. These subsets are created through bootstrapping (random sampling with replacement). Each model is trained independently and in parallel, meaning they do not influence each other.	Involves training models sequentially. Each new model is trained to correct the errors made by the previous models, focusing on the instances that were misclassified. Models are dependent on each other; the training of one model affects the training of the next.
Model Weights	Each model in the ensemble is given equal weight when making the final prediction. The final output is usually determined by majority voting (for classification) or averaging (for regression).	Models are assigned different weights based on their performance. Models that perform poorly are given higher weights in subsequent iterations, emphasizing their predictions.
Focus on Data	Aims to reduce variance by averaging the predictions of several models trained on different subsets of data. It works well when individual models are prone to overfitting.	Focuses on reducing bias by combining weak learners (models that perform slightly better than random guessing) into a strong learner. It enhances the performance by iteratively addressing the errors of previous models.
Parallel vs. Sequential Training	Models can be trained in parallel, making it computationally efficient, especially with large datasets.	Models are trained sequentially, which can lead to longer training times, but typically results in a more accurate model.
Examples	**Random Forest** is a popular bagging algorithm that builds multiple decision trees on bootstrapped samples and averages their predictions.	**AdaBoost** and **Gradient Boosting** are well-known boosting algorithms that sequentially improve upon previous models.

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

Ans- Out-of-bag (OOB) error is a key concept in the context of Random Forests, a popular ensemble learning method that uses bagging (bootstrap aggregating) to improve predictive performance. Here's a detailed explanation of OOB error:

Definition

Out-of-bag error refers to the error estimate for a Random Forest model that is calculated using the samples that were not included in the bootstrap samples for each individual tree in the forest.

How It Works

1. **Bootstrapping**: When training each decision tree in a Random Forest, a bootstrap sample is created by randomly selecting data points with replacement from the training dataset. Typically, about one-third of the data points will not be included in each bootstrap sample.
2. **OOB Samples**: The data points that are not included in the bootstrap sample for a particular tree are referred to as out-of-bag samples for that tree. Each tree in the Random Forest will have its own set of OOB samples.
3. **Error Estimation**:
    - For each OOB sample, the prediction can be made using only the trees that did not have that sample in their bootstrap set.
    - The OOB error is then computed by averaging the predictions of all the trees for each OOB sample and comparing them to the actual values.

Advantages of OOB Error

- **No Additional Validation Set Needed**: OOB error provides an internal validation measure, eliminating the need for a separate validation dataset. This is particularly useful when the dataset is limited in size.
- **Efficient Use of Data**: Since OOB samples are used to estimate the model's performance, it allows for efficient utilization of the entire dataset during training.
- **Good Estimate of Generalization Error**: OOB error serves as a reliable estimate of the model's generalization error, helping to gauge how well the Random Forest is likely to perform on unseen data.

9. What is K-fold cross-validation?

Ans- K-fold cross-validation is a resampling technique used to evaluate the performance of a machine learning model. It helps to assess how well the model will generalize to an independent dataset. Here's a detailed explanation:

Definition

In K-fold cross-validation, the training dataset is divided into K subsets or "folds." The model is trained on K−1 folds and tested on the remaining fold. This process is repeated K times, with each fold serving as the test set once.

Steps Involved

1. **Data Splitting**: The entire dataset is randomly shuffled and divided into K equally (or nearly equally) sized folds.
2. **Training and Validation**:
    - For each fold iii (where iii ranges from 1 to K):
        - Use folds 1,2,…, K excluding fold i to train the model.
        - Use fold iii to test the model.
3. **Performance Measurement**: After completing the training and testing across all K folds, the performance metrics (like accuracy, precision, recall, etc.) are calculated for each fold.
4. **Averaging Results**: The results from all K iterations are then averaged to produce a single performance metric for the model.

Advantages

- **More Reliable Estimate**: K-fold cross-validation provides a more reliable estimate of model performance compared to a single train-test split, as it takes into account different partitions of the data.
- **Efficient Use of Data**: Since every instance in the dataset is used for both training and testing, it maximizes the use of available data.
- **Reduces Variance**: The averaging of the results helps to mitigate the variance associated with a single random split.

Common Choices for KKK

- **K=5 or K=10**: These values are commonly used in practice, as they provide a good balance between bias and variance.
- **Leave-One-Out Cross-Validation (LOOCV)**: A special case where K is equal to the number of instances in the dataset. Each training set consists of all but one data point.

Disadvantages

- **Computationally Expensive**: Training the model K times can be computationally intensive, especially for large datasets or complex models.
- **Not Always Suitable for Time-Series Data**: K-fold cross-validation assumes that the data is independent and identically distributed, which may not hold for time-series data where the order of observations matters.

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

Ans- Hyperparameter tuning is the process of optimizing the hyperparameters of a machine learning model to improve its performance. Hyperparameters are the configuration settings that are not learned from the data but are set before the training process begins. They can significantly influence the performance of the model.

What are Hyperparameters?

Hyperparameters can include:

- **Model parameters**: Such as the number of trees in a Random Forest or the depth of a decision tree.
- **Learning parameters**: Such as the learning rate in gradient descent or the batch size in neural networks.
- **Regularization parameters**: Such as L1 or L2 regularization strength.

Why Hyperparameter Tuning is Done

1. **Improve Model Performance**: Properly tuned hyperparameters can lead to better model accuracy, robustness, and generalization to unseen data. It helps in achieving the best performance for a given model.
2. **Avoid Overfitting and Underfitting**: Hyperparameter tuning helps to find a balance between model complexity and generalization. For instance, a very complex model might overfit the training data, while a too-simple model might underfit.
3. **Adaptation to Different Datasets**: Different datasets may require different hyperparameter settings. Tuning allows the model to adapt its parameters to the specific characteristics of the dataset.
4. **Optimization of Resources**: Efficient hyperparameter tuning can reduce computational costs and training times by finding the best-performing model settings without exhaustive searches.

Common Methods for Hyperparameter Tuning

1. **Grid Search**: This method involves defining a grid of hyperparameter values and evaluating the model performance for each combination of parameters. It's exhaustive but can be computationally expensive.
2. **Random Search**: Instead of checking all combinations, random search selects a random sample of hyperparameter combinations to evaluate. This approach can be more efficient than grid search.
3. **Bayesian Optimization**: A more advanced method that builds a probabilistic model of the function mapping hyperparameters to the model performance, using this model to select the most promising hyperparameters to evaluate next.
4. **Cross-Validation**: Often used in conjunction with the above methods to ensure that the evaluation of hyperparameters is robust. K-fold cross-validation can provide a better estimate of model performance for different hyperparameter settings.

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

Ans- A large learning rate in gradient descent can lead to several issues that negatively impact the training process and the performance of the model. Here are the main problems that can arise:

1. **Divergence**:
   - With a large learning rate, the updates to the model parameters can become excessively large, causing the loss function to increase instead of decrease. This can lead to divergence, where the model fails to converge to a minimum and may oscillate or move away from the optimal solution.
2. **Oscillation**:
   - A high learning rate can cause the optimization process to oscillate around the minimum rather than settling down. Instead of gradually approaching the minimum, the parameters may overshoot, leading to erratic behavior where they oscillate back and forth without stabilizing.
3. **Overshooting Minima**:
   - If the learning rate is too large, the algorithm may skip over the optimal minima entirely. This is especially problematic in complex landscapes with multiple local minima, where the optimizer could miss the best solution due to large parameter updates.
4. **Instability**:
   - Large learning rates can lead to instability in the optimization process. The model may experience sudden jumps in loss values, making it difficult to assess convergence or the effectiveness of the training process.
5. **Poor Generalization**:

- A model trained with a large learning rate may not generalize well to unseen data. The oscillations and divergence can result in a model that fails to capture the underlying patterns in the data, leading to poor performance on the validation or test set.

Solutions

To mitigate these issues, consider the following strategies:

- **Learning Rate Scheduling**: Gradually reduce the learning rate during training (e.g., using learning rate decay).
- **Adaptive Learning Rates**: Use optimization algorithms like Adam, RMSprop, or AdaGrad, which adjust the learning rate dynamically based on the gradient information.
- **Experimentation**: Perform experiments with different learning rates to find an optimal value that allows for effective convergence without causing instability.

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

Ans- Logistic regression is primarily designed for binary classification tasks and works well when the relationship between the features and the target variable is linear. When it comes to classifying non-linear data, logistic regression has some limitations:

Limitations of Logistic Regression for Non-Linear Data

1. **Linear Decision Boundary**:
   - Logistic regression models the probability of a class label as a linear combination of the input features. This results in a linear decision boundary. If the actual relationship between the features and the target is non-linear, logistic regression may fail to capture this relationship, leading to poor classification performance.
2. **Underfitting**:
   - When applied to non-linear data, logistic regression can lead to underfitting, meaning that it cannot adequately model the underlying structure of the data. The model may be too simplistic to represent complex relationships, resulting in high bias.
3. **Limited Complexity**:
   - Logistic regression lacks the flexibility to learn complex patterns in the data. Unlike more advanced models (e.g., decision trees, support vector machines with non-linear kernels, or neural networks), it cannot adapt to the intricacies of non-linear relationships.

Approaches to Handle Non-Linear Data

If you want to use logistic regression for non-linear data, you can consider the following approaches:

1. **Feature Engineering**:
   - Create new features that capture non-linear relationships. This can include polynomial features (e.g., squared or cubic terms), interaction terms between variables, or other transformations of the original features.
2. **Kernel Trick**:
   - Although this is not directly applicable to standard logistic regression, using a kernelized version of logistic regression (like Kernel Logistic Regression) allows for non-linear decision boundaries by mapping the input space into a higher-dimensional space.
3. **Use of Non-Linear Models**:
   - Consider using more sophisticated models that inherently handle non-linear relationships, such as decision trees, random forests, support vector machines with non-linear kernels, or neural networks.

Conclusion

standard logistic regression is not suitable for classifying non-linear data due to its linear nature and the resulting linear decision boundary. However, with appropriate feature engineering or by choosing more flexible models, it is possible to address non-linear classification problems effectively.

13. Differentiate between Adaboost and Gradient Boosting.

Ans-

Comparison	AdaBoost	Gradient Boosting
Basic Concept	AdaBoost focuses on combining multiple weak learners (usually decision trees with one split, known as stumps) to create a strong learner. It adjusts the weights of misclassified instances, giving more emphasis to those that were previously misclassified in each iteration.	Gradient Boosting builds trees sequentially, where each new tree is trained to predict the residuals (the errors) of the combined previous trees. It optimizes a loss function directly, using gradient descent to minimize the errors.
Weighting Mechanism	Adjusts the weights of instances after each round. Misclassified instances receive higher weights, which means that the subsequent weak learners focus more on correcting those errors.	Instead of modifying instance weights, it adds new trees that directly model the errors (residuals) of the previous models. Each tree is trained on the gradients of the loss function with respect to the predictions.
Type of Weak Learners	Typically uses simple models (stumps or shallow trees) as weak learners. The strength comes from combining many of these weak learners.	Can use more complex models (deeper trees) as weak learners. This allows for greater flexibility in fitting the data but may also increase the risk of overfitting.
Loss Function	Primarily uses exponential loss as its objective. The update rules for weights and the final prediction are derived from this loss function.	Can optimize various loss functions (e.g., mean squared error for regression, log loss for binary classification). This makes it more versatile for different types of problems.
Handling of Overfitting	Can be sensitive to noisy data and outliers since it emphasizes misclassified instances. Regularization techniques can be used to mitigate this.	More robust to overfitting through techniques like learning rate adjustments, tree depth control, and regularization parameters (e.g., L1 or L2 regularization).
Performance	Tends to perform well on datasets with a moderate amount of noise. It can struggle when there are many outliers or when the weak learners are too complex.	Generally achieves state-of-the-art performance on a variety of datasets, especially when hyperparameters are carefully tuned. It can handle complex relationships and is widely used in practical applications.

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

Ans- The bias-variance trade-off is a fundamental concept in machine learning that helps to understand the sources of error in a predictive model. It describes the trade-off between two types of errors that can affect the performance of the model: bias and variance. Here's a detailed explanation:

Bias

- **Definition**: Bias refers to the error due to overly simplistic assumptions in the learning algorithm. It represents how far off the model's predictions are from the actual values on average.
- **Characteristics**:
  - High bias models tend to underfit the training data. They make strong assumptions about the data and may fail to capture important patterns.
  - Examples of models with high bias include linear regression applied to non-linear problems.

Variance

- **Definition**: Variance refers to the error due to the model's sensitivity to fluctuations in the training data. It captures how much the model's predictions change when trained on different subsets of the data.
- **Characteristics**:

- High variance models tend to overfit the training data. They learn noise and fluctuations in the data rather than the underlying patterns, resulting in poor generalization to unseen data.
- Examples of models with high variance include deep decision trees or complex neural networks trained on small datasets.

The Trade-Off

- **Trade-Off Explanation**: The bias-variance trade-off illustrates that as you reduce bias (by using a more complex model), variance tends to increase, and vice versa. Ideally, you want to find a model that strikes a balance between the two.
  - **Underfitting**: High bias and low variance occur when the model is too simple. It cannot capture the underlying patterns, leading to poor performance on both the training and test datasets.
  - **Overfitting**: Low bias and high variance occur when the model is too complex. It fits the training data very well but fails to generalize to new, unseen data.

Visual Representation

A common way to visualize the bias-variance trade-off is through a graph showing model complexity on the x-axis and error on the y-axis. The total error can be broken down into three components:

- **Bias Error**: Increases with model complexity.
- **Variance Error**: Decreases with model complexity.
- **Irreducible Error**: Represents the noise inherent in the data and is constant.

Implications for Model Selection

- To achieve optimal model performance, it's important to find the right level of complexity that minimizes total error. Techniques such as cross-validation, regularization, and model selection can help in managing the bias-variance trade-off effectively.

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

Ans-

Support Vector Machines (SVM) use different types of kernels to transform the input data into a higher-dimensional space where it becomes easier to separate classes with a hyperplane. Here's a brief description of the commonly used kernels:

1. Linear Kernel

- **Description**: The linear kernel is the simplest kernel function. It computes the dot product of two input vectors, effectively allowing the SVM to find a linear decision boundary between classes.
- **Formula**: $K(x_i, x_j) = x_i^T x_j$
- **Use Case**: Best suited for linearly separable data. It's computationally efficient and works well when the features are already in a suitable form.

Polynomial Kernel

- **Description**: The polynomial kernel allows for the creation of non-linear decision boundaries by computing a polynomial combination of the input features. It introduces interaction between features, which helps to model more complex relationships.
- **Formula**: $K(x_i, x_j) = (x_i^T x_j + c)^d$
- where ccc is a constant (often set to 1) and d is the degree of the polynomial.
- **Use Case**: Suitable for data where relationships between features are polynomial in nature. The choice of degree d can significantly impact the model's flexibility.

3. RBF (Radial Basis Function) Kernel

- **Description**: The RBF kernel, also known as the Gaussian kernel, transforms the input space into a higher-dimensional space using a radial basis function. It measures the distance between input vectors, allowing the SVM to create complex decision boundaries.
- **Formula**: $K(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
- where σ is a parameter that controls the width of the Gaussian distribution.
- **Use Case**: Effective for cases where the relationship between features is highly non-linear. It's widely used due to its ability to handle a wide variety of data distributions.