CS60050

Machine Learning Assignment-3

Name-Diganta Mandal
Roll-22CS30062

Part-1

HIGGS UCI Dataset

Dataset Overview

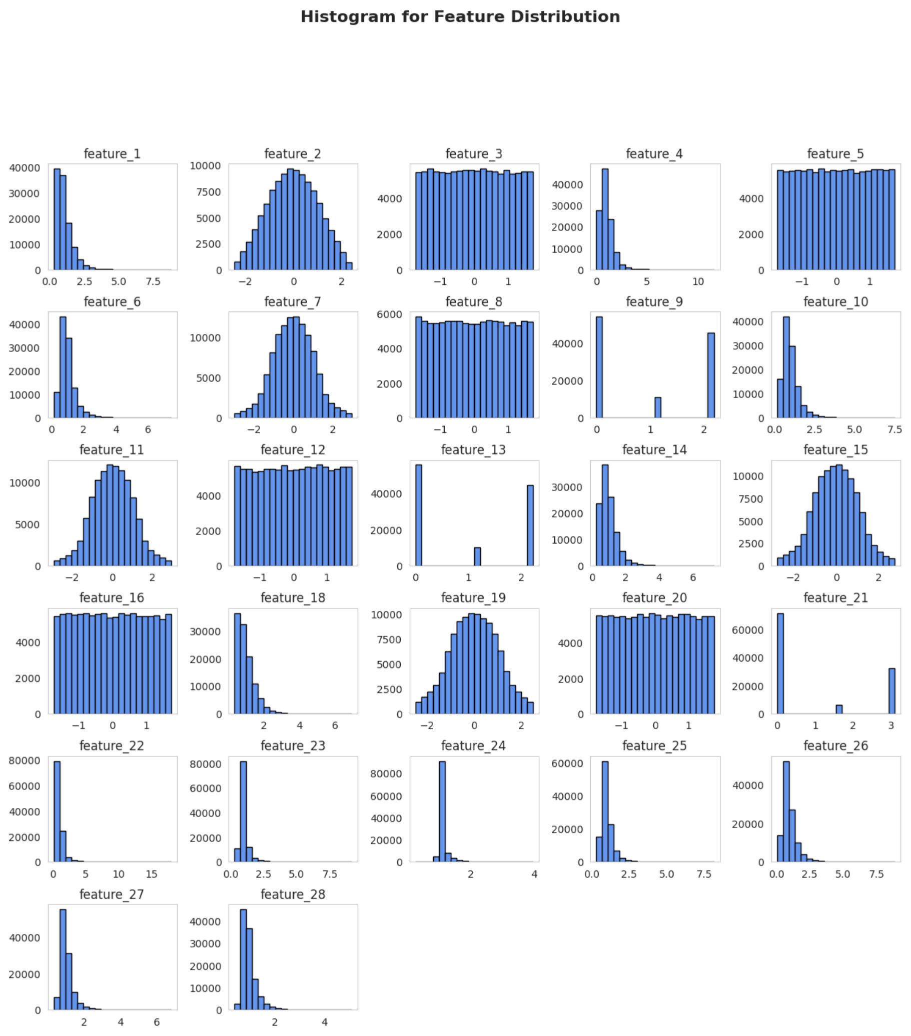
Features Overview

The dataset consists of kinematic properties and derived features related to particle interactions in an accelerator. The first 21 features represent kinematic properties measured by particle detectors, capturing critical attributes of particle movement and interactions essential for understanding particle physics experiments. The following seven features are high-level functions created from the initial 21, designed by physicists to enhance the dataset's ability to distinguish between two different classes of particle events.

Objectives

The primary objective of utilising this dataset is to apply Support Vector Machines (SVM) to classify the data. This approach aims to leverage SVM's capability to handle complex, high-dimensional spaces effectively, making it well-suited for the classification tasks inherent in particle physics. By employing SVM, the goal is to accurately classify the particle events into their respective categories, thereby providing insights into the underlying patterns of the data.

Feature Distribution

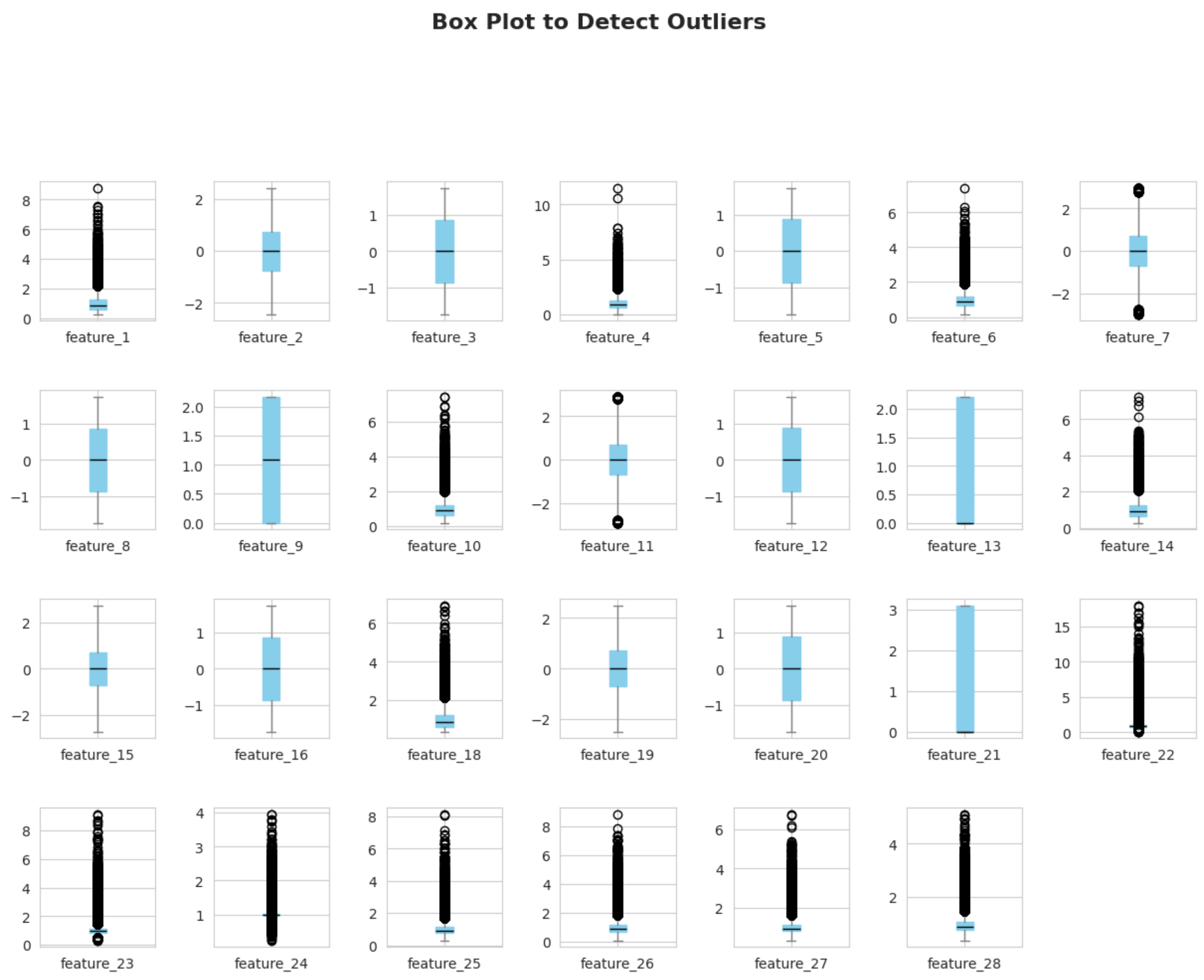


The feature distribution analysis reveals a mix of patterns:

- **Skewed Distributions**: Features like `feature_1`, `feature_4`, and `feature_6` are positively skewed, suggesting most values are concentrated on the left with a long right tail. **Normal Distributions**: Features such as `feature_2`, `feature_7`, and `feature_11` exhibit a Gaussian shape, which may benefit models that assume normality.
- **Uniform Distributions**: Features `feature_3`, `feature_8`, and `feature_13` are uniformly distributed, potentially lacking predictive power in raw form.
- **Discrete and Sparse**: Some, like `feature_13` and `feature_14`, have limited distinct values, suggesting they may be categorical or binary.
- **Outliers**: Features with extreme values, like `feature_22` and `feature_23`, may need outlier handling.

Preprocessing these features through scaling, transformations, and dimensionality reduction helped me enhance model performance.

Outlier Detection



The box plot analysis highlights several features with significant outliers:

- **Numerous Outliers**: Features like `feature_1`, `feature_6`, `feature_10`, `feature_14`, `feature_18`, `feature_22`, and `feature_25` have a large number of outliers, suggesting the presence of extreme values far from the median.
- **Minimal or No Outliers**: Features such as `feature_2`, `feature_3`, and `feature_8` have fewer outliers, indicating more stable distributions around the median.
- **High Variability**: Features like `feature_22` and `feature_24` exhibit wide ranges, which might need scaling or normalization.

These factors are taken care during the process of standardisation,normalization and feature engineering .

Linear SVM

```
Best Model Performance Metrics
C : 0.01
Accuracy : 0.65
Precision : 0.64
Recall : 0.76
F1 Score : 0.70
ROC AUC Score : 0.70
```

First, I implemented SVM with linear kernel and trained it for different values of C to obtain the best model among them.I used the ROC AUC score to evaluate the model's capability to distinguish between classes.These metrics indicate a satisfactory balance between precision and recall, as evidenced by the F1 score of 0.70. The ROC AUC score of 0.70 also highlights a reasonable capacity of the model to separate classes.

Although the Linear SVM provides a good baseline for classification, it is not a practical solution for handling large datasets as in this case.

Scalability and Efficiency

To overcome this, I used **Stochastic Gradient Descent** to approximate the SVM solution. In this method, we update the model parameters incrementally for each individual data point, rather than computing gradients over the entire dataset. This is particularly beneficial for large datasets, as it reduces memory usage and accelerates training.

```
Performance Metrics for SGD-based SVM:
Accuracy      : 0.65
Precision     : 0.63
Recall        : 0.81
F1 Score      : 0.71
ROC AUC Score : 0.70
```

The SGD classifier provides similar result but works much faster compared to vanilla SVM.
We can also use mini batch gradient descent to make the process faster instead of SGD but It requires enough memory to load and process mini-batches, which may not be feasible for very large datasets or limited resources as in our case.

SVM with Kernels

Polynomial Kernel

```
Performance Metrics for Polynomial Kernel based SVM:
Degree: 2, C: 100
Accuracy: 0.6552
Precision: 0.6421
Recall: 0.7845
F1 Score: 0.7062
Area Under the ROC Curve: 0.7136
```

Computational Cost=O(m^2(n+d))

m: No. Of samples in the dataset

n: Cost of computing the dot product

d: Cost of polynomial kernel of degree d

RBF Kernel

```
Performance Metrics for RBF Kernel based SVM:

Gamma 0.01, C: 10
Accuracy: 0.6730
Precision: 0.6731
Recall: 0.7406
F1 Score: 0.7052
Area Under the ROC Curve: 0.7349
```

Computational Cost=O(m^2n)

m: No. Of samples in the dataset

n: Cost of computing the dot product

Custom Kernel

My custom kernel involves a combination of RBF kernel(gamma=0.5) and Linear Kernel.

```
Performance Metrics for RBF Kernel based SVM:
C: 0.1
Accuracy: 0.6450
Precision: 0.6444
Recall: 0.7316
F1 Score: 0.6852
Area Under the ROC Curve: 0.6833

Computational Cost=O(m^2n)
```

m: No. Of samples in the dataset

n: Cost of computing the dot product

***Details regarding time complexity are mentioned in .ipynb file*

Hyperparameter Tuning

We tuned the following hyperparameters:

- **C**: Regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification error.
- d: Degree of the polynomial kernel
- **γ**: Kernel coefficient for the RBF kernel, influencing the shape of the decision boundary.

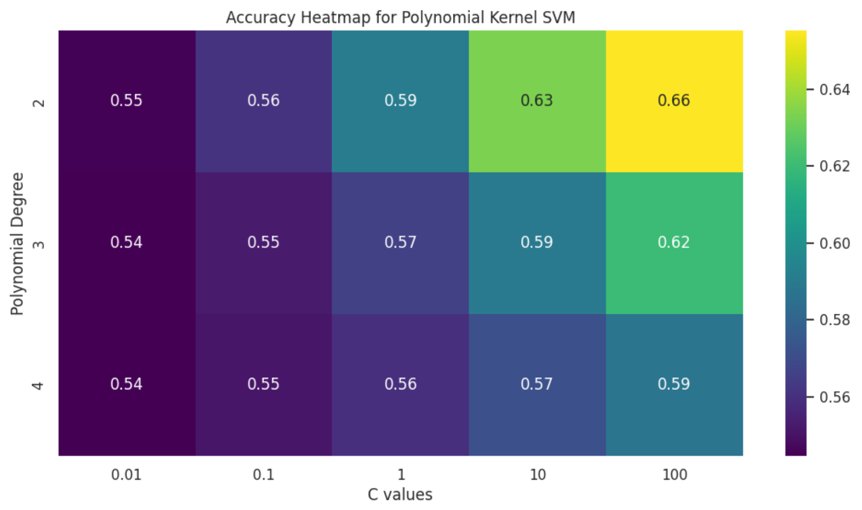
I used **RandomSearch** to tune the hyper-parameters with a 5-fold cross validation.

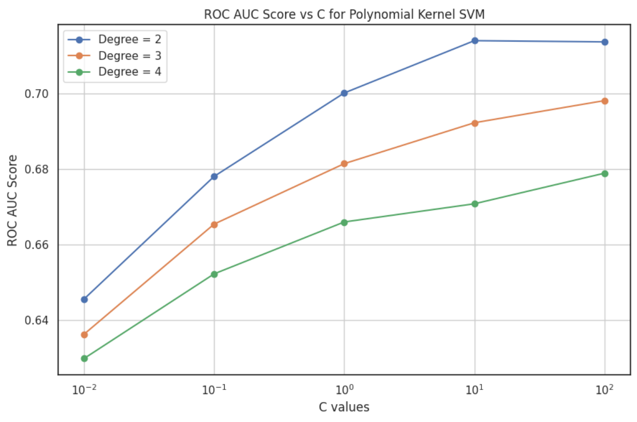
This process I restricted to just 10 iterations due to the high RAM usage of the process which resulted in decrease in accuracy compared to the normal method mentioned above.

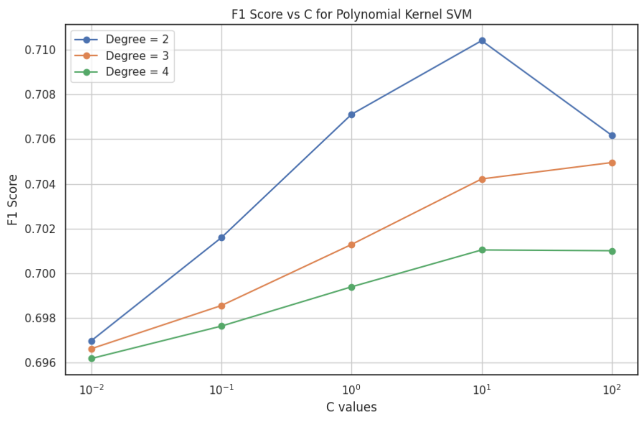
- Polynomial Kernel -> C=2, d=10
- RBF Kernel -> γ=0.001, C=1
- Custom Kernel -> C=0.1

Hyperparameter Sensitivity Analysis

<u>Polynomial Kernel</u>

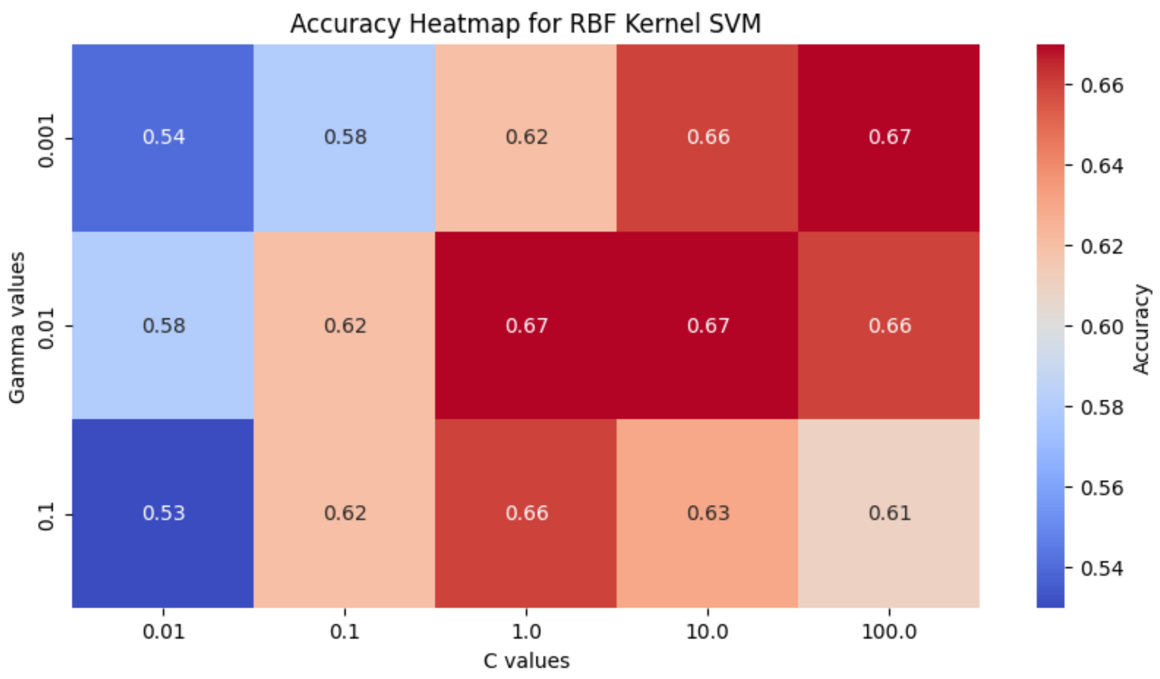


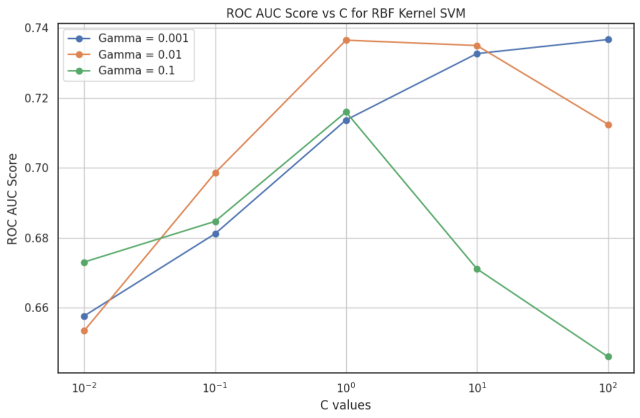


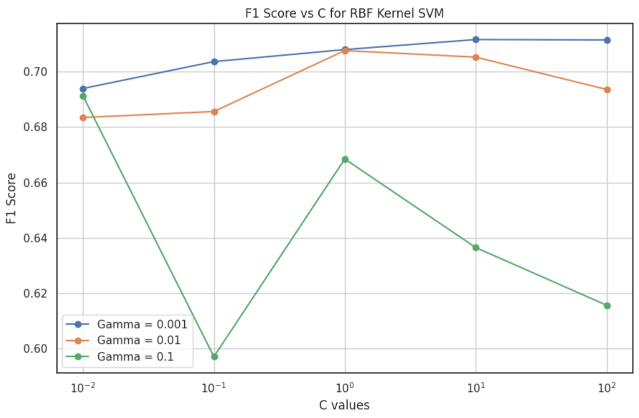


- All the metrics increases with increase in C upto a certain value and then tend to stabilise.
- It is observed that degree 2 polynomial kernel outperforms higher degree. It is likely because of overfitting caused by higher degree kernel.
- Accuracy and ROC AUC Score peaks at degree=2 and C=10 thus making it our to optimal model

RBF Kernel

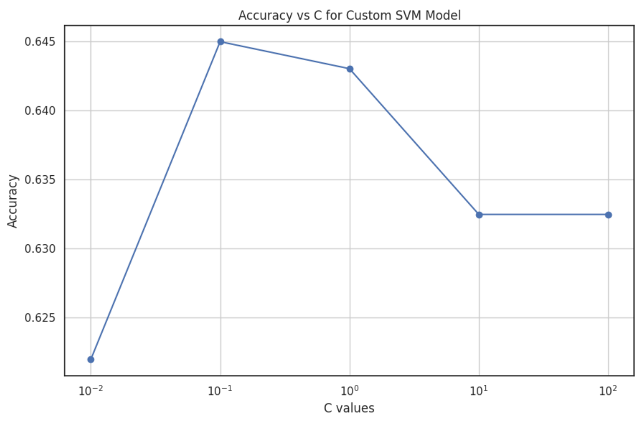


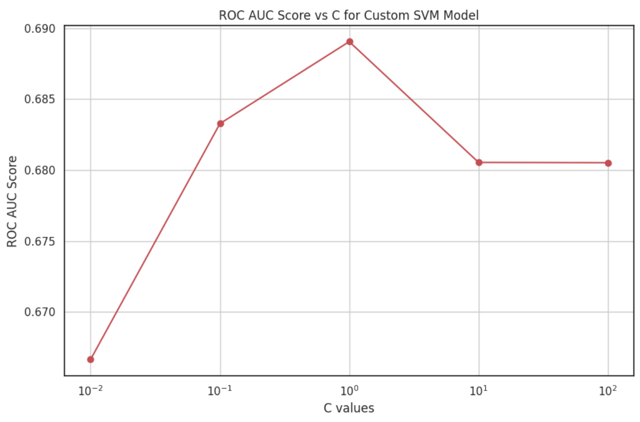




- Here gamma values significantly impact the result obtained. A smaller gamma value tends to give better result since it leads to wider kernel which can capture more global patterns in the data.
- Accuracy tends to increase with increase in value of C but very high value of C may lead to overfitting.Thus, C=1 or 10 is the optimal choice.

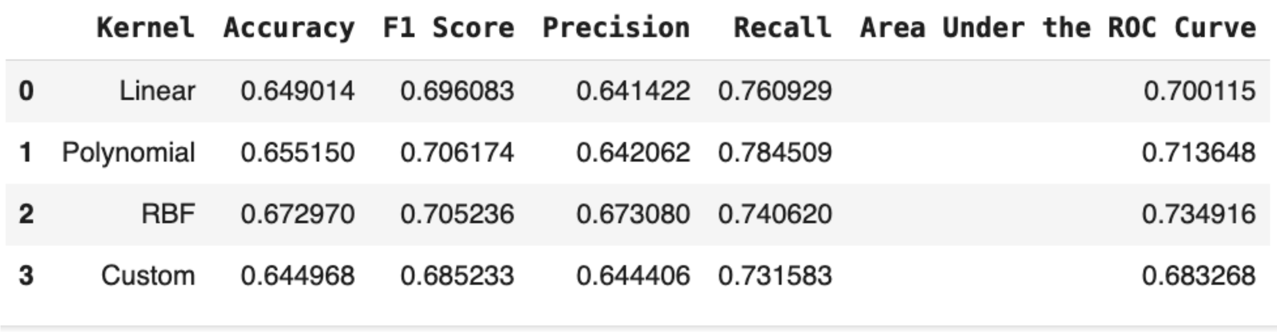
Custom Kernel





- In case of custom kernel, too high or too low value of C leads to bad clustering. Desirable results are produced for C=0.1 or 1
- Both ROC score and accuracy tends to not improve after C=1 and stabilises at a lower value.

Comparison



	Kernel	Accuracy	F1 Score	Precision	Recall	Area Under the ROC Curve
0	Linear	0.649014	0.696083	0.641422	0.760929	0.700115
1	Polynomial	0.655150	0.706174	0.642062	0.784509	0.713648
2	RBF	0.672970	0.705236	0.673080	0.740620	0.734916
3	Custom	0.644968	0.685233	0.644406	0.731583	0.683268

- RBF kernel gives the best accuracy as well as Area Unser ROC Curve.
- But if cost of execution is very crucial then linear kernel is also a sufficiently good option.

Conclusion

Based on the experimental results, the RBF kernel exhibited superior performance compared to other kernel for the given dataset. This can be attributed to the RBF kernel's ability to implicitly map data into a high-dimensional feature space, enabling the separation of non-linearly separable data.
Furthermore, the RBF kernel's sensitivity to the gamma hyperparameter provides granular control over the model's capacity. By carefully tuning the gamma parameter, we can balance the model's ability to fit the training data while avoiding overfitting.

Therefore, the RBF kernel with appropriately tuned hyperparameters was selected as the optimal choice for this specific classification task.