Machine Learning SVM Project

Problem definition and motivation:

In the field of machine learning, classification problems are fundamental to a wide array of applications, ranging from image recognition to medical diagnosis. This project is centered around utilizing and evaluating Support Vector Machines (SVM), Our challenge lies in selecting an optimal kernel function and an appropriate degree of regularization. These choices are crucial in shaping the SVM's ability to make accurate predictions and adapt effectively to various datasets. The kernel function determines the transformation of data into a suitable feature space, thereby influencing the decision boundary, while regularization helps in balancing the model's complexity against its ability to generalize. The project aims to present these crucial parameters, optimizing SVMs to enhance their predictive accuracy across diverse applications.

Dataset:

These datasets, designed for evaluating clustering algorithms, consist of two primary features and a label for each data point. The features typically represent coordinates or measurements in a two-dimensional space, and the label indicates the cluster to which each point belongs. The variety in their structures — from simple, distinct clusters to more complex, overlapping ones — provides a comprehensive platform for testing the effectiveness of various clustering techniques, including SVMs with different kernel functions. These datasets are ideal for exploring how algorithms perform under different clustering scenarios.

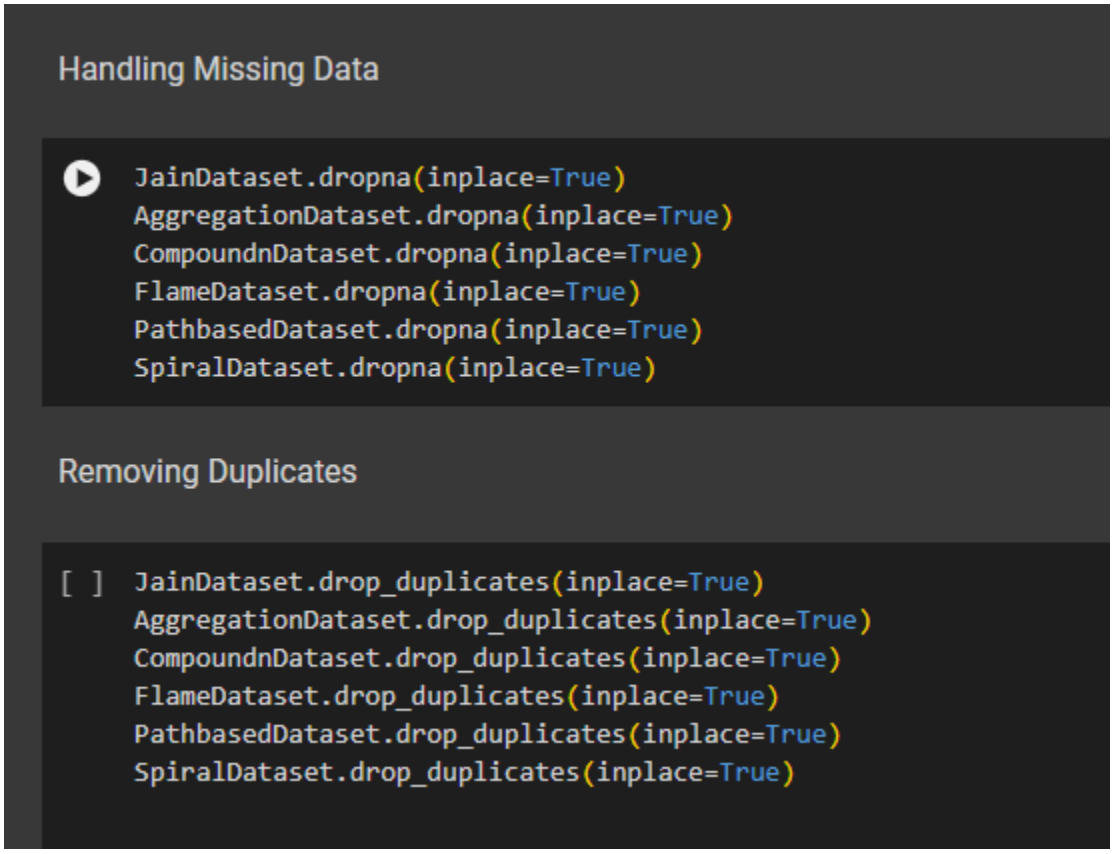
Approach and Methodology:

Data Pre-processing

My approach includes multiple steps in data pre-processing. After importing necessary libraries such as Numpy, Pandas, Scikit-learn, and Matplotlib. My Key pre-processing steps include:

1. Data Cleaning: data cleaning steps, are essential for preparing the dataset for effective model training and evaluation. These steps include Handling Missing Values using `dropna` to remove any rows with missing values from the datasets (`JainDataset`, `AggregationDataset`, `CompoundnDataset`, `FlameDataset`, `PathbasedDataset`, `SpiralDataset`). This ensures that the SVM models do not encounter any null or NaN values that could adversely affect their performance. Then remove duplicates by using `drop_duplicates` to remove any duplicate entries

in the datasets. This step is crucial to prevent the model from being biased towards repeated observations.



2. Data Splitting: done through using `train_test_split` from Scikit-learn to divide the dataset into training and testing sets, ensuring a mixture of data points in both sets for effective training and evaluation.

Spliting The Data

```
[ ]  Jain_data=JainDataset.drop('Label', axis=1)
     Jain_label=JainDataset['Label'].astype(np.int_)


     # For AggregationDataset
     Aggregation_data = AggregationDataset.drop('Label', axis=1)
     Aggregation_label = AggregationDataset['Label'].astype(np.int_)

     # For CompoundnDataset
     Compound_data = CompoundnDataset.drop('Label', axis=1)
     Compound_label = CompoundnDataset['Label'].astype(np.int_)

     # For FlameDataset
     Flame_data = FlameDataset.drop('Label', axis=1)
     Flame_label = FlameDataset['Label'].astype(np.int_)

     # For PathbasedDataset
     Pathbased_data = PathbasedDataset.drop('Label', axis=1)
     Pathbased_label = PathbasedDataset['Label'].astype(np.int_)

     # For SpiralDataset
     Spiral_data = SpiralDataset.drop('Label', axis=1)
     Spiral_label = SpiralDataset['Label'].astype(np.int_)
```

```
Jain_train_data, Jain_test_data, Jain_train_labels, Jain_test_labels = train_test_split(Jain_data, Jain_label,  test_si
                                                    shuffle = True,
                                                    random_state = 0)



#'Aggregation_data'
Aggregation_train_data, Aggregation_test_data, Aggregation_train_labels, Aggregation_test_labels = train_test_split(
    Aggregation_data, Aggregation_label, test_size=0.2, shuffle=True, random_state=0)

# Compound_data'
Compound_train_data, Compound_test_data, Compound_train_labels, Compound_test_labels = train_test_split(
    Compound_data, Compound_label, test_size=0.2, shuffle=True, random_state=0)

# Flame_data
Flame_train_data, Flame_test_data, Flame_train_labels, Flame_test_labels = train_test_split(
    Flame_data, Flame_label, test_size=0.2, shuffle=True, random_state=0)

# Pathbased_data
Pathbased_train_data, Pathbased_test_data, Pathbased_train_labels, Pathbased_test_labels = train_test_split(
    Pathbased_data, Pathbased_label, test_size=0.2, shuffle=True, random_state=0)

# Spiral_data
Spiral_train_data, Spiral_test_data, Spiral_train_labels, Spiral_test_labels = train_test_split(
    Spiral_data, Spiral_label, test_size=0.2, shuffle=True, random_state=0)
```

3. Data Normalization: Using MinMaxScaler from Scikit-learn, the data is normalized to a range between 0 and 1. This is particularly useful for algorithms that are sensitive to the scale of the data, like SVMs. But by observing the varias datasets it was not needed

Data Normalization

```
# Create a MinMaxScaler instance
scaler = MinMaxScaler()

# Scaled training data for Jain dataset
scaled_Jain_train_data = pd.DataFrame(scaler.fit_transform(Jain_train_data))
scaled_Jain_test_data = scaler.transform(Jain_test_data)

# Scaled training data for Aggregation dataset
scaled_Aggregation_train_data = pd.DataFrame(scaler.fit_transform(Aggregation_train_data))
scaled_Aggregation_test_data = scaler.transform(Aggregation_test_data)

# Scaled training data for Compound dataset
scaled_Compound_train_data = pd.DataFrame((scaler.fit_transform(Compound_train_data)))
scaled_Compound_test_data = scaler.transform(Compound_test_data)

# Scaled training data for Flame dataset
scaled_Flame_train_data = pd.DataFrame(scaler.fit_transform(Flame_train_data))
scaled_Flame_test_data = scaler.transform(Flame_test_data)

# Scaled training data for Pathbased dataset
scaled_Pathbased_train_data = pd.DataFrame(scaler.fit_transform(Pathbased_train_data))
scaled_Pathbased_test_data = scaler.transform(Pathbased_test_data)

# Scaled training data for Spiral dataset
scaled_Spiral_train_data = pd.DataFrame(scaler.fit_transform(Spiral_train_data))
scaled_Spiral_test_data = scaler.transform(Spiral_test_data)
```

Model Parameters and Evaluation

1. Model Training: The SVM models are trained using two functions: `Applying_SVM_Without_regularization` and `Applying_SVM_With_regularization`. Both functions start by defining a hyperparameter grid that includes various kernel types (linear, polynomial, radial basis function, and sigmoid) and different levels of the regularization parameter C. An SVM classifier is then initialized, and a grid search is performed using GridSearchCV to identify the best combination of hyperparameters that maximizes accuracy. The models are trained on the provided `train_data` and `train_labels`, and the best SVM model is extracted based on cross-validation performance. Additionally, decision regions of the best models are plotted to visualize their classification behaviorr. These functions serve as universal functions to be used latter with each dataset.

2. Parameter Selection: In the context of training Support Vector Machine (SVM) models, the *hyperparameter C* plays a key role in controlling the trade-off between maximizing the margin between data points and minimizing classification errors; a smaller C emphasizes a larger margin (regularization), while a larger C allows for more data points to be classified correctly (No

regularization). *Kernels*, on the other hand, determine the mathematical transformation applied to input data, with options such as 'linear,' 'poly,' 'rbf' (radial basis function), and 'sigmoid,' influencing how the SVM separates data in complex patterns. *Cross-validation (cv)* is a method to test how well a machine learning model works. It splits the data into parts and tests the model on different parts. For example, in 5-fold cross-validation, the data is divided into five sets. The model is trained and tested five times, each time using a different set for testing and the rest for training. This helps ensure the model can make good predictions on various data, not just what it was trained on. *'Accuracy'* is a measure of how often the model is correct in its predictions and is commonly used to evaluate its performance.

3. **Model Evaluation:** The `Module_Evaluation` function plays a critical role in evaluating the performance of a Support Vector Machine (SVM) model trained on a specific dataset. The key aspects of the function's operation are as following The SVM model, previously identified as the best, is then employed to make predictions on a designated test dataset (`test_data`). Then the function calculates the accuracy of the SVM model's predictions on the test dataset using the `accuracy_score` metric. The result is presented as a percentage, offering a clear indication of the model's correctness in its classification. In addition to accuracy, the function provides a comprehensive error analysis. It calculates both the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE). These metrics quantify the model's predictive accuracy and the magnitude of prediction errors. The MSE indicates the average squared error between predicted and actual values, while RMSE provides a more interpretable measure by taking the square root of the MSE.

Implementation:

The implementation of SVM-based classification involved a systematic approach to training and evaluation, utilizing custom functions tailored for our datasets.

1. Training with Regularization: The `Applying_SVM_With_regularization` function was designed to automate the process of SVM model training with regularization. This function accepts training data and labels as inputs and utilizes a hyperparameter grid to determine the optimal combination of kernel functions and regularization strengths. The grid includes kernel types such as linear, polynomial, radial basis function (RBF), and sigmoid, and varies the regularization parameter C to balance the margin size and classification error. GridSearchCV is employed to find the best parameters based on cross-validation accuracy. This function is applied to datasets including `Pathbased` and `Spiral`.

```python
def Applying_SVM_With_regularization(train_data,train_labels):

    varias_parameters = {
        "kernel":['linear', 'poly', 'rbf',"sigmoid"],
        "C":  np.logspace(-3, 3, 7)

    }
    svm_object= SVC()
    model_svm = GridSearchCV(svm_object, varias_parameters,cv=5,n_jobs=-1,scoring='accuracy')


    model_svm.fit(train_data,train_labels)

    # Get the best model
    best_svm_model = model_svm.best_estimator_

     # Plot decision regions using the best model as GridSearchCV find the best parameters and avoid fitting each model multiples of times
    fig = plot_decision_regions(X=train_data.values, y=train_labels.values, clf=best_svm_model, legend=2, markers='x^sv<>')
    plt.title(f"Best SVM Model with {best_svm_model.kernel.capitalize()} Kernel and C = {best_svm_model.C}")
    plt.show()

    return model_svm
```

2. Training without Regularization: Similarly, the `Applying_SVM_Without_regularization` function focuses on training SVM models without the regularization term. The process is akin to the regularized version but focuses on maximizing the classification accuracy without penalizing for margin size.

```python
def Applying_SVM_Without_regularization(train_data,train_labels):


    varias_parameters = {
        "kernel":['linear', 'poly', 'rbf',"sigmoid"],
        "C": [1e12]  # Extremely high C value to minimize regularization

    }
    svm_object= SVC()
    model_svm = GridSearchCV(svm_object, varias_parameters,cv=5,n_jobs=-1,scoring='accuracy')


    model_svm.fit(train_data,train_labels)

    # Get the best model
    best_svm_model = model_svm.best_estimator_

     # Plot decision regions using the best model as GridSearchCV find the best parameters and avoid fitting each model multiples of times
    fig = plot_decision_regions(X=train_data.values, y=train_labels.values, clf=best_svm_model, legend=2, markers='x^sv<>')
    plt.title(f"Best SVM Model with Kernel: {best_svm_model.kernel} and Extremely High C Value")
    plt.show()

    return model_svm
```

3. Model Evaluation: The evaluation of the models is conducted through the `Module_Evalwation` function. After training, this function takes test data and labels along with the trained model to compute and return performance metrics such as accuracy, mean squared error (MSE), and root mean squared error

(RMSE). These metrics provide a comprehensive view of the model's predictive performance and error magnitude.

```
[ ]  def Module_Evalwation(test_data,test_label,best_svm_model):


       print(f" using this module paramater {best_svm_model.best_params_}")

       # Make predictions on test set
       predictions = best_svm_model.predict(test_data)

       # Calculate accuracy on test set
       test_accuracy = accuracy_score(test_label, predictions)

       # Convert to percentages if desired
       test_accuracy_percentage = test_accuracy * 100

       print(f' We Got the  Accuracy: {test_accuracy_percentage:.2f}%')

       # Calculate Mean Squared Error
       mse = mean_squared_error(test_label, predictions)

       #Root Mean Squared Error (RMSE)
       rmse = mse ** 0.5

       print(f'Mean Squared Error: {mse}')
       print(f'Root Mean Squared Error: {rmse}')
```

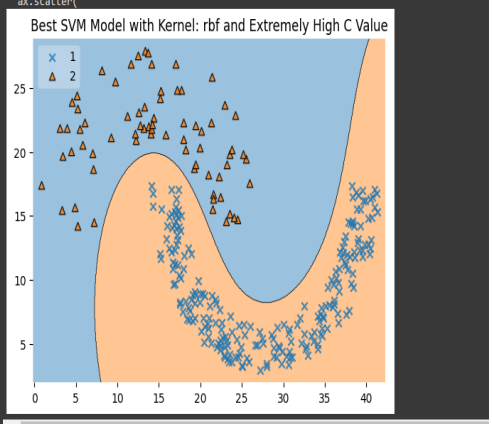
4. Visualization and Analysis: Decision regions for the best-performing models are plotted to assess their classification behavior visually. This step is crucial for understanding the model's decision boundaries and the impact of kernel and regularization choices on the model's ability to separate data points in the feature space.

The Jain dataset:

Please Note That I have changed the Parameters for C = 1e6 in the Without Realization function and the CV to 3 as my device could not handle higher values through the run

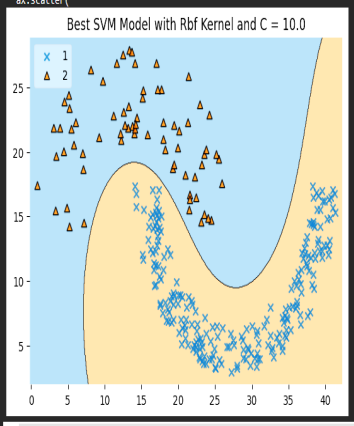
```
best_module=Applying_SVM_Without_regularization(Jain_train_data,Jain_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not hav
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWa
  ax.scatter(
```




```
best_module_with_regularization=Applying_SVM_With_regularization(Jain_train_data,Jain_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid featur
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You pas
  ax.scatter(
```

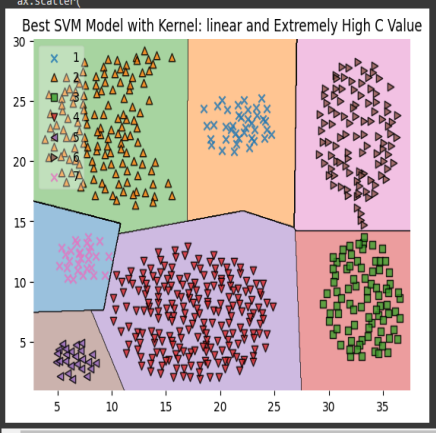



```
 without the requlariza tion
 using this module paramater {'C': 1000000.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
 ============================================================
 with the requlariza tion
 using this module paramater {'C': 10.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
```

The Aggregation dataset:

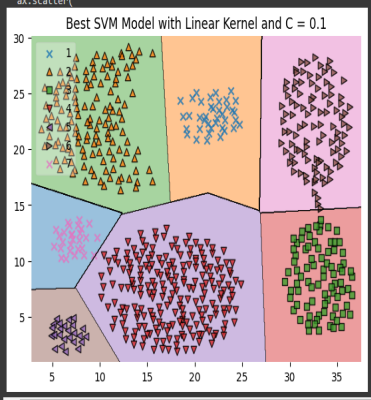
```
best_module=Applying_SVM_Without_regularization(Aggregation_train_data,Aggregation_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid fe
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: Yo
  ax.scatter(
```




```
best_module_with_regularization=Applying_SVM_With_regularization(Aggregation_train_data,Aggregation_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SV
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You passed a edgecolor
  ax.scatter(
```

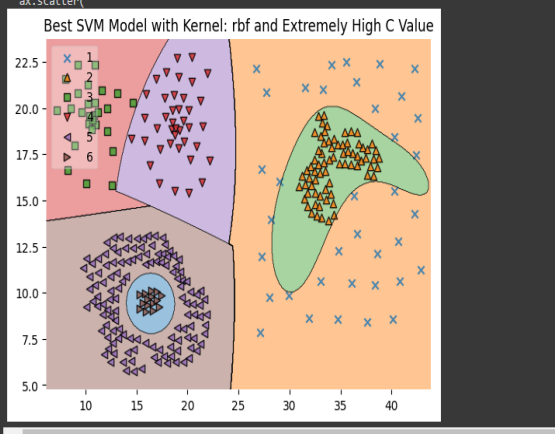



```
 without the requlariztion
 using this module paramater {'C': 1000000.0, 'kernel': 'linear'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
 ============================================================
 with the requlariztion
 using this module paramater {'C': 0.1, 'kernel': 'linear'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
```

The Compound dataset:

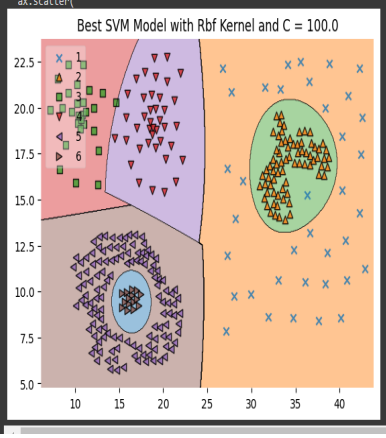
```
best_module=Applying_SVM_Without_regularization(Compound_train_data,Compound_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarnir
  ax.scatter(
```




```
best_module_with_regularization=Applying_SVM_With_regularization(Compound_train_data,Compound_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, b
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You passed a edge
  ax.scatter(
```

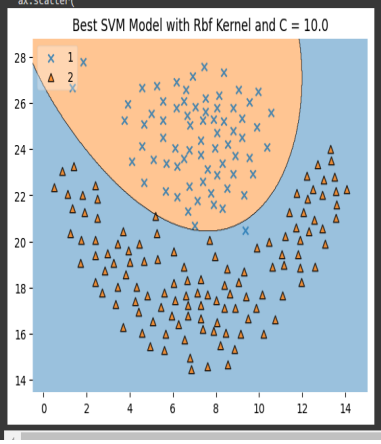



```
model without the requlariztion
 using this module paramater {'C': 1000000.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 95.00%
Mean Squared Error: 0.05
Root Mean Squared Error: 0.22360679774997896
==========================================
model with the requlariztion
 using this module paramater {'C': 100.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 96.25%
Mean Squared Error: 0.0375
Root Mean Squared Error: 0.19364916731037085
```

The Flame dataset:

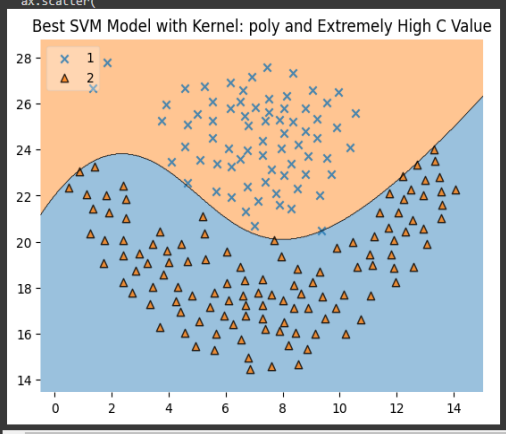
```
best_module_using_regularization=Applying_SVM_With_regularization(Flame_train_data,Flame_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You passe
  ax.scatter(
```




```
best_module=Applying_SVM_Without_regularization(Flame_train_data,Flame_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarn
  ax.scatter(
```



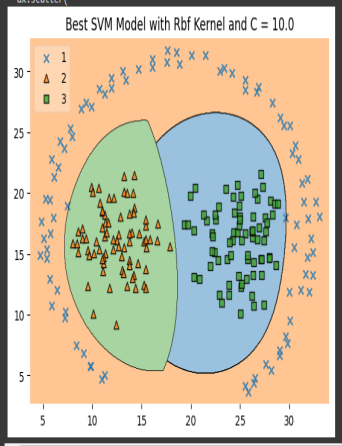

```
print('model without the requlariza tion ')
Module_Evalwation(Flame_test_data,Flame_test_labels,best_module)
print('==============================================')
print('model with the requlariza tion ')

Module_Evalwation(Flame_test_data,Flame_test_labels,best_module_using_requlariza tion
)
```

```
model without the requlariza tion
 using this module paramater {'C': 1000000.0, 'kernel': 'poly'}
 We Got the  Accuracy: 97.92%
Mean Squared Error: 0.020833333333333332
Root Mean Squared Error: 0.14433756729740643
==============================================
model with the requlariza tion
 using this module paramater {'C': 10.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 97.92%
Mean Squared Error: 0.020833333333333332
Root Mean Squared Error: 0.14433756729740643
```

The Pathbased dataset:

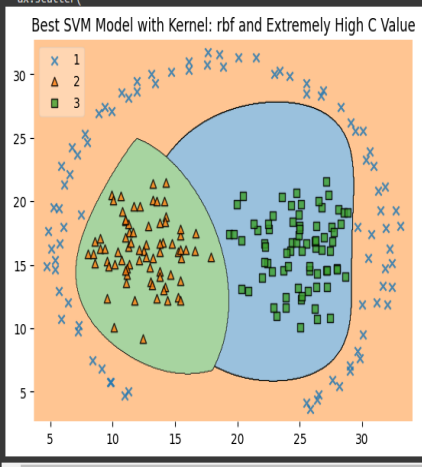
```
best_module_with_regularization=Applying_SVM_With_regularization(Pathbased_train_data,Pathbased_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVC wa
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You passed a edgecolor/edg
  ax.scatter(
```




```
[30] best_module=Applying_SVM_Without_regularization(Pathbased_train_data,Pathbased_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid f
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: Yo
  ax.scatter(
```



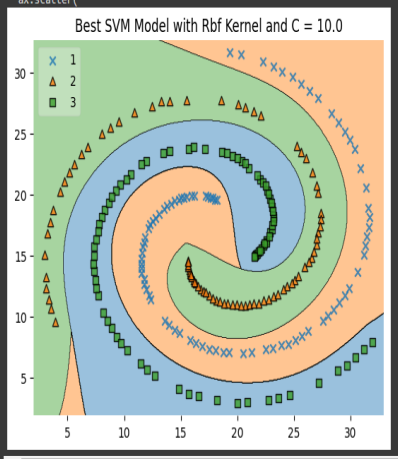

```
[32] print('model without the requlariztion ')
     Module_Evalwation(Pathbased_test_data,Pathbased_test_labels,best_module)
     print('==========================================')
     print('model with the requlariztion ')


     Module_Evalwation(Pathbased_test_data,Pathbased_test_labels,best_module_with_regularization
     )
```

```
model without the requlariztion
 using this module paramater {'C': 1000000.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 95.00%
Mean Squared Error: 0.15
Root Mean Squared Error: 0.3872983346207417
==========================================
model with the requlariztion
 using this module paramater {'C': 10.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 96.67%
Mean Squared Error: 0.13333333333333333
Root Mean Squared Error: 0.3651483716701107
```

And the Spiral dataset :

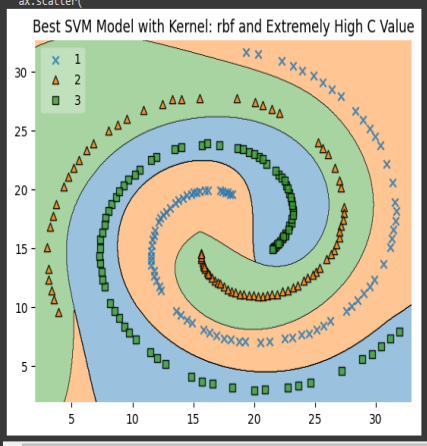
```
best_module_with_regularization=Applying_SVM_With_regularization(Spiral_train_data,Spiral_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature nam
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: You passed a
  ax.scatter(
```




```
best_module=Applying_SVM_Without_regularization(Spiral_train_data,Spiral_train_labels)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/mlxtend/plotting/decision_regions.py:346: UserWarning: Yo
  ax.scatter(
```




```
print('model without the requlariza tion ')
Module_Evalwation(Spiral_test_data,Spiral_test_labels,best_module)
print('==============================================')
print('model with the requlariza tion ')

Module_Evalwation(Spiral_test_data,Spiral_test_labels,best_module_with_regularization
)
```

```
model without the requlariza tion
 using this module paramater {'C': 1000000.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
==============================================
model with the requlariza tion
 using this module paramater {'C': 10.0, 'kernel': 'rbf'}
 We Got the  Accuracy: 100.00%
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0
```

Conclusion:

In conclusion, the evaluation of SVM classifiers with and without regularization provided valuable insights into their performance across different datasets.

Accuracy Analysis: The models' accuracies were carefully calculated to assess their prediction correctness.

Error Metric: The MSE and RMSE offered a quantitative measure of the prediction errors, providing a detailed error analysis.

Performance Insights:
Through using the module on variate datasets I can see that most of the time RBF kernel gives the best result

Learnings:

- I learned the significant role of GridSearchCV in optimizing the model by comparing different parameters simultaneously and selecting the best model.

- I learned how to visualize the results, which provided a clear and intuitive representation of the models' decision boundaries and their performance implications.

- Through the implementation of cross-validation, I gained a practical understanding of its operation. Cross-validation works by splitting the dataset into 'k' parts (or 'folds'), then running 'k' separate learning experiments. In each experiment, the model is trained on 'k-1' folds and validated on the remaining fold. This process is repeated 'k' times, with each fold serving as the validation set once. The results are then averaged out to produce a single estimation. This technique is instrumental in safeguarding against overfitting and ensuring that the model generalizes well to new data.

- I explored the intricacies of the hyperparameter C, which inversely regulates the strength of regularization in SVM models. Unlike the standard equation where higher C values indicate stronger regularization, in SVM, a higher C value signifies less regularization, allowing the model to fit closer to the training data by penalizing the cost of misclassification more severely.

Improvement Tips:

- Explore Kernels: There are many options for the SVM kernel. Trying different ones might help the model work better for your problem.