Exploring SVM Classification: Impact of Different Kernels and Regularization on Diverse Datasets

Course Instructor : Khaled Mostafa El Sayed
TAs : Eng. Gamal Zayed & Eng. Marwa Monier

Sama Youssef 202-000-819

December 15th, 2023

1 Abstract

The task involves utilizing Support Vector Machine (SVM) classification models with different kernels and regularization to classify datasets. SVM is a powerful algorithm for classification tasks, and understanding the impact of different kernels and regularization on model performance is crucial for effective model selection and deployment. The motivation lies in the need to comprehend how different parameters affect SVM classification performance and to make informed decisions in real-world applications.

2 Dataset Description

2.1 Aggregation

Number of Features: 2
Number of Samples: 788
Target Column: the 1st column
Number of Classes: 7

First Feature Statistics:

- Mean: 19.566815
- Standard Deviation (std):9.922042
- Minimum (min): 3.350000
- Maximum (max): 36.550000

Second Feature Statistics:

- Mean: 14.171764
- Standard Deviation (std): 8.089683
- Minimum (min): 1.950000
- Maximum (max): 29.150000

2.2 Compound

Number of Features: 2
Number of Samples: 399
Target Column: the 1st column
Number of Classes: 6

First Feature Statistics:

- Mean: 22.22
- Standard Deviation (std): 9.7
- Minimum (min): 7.150000
- Maximum (max): 42.900000

Second Feature Statistics:

- Mean: 13.970677
- Standard Deviation (std): 4.743516
- Minimum (min): 5.750000
- Maximum (max): 22.750000

2.3 Flame

Number of Features: 2
Number of Samples: 240
Target Column: the 1st column
Number of Classes: 2

First Feature Statistics:

- Mean: 7.323750
- Standard Deviation (std): 3.202509
- Minimum (min): 0.500000
- Maximum (max): 14.200000

Second Feature Statistics:

- Mean: 20.928542
- Standard Deviation (std):3.383390
- Minimum (min): 14.450000
- Maximum (max): 27.800000

2.4 Jain

Number of Features: 2
Number of Samples: 373
Target Column: the 1st column
Number of Classes: 2;

First Feature Statistics:

- Mean: 24.330697
- Standard Deviation (std): 9.853372
- Minimum (min): 0.750000
- Maximum (max): 41.300000

Second Feature Statistics:

- Mean: 12.145979
- Standard Deviation (std): 6.605375
- Minimum (min): 2.950000
- Maximum (max): 27.850000

2.5 Spiral

Number of Features: 2
Number of Samples: 312
Target Column: the 1st column
Number of Classes: 3

First Feature Statistics:

- Mean: 18.408173
- Standard Deviation (std): 7.299923
- Minimum (min): 3.000000
- Maximum (max): 31.950000

Second Feature Statistics:

- Mean: 16.344712
- Standard Deviation (std): 6.867232
- Minimum (min): 2.900000
- Maximum (max): 31.650000

2.6 Pathbased

Number of Features: 2
Number of Samples: 300
Target Column: the 1st column
Number of Classes:3

First Feature Statistics:

- Mean: 18.846500
- Standard Deviation (std): 8.253238
- Minimum (min): 4.700000
- Maximum (max): 33.050000

Second Feature Statistics:

- Mean:17.188500
- Standard Deviation (std): 5.855957
- Minimum (min): 3.650000
- Maximum (max): 31.750000

3 Approach and Methodology

3.1 Data Pre-processing

3.1.1 Loading the datasets

This is function to selet the neede file and performing any necessary data cleaning.

```
def xy_of_dataset(set_name):
    if set_name == 'Aggregation':
        data = "Aggregation.csv"
    elif set_name == 'Compound':
        data = "Compound.csv"
    elif set_name == 'Flame':
        data = "Flame.csv"
    elif set_name == 'Jain':
        data = "Jain.csv"
    elif set_name == 'Spiral':
        data = "Spiral.csv"
    elif set_name == 'Pathbased':
        data = "Pathbased.csv"
    df = pd.read_csv(data, header=None)
    print( df.describe())
    df.dropna()
    df.columns = ['target', 'x1', 'x2']
    x = df[['x1', 'x2']]
    y = df['target']
    num_classes = y.nunique()

    print("Number of classes:", num_classes)
    x = x.values
    y = y.values
    return x, y
```

3.1.2 Splitting the data into training and testing sets

```
X_train, X_temp, y_train, y_temp = train_test_split(x, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
    random_state=42)
```

3.1.3 Scaling the features if required

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

3.2 Model Parameters and Evaluation

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

3.2.1 Utilizing different SVM kernels without and without regularization

```

scaler = StandardScaler()
X_train2 = scaler.fit_transform(X_train)
X_test2 = scaler.transform(X_test)

svm_linear_no_reg = SVC(kernel='linear', C=1.0)
svm_linear_with_reg = SVC(kernel='linear', C=0.1)
svm_poly_no_reg = SVC(kernel='poly', degree=3, C=1.0)
svm_poly_with_reg = SVC(kernel='poly', degree=3, C=0.1)
svm_rbf_no_reg = SVC(kernel='rbf', gamma=0.7, C=1.0)
```

```
svm_rbf_with_reg = SVC(kernel='rbf', gamma=0.7, C=0.1)


svm_linear_no_reg.fit(X_train2, y_train)


svm_linear_with_reg.fit(X_train2, y_train)
svm_poly_no_reg.fit(X_train2, y_train)
svm_poly_with_reg.fit(X_train2, y_train)
svm_rbf_no_reg.fit(X_train2, y_train)
svm_rbf_with_reg.fit(X_train2, y_train)
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

classifiers = [svm_linear_no_reg, svm_linear_with_reg, svm_poly_no_reg, svm_poly_with_reg, svm_rbf_no_reg, svm_rbf_with_reg]
titles = ['Linear (No Regularization)', 'Linear (With Regularization)', 'Polynomial (No Regularization)', 'Polynomial (With Regularization)', 'RBF (No Regularization)', 'RBF (With Regularization)']

for clf, ax, title in zip(classifiers, axes.flatten(), titles):
    plot_decision_regions(X=X_train2, y=y_train, clf=clf, legend=2, markers='x^sv<>', ax=ax)
    ax.set_title(title)

plt.tight_layout()
plt.show()

linear_no_reg_accuracy = accuracy_score(y_test, svm_linear_no_reg.predict(X_test2))
linear_with_reg_accuracy = accuracy_score(y_test, svm_linear_with_reg.predict(X_test2))
poly_no_reg_accuracy = accuracy_score(y_test, svm_poly_no_reg.predict(X_test2))
poly_with_reg_accuracy = accuracy_score(y_test, svm_poly_with_reg.predict(X_test2))
rbf_no_reg_accuracy = accuracy_score(y_test, svm_rbf_no_reg.predict(X_test2))
rbf_with_reg_accuracy = accuracy_score(y_test, svm_rbf_with_reg.predict(X_test2))

print("Linear Kernel without regularization accuracy:", linear_no_reg_accuracy)
print("Linear Kernel with regularization accuracy:", linear_with_reg_accuracy)
print("Polynomial Kernel without regularization accuracy:", poly_no_reg_accuracy)
print("Polynomial Kernel with regularization accuracy:", poly_with_reg_accuracy)
print("RBF Kernel without regularization accuracy:", rbf_no_reg_accuracy)
print("RBF Kernel with regularization accuracy:", rbf_with_reg_accuracy)
```

3.2.2 Plotting different SVM kernels without regularization

such as linear, polynomial, and Gaussian (RBF).

```
gs = gridspec.GridSpec(2, 2)

fig = plt.figure(figsize=(16,16))

clf1 = SVC( kernel='linear',C=1)
clf3 = SVC( kernel='poly',C=1)
clf4 = SVC( kernel='rbf',C=1)
scatter_highlight_kwargs = {'s': 50, 'label': 'Test data', 'alpha': 0.7, 'linestyle':'-'}

labels = [' Linear Kernel' , ' Polynomial Kernel'  ,' RBF Kernel']
for clf, lab, grd in zip([clf1, clf3, clf4],
                         labels,
                         itertools.product([0, 1], repeat=2)):

    clf.fit(X_train, y_train)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X_train, y=y_train, clf=clf, legend=2, markers='x^sv<>')
    plt.title(lab)

plt.show()
```

3.3 Evaluating model performance

using appropriate metrics such as accuracy, precision, recall, and F1-score we can Evaluate model performance.

```
X_val = scaler.transform(X_val)
y_pred = best_svm_model.predict(X_val)
test_accuracy = accuracy_score(y_val, y_pred)*100
print(f"The accuracy of the best SVM model on the validation set is: {test_accuracy:.4f}%
    ")
best_svm_model.get_params()
X_test = scaler.transform(X_test)
y_pred = best_svm_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)*100
print(f"The accuracy of the best SVM model on the test set is: {test_accuracy:.4f}%")
```

3.4 Visualizations

Visualizing decision boundaries and support vectors to gain insights into model behavior.

```
# List of dataset names
dataset_names = ['Aggregation', 'Compound', 'Flame', 'Jain', 'Spiral', 'Pathbased']

# Initialize an empty list to store the X_train and y_train datasets
datasets = []

for name in dataset_names:
    # Get X and y for the current dataset
    X, y = xy_of_dataset(name)

    # Write dataset description
    print(f"Dataset Description for {name}:")
    print(f"Number of samples: {len(X)}")
    print(f"Number of features: {X.shape[1]}")
    print(f"Number of classes: {len(np.unique(y))}")
    print()  # Add a newline for separation

    # Split the dataset and add X_train, y_train to the list
    X_train, _, y_train, _ = train_test_split(X, y, test_size=0.4, random_state=42)
    datasets.append((X_train, y_train))

# Output the dataset descriptions and store X_train, y_train sets in the 'datasets' list
```

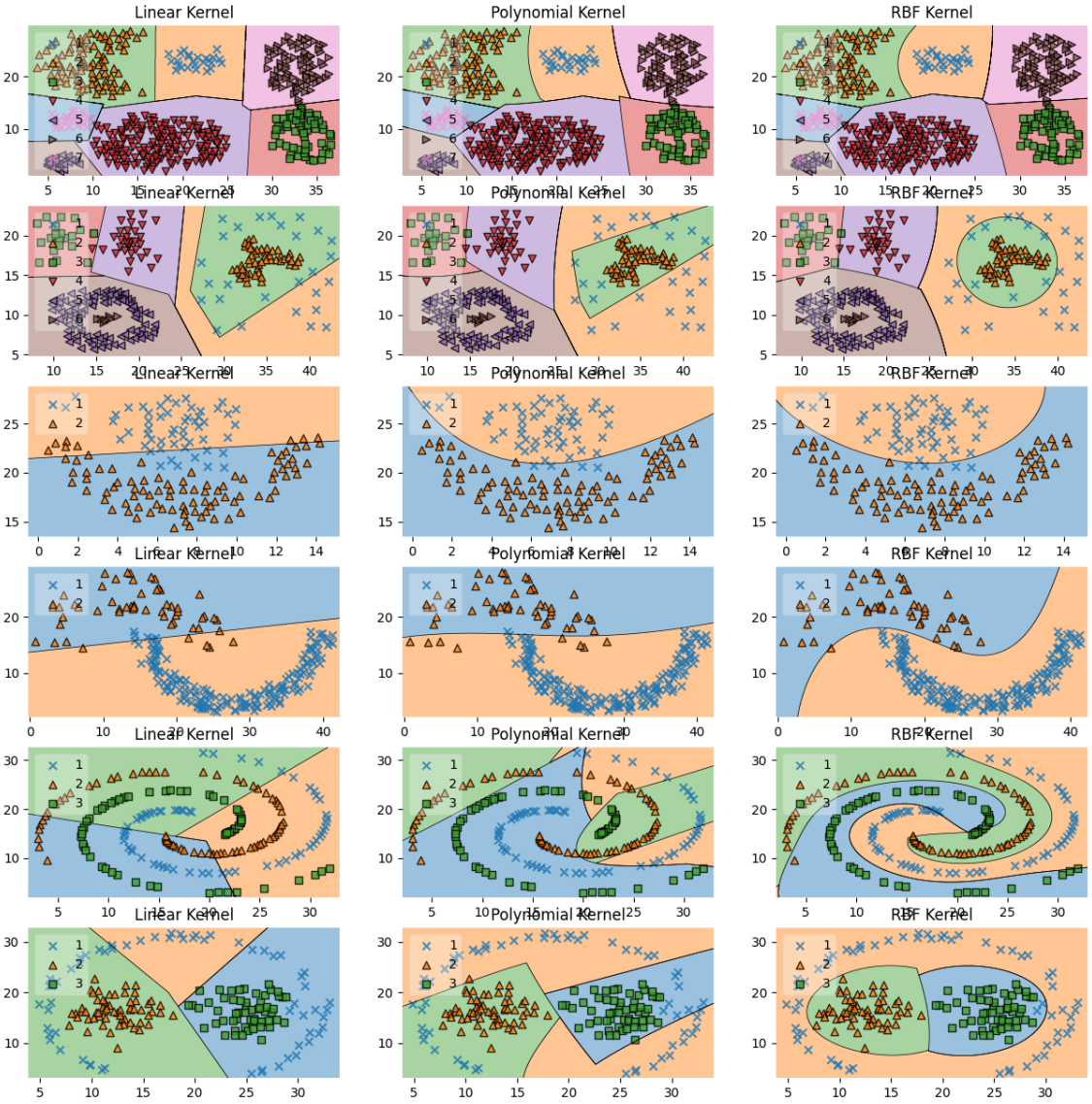
Linear Kernel
Polynomial Kernel
RBF Kernel
Linear Kernel
Polynomial Kernel
RBF Kernel
Linear Kernel
Polynomial Kernel
RBF Kernel
Linear Kernel
Polynomial Kernel
RBF Kernel
Linear Kernel
Polynomial Kernel
RBF Kernel
Linear Kernel
Polynomial Kernel
RBF Kernel

Figure 1:

4 Implementation

4.1 Tools and Libraries

Utilization of Python programming language.Implementation with popular libraries such as numpy, pandas,sklearn,matplotlib,mlxtend, and itertools for data manipulation, model building, and visualization.

```
import numpy as np
import pandas as pd
import itertools
from sklearn import datasets
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.gridspec as gridspec
from mlxtend.preprocessing import shuffle_arrays_unison
import seaborn as sns
from tqdm.notebook import tqdm_notebook as tqdm
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVR
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
from matplotlib.colors import ListedColormap
```

4.2 Hyperparameter tuning

Hyperparameter tuning using cross-validation and grid search techniques.

4.2.1 Selecting Best kernel

```
param_grid_kernel = {'kernel': ['linear', 'poly', 'rbf']}
svm_model = SVC()
grid_search = GridSearchCV(svm_model, param_grid_kernel, cv=5)
grid_search.fit(X_train, y_train)
best_kernel = grid_search.best_params_['kernel']
```

4.2.2 Selecting Best C

```
param_grid_C = {'C': list(np.arange(0, 10, 0.1))}

grid_search_C = GridSearchCV(svm_model, param_grid_C,cv=5)
grid_search_C.fit(X_train, y_train)

best_C = grid_search_C.best_params_['C']
```

4.2.3 Selecting Best Gamma

If the best kernel was RBF then we need to get the best gamma for this kernel

```
if best_kernel=='rbf':
  param_grid_g = {'gamma':  np.append(np.arange(0.001, 1, 0.0001), 'scale')}
  #param_grid_g = {'gamma': list(np.arange(0, 10, 0.01))}


  grid_search_g = GridSearchCV(svm_model, param_grid_g,cv=5)
  grid_search_g.fit(X_train, y_train)

  best_gamma = grid_search_g.best_params_['gamma']
```

4.2.4 Selecting best degree

If the best kernel was poly then we need to get the best degree for this kernel

```
if best_kernel=='poly':
  grid_search_degree = {'degree': np.arange(1, 10)}
  svm_model = SVC(kernel='poly')
  grid_search_degree = GridSearchCV(svm_model, grid_search_degree, cv=5)
  grid_search_degree.fit(X_train, y_train)
  best_degree = grid_search_degree.best_params_['degree']
```

5 Results

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

if best_kernel=='linear':
  best_svm_model = SVC(C=best_C,  kernel=best_kernel)

elif best_kernel=='rbf':
  best_svm_model = SVC(C=best_C, gamma=best_gamma, kernel=best_kernel)

elif best_kernel=='poly':
  best_svm_model = SVC(C=best_C, degree=best_degree, kernel=best_kernel)
best_svm_model.fit(X_train, y_train)

fig = plot_decision_regions(X=X_train, y=y_train, clf=best_svm_model, legend=2, markers='
    x^sv<>')
plt.title("best model")

plt.show()
X_val = scaler.transform(X_val)
y_pred = best_svm_model.predict(X_val)
test_accuracy = accuracy_score(y_val, y_pred)*100
print(f"The accuracy of the best SVM model on the validation set is: {test_accuracy:.4f}%
    ")
best_svm_model.get_params()
X_test = scaler.transform(X_test)
y_pred = best_svm_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)*100
print(f"The accuracy of the best SVM model on the test set is: {test_accuracy:.4f}%")
```

5.1 Aggregation

- C: 0.30
- kernel : rbf
- Gamma: scale
- validation accuracy: 100.0000%
- testing accuracy:98.7342%

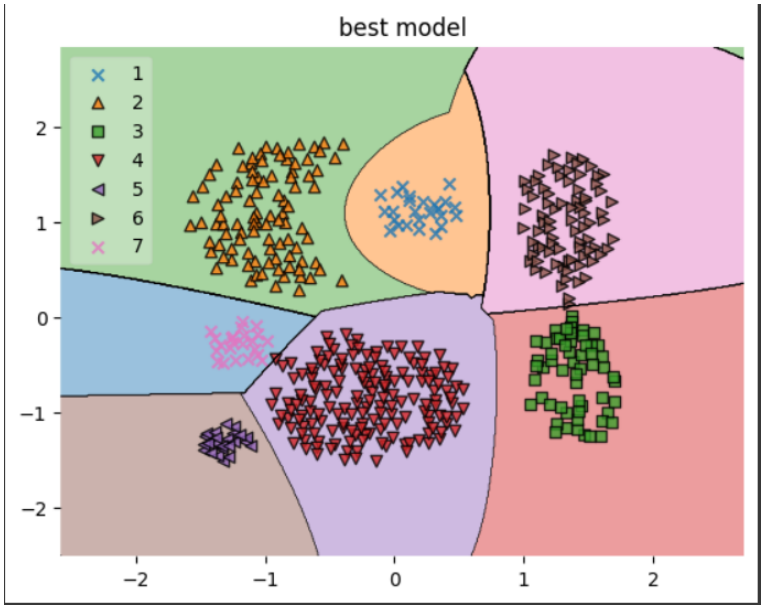

Figure 2:

5.2 Compound

- C: 100
- kernel : RBF
- Gamma: scale
- validation accuracy: 98.7500%
- testing accuracy: 97.5000%

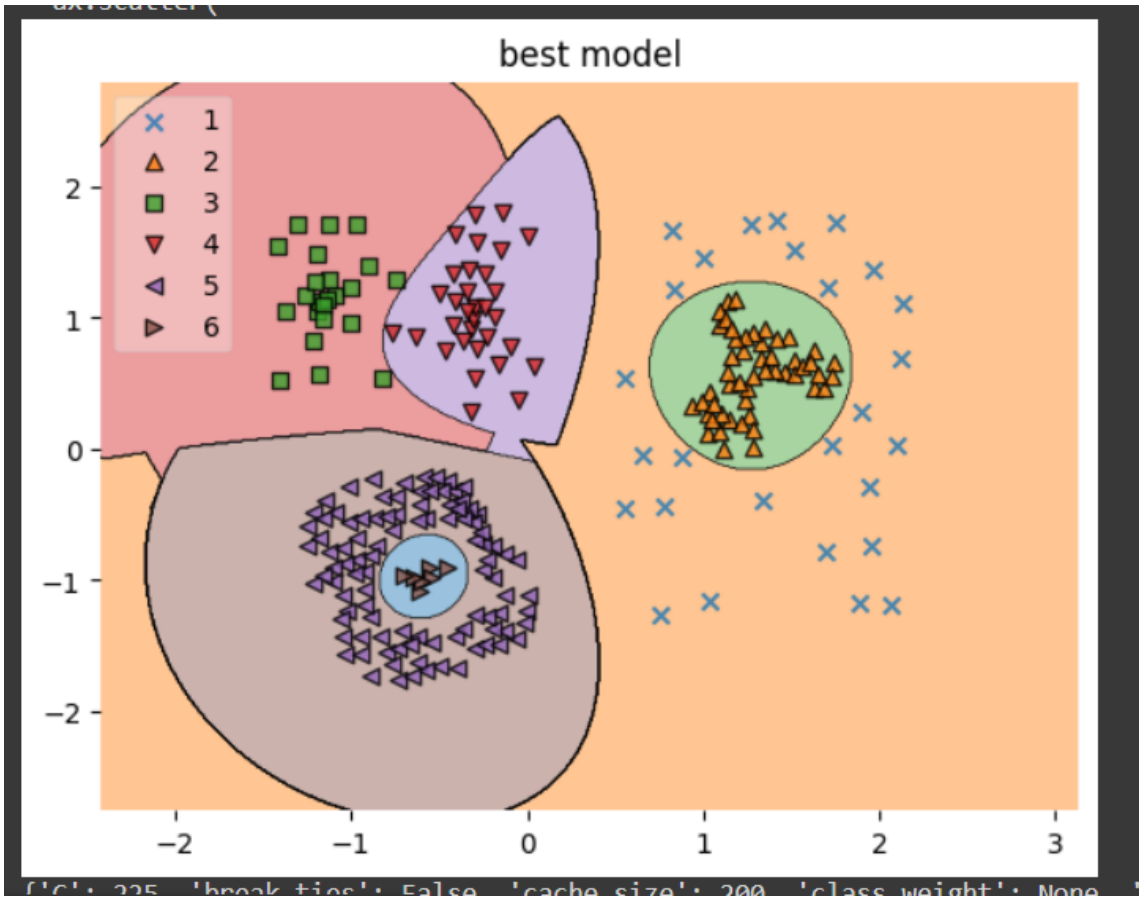

Figure 3:

5.3 Flame

- C: 4.800000000000001
- kernel :rbf
- Gamma: scale
- validation accuracy: 100.0000%
- testing accuracy: 100.0000%

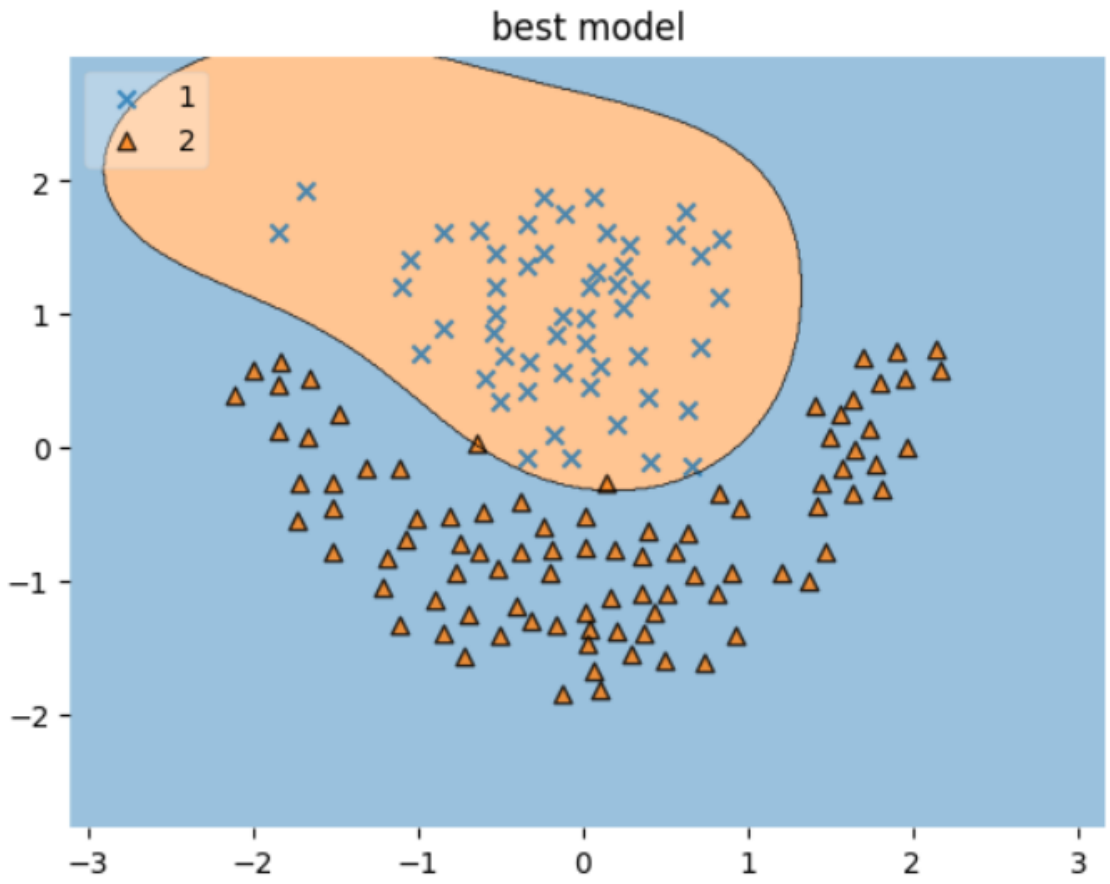

Figure 4:

5.4 Jain

- C: 1.2000000000000002
- kernel :rbf
- Gamma: scale
- validation accuracy: 100.0000%
- testing accuracy: 100.0000%

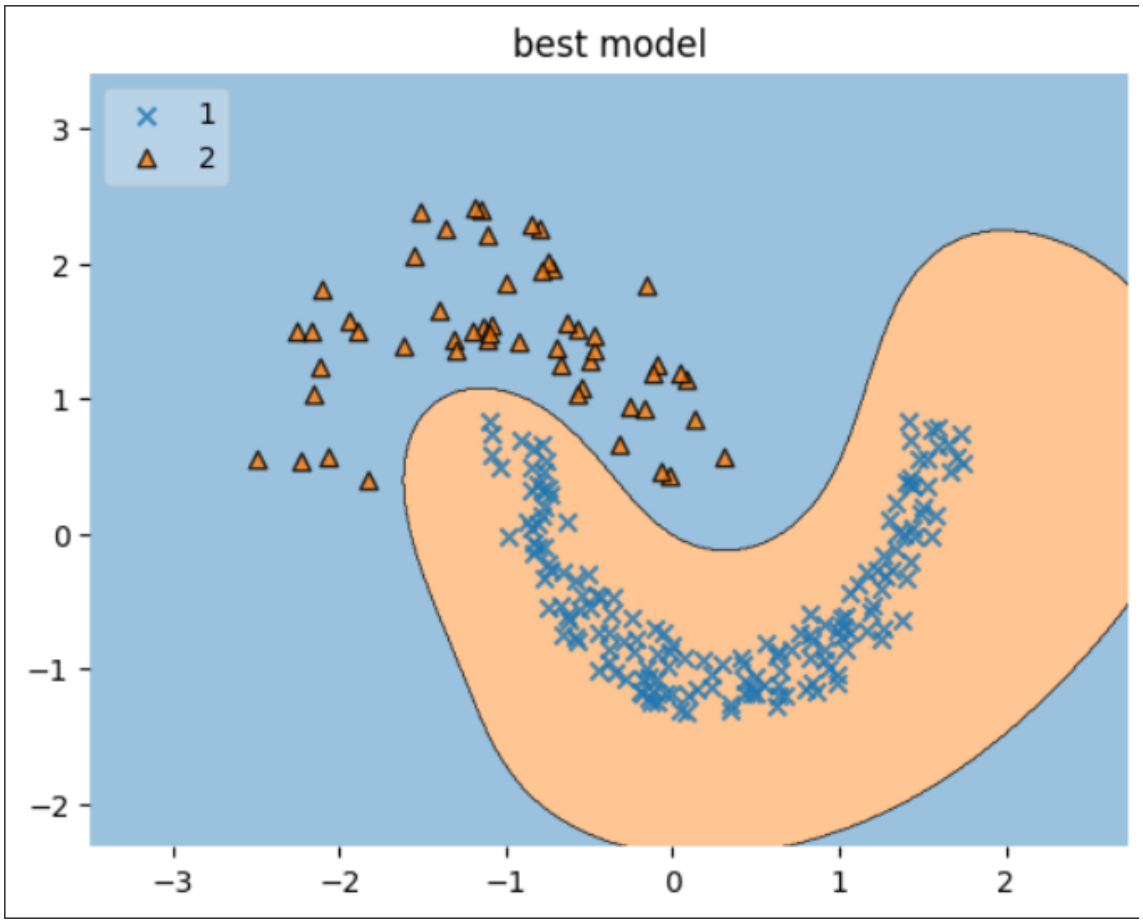

Figure 5:

5.5 Spiral

- C: 9.5
- kernel :rbf
- Gamma:'scale'
- validation accuracy: 100.0000%
- testing accuracy: 100.0000%

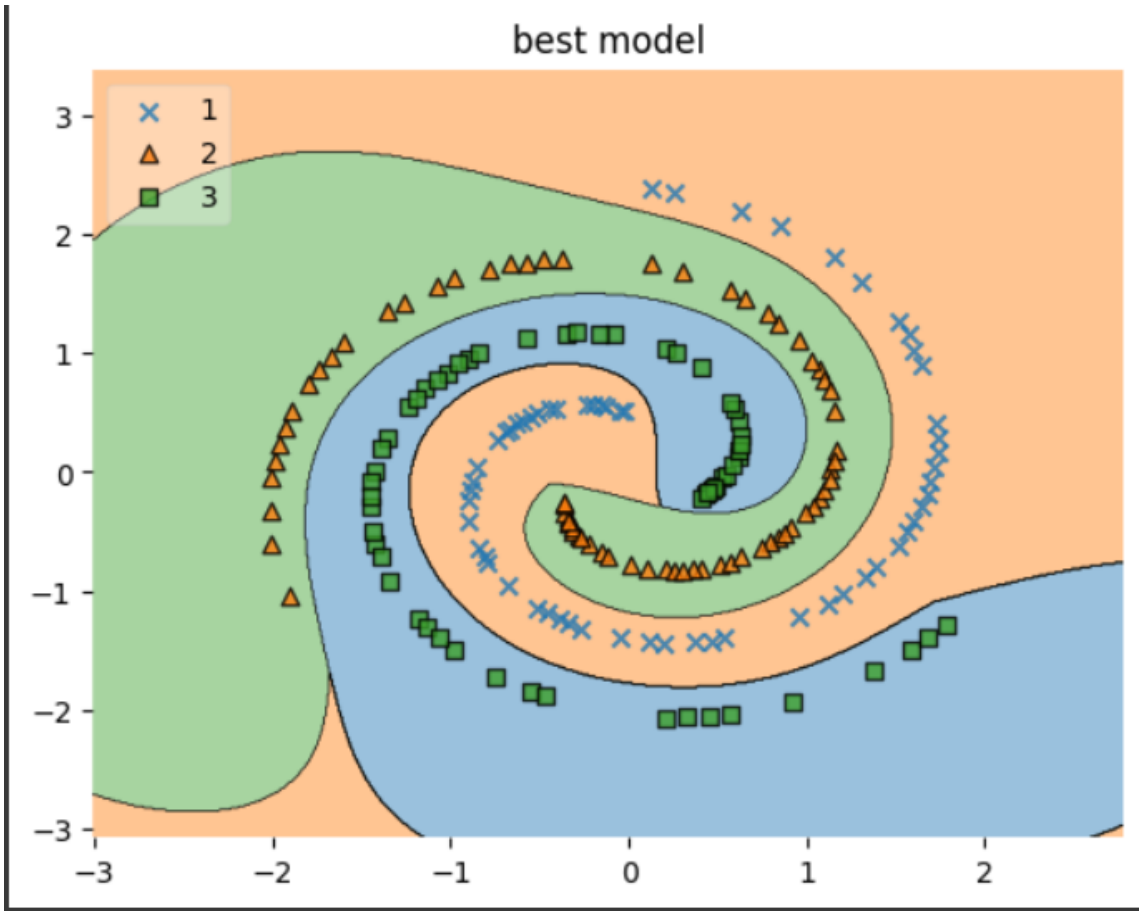

Figure 6:

5.6 Pathbased

- C: 5.6000000000000005
- kernel :rbf
- Gamma: scale
- validation accuracy: 100.0000%
- testing accuracy: 98.3333%

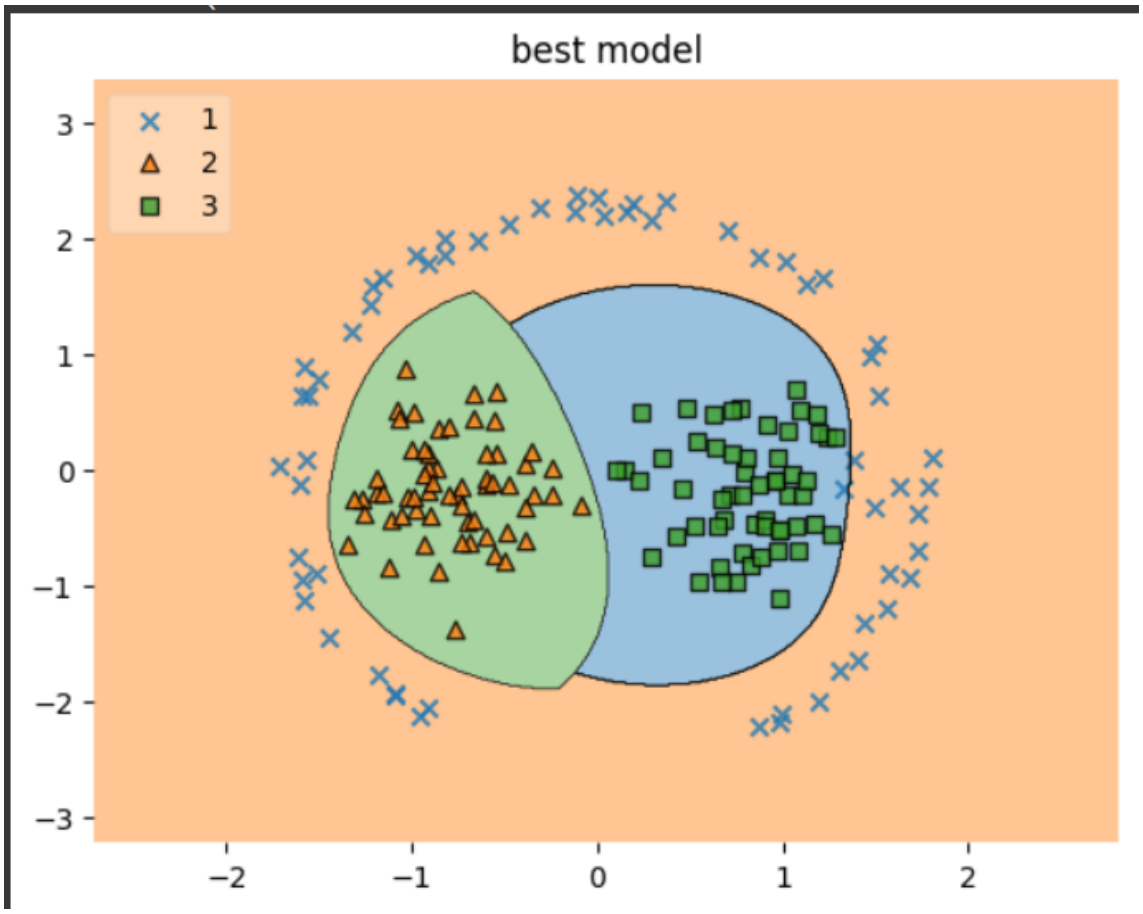

Figure 7:

6 Conclusion

The approach revealed the impact of different kernels and regularization on SVM classification. Insights into the behavior of the models with varying parameters were obtained through comprehensive visualization. The study provided valuable learning experiences and highlighted the significance of proper parameter selection in building effective classification models. Areas for improvement include exploring additional kernels and regularization strategies, and optimizing feature engineering techniques for enhanced model performance.