“In The Name Of God”

HW#6 Report

Professor: Dr.Azmifar

Presenters:

Narges Dehghan

Masome Jafari

Table of Contents

Abstract

This assignment encompasses a comprehensive exploration of implement SVM and how to visualize the resulted probability density functions., fundamental techniques in the realm of machine learning and data analysis. The primary objectives are twofold: to understand the fundamental concepts and application of these methods and to apply this knowledge to real-world datasets.

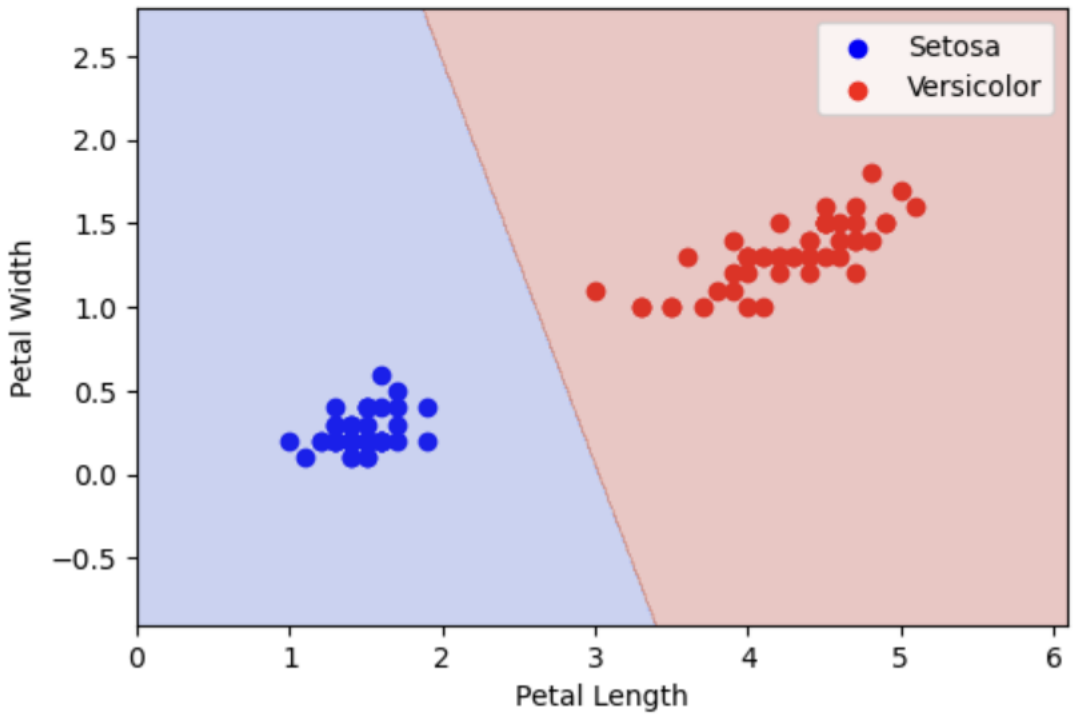
Datasets

Part one,This implementation uses the popular Iris dataset, available in the scikit-learn library, to demonstrate logistic regression for binary classification. The dataset contains four features (sepal length, sepal width, petal length, and petal width) and three target classes (Setosa, Versicolor, and Virginica). In this case, we focus on classes 0 (Setosa) and 1 (Versicolor).

1. **Importing Libraries:**
    - The code starts by importing the necessary libraries, including Pandas for data manipulation, NumPy for numerical operations, Matplotlib for plotting, scikit-learn for dataset and logistic regression model, and sklearn.datasets for accessing the Iris dataset.
2. **Loading and Exploring the Iris Dataset:**
    - The Iris dataset is loaded using **data.load_iris()**. The feature names and target names are then printed to understand the structure of the dataset.
3. **Data Preparation:**
    - The target classes to be considered are selected as [0, 1], representing Setosa and Versicolor.
    - A boolean mask is created using NumPy to filter the dataset for the selected classes.
    - Features (petal length and petal width) and corresponding target values are extracted based on the mask.
4. **Logistic Regression Model:**
    - A logistic regression model is instantiated using **LogisticRegression()** from scikit-learn.

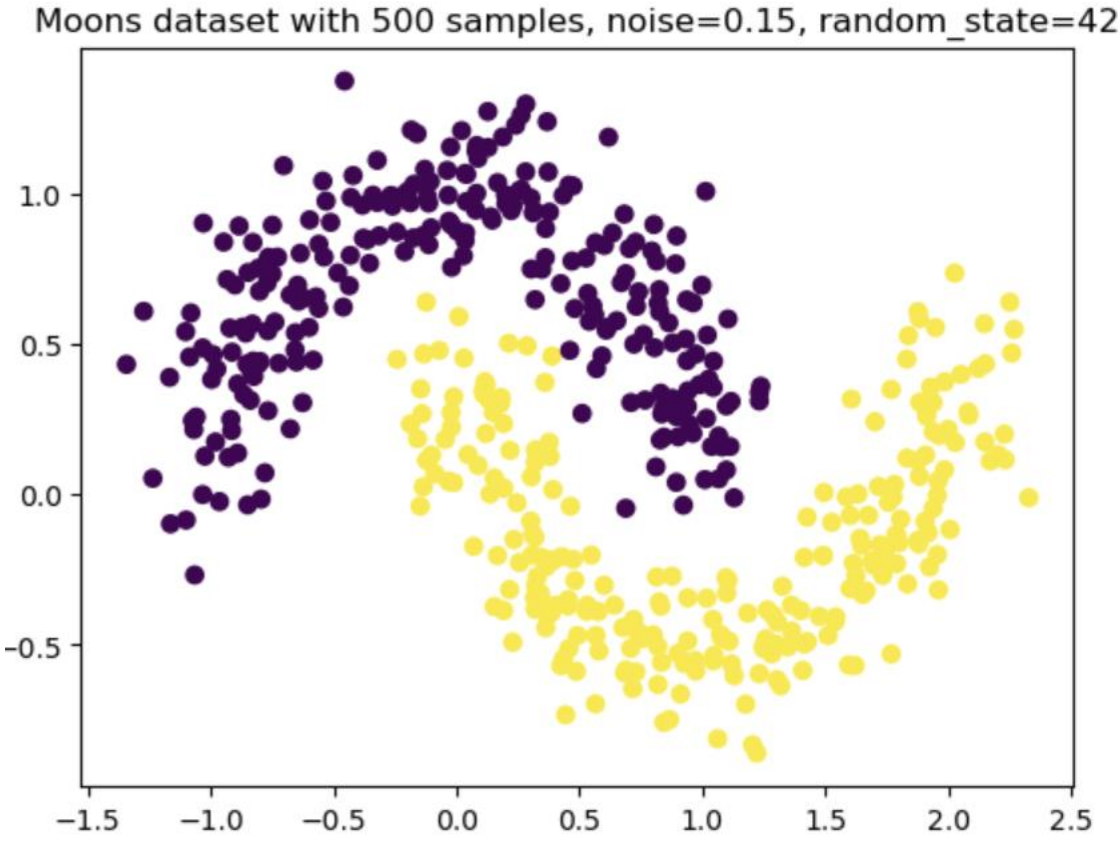
- The model is trained on the selected features (**x**) and target values (**y**) using the **fit** method.

5. **Visualization:**
    - Matplotlib is used to create a scatter plot of the data points, where Setosa is shown in blue and Versicolor in red.
    - Decision boundaries are plotted using the logistic regression model to visually represent the classification regions.
    - The **contourf** function is used to fill the decision regions with color, creating a clear separation between Setosa and Versicolor.



Part 2:

This implementation successfully generates and visualizes a synthetic moons dataset, which can be utilized for testing and evaluating binary classification algorithms. The provided parameters allow for customization, enabling the creation of datasets with varying characteristics for experimental purposes.



SVM:

The goal of this implementation is to train a Support Vector Machine (SVM) with a linear kernel on the Iris dataset to classify samples into two classes (0 or 1). The implementation includes tuning the hyperparameters using RandomizedSearchCV and visualizing the decision boundary, margin, and support vectors.

Dataset:

The Iris dataset is used for this implementation. It is loaded using the **load_iris** function from **sklearn.datasets**, and only features corresponding to the second and third columns (indices 2:4) are considered. Class labels are binary, representing either class 0 or class 1.

Data Preprocessing:

The dataset is split into training and testing sets using **train_test_split**. Standard scaling is applied to normalize the feature values using **StandardScaler** from **sklearn.preprocessing**.

Model Training and Hyperparameter Tuning:

A linear Support Vector Machine (SVM) model is used with the **SVC** class from **sklearn.svm**. Hyperparameter tuning is performed using RandomizedSearchCV, exploring different values of the regularization parameter **C** through a uniform distribution ranging from 0 to 20. The best hyperparameter values are printed, and the best model is selected for further analysis.

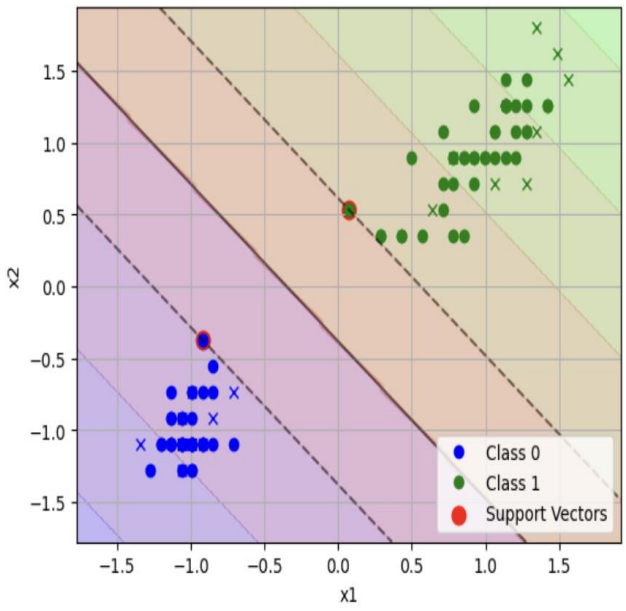
Evaluation:

The trained SVM model is evaluated on the test set, and predictions are obtained using the **predict** method. The **plot_svm** function is used to create a visual representation of the SVM decision boundary, margin, and support vectors on both the training and test sets.

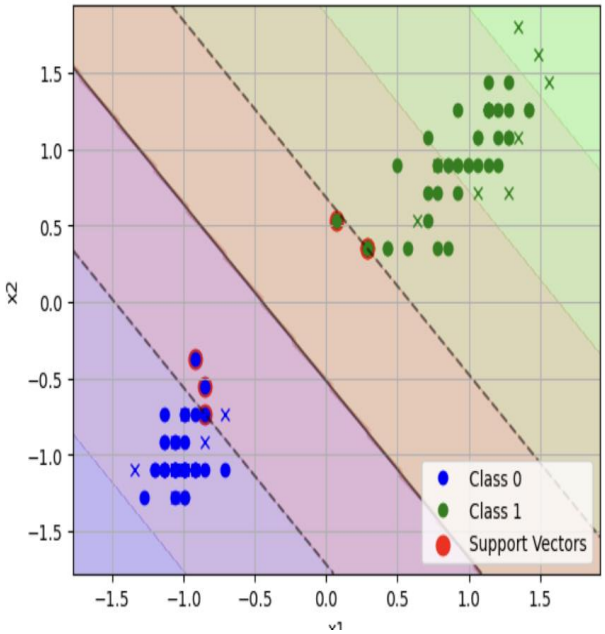
The SVM model with a linear kernel demonstrates its ability to classify samples from the Iris dataset. The hyperparameters are tuned using RandomizedSearchCV, and the resulting model is

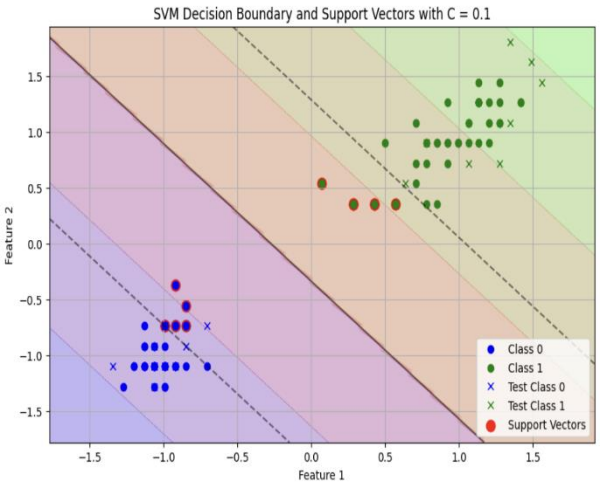
evaluated and visually presented. This implementation provides a comprehensive understanding of the SVM's performance and decision boundaries on the given dataset. We plot it for different c as follow:

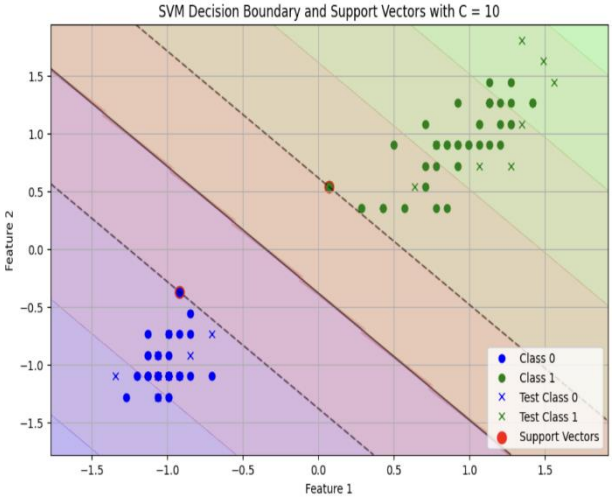
```
best hyper-parameter value:  {'C': 8.153740561605723}
```




```
best hyper-parameter value:  {'C': 0.4076870280802861}
```








```
Recall: 1.0
Precision: 1.0
Confusion Matrix:
 [[12  0]
 [ 0  8]]
```

SVM(Non-Liner)

1. Data Preprocessing:

- The moons dataset is split into training and test sets using **train_test_split** from **sklearn.model_selection**.
- Standardization is applied to the feature sets using **StandardScaler** to ensure uniform scale in training and test data.

2. Model Selection and Hyperparameter Tuning:

- Two Support Vector Machine (SVM) models are considered with different kernels: Polynomial Kernel and Radial Basis Function (RBF) Kernel.
- Hyperparameter tuning is performed using **tune** function (not provided in the code snippet) with the specified parameter distributions for each kernel type.

3. Model Evaluation:

- The best models (tuned for both Polynomial and RBF kernels) are used to predict labels on the test set.
- Classification reports and confusion matrices are generated for both models using **classification_report** and **confusion_matrix** from **sklearn.metrics**. These provide detailed information on precision, recall, F1-score, and support for each class.

4. Decision Boundary Visualization:

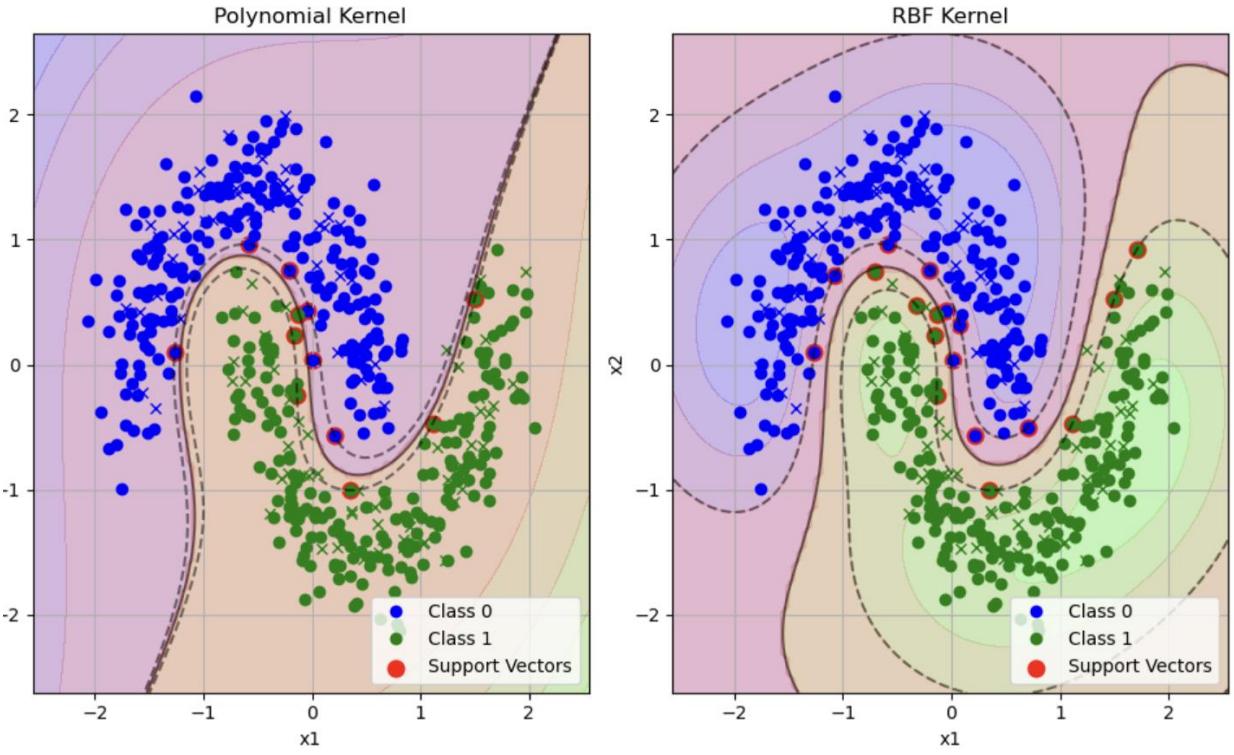
- A custom function **plot_svm** is implemented to visualize the SVM decision boundaries, margins, support vectors, and the distribution of training and test data.
- The decision boundaries for both the Polynomial and RBF kernels are plotted using the **plot_svm** function.

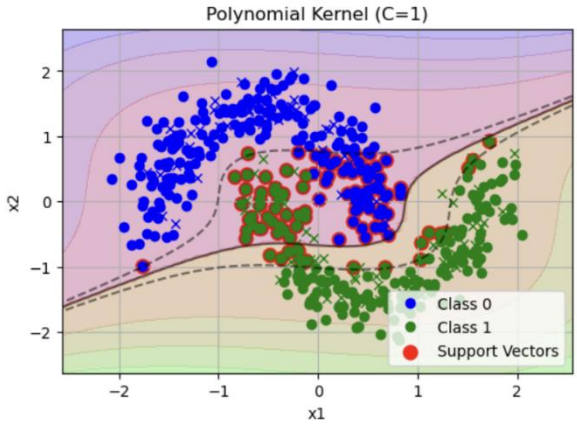
5. Results and Interpretation:

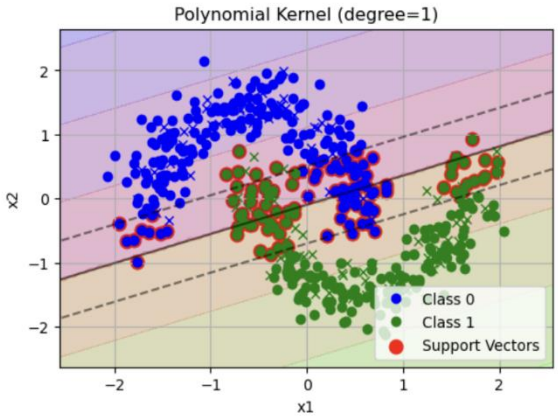
- Classification reports and confusion matrices offer insights into the performance of each SVM model.
- Decision boundary plots provide a visual representation of how well the models separate the two classes in the feature space.

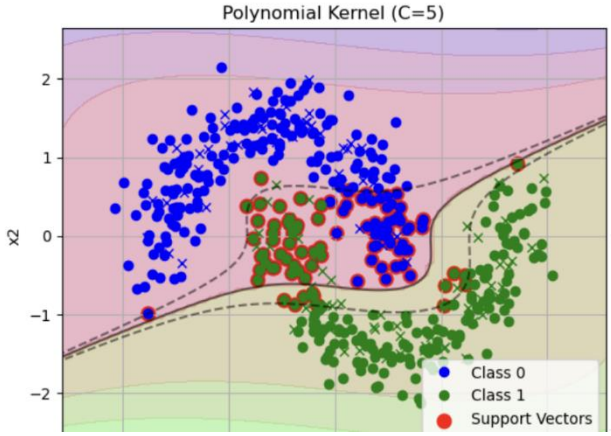
The implementation demonstrates the process of splitting data, preprocessing, hyperparameter tuning, model evaluation, and decision boundary visualization for SVMs with Polynomial and RBF kernels.

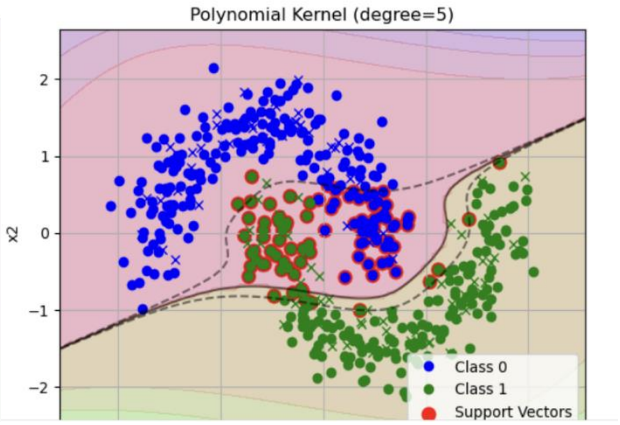
The results and visualizations should be carefully examined to make informed decisions about the model's performance and generalization to new, unseen data.

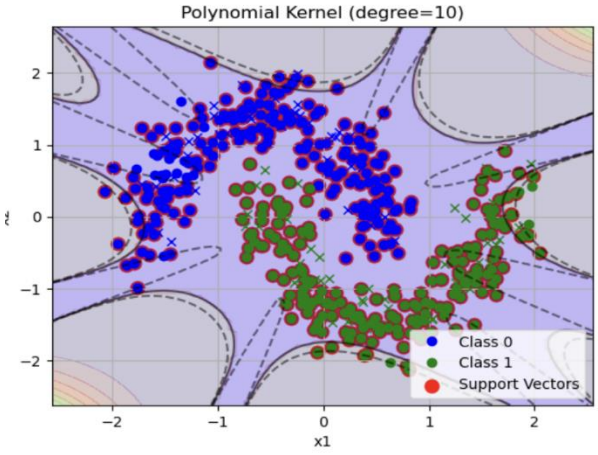


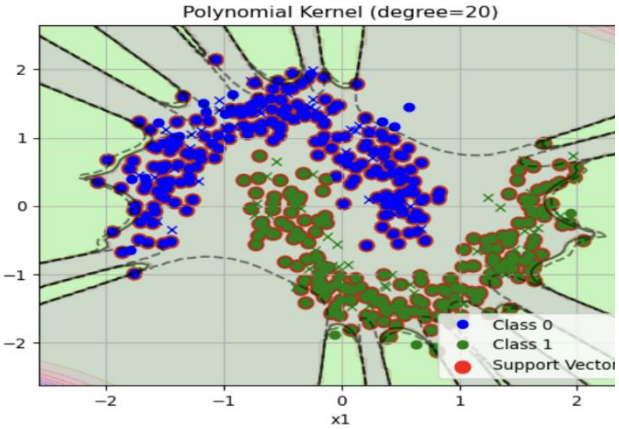

Polynomial Kernel (C=1)
Class 0
Class 1
Support Vectors
x1
x2

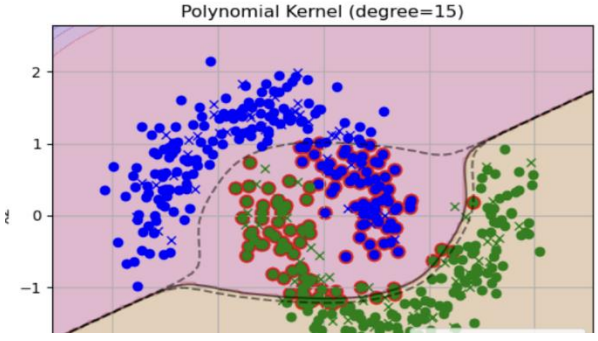

Polynomial Kernel (degree=1)
Class 0
Class 1
Support Vectors
x1
x2

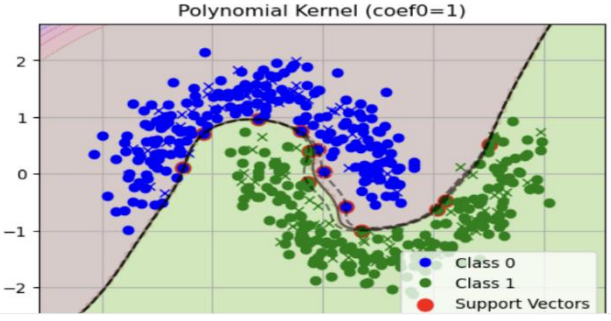

Polynomial Kernel (C=5)
Class 0
Class 1
Support Vectors
x2


Polynomial Kernel (degree=5)
Class 0
Class 1
Support Vectors
x2


Polynomial Kernel (degree=10)
Class 0
Class 1
Support Vectors
x1


Polynomial Kernel (degree=20)
Class 0
Class 1
x1


Polynomial Kernel (degree=15)


Polynomial Kernel (coef0=1)
Class 0
Class 1
Support Vectors

Bouns:

1. Data Generation:

- Synthetic data is generated using NumPy to create training data (**X_train**), regular novel observations (**X_test**), and abnormal novel observations (**X_outliers**).
- Training data consists of two sets shifted by a constant.

2. Model Training:

- A One-Class SVM model (**svm.OneClassSVM**) is employed with a radial basis function (RBF) kernel.
- The model is trained on the training data (**X_train**) with a specified nu value (**nu=0.1**) and gamma value (**gamma=0.1**).

3. Prediction and Error Computation:

- Predictions are made on the training data, regular novel observations, and abnormal novel observations.
- Errors are computed based on predictions for training data (**n_error_train**), regular novel observations (**n_error_test**), and abnormal novel observations (**n_error_outliers**).

4. Decision Boundary Visualization:

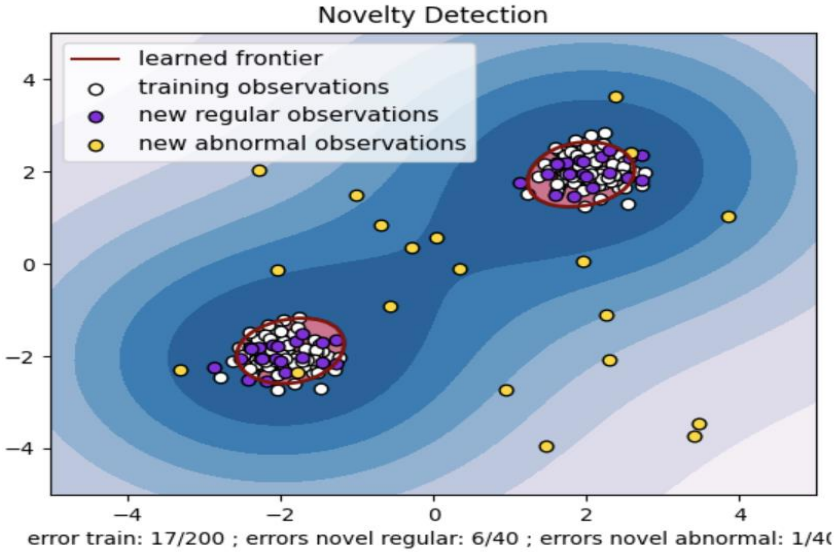
- Decision boundary displays are generated using **DecisionBoundaryDisplay** from **sklearn.inspection**.
- The decision function is used for plotting contours, and different levels and colors are specified for clear visualization.
- Scatter plots are created for training observations (**X_train**), new regular observations (**X_test**), and new abnormal observations (**X_outliers**).
- A legend is added to the plot to identify the learned frontier and different types of observations.

5. Results Display:

- The final plot includes visual representations of the decision boundaries, training, regular novel, and abnormal novel observations.
- The legend provides information about the learned frontier and different types of observations.
- Additional information such as error rates for training and novel observations is displayed in the plot title and x-axis label.

6. Conclusion:

- The implementation demonstrates the use of One-Class SVM for novelty detection.
- Visualization of decision boundaries helps in understanding how the model separates different types of observations.
- Error rates provide quantitative information on the model's performance on training and novel data.
- The code and visualizations offer insights into the novelty detection process and showcase the effectiveness of the One-Class SVM model in identifying abnormal observations.



Conclusion

Both codes serve as educational examples of how to implement and apply fundamental machine learning algorithms. They highlight the importance of data preprocessing, parameter optimization, and result visualization in the context of regression and classification tasks. These codes can be a starting point for more advanced modeling and analysis of the respective domains they address.