SVM(Kernels - Simple)

Dear Professor : Mr.Manthouri

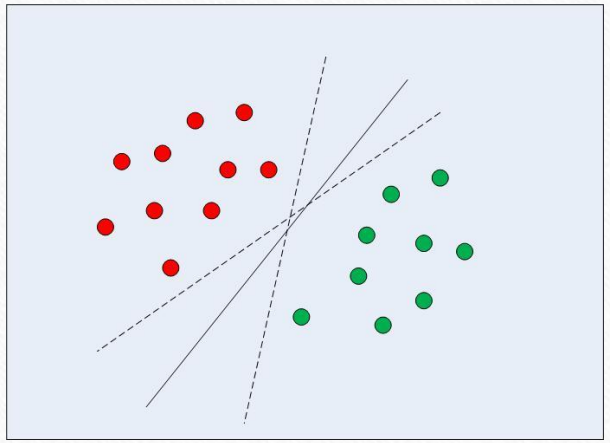
Produced By : Ghasemi,morteza

https://github.com/Morteza-Ghasemi

January 2021

SVM(Kernels - Simple)

- In case of linearly separable data in two dimensions



- SVM differs from the other classification algorithms in the way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes.

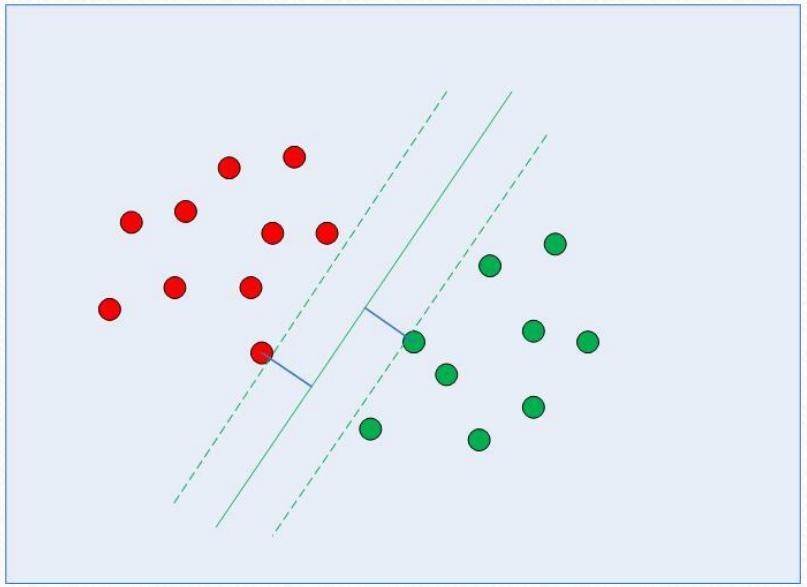
SVM(Kernels - Simple)

- An SVM doesn't merely find a decision boundary; it finds the most optimal decision boundary.

- The most optimal decision boundary is

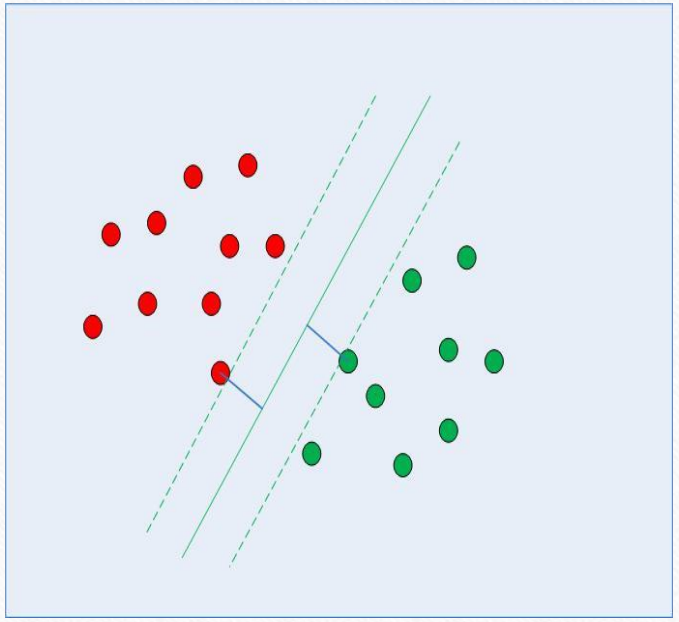
  the one which has maximum margin from

  the nearest points of all the classes.



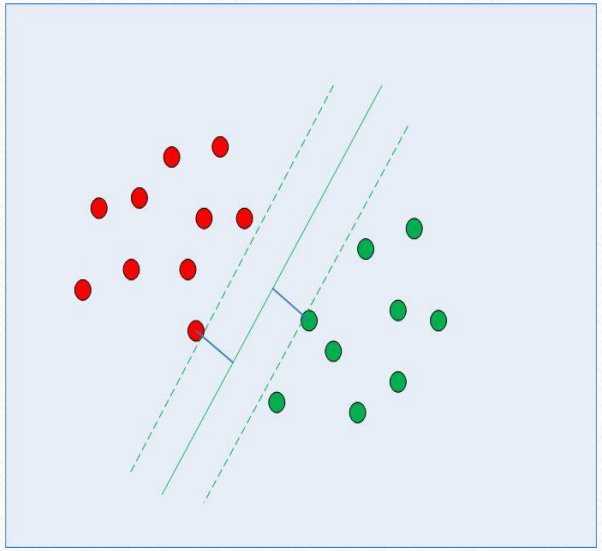
SVM(Kernels - Simple)

- he nearest points from the decision boundary that maximize the distance between the decision boundary and the points are called support vectors.
- The decision boundary in case of support vector machines is called the maximum margin classifier, or the maximum margin hyper plane.



SVM(Kernels - Simple)

- There is complex mathematics involved behind finding the support vectors, calculating the margin between decision boundary and the support vectors and maximizing this margin.



SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- The task is to predict whether a bank currency note is authentic or not based upon four attributes of the note i.e. skewness of the wavelet transformed image, variance of the image, entropy of the image, and curtosis of the image.
- This is a binary classification problem and then will use SVM algorithm to solve this problem.

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Importing the Dataset**
  - ✓ The data is available for download at the following link: https://drive.google.com/file/d/13nw-uRXPY8XIZQxKRNZ3yYlho-CYm_Qt/view
  - ✓ The detailed information about the data is available at the following link: https://archive.ics.uci.edu/ml/datasets/banknote+authentication

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Exploratory Data Analysis**
  - ✓ The following code reads bank currency note data into pandas dataframe:

    bankdata = pd.read_csv(r"D:\...\bill_authentication.csv")
  - ✓ To see the rows and columns and of the data, execute the following command:

    bankdata.shape

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Exploratory Data Analysis**
  - ✓ To get a feel of how our dataset actually looks, execute the following command:
    bankdata.head()
  - ✓ You can see that all of the attributes in the dataset are numeric. The label is also numeric i.e. 0 and 1.

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Data Preprocessing**
  - ✓ Dividing the data into attributes and labels :
    - ✓ X = bankdata.drop('Class', axis=1)
    - ✓ y = bankdata['Class']
  - ✓ dividing the data into training and testing sets:
    - ✓ from sklearn.model_selection import train_test_split
    - ✓ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Training the Algorithm**
  - ✓ we will use the support vector classifier class, which is written as SVC in the Scikit-Learn's svm library.
  - ✓ This class takes one parameter, which is the kernel type.
  - ✓ This is very important. In the case of a simple SVM we simply set this parameter as "linear" since simple SVMs can only classify linearly separable data.

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Training the Algorithm**
  - ✓ The fit method of SVC class is called to train the algorithm on the training data, which is passed as a parameter to the fit method.
    - ✓ from sklearn.svm import SVC
    - ✓ svclassifier = SVC(kernel='linear')
    - ✓ svclassifier.fit(X_train, y_train)

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Making Predictions**
  - ✓ To make predictions, the predict method of the SVC class is used.
    - ✓ y_pred = svclassifier.predict(X_test)

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

- **Evaluating the Algorithm**
  - ✓ Confusion matrix, precision, recall, and F1 measures are the most commonly used metrics for classification tasks.
    - ✓ from sklearn.metrics import classification_report, confusion_matrix
    - ✓ print(confusion_matrix(y_test,y_pred))
    - ✓ print(classification_report(y_test,y_pred))

SVM(Kernels - Simple)

Implementing SVM with Scikit-Learn

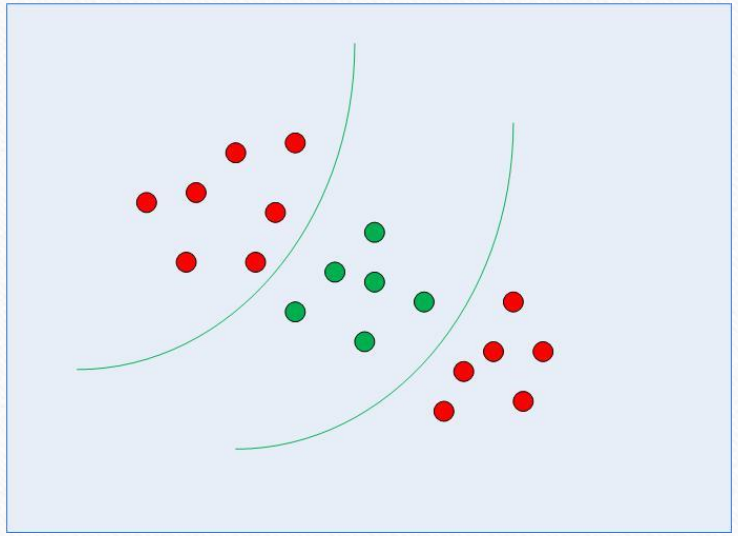
- **Evaluating the Algorithm**
  - ✓ The output of the SVM with linear kernel looks like this:

```
[[152   0]
 [  1 122]]
             precision    recall  f1-score   support

          0       0.99      1.00      1.00       152
          1       1.00      0.99      1.00       123

avg / total       1.00      1.00      1.00       275
```

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

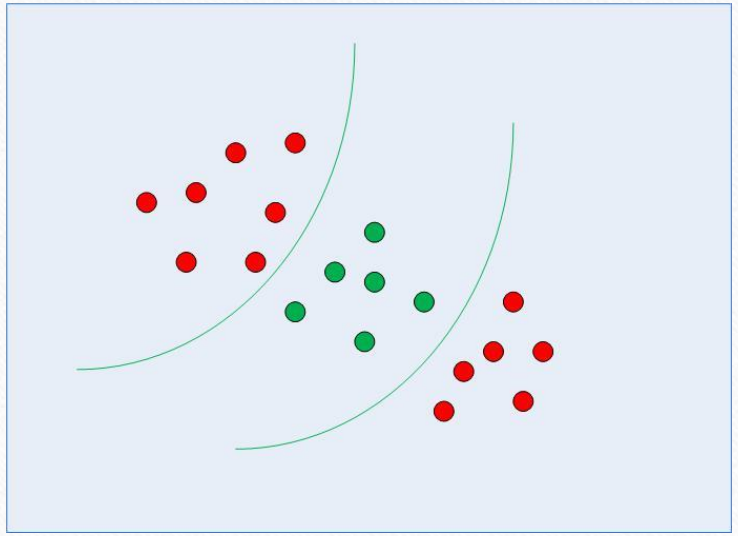
- in the case of non-linearly separable data, a straight line cannot be used as a decision boundary. Rather, a modified version of SVM, called Kernel SVM, is used.



SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- Basically, the kernel SVM projects the non-linearly separable data lower dimensions to linearly separable data in higher dimensions in such a way that data points belonging to different classes are allocated to different dimensions.



SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Importing libraries**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Importing the Dataset**
  - ✓ url = https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
  - ✓ colnames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
  - ✓ irisdata = pd.read_csv(url, names=colnames)

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Preprocessing**
  - ✓ X = irisdata.drop('Class', axis=1)
  - ✓ y = irisdata['Class']

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Train Test Split**
  - ✓ from sklearn.model_selection import train_test_split
  - ✓ X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Training the Algorithm**
  - ✓ In the case of the simple SVM we used "linear" as the value for the kernel parameter. However, for kernel SVM you can use Gaussian, polynomial, sigmoid, or computable kernel.
  - ✓ We will implement polynomial, Gaussian, and sigmoid kernels to see which one works better for our problem.

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Polynomial Kernel**
  - ✓ In the case of polynomial kernel, you also have to pass a value for the degree parameter of the SVC class. This basically is the degree of the polynomial.
  - ✓ from sklearn.svm import SVC
  - ✓ svclassifier = SVC(kernel='poly', degree=8)
  - ✓ svclassifier.fit(X_train, y_train)

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Making Predictions**
  - ✓ Now once we have trained the algorithm, the next step is to make predictions on the test data
  - ✓ y_pred = svclassifier.predict(X_test)

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Evaluating the Algorithm**
  - ✓ As usual, the final step of any machine learning algorithm is to make evaluations for polynomial kernel.
  - ✓ from sklearn.metrics import classification_report, confusion_matrix
  - ✓ print(confusion_matrix(y_test, y_pred))
  - ✓ print(classification_report(y_test, y_pred))

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Evaluating the Algorithm**
  - ✓ The output for the kernel SVM using polynomial kernel looks like this:

```
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

    avg / total       0.97      0.97      0.97        30
```

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Gaussian Kernel**
  - ✓ To use Gaussian kernel, you have to specify 'rbf' as value for the Kernel parameter of the SVC class.
  - ✓ svclassifier = SVC(kernel='rbf')

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Evaluating the Algorithm**
  - ✓ The output of the Kernel SVM with Gaussian kernel looks like this:

```
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

    avg / total       1.00      1.00      1.00        30
```

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Sigmoid Kernel**
  - ✓ To use sigmoid kernel, you have to specify ' sigmoid ' as value for the Kernel parameter of the SVC class.
  - ✓ svclassifier = SVC(kernel='sigmoid')

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Evaluating the Algorithm**
  - ✓ The output of the Kernel SVM with Sigmoid kernel looks like this:

```
[[ 0  0 11]
 [ 0  0 13]
 [ 0  0  6]]
                precision    recall  f1-score   support

    Iris-setosa       0.00      0.00      0.00        11
Iris-versicolor       0.00      0.00      0.00        13
 Iris-virginica       0.20      1.00      0.33         6

    avg / total       0.04      0.20      0.07        30
```

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Comparison of Kernel Performance**
  - ✓ If we compare the performance of the different types of kernels we can clearly see that the sigmoid kernel performs the worst. This is due to the reason that sigmoid function returns two values, 0 and 1, therefore it is more suitable for binary classification problems. However, in our case we had three output classes.

SVM(Kernels - Simple)

Implementing Kernel SVM with Scikit-Learn

- **Comparison of Kernel Performance**
  - ✓ Amongst the Gaussian kernel and polynomial kernel, we can see that Gaussian kernel achieved a perfect 100% prediction rate while polynomial kernel misclassified one instance. Therefore the Gaussian kernel performed slightly better. However, there is no hard and fast rule as to which kernel performs best in every scenario. It is all about testing all the kernels and selecting the one with the best results on your test dataset.

SVM(Kernels - Simple)- **Results**
Comparison of Kernels Performance

Hints :

- **the dataset is banknote authentication**

Abstract: Data were extracted from images that were taken for the evaluation of an authentication procedure for banknotes.

Data Set Characteristics:	Multivariate	**Number of Instances:**	1372	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	5	**Date Donated**	2013-04-16
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	290722

- **Kernels : Linear, Gaussian(Rbf), Polynomial, Sigmoid**

SVM(Kernels - Simple)- **Results**

Comparison of Kernels Performance

```
############ SVC(kernel='linear') ############
[[144   3]
 [  0 128]]
             precision    recall  f1-score   support

          0       1.00      0.98      0.99       147
          1       0.98      1.00      0.99       128

avg / total       0.99      0.99      0.99       275
```

SVM(Kernels - Simple)- **Results**

Comparison of Kernels Performance

```
############# SVC(kernel='poly') #############
[[146   1]
 [  0 128]]
             precision    recall  f1-score   support

          0       1.00      0.99      1.00       147
          1       0.99      1.00      1.00       128

avg / total       1.00      1.00      1.00       275
```

SVM(Kernels - Simple)- **Results**

Comparison of Kernels Performance

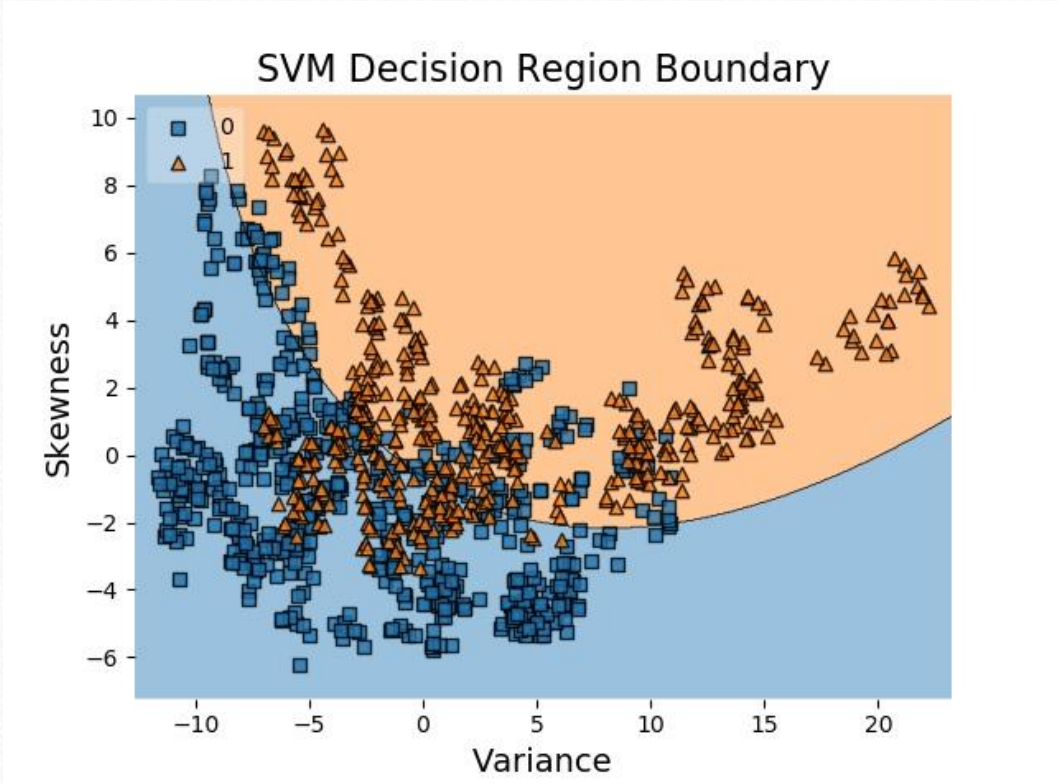
```
############ SVC(kernel='rbf') ############
[[147   0]
 [  0 128]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       147
          1       1.00      1.00      1.00       128

avg / total       1.00      1.00      1.00       275
```

SVM(Kernels - Simple)- **Results**

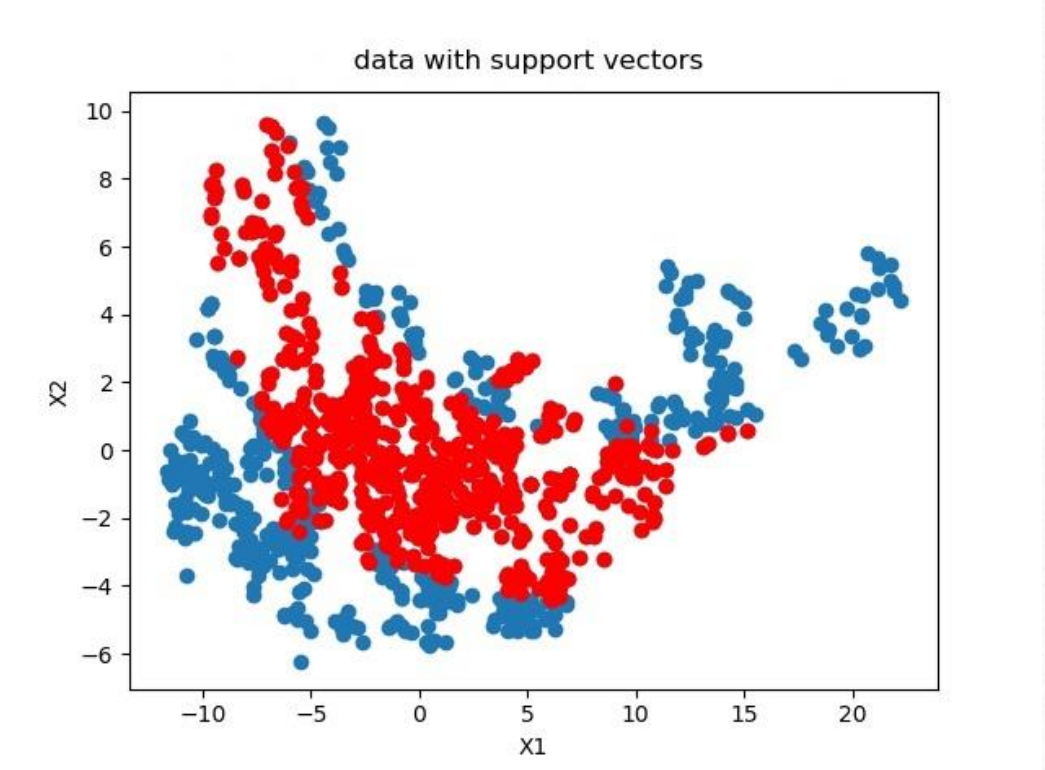
Comparison of Kernels Performance

```
############ SVC(kernel='sigmoid') ############
[[122  25]
 [ 32  96]]
             precision    recall  f1-score   support

          0       0.79      0.83      0.81       147
          1       0.79      0.75      0.77       128

avg / total       0.79      0.79      0.79       275
```

Decision Region Boundary



Data With Support Vectors



The End