**Implementing Kernel SVM with Scikit-Learn**

Implementing Kernel SVM with Scikit-Learn is similar to the simple SVM. In this section, we will use the famous [iris dataset](https://en.wikipedia.org/wiki/Iris_flower_data_set) to predict the category to which a plant belongs based on four attributes: sepal-width, sepal-length, petal-width and petal-length.

The dataset can be downloaded from the following link:

<https://archive.ics.uci.edu/ml/datasets/iris4>

The rest of the steps are typical machine learning steps and need very little explanation until we reach the part where we train our Kernel SVM.

**Importing Libraries**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

**Importing the Dataset**

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

*# Assign colum names to the dataset*

colnames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

*# Read dataset to pandas dataframe*

irisdata = pd.read\_csv(url, names=colnames)

**Preprocessing**

X = irisdata.drop('Class', axis=1)

y = irisdata['Class']

**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)

**Training the Algorithm**

To train the kernel SVM, we use the same SVC class of the Scikit-Learn's svm library. The difference lies in the value for the kernel parameter of the SVC class. 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.

**1. 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. Take a look at how we can use a polynomial kernel to implement kernel SVM:

from sklearn.svm import SVC

svclassifier = SVC(kernel='poly', degree=8)

svclassifier.fit(X\_train, y\_train)

**Making Predictions**

Now once we have trained the algorithm, the next step is to make predictions on the test data.

Execute the following script to do so:

y\_pred = svclassifier.predict(X\_test)

**Evaluating the Algorithm**

As usual, the final step of any machine learning algorithm is to make evaluations for polynomial kernel. Execute the following script:

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

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

Now let's repeat the same steps for Gaussian and sigmoid kernels.

**2. Gaussian Kernel**

Take a look at how we can use polynomial kernel to implement kernel SVM:

from sklearn.svm import SVC

svclassifier = SVC(kernel='rbf')

svclassifier.fit(X\_train, y\_train)

To use Gaussian kernel, you have to specify 'rbf' as value for the Kernel parameter of the SVC class.

**Prediction and Evaluation**

y\_pred = svclassifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

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

**3. Sigmoid Kernel**

Finally, let's use a sigmoid kernel for implementing Kernel SVM. Take a look at the following script:

from sklearn.svm import SVC

svclassifier = SVC(kernel='sigmoid')

svclassifier.fit(X\_train, y\_train)

To use the sigmoid kernel, you have to specify 'sigmoid' as value for the kernel parameter of the SVC class.

**Prediction and Evaluation**

y\_pred = svclassifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

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

**4. 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.

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.

### **5. Grid Search**

Now we will tune the parameters, check for the improvement

Tuning parameters value for machine learning algorithms effectively improves the model performance.

Import Gridsearch from Scikit Learn.

from sklearn.model\_selection import GridSearchCV

Create a dictionary called param\_grid and fill out some parameters for C and Gamma

param\_grid = {'C':[0.1,1,10,100], 'gamma':[1,0.1,0.01,0.001]}

Create a GridSearchCV object and fit it to the training data

Grid search is a model hyperparameter optimization technique.

grid = GridSearchCV(SVC(), param\_grid, refit = True, verbose=3)

grid.fit(X\_train, y\_train)

Let us predict using the Grid model

pred\_grid = grid.predict(X\_test)

Let us compute the confusion matrix

print(confusion\_matrix(y\_test, pred\_grid))

[[12 0 0]

[ 0 11 0]

[ 0 1 6]]

Let us print the report also

print(classification\_report(y\_test, pred\_grid))

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 12

Iris-versicolor 0.92 1.00 0.96 11

Iris-virginica 1.00 0.86 0.92 7

accuracy 0.97 30

macro avg 0.97 0.95 0.96 30

weighted avg 0.97 0.97 0.97 30

**SVR in 6 Steps with Python:**

Let’s jump to the Python practice on this topic. Here is the link you can reach the dataset for this problem.

#1 Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

#2 Importing the dataset

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:,1:2].values.astype(float)

y = dataset.iloc[:,2:3].values.astype(float)

#3 Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

sc\_y = StandardScaler()

X = sc\_X.fit\_transform(X)

y = sc\_y.fit\_transform(y)

#4 Fitting the Support Vector Regression Model to the dataset

# Create your support vector regressor here

from sklearn.svm import SVR

# most important SVR parameter is Kernel type. It can be linear,polynomial or gaussian.

#SVR. We have a non-linear condition so we can select polynomial or gaussian but here

#we select RBF(a gaussian type) kernel.

regressor = SVR(kernel='rbf')

regressor.fit(X,y)

#5 Predicting a new result

y\_pred = sc\_y.inverse\_transform((regressor.predict(sc\_X.transform(np.array([[6.5]])))))

#6 Visualising the Support Vector Regression results

plt.scatter(X, y, color = 'magenta')

plt.plot(X, regressor.predict(X), color = 'green')

plt.title('Truth or Bluff (Support Vector Regression Model)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()

#6 Visualising the Regression results (for higher resolution and smoother curve)

X\_grid = np.arange(min(X), max(X), 0.1)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.title('Truth or Bluff (Support Vector Regression Model(High Resolution))')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()