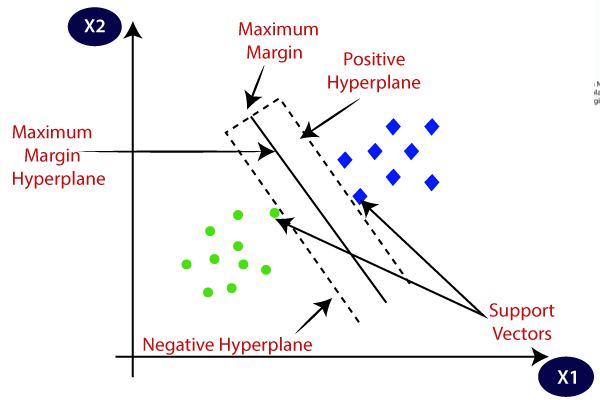
**Digit Recognition using SVM**

**Support Vector Machine Algorithm**

Support Vector Machine or SVM is a Supervised Learning algorithm, which is used for Classification and Regression problems. but it is mostly used for Classification problems in Machine Learning.

an SVM separates data across a decision boundary (plane) determined by only a small subset of the data (feature vectors). The data subset that supports the decision boundary is called the support vector. and The best decision boundary is called a hyperplane. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



The distance between the decision–boundary and the closest data points is called margin. And The optimization during training finds a hyperplane that has the maximum margin.

SVM algorithm can be used for **image classification, Face detection, text categorization,** etc.

**SVM can be of two types:**

* **Linear SVM:** if a dataset can be classified into two classes by using a single straight line, then such data is linearly separable data, and the classifier is used called a Linear SVM classifier. and the kernel is also used “linear”
* **Non-linear SVM:** if a dataset cannot be classified by using a straight line, then such data is non-linear data, and the classifier used is called a Non-linear SVM classifier. and the kernel is used “poly” or “rbf”.

**Python Implementation of Support Vector Machine of the Image Classification Problem**

Now we will implement the SVM algorithm using Python. Here we will use the digit recognizer data. You can download the data-set from the [kaggle](https://www.kaggle.com/khotijahs1/digitrecognizer/download) itself

 I have used 42000 samples and 28000 samples from the training and test data-sets just to reduce the time of computation

**Input the data**

import numpy as np   
import pandas as pd   
import matplotlib.pyplot as plt

train\_df = pd.read\_csv("train.csv")   
test\_df = pd.read\_csv("test.csv")

**Extracting Independent and dependent Variable.** Now we separate label and pixel columns and, the label is the 1st column of the data frame.

X = train\_df.drop('label', axis=1)  
y = train\_df['label']

**Model Training**

Then I have separate training and test data with 30% samples reserved for test data.

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 100)

**Data preprocessing**

I have used StandardScaler which standardizes features by removing mean and scaling it to unit variance. StandardScaler is that it will transform your data such that its distribution will have a mean value of 0 and a standard deviation of 1

from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()

# fit\_transform use to do some calculation and then do transformation  
X\_train = scaler.fit\_transform(X\_train)   
X\_test = scaler.transform(X\_test)

**Model construct**

Now the training set will be fitted to the SVM classifier and construct the model. To create the SVM classifier, we will import the **SVC** class from **Sklearn.svm** library. Below is the code for it:

from sklearn.svm import SVC  
rbf\_model = SVC(kernel=’rbf’)   
rbf\_model.fit(X\_train, y\_train)

In the above code, we have used **kernel=’rbf’**, as here we are creating SVM for linearly separable data. and Fit function is used to adjust weights according to data values

Note -*Radial base kernel support vector machine is a good approach when the data is not linearly separable. RBF Kernel is popular because of its similarity to K-Nearest Neighborhood Algorithm. It has overcome the space complexity problem as RBF Kernel.*

**Model Testing**

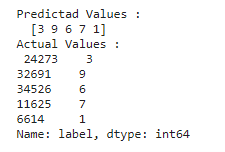
Now, we will predict the output for the test set. For this, we will create a new variable y\_rbf\_pred. Below is the code for it:

y\_rbf\_pred = rbf\_model.predict(X\_test)

After getting the y\_rbf\_pred vector, we can compare the result of **y\_rbf\_pred** and **y\_test** to check the difference between the actual value and predicted value.

print(“Predictad Values :\n “,y\_rbf\_pred[10:15])  
print (“Actual Values :\n”,y\_test[10:15])

**Output:-**



The accuracy of the model can be calculated using the **accuracy\_score()** method from sklearn.metrics

from sklearn import metrics  
acc\_rbf= metrics.accuracy\_score(y\_test, y\_rbf\_pred)  
print("accuracy:","{:.2f}".format(acc\_rbf\*100),"%")

**Output:-**

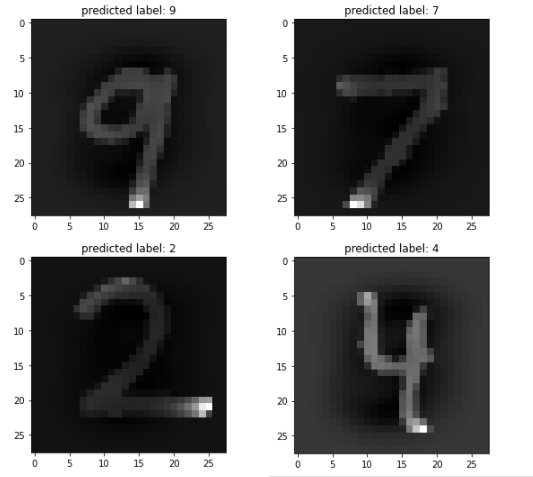
accuracy: 95.66 %

We can now plot the digits using python matplotlib.pyplot . We use the prediction list and the pixel values from the test list for comparison.

for i in (np.random.randint(0,270,4)):   
 two\_d = (np.reshape(X\_test[i], (28, 28)))  
 plt.title('predicted label: {0}'. format(y\_rbf\_pred[i])  
 plt.imshow(two\_d, cmap='gray')   
 plt.show()

Let me briefly explain the second line of the code. As the pixel values are arranged in a row with 784 columns in the data set, first we use NumPy ‘reshape’ module to arrange it in 28\*28 array

**Output:-**



## **End Notes**

In this document, we looked at the machine learning algorithm, Support Vector Machine in detail.  I discussed its concept of working, the process of implementation in python, the tricks to make the model efficient by tuning its parameters, and finally a problem to solve. I would suggest you use SVM and analyze the power of this model by tuning the parameters.