Sardar Patel Institute of Technology,Mumbai
Department of Electronics and Telecommunication Engineering
B.E. Sem-VII (2021-2022)
Data Analytics

**Experiment: Exploratory Data Analysis (EDA)**

**Name: Sneha Ghuge** **UID:** 2019110015 **BE ETRX** **DA LAB 10**

**Aim:** Understanding Support Vector Machine algorithm through building SVM algorithm in Python

**CODE & OUTPUT:**

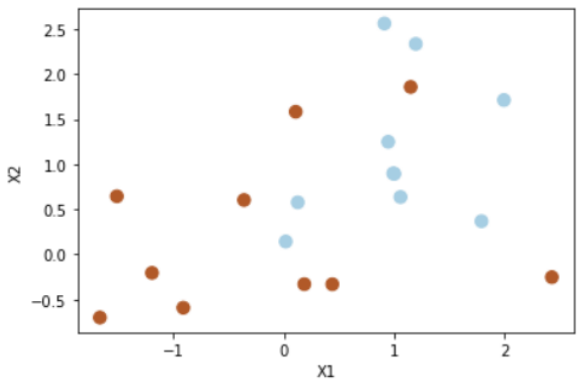
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

%matplotlib inline

# We'll define a function to draw a nice plot of an SVM
def plot_svc(svc, X, y, h=0.02, pad=0.25):
    x_min, x_max = X[:, 0].min()-pad, X[:, 0].max()+pad
    y_min, y_max = X[:, 1].min()-pad, X[:, 1].max()+pad
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.2)

    plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
    # Support vectors indicated in plot by vertical lines
    sv = svc.support_vectors_
    plt.scatter(sv[:,0], sv[:,1], c='k', marker='x', s=100, linewidths='1')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xlabel('X1')
    plt.ylabel('X2')
    plt.show()
    print('Number of support vectors: ', svc.support_.size)
```

```python
from sklearn.svm import SVC
# Generating random data: 20 observations of 2 features and divide into two classes.
np.random.seed(5)
X = np.random.randn(20,2)
y = np.repeat([1,-1], 10)

X[y == -1] = X[y == -1]+1
```

```python
#Let's plot the data to see whether the classes are linearly separable:
```

```python
plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
plt.xlabel('X1')
plt.ylabel('X2')
```

```
Text(0, 0.5, 'X2')
```




```python
#Since the classes are not linearly seperable; we fit the support vector classifier:
```

```python
svc = SVC(C=1, kernel='linear')
svc.fit(X, y)
```

```
SVC(C=1, kernel='linear')
```

```python
plot_svc(svc, X, y)
```

```
-

Number of support vectors:  16
```

```python
from sklearn.model_selection import GridSearchCV

# Select the optimal C parameter by cross-validation
tuned_parameters = [{'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}]
clf = GridSearchCV(SVC(kernel='linear'), tuned_parameters, cv=10, scoring='accuracy')
clf.fit(X, y)
```

```
GridSearchCV(cv=10, estimator=SVC(kernel='linear'),
             param_grid=[{'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}],
             scoring='accuracy')
```

```python
# Now let's see the cross-validation errors for each of these models:
```

```python
clf.cv_results_
```

```
{'mean_fit_time': array([0.0015589 , 0.00214555, 0.0033421 , 0.00355594, 0.00323312,
        0.00203841, 0.00372868]),
 'std_fit_time': array([0.00290138, 0.00297289, 0.00360184, 0.00444351, 0.0044069 ,
        0.00325821, 0.00395491]),
 'mean_score_time': array([0.00135007, 0.00154669, 0.        , 0.00081351, 0.00091844,
        0.        , 0.00050769]),
 'std_score_time': array([0.00276111, 0.00255209, 0.        , 0.00179364, 0.00239126,
        0.        , 0.00123364]),
 'param_C': masked_array(data=[0.001, 0.01, 0.1, 1, 5, 10, 100],
              mask=[False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.001},
  {'C': 0.01},
  {'C': 0.1},
  {'C': 1},
  {'C': 5},
  {'C': 10},
  {'C': 100}],
 'split0_test_score': array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
 'split1_test_score': array([0.5, 0.5, 0.5, 0. , 0. , 0. , 0. ]),
 'split2_test_score': array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
 'split3_test_score': array([1., 1., 1., 1., 1., 1., 1.]),
 'split4_test_score': array([1., 1., 1., 1., 1., 1., 1.]),
 'split5_test_score': array([1., 1., 1., 1., 1., 1., 1.]),
 'split6_test_score': array([1., 1., 1., 1., 1., 1., 1.]),
 'split7_test_score': array([1., 1., 1., 1., 1., 1., 1.]),
 'split8_test_score': array([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]),
```

```python
#The  GridSearchCV()  function stores the best parameters obtained; lets see those
clf.best_params_
```

```
{'C': 0.001}
```

```python
#c=0.001 is best according to  GridSearchCV .
```

```python
#The  predict()  function can be used to predict the class label on a set of test observation. Let's generate a test data set:
np.random.seed(1)
X_test = np.random.randn(20,2)
y_test = np.random.choice([-1,1], 20)
X_test[y_test == 1] = X_test[y_test == 1]-1
```

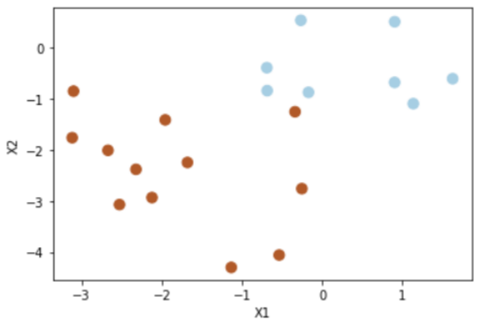
```python
#Now we predict the class labels of these test observations.
#Here we use the best model obtained through cross-validation in order to make predictions:

svc2 = SVC(C=0.001, kernel='linear')
svc2.fit(X, y)
y_pred = svc2.predict(X_test)
pd.DataFrame(confusion_matrix(y_test, y_pred), index=svc2.classes_, columns=svc2.classes_)
```

| | -1 | 1 |
|---|---|---|
| **-1** | 2 | 6 |
| **1** | 0 | 12 |

```python
#14 of the test observations are correctly classified.
```

```python
#Now consider a situation in which the two classes are linearly separable.
#Then we can find a separating hyperplane using the svm()  function.

X_test[y_test == 1] = X_test[y_test == 1] -1
plt.scatter(X_test[:,0], X_test[:,1], s=70, c=y_test, cmap=mpl.cm.Paired)
plt.xlabel('X1')
plt.ylabel('X2')
```

```
Text(0, 0.5, 'X2')
```




```python
#Now the observations are just barely linearly separable.
#We fit the support vector classifier and plot the resulting hyperplane, using a very large value of  cost  so that no observ
svc3 = SVC(C=1e5, kernel='linear')
svc3.fit(X_test, y_test)
plot_svc(svc3, X_test, y_test)
```

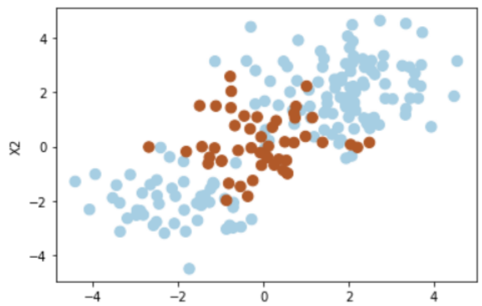
```
Number of support vectors:  3
```

```python
# No training errors were made and only three support vectors were used.
```

```
# Support Vector Machine
#In order to fit an SVM using a non-linear kernel, we once again use the  SVC()  function.
#However, now we use a different value of the parameter kernel.
#To fit an SVM with a polynomial kernel we use  kernel="poly" , and to fit an SVM with a radial kernel we use  kernel="rbf" .

#Let's generate some data with a non-linear class boundary:
```

```
from sklearn.model_selection import train_test_split

np.random.seed(8)
X = np.random.randn(200,2)
X[:100] = X[:100] +2
X[101:150] = X[101:150] -2
y = np.concatenate([np.repeat(-1, 150), np.repeat(1,50)])

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.5, random_state=2)

plt.scatter(X[:,0], X[:,1], s=70, c=y, cmap=mpl.cm.Paired)
plt.xlabel('X1')
plt.ylabel('X2')
```

```
Text(0, 0.5, 'X2')
```



```
#One class is kind of stuck in the middle of another class.
#This suggests that we might want to use a radial kernel in our SVM.
#Now let's fit the training data using the  SVC()  function with a radial kernel and  γ=1 :
svm = SVC(C=1.0, kernel='rbf', gamma=1)
svm.fit(X_train, y_train)
plot_svc(svm, X_test, y_test)
```

**Number of support vectors:  51**

```
# Increasing C parameter, allowing more flexibility
svm2 = SVC(C=100, kernel='rbf', gamma=1.0)
svm2.fit(X_train, y_train)
plot_svc(svm2, X_test, y_test)
```

Number of support vectors:  36

In [26]:

```
#However, this comes at the price of a more irregular decision boundary that seems to be at risk of overfitting the data.
#We can perform cross-validation using  GridSearchCV()  to select the best choice of  γ  and cost for an SVM with a radial ker

tuned_parameters = [{'C': [0.01, 0.1, 1, 10, 100],
                     'gamma': [0.5, 1,2,3,4]}]
clf = GridSearchCV(SVC(kernel='rbf'), tuned_parameters, cv=10, scoring='accuracy')
clf.fit(X_train, y_train)
clf.best_params_
```

```
{'C': 10, 'gamma': 0.5}
```

```
Number of support vectors:  32
[[66  7]
 [ 6 21]]
0.87
```

8]:
```
#87% of test observations are correctly classified by this SVM
```

9]:
```
#ROC Curves
from sklearn.metrics import auc
from sklearn.metrics import roc_curve
# More constrained model
svm3 = SVC(C=1, kernel='rbf', gamma=1)
svm3.fit(X_train, y_train)
```

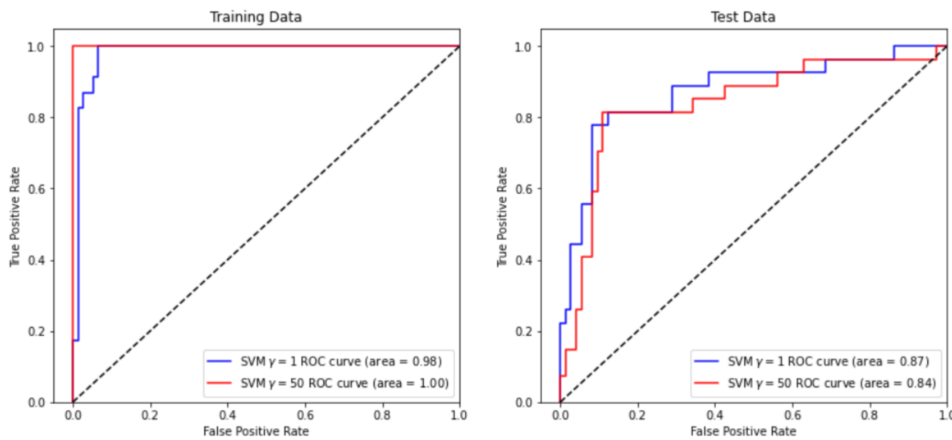
9]:
```
SVC(C=1, gamma=1)
```

0]:
```
# More flexible model
svm4 = SVC(C=1, kernel='rbf', gamma=50)
svm4.fit(X_train, y_train)
```

0]:
```
SVC(C=1, gamma=50)
```

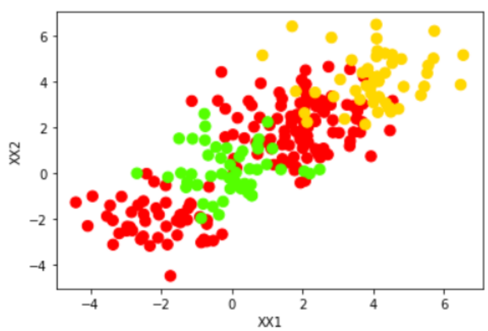
```
y_train_score3 = svm3.decision_function(X_train)
y_train_score4 = svm4.decision_function(X_train)
```

```
#ROC plot to see how the models perform on both the training and the test data:

y_train_score3 = svm3.decision_function(X_train)
y_train_score4 = svm4.decision_function(X_train)

false_pos_rate3, true_pos_rate3, _ = roc_curve(y_train, y_train_score3)
roc_auc3 = auc(false_pos_rate3, true_pos_rate3)

false_pos_rate4, true_pos_rate4, _ = roc_curve(y_train, y_train_score4)
roc_auc4 = auc(false_pos_rate4, true_pos_rate4)

fig, (ax1,ax2) = plt.subplots(1, 2, figsize=(14,6))
ax1.plot(false_pos_rate3, true_pos_rate3, label='SVM $\gamma = 1$ ROC curve (area = %0.2f)' % roc_auc3, color='b')
ax1.plot(false_pos_rate4, true_pos_rate4, label='SVM $\gamma = 50$ ROC curve (area = %0.2f)' % roc_auc4, color='r')
ax1.set_title('Training Data')

y_test_score3 = svm3.decision_function(X_test)
y_test_score4 = svm4.decision_function(X_test)

false_pos_rate3, true_pos_rate3, _ = roc_curve(y_test, y_test_score3)
roc_auc3 = auc(false_pos_rate3, true_pos_rate3)

false_pos_rate4, true_pos_rate4, _ = roc_curve(y_test, y_test_score4)
roc_auc4 = auc(false_pos_rate4, true_pos_rate4)

ax2.plot(false_pos_rate3, true_pos_rate3, label='SVM $\gamma = 1$ ROC curve (area = %0.2f)' % roc_auc3, color='b')
ax2.plot(false_pos_rate4, true_pos_rate4, label='SVM $\gamma = 50$ ROC curve (area = %0.2f)' % roc_auc4, color='r')
ax2.set_title('Test Data')

for ax in fig.axes:
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([-0.05, 1.0])
    ax.set_ylim([0.0, 1.05])
```




```python
#SVM with Multiple Classes
np.random.seed(8)
XX = np.vstack([X, np.random.randn(50,2)])
yy = np.hstack([y, np.repeat(0,50)])
XX[yy ==0] = XX[yy == 0] +4

plt.scatter(XX[:,0], XX[:,1], s=70, c=yy, cmap=plt.cm.prism)
plt.xlabel('XX1')
plt.ylabel('XX2')
```

```
Text(0, 0.5, 'XX2')
```




**Conclusion:**

1. Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.
2. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
3. Successfully performed SVM using python
4. Understood Support Vector Machine algorithm through building SVM algorithm in Python.