Name : Kshitij V Darwhekar

Roll No: TETB19

Sub: Soft Computitng

Batch: B2

Experiment 4: Machine Learning for Image Classification (Support Vector Machine Tutorial Using Python Sklearn)

```
import pandas as pd
from sklearn.datasets import load_iris
iris = load_iris()
```

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df = pd.DataFrame(iris.data,columns=iris.feature_names)
```

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df['target'] = iris.target
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
df[df.target==1].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

```
df[df.target==2].head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2

```
df['flower_name'] =df.target.apply(lambda x: iris.target_names[x])
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

```
df[45:55]
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
45	4.8	3.0	1.4	0.3	0	setosa
46	5.1	3.8	1.6	0.2	0	setosa
47	4.6	3.2	1.4	0.2	0	setosa
48	5.3	3.7	1.5	0.2	0	setosa
49	5.0	3.3	1.4	0.2	0	setosa
50	7.0	3.2	4.7	1.4	1	versicolor
51	6.4	3.2	4.5	1.5	1	versicolor
52	6.9	3.1	4.9	1.5	1	versicolor
53	5.5	2.3	4.0	1.3	1	versicolor
54	6.5	2.8	4.6	1.5	1	versicolor

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

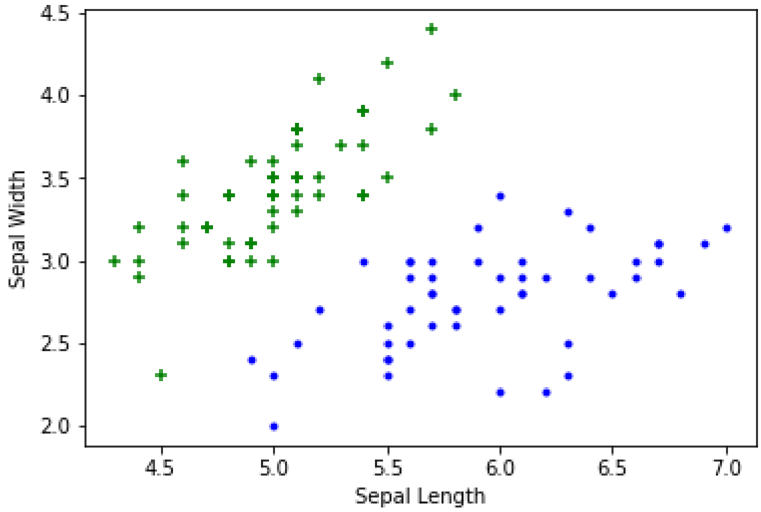
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Sepal length vs Sepal Width (Setosa vs Versicolor)

```
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
```

```
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'],color="green",marker='+')
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'],color="blue",marker='.')
```

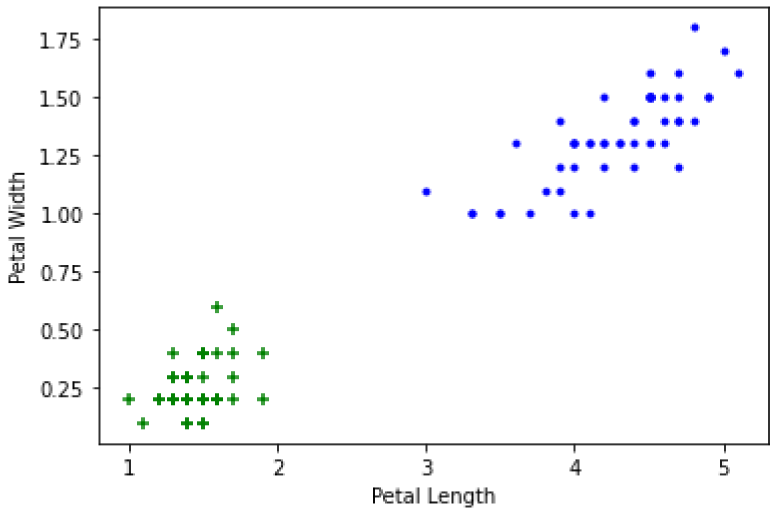
```
<matplotlib.collections.PathCollection at 0x7f14938e3a10>
```



Petal length vs Pepal Width (Setosa vs Versicolor)

```
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.scatter(df0['petal length (cm)'], df0['petal width (cm)'],color="green",marker='+')
plt.scatter(df1['petal length (cm)'], df1['petal width (cm)'],color="blue",marker='.')
```

```
<matplotlib.collections.PathCollection at 0x7f14933d5b10>
```



Train Using Support Vector Machine (SVM)

```
from sklearn.model_selection import train_test_split
```

```
X = df.drop(['target','flower_name'], axis='columns')
y = df.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
len(X_train)
```

```
120
```

```
len(X_test)
```

```
30
```

```
from sklearn.svm import SVC
model = SVC()
```

```
model.fit(X_train, y_train)
```

```
SVC()
```

```
model.score(X_test, y_test)
```

```
0.9666666666666667
```

```
model.predict([[4.8,3.0,1.5,0.3]])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not h
  "X does not have valid feature names, but"
array([0])
```

Tune parameters

1. Regularization (C)

The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example.

```
model_C = SVC(C=1)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

```
0.9666666666666667
```

```
model_C = SVC(C=10)
model_C.fit(X_train, y_train)
model_C.score(X_test, y_test)
```

```
0.9666666666666667
```

2. Gamma

Gamma parameter: gamma determines the distance a single data sample exerts influence. That is, the gamma parameter can be said to adjust the curvature of the decision boundary.

```
model_g = SVC(gamma=10)
model_g.fit(X_train, y_train)
model_g.score(X_test, y_test)
```

```
0.9666666666666667
```

3. Kernel

A kernel is a specialized kind of similarity function. It takes two points as input, and returns their similarity as output, just as a similarity metric does. A mathematical result from linear algebra known as Mercer's theorem has the implication that a broad class of functions (e.g. similarity metrics) may be expressed in terms of a dot product in some (possibly very and even infinitely) high dimensional space. This means that calculations performed on points in high-dimensional spaces may be restated in terms of dot products

```
model_linear_kernal = SVC(kernel='linear')
model_linear_kernal.fit(X_train, y_train)
```

```
SVC(kernel='linear')
```

```
model_linear_kernal.score(X_test, y_test)
```

```
0.9666666666666667
```

Exercise

Train SVM classifier using sklearn digits dataset (i.e. from sklearn.datasets import load_digits) and then,

1. Measure accuracy of your model using different kernels such as rbf and linear.
2. Tune your model further using regularization and gamma parameters and try to come up with highest accurancy score
3. Use 80% of samples as training data size

```
import pandas as pd

import numpy as np

import sklearn

import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_digits

import seaborn as sns


digits=load_digits()

print(digits)
```

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]]), 'target': array([0, 1, 2, ..., 8, 9, 8]
        [ 0.,  0., 13., ..., 15.,  5.,  0.],
        [ 0.,  3., 15., ..., 11.,  8.,  0.],
        ...,
        [ 0.,  4., 11., ..., 12.,  7.,  0.],
        [ 0.,  2., 14., ..., 12.,  0.,  0.],
        [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  0.,  3., ...,  6.,  0.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
        [ 0.,  0.,  3., ..., 14.,  0.,  0.],
        [ 0.,  0.,  8., ..., 16.,  0.,  0.],
        ...,
        [ 0.,  9., 16., ...,  0.,  0.,  0.],
        [ 0.,  3., 13., ..., 11.,  5.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

       ...,

       [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
        [ 0.,  0., 13., ...,  2.,  1.,  0.],
        [ 0.,  0., 16., ..., 16.,  5.,  0.],
        ...,
        [ 0.,  0., 16., ..., 15.,  0.,  0.],
        [ 0.,  0., 15., ..., 16.,  0.,  0.],
        [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

       [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
        [ 0.,  0., 14., ..., 15.,  1.,  0.],
        [ 0.,  4., 16., ..., 16.,  7.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  2.,  0.],
        [ 0.,  0.,  4., ..., 16.,  2.,  0.],
        [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

       [[ 0.,  0., 10., ...,  1.,  0.,  0.],
```

```
        [ 0.,  2., 16., ...,  1.,  0.,  0.],
        [ 0.,  0., 15., ..., 15.,  0.,  0.],
        ...,
        [ 0.,  4., 16., ..., 16.,  6.,  0.],
        [ 0.,  8., 16., ..., 16.,  8.,  0.],
        [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]), 'DESCR': ".. _digits_dataset:\n\nOpti
```

```
digits.keys()
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DES
```

```
df=pd.DataFrame(digits.data)

print(df.head())

print(df.shape)
```

```
     0    1    2     3     4     5    6    7    8    9  ...   54   55   56  \
0  0.0  0.0  5.0  13.0   9.0   1.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
1  0.0  0.0  0.0  12.0  13.0   5.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0
2  0.0  0.0  0.0   4.0  15.0  12.0  0.0  0.0  0.0  0.0  ...  5.0  0.0  0.0
3  0.0  0.0  7.0  15.0  13.0   1.0  0.0  0.0  0.0  8.0  ...  9.0  0.0  0.0
4  0.0  0.0  0.0   1.0  11.0   0.0  0.0  0.0  0.0  0.0  ...  0.0  0.0  0.0

     57   58    59    60    61   62   63
0  0.0  6.0  13.0  10.0   0.0  0.0  0.0
1  0.0  0.0  11.0  16.0  10.0  0.0  0.0
2  0.0  0.0   3.0  11.0  16.0  9.0  0.0
3  0.0  7.0  13.0  13.0   9.0  0.0  0.0
4  0.0  0.0   2.0  16.0   4.0  0.0  0.0

[5 rows x 64 columns]
(1797, 64)
```

```
df.columns
```

```
RangeIndex(start=0, stop=64, step=1)
```

```
df.isnull().sum()
```

```
0     0
1     0
2     0
3     0
4     0
     ..
59    0
60    0
61    0
62    0
63    0
Length: 64, dtype: int64
```

```
df['target']=digits.target
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0

5 rows × 65 columns

```
print(digits.data.shape)
```

```
print(digits.target.shape)
```

```
(1797, 64)
(1797,)
```

```
df.target
```

```
0       0
1       1
2       2
3       3
4       4
       ..
1792    9
1793    0
1794    8
1795    9
1796    8
Name: target, Length: 1797, dtype: int64
```

```
df.values
```

```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  1.],
       [ 0.,  0.,  0., ...,  9.,  0.,  2.],
       ...,
       [ 0.,  0.,  1., ...,  0.,  0.,  8.],
       [ 0.,  0.,  2., ...,  0.,  0.,  9.],
       [ 0.,  0., 10., ...,  1.,  0.,  8.]])
```

```
from sklearn.model_selection import train_test_split
```

```
x=df.drop(['target'],axis='columns')
```

```
y=df.target

x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state=12)



print(len(x_train))

print(len(x_test))
```

```
1437
360
```

```
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC

model1=SVC(kernel='rbf',random_state=0, probability=True)

model1.fit(x_train,y_train)

y_pred_1=model1.predict(x_test)

print("Model Score of Kernal(rbf) :", model1.score(x_test,y_test))
```

```
Model Score of Kernal(rbf) : 0.9916666666666667
```

```
model2=SVC(kernel='linear',random_state=0, probability=True)

model2.fit(x_train,y_train)

y_pred_2=model2.predict(x_test)

print("Model Score of Kernal(linear) :", model2.score(x_test,y_test))
```

```
Model Score of Kernal(linear) : 0.975
```

```
model3=SVC(kernel='poly',random_state=0, probability=True)

model3.fit(x_train,y_train)

y_pred_3=model3.predict(x_test)

print("Model Score of Kernal(poly) :", model3.score(x_test,y_test))
```

```
Model Score of Kernal(poly) : 0.9944444444444445
```

```
accuracy=accuracy_score(y_test,y_pred_3)
```

```
print('ACCURACY is',accuracy)

    ACCURACY is 0.9944444444444445

from sklearn.metrics import confusion_matrix

cm=np.array(confusion_matrix(y_test,y_pred_3))

cm

    array([[37,  0,  0,  0,  0,  0,  0,  0,  0,  0],
           [ 0, 32,  0,  0,  0,  0,  0,  0,  0,  0],
           [ 0,  0, 38,  0,  0,  0,  0,  0,  0,  0],
           [ 0,  0,  0, 43,  0,  0,  0,  0,  0,  0],
           [ 0,  0,  0,  0, 39,  0,  0,  0,  0,  0],
           [ 0,  0,  0,  0,  0, 32,  0,  0,  0,  2],
           [ 0,  0,  0,  0,  0,  0, 29,  0,  0,  0],
           [ 0,  0,  0,  0,  0,  0,  0, 42,  0,  0],
           [ 0,  0,  0,  0,  0,  0,  0,  0, 32,  0],
           [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 34]])

from sklearn.metrics import mean_squared_error

mse=mean_squared_error(y_test,y_pred_3)

mse

    0.08888888888888889

model1_C=SVC(C=3)

model1_C.fit(x_train,y_train)

model1_C.score(x_test,y_test)

    0.9944444444444445

model2_C=SVC(C=3)

model2_C.fit(x_train,y_train)

model2_C.score(x_test,y_test)

    0.9944444444444445

model3_C=SVC(C=3)

model3_C.fit(x_train,y_train)

model3_C.score(x_test,y_test)

    0.9944444444444445
```

```
plt.figure(figsize=(5,5))

sns.heatmap(cm, annot=True, fmt=".2f", linewidths=.5, square = True, cmap = 'Blues_r')

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

A=f'Accuracy Score :{accuracy:.2f}'

plt.title(A)

plt.show()
```