200010021

October 8, 2022

1 LAB 8 : Classification

1. Support Vector Machines
2. K-Nearest Neighbors
3. Classification on MNIST Digit

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

2 Support Vector Machines (SVM)

1. Try to maximize the margin of separation between data.
2. Instead of learning wx+b=0 separating hyperplane directly (like logistic regression), SVM try to learn wx+b=0, such that, the margin between two hyperplanes wx+b=1 and wx+b=-1 (also known as support vectors) is maximum.
3. Margin between wx+b=1 and wx+b=-1 hyperplane is $\frac{2}{||w||}$
4. we have a constraint optimization problem of maximizing $\frac{2}{||w||}$, with constraints wx+b>=1 (for +ve class) and wx+b<=-1 (for -ve class).
5. As $y_i = 1$ for +ve class and $y_i = -1$ for -ve class, the constraint can be re-written as:

$$y(wx + b) >= 1$$

6. Final optimization is (i.e to find w and b):

$$\min_{||w||} \frac{1}{2}||w||,$$

$$y(wx + b) \geq 1, \ \forall \ data$$

Acknowledgement:

https://pythonprogramming.net/predictions-svm-machine-learning-tutorial/

https://medium.com/deep-math-machine-learning-ai/chapter-3-1-svm-from-scratch-in-python-86f93f853dc

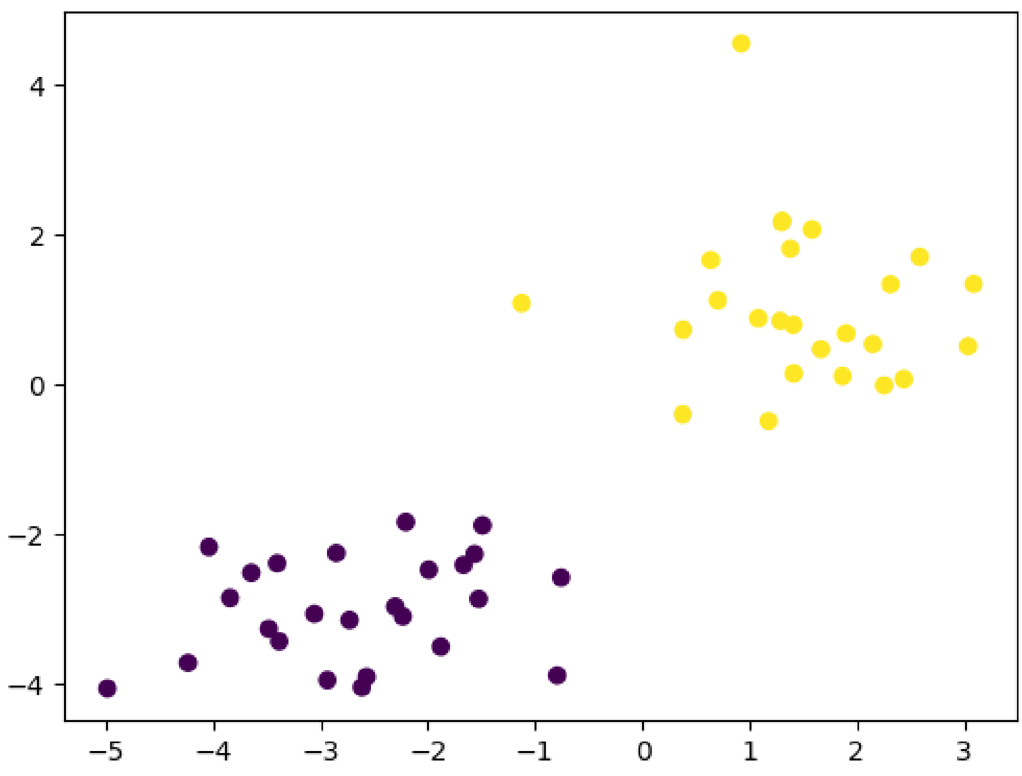
2.1 Data generation:

1. Generate 2D gaussian data with fixed mean and variance for 2 class.(var=Identity, class1: mean[-4,-4], class2: mean[1,1], No. of data 25 from each class)
2. create the label matrix
3. Plot the generated data

[2]:
```
No_sample=50
mean1=np.array([-3,-3])
var1=np.array([[1,0],[0,1]])
mean2=np.array([1,1])
var2=var1
data1=np.random.multivariate_normal(mean1,var1,int(No_sample/2))
data2=np.random.multivariate_normal(mean2,var2,int(No_sample/2))
X=np.concatenate((data1,data2))
print(X.shape)
y=np.concatenate((-1*np.ones(data1.shape[0]),np.ones(data2.shape[0])))
print(y.shape)

plt.figure()
plt.scatter(X[:,0],X[:,1],marker='o',c=y)
```

```
(50, 2)
(50,)
```

[2]:
```
<matplotlib.collections.PathCollection at 0x7f5e76bfba30>
```



Create a data dictionary, which contains both label and data points.

[3]:
```
postiveX=[]
negativeX=[]
for i,v in enumerate(y):
    if v==-1:
        negativeX.append(X[i])
    else:
        postiveX.append(X[i])

#our data dictionary
data_dict = {-1:np.array(negativeX), 1:np.array(postiveX)}
```

2.2 SVM training

1. create a search space for w (i.e w1=w2),[0, 0.5*max((abs(feat)))] and for b, [-max((abs(feat))),max((abs(feat)))], with appropriate step.
2. we will start with a higher step and find optimal w and b, then we will reduce the step and again re-evaluate the optimal one.
3. In each step, we will take transform of w, [1,1], [-1,1],[1,-1] and [-1,-1] to search arround the w.
4. In every pass (for a fixed step size) we will store all the w, b and its corresponding ||w||, which make the data correctly classified as per the condition $y(wx + b) \geq 1$.
5. Obtain the optimal hyperplane having minimum ||w||.
6. Start with the optimal w and repeat the same (step 3,4 and 5) for a reduced step size.

[8]:
```
class SVM:
    def __init__(self, data_dict) -> None:
        self.data_dict = data_dict
        self.w = []
        self.b = []
        self.max_feature_value=float('-inf')
        self.min_feature_value=float('+inf')
        self._min_max_fv(data_dict)
        self.learning_rate = [self.max_feature_value * 0.1, self.
 ↪max_feature_value * 0.01, self.max_feature_value * 0.001,]

    def _min_max_fv(self, data_dict):
        for yi in data_dict:
            if np.amax(data_dict[yi])>self.max_feature_value:
                self.max_feature_value=np.amax(data_dict[yi])

            if np.amin(data_dict[yi])<self.min_feature_value:
```

```
                self.min_feature_value=np.amin(data_dict[yi])

    def train(self, data_dict):
        i=1
        w = []
        b = []

        length_Wvector = {}
        transforms = [[1,1],[-1,1],[-1,-1],[1,-1]]

        b_step_size = 2
        b_multiple = 5
        w_optimum = self.max_feature_value*0.5

        for lrate in self.learning_rate:

            w = np.array([w_optimum,w_optimum])
            optimized = False
            while not optimized:

                for b in np.arange(-1*(self.max_feature_value*b_step_size),
↪self.max_feature_value*b_step_size, lrate*b_multiple):
                    for transformation in transforms:
                        w_t = w*transformation

                        correctly_classified = True


                        for yi in data_dict:
                            for xi in data_dict[yi]:
                                if yi*(np.dot(w_t,xi)+b) < 1:  # we want
↪yi*(np.dot(w_t,xi)+b) >= 1 for correct classification
                                    correctly_classified = False

                        if correctly_classified:
                            length_Wvector[np.linalg.norm(w_t)] = [w_t,b]
↪#store w, b for minimum magnitude

                if w[0] < 0:
                    optimized = True
                else:
                    w = w - lrate

            norms = sorted([n for n in length_Wvector])

            minimum_wlength = length_Wvector[norms[0]]
            w = minimum_wlength[0]
```

```
            b = minimum_wlength[1]

            w_optimum = w[0]+lrate*2

        self.w = w
        self.b = b

        return w,b
```

Training

[13]:
```
# All the required variables
w=[] # Weights 2 dimensional vector
b=[] # Bias
SVC = SVM(data_dict)
w,b = SVC.train(data_dict)
print(w)
print(b)
```

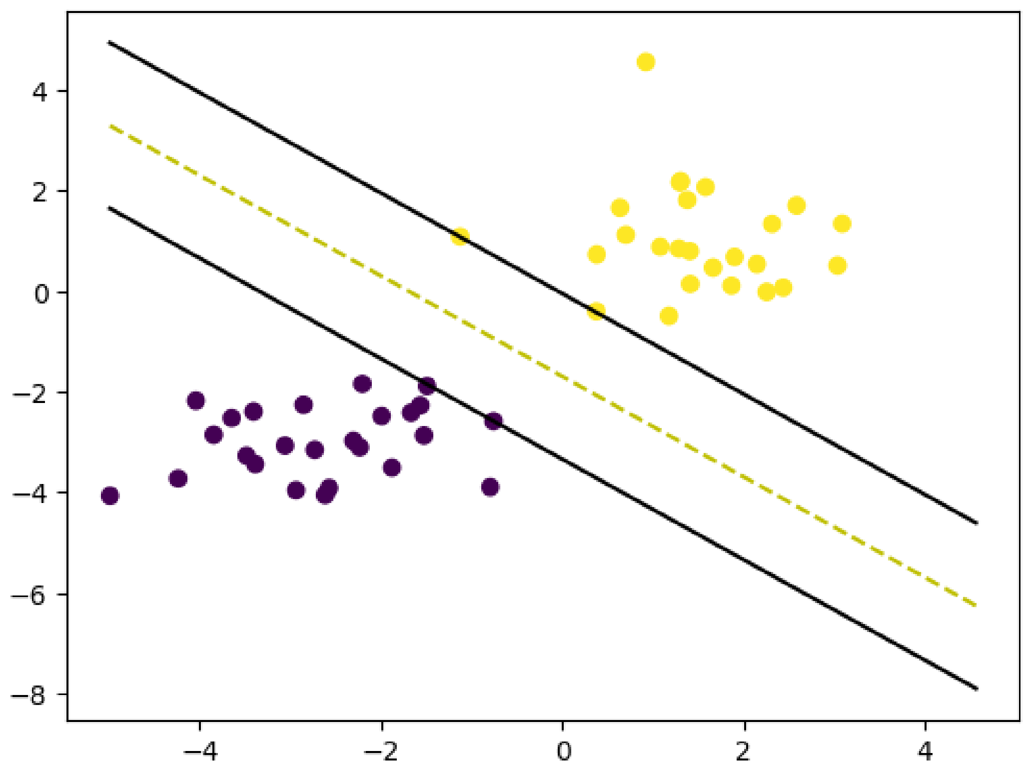
```
[0.60693615 0.60693615]
1.02677167662862
```

2.3 Visualization of the SVM separating hyperplanes (after training)

[11]:
```
def visualize(data_dict):



        plt.scatter(X[:,0],X[:,1],marker='o',c=y)

        # hyperplane = x.w+b
        # v = x.w+b
        # psv = 1
        # nsv = -1
        # dec = 0
        def hyperplane_value(x,w,b,v):
            return (-w[0]*x-b+v) / w[1]


        hyp_x_min = np.min([np.min(data_dict[1]),np.min(data_dict[-1])])
        hyp_x_max = np.max([np.max(data_dict[1]),np.max(data_dict[-1])])

        # (w.x+b) = 1
        # positive support vector hyperplane
        psv1 = hyperplane_value(hyp_x_min, w, b, 1)
        psv2 = hyperplane_value(hyp_x_max, w, b, 1)
        plt.plot([hyp_x_min,hyp_x_max],[psv1,psv2], 'k')
```

```
        # (w.x+b) = -1
        # negative support vector hyperplane
        nsv1 = hyperplane_value(hyp_x_min, w, b, -1)
        nsv2 = hyperplane_value(hyp_x_max, w, b, -1)
        plt.plot([hyp_x_min,hyp_x_max],[nsv1,nsv2], 'k')

        # (w.x+b) = 0
        # positive support vector hyperplane
        db1 = hyperplane_value(hyp_x_min, w, b, 0)
        db2 = hyperplane_value(hyp_x_max, w, b, 0)
        plt.plot([hyp_x_min,hyp_x_max],[db1,db2], 'y--')
```

[12]:
```
fig = plt.figure()
visualize(data_dict)
```



Testing

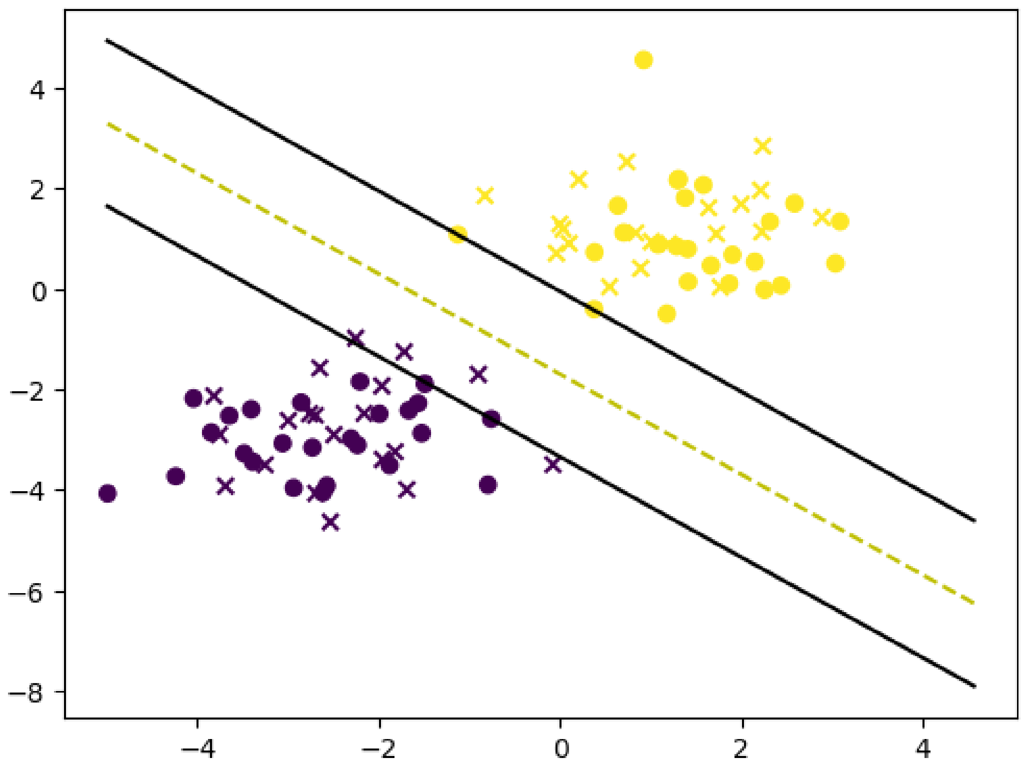
[14]:
```
def predict(data,w,b):
  y_pred=np.sign(np.dot(data,w)+b)
```

```python
    return y_pred
```

[15]:
```python
No_test_sample=40
data1=np.random.multivariate_normal(mean1,var1,int(No_test_sample/2))
data2=np.random.multivariate_normal(mean2,var2,int(No_test_sample/2))
test_data=np.concatenate((data1,data2))
y_gr=np.concatenate((-1*np.ones(data1.shape[0]),np.ones(data2.shape[0])))

# evaluate with the trained model

y_pred=predict(test_data,w,b)
accuracy=(1-(np.abs(0.5*np.sum(y_pred-y_gr))/y_pred.shape[0]))*100
print('test accuracy=',accuracy)

#  Visualization
plt.figure()
visualize(data_dict)
plt.scatter(test_data[:,0],test_data[:,1],marker='x',c=y_gr)
```

```
test accuracy= 100.0
```

[15]:
```
<matplotlib.collections.PathCollection at 0x7f5e75edbd90>
```



Use the Sci-kit Learn Package and perform Classification on the above dataset using the SVM algorithm

[16]:
```
from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X,y)
tr_Acc = svm.score(X,y)
print('Train accuracy SVM =',tr_Acc*100)
```

```
Train accuracy SVM = 100.0
```

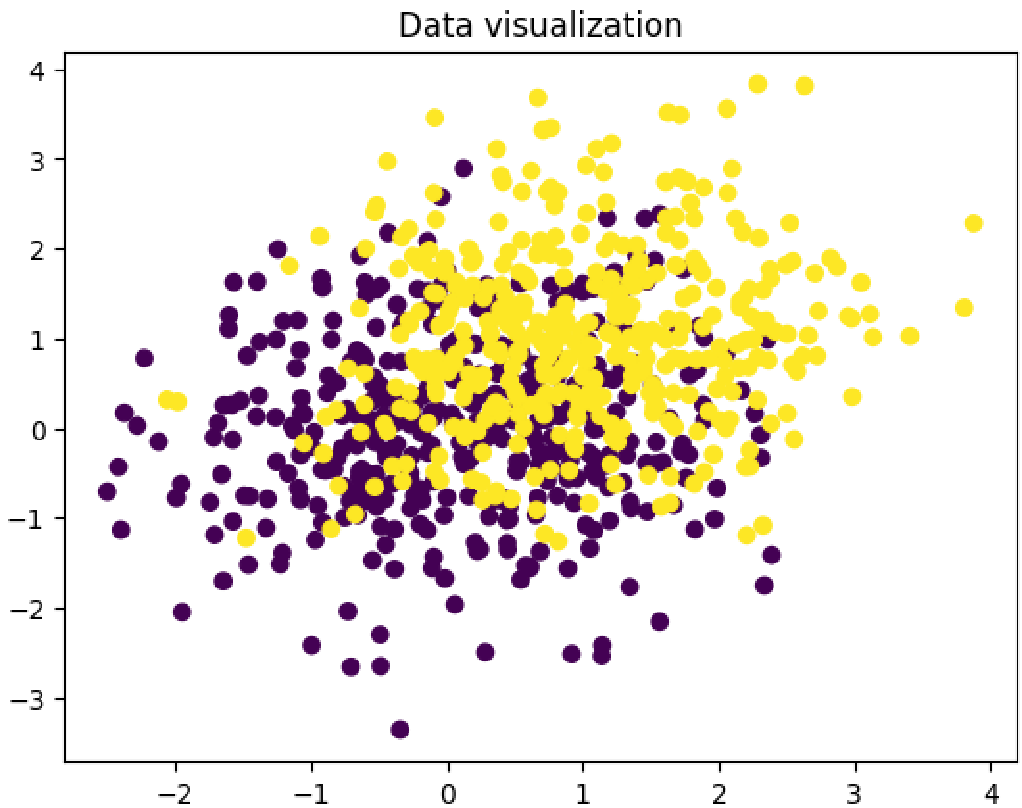
[14]:
```
# svm testing
from sklearn.metrics import confusion_matrix as conf_mat
y_pred=svm.predict(test_data)
svm_Acc=svm.score(test_data,y_gr)
print('Test accuracy SVM=',svm_Acc*100)
print('Confusion matrix=\n',conf_mat(y_gr,y_pred))
```

```
Test accuracy SVM= 100.0
Confusion matrix=
 [[20  0]
 [ 0 20]]
```

3 K-Nearest Neighbours (KNN)

[17]:
```
import numpy as np
import matplotlib.pyplot as plt

mean1=np.array([0,0])
mean2=np.array([1,1])
var=np.array([[1,0.1],[0.1,1]])
np.random.seed(0)
data1=np.random.multivariate_normal(mean1,var,500)
data2=np.random.multivariate_normal(mean2,var,500)
data_train=np.concatenate((data1[:-100,],data2[:-100]))
label=np.concatenate((np.zeros(data1.shape[0]-100),np.ones(data2.shape[0]-100)))

plt.figure()
plt.scatter(data_train[:,0],data_train[:,1],c=label)
plt.title('Data visualization')
```

[17]:
```
Text(0.5, 1.0, 'Data visualization')
```



[18]:
```
def euclidean_distance(row1, row2):
  return np.linalg.norm(row1-row2)
```

[22]:
```
def get_neighbors(train,label_train, test_row, num_neighbors):
  distances = list()
  for i in range(train.shape[0]):
    train_row=train[i,:]
    label_row=label_train[i]
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist,label_row))
  distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i])
  return neighbors
```

[23]:
```
def predict_classification(neigbors):
  pred=list()
  for i in range(len(neigbors)):
    pred.append(neigbors[i][2])
  prediction = max(set(pred), key=pred.count)
```

```
  return prediction
```

[24]:
```
data_test=np.concatenate((data1[-100:],data2[-100:]))
label_test=np.concatenate((np.zeros(100),np.ones(100)))
```

[25]:
```
K=2

pred_label=np.zeros(data_test.shape[0])
for i in range(data_test.shape[0]):
  neig=get_neighbors(data_train,label, data_test[i,:], K)
  pred_label[i]=predict_classification(neig)

accuracy=(len(np.where(pred_label==label_test)[0])/len(label_test))*100
print('Testing Accuracy=',accuracy,'%')
```

```
Testing Accuracy= 65.5 %
```

Use the Sci-kit Learn Package and perform Classification on the above dataset using the K-Nearest Neighbour algorithm

[26]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=2)
model.fit(data_train,label)
pred_label = model.predict(data_test)

accuracy=(len(np.where(pred_label==label_test)[0])/len(label_test))*100
print('Testing Accuracy=',accuracy,'%')
```

```
Testing Accuracy= 65.5 %
```

4 Classification on MNIST Digit Data

1. Read MNIST data and perform train-test split
2. Select any 2 Classes and perform classification task using SVM, KNN and Logistic Regression algorithms with the help of Sci-Kit Learn tool
3. Report the train and test accuracy and also display the results using confusion matrix
4. Repeat steps 2 and 3 for all 10 Classes and tabulate the results

[23]:
```
%pip install idx2numpy
```

```
Collecting idx2numpy
  Downloading idx2numpy-1.2.3.tar.gz (6.8 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from idx2numpy) (1.19.5)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from idx2numpy) (1.15.0)
Building wheels for collected packages: idx2numpy
  Building wheel for idx2numpy (setup.py) … done
  Created wheel for idx2numpy: filename=idx2numpy-1.2.3-py3-none-any.whl
```

```
size=7919
sha256=fe3b197928fffeca37b34dc0e7f469cb48191e3629e835dd7e3db4ba3f455ef5
  Stored in directory: /root/.cache/pip/wheels/1a/ce/ad/d5e95a35cfe34149aade5e50
0f2edd535c0566d79e9a8e1d8a
Successfully built idx2numpy
Installing collected packages: idx2numpy
Successfully installed idx2numpy-1.2.3
```

[98]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle


file1='./t10k-images-idx3-ubyte'
file2='./t10k-labels-idx1-ubyte'

import idx2numpy


x_train= idx2numpy.convert_from_file(file1)
y_train= idx2numpy.convert_from_file(file2)
```

[99]:
```python
print(x_train.shape)
print(y_train.shape)
```

```
(10000, 28, 28)
(10000,)
```

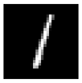
[100]:
```python
indx1 = np.where(y_train == 1)[0]

indx4 = np.where(y_train == 4)[0]
```

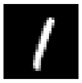
[101]:
```python
x1 = x_train[indx1]
x4 = x_train[indx4]

y1 = y_train[indx1]
y2 = y_train[indx4]
```

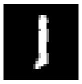
[102]:
```python
X = []
for x in x1:
    X.append([x.flatten()])
for x in x4:
    X.append([x.flatten()])
X = np.concatenate(X)
print(X.shape)

Y = np.concatenate((y1,y2))
print(Y.shape)
```

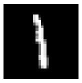
```
(2117, 784)
(2117,)
```

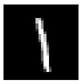
[103]:
```python
from PIL import Image
for im in x1[:5]:
    img = Image.fromarray(im)
    img.show()
for im in x4[:5]:
    img = Image.fromarray(im)
    img.show()
```

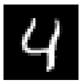


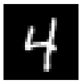


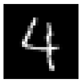












[104]:
```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
```

[105]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

X_tr, X_tst, Y_tr, Y_tst = train_test_split(X,Y)
```

5 SVM

[106]:
```
clf = make_pipeline(StandardScaler(), SVC())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 1.0
Accuracy of this model is 99.62264150943396%
```

[106]:
```
array([[280,   2],
       [  0, 248]])
```

6 Logistic

[107]:
```
clf = make_pipeline(StandardScaler(), LogisticRegression())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 1.0
Accuracy of this model is 99.81132075471699%
```

[107]:
```
array([[281,   1],
       [  0, 248]])
```

7 Knn

[108]:
```
clf = make_pipeline(StandardScaler(), KNeighborsClassifier())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 0.9899180844360429
Accuracy of this model is 99.43396226415095%
```

[108]:
```
array([[282,   0],
       [  3, 245]])
```

8 All 10 classes

8.1 Test train split

[112]:
```
file1='./t10k-images-idx3-ubyte'
file2='./t10k-labels-idx1-ubyte'

import idx2numpy


x_train= idx2numpy.convert_from_file(file1)
y_train= idx2numpy.convert_from_file(file2)

X = []
Y = []

for x in x_train:
```

```
    X.append([x.flatten()])

X = np.concatenate(X)
print(X.shape)

Y = y_train
print(Y.shape)

X_tr, X_tst, Y_tr, Y_tst = train_test_split(X,Y)
```

```
(10000, 784)
(10000,)
```

9 SVM

[113]:
```
clf = make_pipeline(StandardScaler(), SVC())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 0.9846666666666667
Accuracy of this model is 93.32000000000001%
```

[113]:
```
array([[207,   0,   6,   1,   0,   0,   0,   0,   0,   0],
       [  0, 258,   1,   0,   0,   1,   1,   0,   0,   0],
       [  2,   0, 242,   0,   1,   0,   1,   5,   4,   0],
       [  0,   0,   9, 232,   1,   3,   1,   5,   3,   0],
       [  0,   0,   5,   0, 232,   0,   1,   0,   1,   3],
       [  1,   1,   5,   4,   0, 208,   8,   2,   1,   1],
       [  2,   1,  10,   0,   1,   0, 231,   0,   1,   0],
       [  1,   3,   8,   2,   3,   0,   0, 252,   0,   6],
       [  2,   1,   5,   1,   2,  10,   2,   0, 226,   0],
       [  4,   1,   5,   1,   6,   2,   0,   9,   0, 245]])
```

10 Logistic

[114]:
```
clf = make_pipeline(StandardScaler(), LogisticRegression())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 0.9998666666666667
Accuracy of this model is 88.44%
```

```
/home/abhishekj/.local/lib/python3.9/site-
packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[114]:

```
array([[205,   0,   3,   1,   1,   2,   2,   0,   0,   0],
       [  0, 254,   1,   0,   0,   3,   0,   1,   2,   0],
       [  3,   5, 211,  11,   1,   1,   3,   8,  10,   2],
       [  0,   0,   8, 218,   2,  15,   0,   3,   8,   0],
       [  0,   0,   6,   0, 217,   1,   1,   5,   2,  10],
       [  1,   2,   3,   9,   3, 188,   7,   7,   9,   2],
       [  3,   1,   5,   1,   2,   4, 229,   0,   1,   0],
       [  1,   2,   4,   2,   8,   0,   1, 245,   2,  10],
       [  4,   3,   4,   4,   4,  11,   4,   1, 209,   5],
       [  4,   1,   2,   4,  14,   3,   0,  10,   0, 235]])
```

11 KNN

[115]:

```
clf = make_pipeline(StandardScaler(), KNeighborsClassifier())
clf.fit(X=X_tr,y=Y_tr)
print(f"Test accuracy is {clf.score(X_tr,Y_tr)}")
y_pred = clf.predict(X_tst)
print(f"Accuracy of this model is {accuracy_score(Y_tst,y_pred)*100}%")
confusion_matrix(Y_tst,y_pred)
```

```
Test accuracy is 0.9410666666666667
Accuracy of this model is 91.36%
```

[115]:

```
array([[211,   1,   1,   0,   0,   0,   0,   1,   0,   0],
       [  0, 259,   1,   0,   0,   0,   1,   0,   0,   0],
       [  8,   4, 224,   9,   0,   0,   1,   5,   3,   1],
       [  0,   0,   1, 236,   3,   5,   0,   4,   3,   2],
       [  0,   4,   4,   0, 227,   0,   0,   1,   0,   6],
       [  5,   2,   1,   9,   2, 200,   7,   2,   1,   2],
       [  7,   1,   2,   0,   0,   0, 235,   0,   1,   0],
       [  0,   9,   2,   1,   3,   0,   0, 246,   0,  14],
       [  7,   3,   4,   8,   3,  17,   0,   3, 204,   0],
       [  3,   2,   0,   2,   9,   3,   0,  12,   0, 242]])
```