LAB 8 : Classification

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

In [30]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

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

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

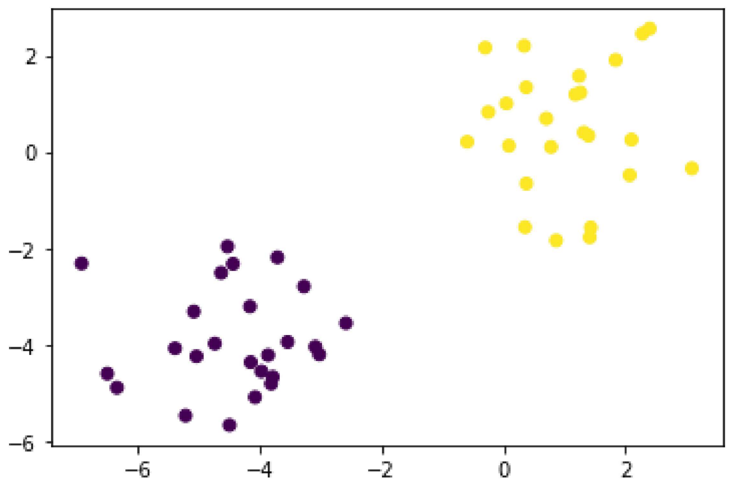
Data generation:

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

In [31]:
```
No_sample=50
mean1=np.array([-4,-4])
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,)
```

Out[31]:
```
<matplotlib.collections.PathCollection at 0x7f6194d57750>
```



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

In [32]:
```
postiveX = []
negativeX = []

for i,j in enumerate(y):
    if j>0:
        postiveX.append(X[i])
    else:
        negativeX.append(X[i])



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

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.

In [33]:
```
# it is just a searching algorithem, not a complicated optimization algorithm, (just for understanding of con
cepts through visualization)

def SVM_Training(data_dict):



    # { ||w||: [w,b] } is dictionary contains norms of w and corresponding w and b value,
    # where all the data points are correctly classified
    norm_w_b = {}
    transforms = [[1,1],[-1,1],[-1,-1],[1,-1]]

    max_feature_value=np.max([np.max(np.abs(data_dict[1])),np.max(np.abs(data_dict[-1]))])
    steps = [max_feature_value * 0.1, max_feature_value * 0.01, max_feature_value * 0.001]

    b_step_size = 2
    b_multiple = 5
    w_optimum = max_feature_value*0.5

    for step in steps:

        w = np.array([w_optimum,w_optimum])
        flag = True
        while flag:
            #b=[-maxvalue to maxvalue] we wanna maximize the b values so check for every b value
            for b in np.arange(-1*(max_feature_value*b_step_size), max_feature_value*b_step_size, step*b_mult
iple):
                for transformation in transforms:  # transforms = [[1,1],[-1,1],[-1,-1],[1,-1]]
                    w_t = w*transformation

                    correctly_classified = True

                    # every data point should be correct
                    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:
                        norm_w_b[np.linalg.norm(w_t)] = [w_t,b] #store w, b for minimum magnitude
```

```
            if w[0] < 0:
                flag = False
            else:
                w = w - step

        norms = sorted([n for n in norm_w_b]) # sort the heated norms

        minimum_wlength = norm_w_b[norms[0]]
        w = minimum_wlength[0]
        b = minimum_wlength[1]

        w_optimum = w[0] # w1 and w2 are same

    return w,b
```

Training

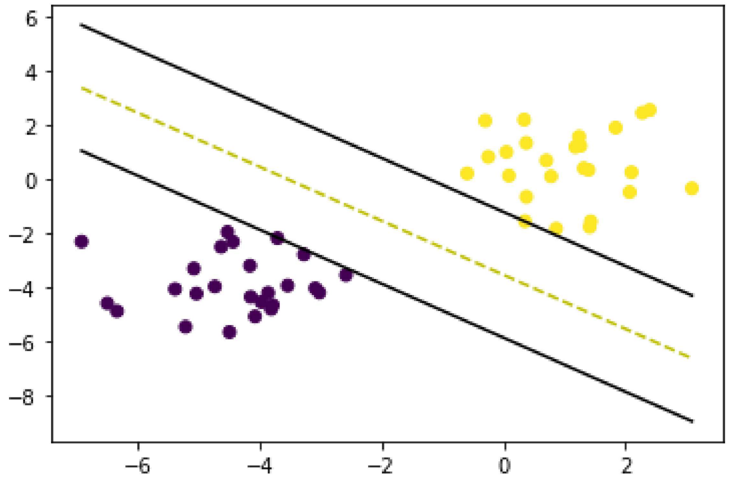
In [34]:
```
# All the required variables
w=[] # Weights 2 dimensional vector
b=[] # Bias
w,b=SVM_Training(data_dict)
print(w)
print(b)
```

```
[0.42929749 0.42929749]
1.5233136912376946
```

Visualization of the SVM separating hyperplanes (after training)

In [35]:
```
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--')
```

In [36]:
```
fig = plt.figure()
visualize(data_dict)
```



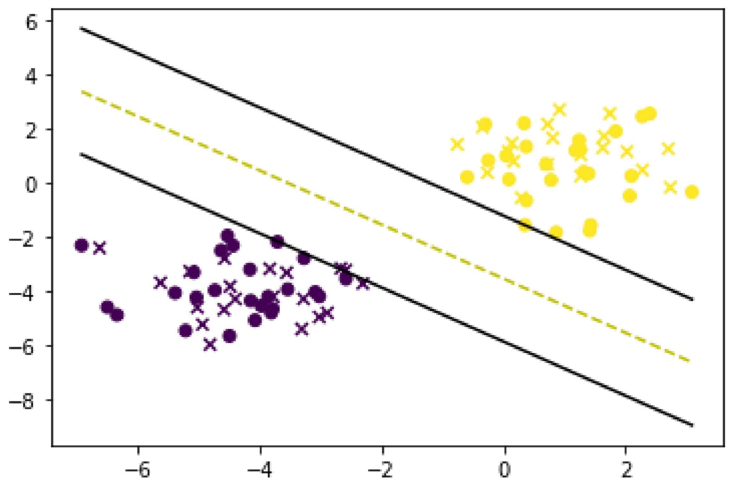
Testing

In [37]:
```
def predict(data,w,b):
  y_pred = y_pred=np.sign(np.dot(data,w)+b)
  return y_pred
```

In [38]:
```
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 = 100*(sum(y_pred==y_gr)/len(y_gr))
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
```

Out[38]:
```
<matplotlib.collections.PathCollection at 0x7f6194d57090>
```



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

In [39]:
```
from sklearn.svm import LinearSVC
svm = LinearSVC()
svm.fit(X,y)

print('Train accuracy SVM =',svm.score(X,y)*100)
```

```
Train accuracy SVM = 100.0
```

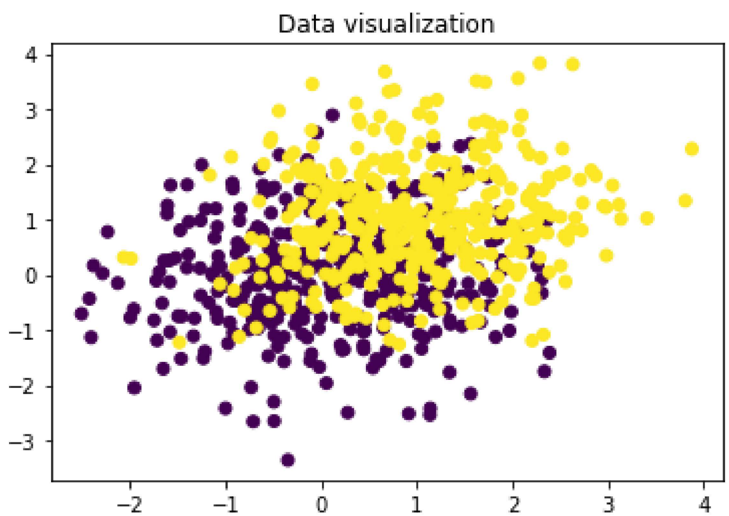
In [40]:
```
# svm testing
from sklearn.metrics import confusion_matrix
y_pred = svm.predict(test_data)

print('Test accuracy SVM=',svm.score(test_data,y_gr)*100)
print('Confusion matrix=\n',confusion_matrix(y_gr,y_pred))
```

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

K-Nearest Neighbours (KNN)

In [41]:
```
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')
```

Out[41]: Text(0.5, 1.0, 'Data visualization')



In [42]:
```
def euclidean_distance(row1, row2):
  return np.linalg.norm(row1-row2)
```

In [43]:
```
def get_neighbors(train,label_train, test_row, num_neighbors):
  distances = []
  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 tuple: tuple[1])
  neighbors = []
  for i in range(num_neighbors):
    neighbors.append(distances[i])
  return neighbors
```

In [44]:
```
def predict_classification(neigbors):
  pred = []
  for i in range(len(neigbors)):
    pred.append(neigbors[i][2])
  prediction = max(set(pred), key=pred.count)

  return prediction
```

In [45]:
```
# test data generation
data_test=np.concatenate((data1[-100:],data2[-100:]))
label_test=np.concatenate((np.zeros(100),np.ones(100)))
```

In [46]:
```
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

In [47]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=15)
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= 74.0 %
```

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

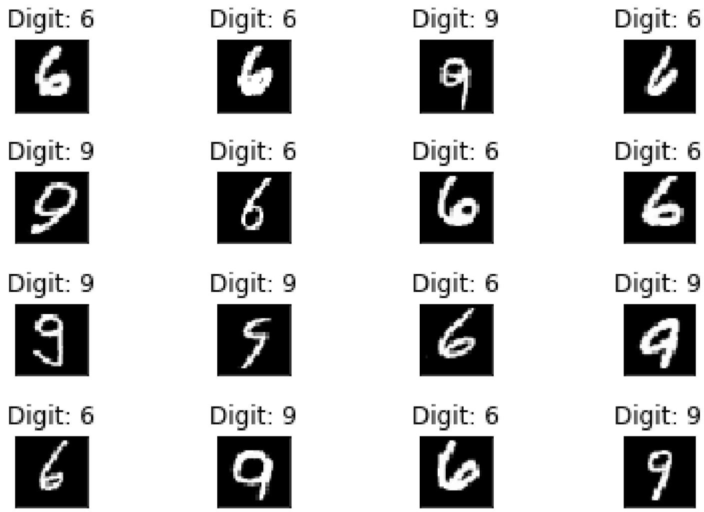
In [48]:
```
import numpy as np
import matplotlib.pyplot as plt
import keras
import tensorflow as tf

tf.keras.datasets.mnist.load_data(path="mnist.npz")
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

Note : If you are interested, also try classifying MNIST digit data using the code you have written for SVM, KNN and Logistic Regression

In [49]:
```
import numpy as np
import matplotlib.pyplot as plt
import keras
import tensorflow as tf

tf.keras.datasets.mnist.load_data(path="mnist.npz")
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

In [50]:
```
from sklearn.utils import shuffle


# input image dimensions
img_rows, img_cols = 28, 28

cl1, cl2 = 6, 9 #choose two class you want to evaluate
```

In [51]:
```
#############################################################################################
##33
i, = np.where(y_train == cl1)   #used to separate index information of class 1
j, = np.where(y_train == cl2)   #used to separate index information of class 2

cl1_train=x_train[i,:,:]        #pooled out the data corresponds to class1
cl1_label=y_train[i]            #pooled out the data labels corresponds to class1

cl2_train=x_train[j,:,:]        #pooled out the data corresponds to class2
cl2_label=y_train[j]            #pooled out the data labels corresponds to class2

train_com = np.concatenate((cl1_train,cl2_train),axis=0)  #Merge the class1 and class2 data
train_lab=np.concatenate((cl1_label,cl2_label),axis=0)   #Merge the labels of class1 and class2

[train_sff,train_labs]=shuffle(train_com,train_lab)     # Shuffle the data and label (to properly train the n
etwork)
#%%
############################################# Plot to shuffled data ###########################
fig = plt.figure()
for i in range(16):
  plt.subplot(4,4,i+1)
  plt.tight_layout()
  plt.imshow(train_sff[i], cmap='gray', interpolation='none')
  plt.title("Digit: {}".format(train_labs[i]))
  plt.xticks([])
  plt.yticks([])
fig
#%%
########################################### Change the labels to 0 and 1 ( as dealing with 2 class), for easy
conversion of categorial ###############################
np.place(train_labs, train_labs==cl1, [0])
np.place(train_labs, train_labs==cl2, [1])

#############################################################################################
#train_labs_cat = keras.utils.to_categorical(train_labs, 2)          # make the output label categorical
#
train_sff = train_sff.astype('float32')

train_sff /= 255

ftrain_sff=train_sff.reshape(train_labs.shape[0],img_rows*img_cols)  # flattern the input data
```


Digit: 6
Digit: 6
Digit: 9
Digit: 6
Digit: 9
Digit: 6
Digit: 6
Digit: 6
Digit: 9
Digit: 9
Digit: 6
Digit: 9
Digit: 6
Digit: 9
Digit: 6
Digit: 9

In [52]:
```
i, = np.where(y_test == cl1)
j, = np.where(y_test == cl2)
cl1_test=x_test[i,:,:]
cl1_label=y_test[i]

#cl1_test=x_test[0:3,:,:]
#cl1_label=y_test[0:3]


cl2_test=x_test[j,:,:]
cl2_label=y_test[j]

#cl2_test=x_test[0:3,:,:]
#cl2_label=y_test[0:3]

test_com = np.concatenate((cl1_test,cl2_test),axis=0)
test_lab=np.concatenate((cl1_label,cl2_label),axis=0)

np.place(test_lab, test_lab==cl1, [0])
np.place(test_lab, test_lab==cl2, [1])


test_com = test_com.astype('float32')

test_com /= 255

ftest_com=test_com.reshape(test_lab.shape[0],img_rows*img_cols)
```

In [53]:
```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix as conf_mat
```

In [54]:
```
################################################################################################################
###
#LR Training
Lreg = LogisticRegression(solver='liblinear')
Lreg.fit(ftrain_sff[0:2000,:],train_labs[0:2000])

LR_tr_Acc=Lreg.score(ftrain_sff[0:2000,:],train_labs[0:2000])

print('Train accuracy Logistic regression=',LR_tr_Acc*100)
```

```
Train accuracy Logistic regression= 100.0
```

In [55]:
```
#LR testing
y_pred=Lreg.predict(ftest_com)
Lreg_Acc=Lreg.score(ftest_com,test_lab)
print('Test accuracy Logistic regression=',Lreg_Acc*100)
print('Confusion matrix=\n',conf_mat(test_lab,y_pred))
```

```
Test accuracy Logistic regression= 99.49161159125572
Confusion matrix=
 [[ 951    7]
 [   3 1006]]
```

In [56]:
```
# svm training
svm = LinearSVC()
svm.fit(ftrain_sff[0:2000,:],train_labs[0:2000])

tr_Acc=svm.score(ftrain_sff[0:2000,:],train_labs[0:2000])
print('Train accuracy SVM=',tr_Acc*100)
```

```
Train accuracy SVM= 100.0
```

In [57]:
```
# svm testing
y_pred=svm.predict(ftest_com)
svm_Acc=svm.score(ftest_com,test_lab)
print('Test accuracy SVM=',svm_Acc*100)
print('Confusion matrix=\n',conf_mat(test_lab,y_pred))
```

```
Test accuracy SVM= 99.44077275038129
Confusion matrix=
 [[ 951    7]
 [   4 1005]]
```