LAB 8 : Classification

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

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from jupyterthemes import jtplot
jtplot.style(theme='gruvboxd',context='notebook',grid=False)
```

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

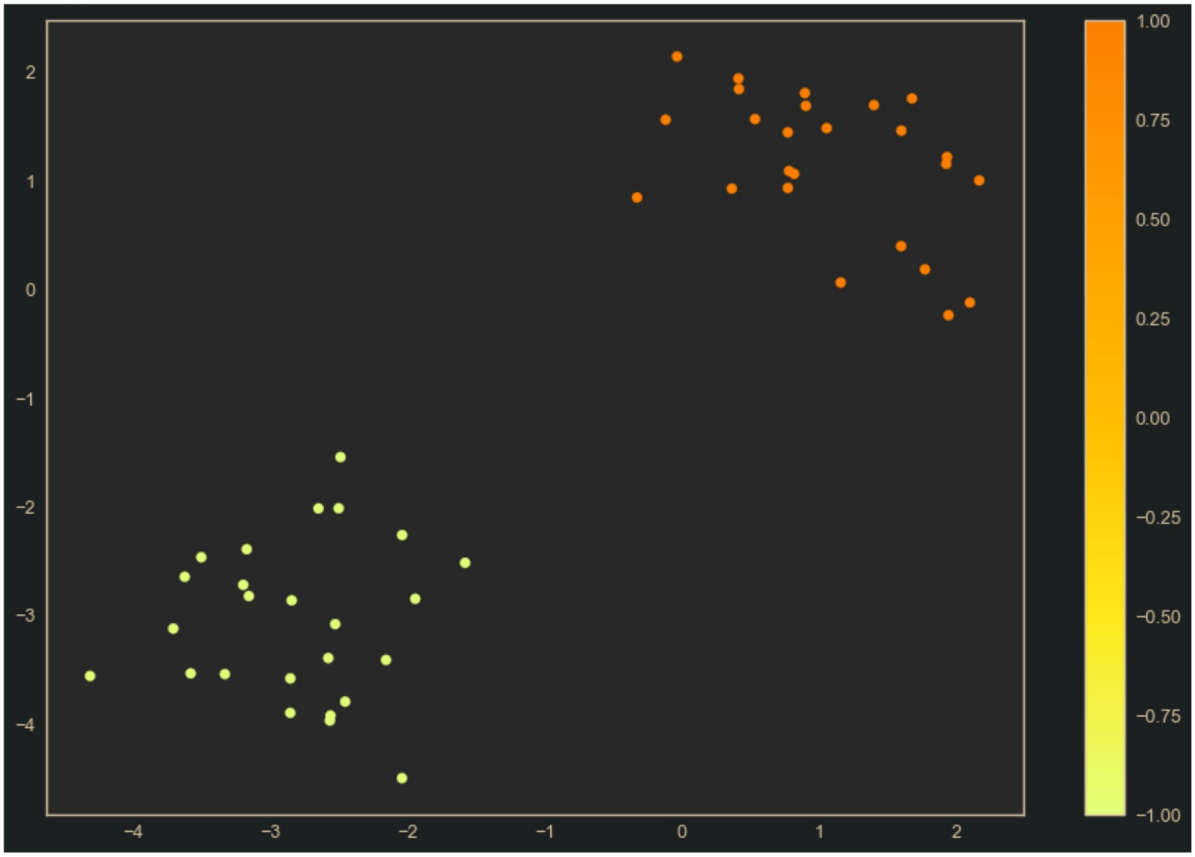
Data generation:

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 [2]:

```
No_sample=50
mean1=np.array([-3,-3])
var1=np.array([[0.5,0],[0,0.5]])
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)
#visualising generated data
plt.figure(figsize=(15,10))
plt.scatter(X[:,0],X[:,1],marker='o',c=y,cmap='Wistia');
plt.colorbar();
```

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



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

In [3]:

```
positiveX=[]
negativeX=[]

## Write your code here
positiveX = X[np.where(y==1)];
negativeX = X[np.where(y==-1)];
# print(positiveX.shape)
# print(negativeX.shape)

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

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 [4]:

```
# it is just a searching algorithem, not a complicated
optimization algorithem, (just for understanding of concepts
through visualization)
# seting max feature to -inf and min feature to +inf
max_feature_value=float('-inf')
min_feature_value=float('+inf')
#looping to get max and  n feature value
for yi in data_dict:
    if np.amax(data_dict[yi])>max_feature_value:
        max_feature_value=np.amax(data_dict[yi])

    if np.amin(data_dict[yi])<min_feature_value:
        min_feature_value=np.amin(data_dict[yi])
# learning rate array
learning_rate = [max_feature_value * 0.1, max_feature_value *
0.01]
# the svm training function
def SVM_Training(data_dict):
    # insert your code here
    length_Wvector = {} # |w|
    transforms = [[1,1],[-1,1],[-1,-1],[1,-1]] # some transforms
to be done to w
    # step size for b
    b_step_size = 2
    b_multiple = 5 # multiplication factor for b
    w_optimum = max_feature_value*0.5 # initial value of w
    #looping over lr array
    for lrate in learning_rate:
        # initialising w
        w = np.array([w_optimum,w_optimum])
        optimized = False #setting optimiser flag as false
        while not optimized: # looping till optimised
            for b in np.arange(-1*
(max_feature_value*b_step_size), max_feature_value*b_step_size,
lrate*b_multiple): #looping over all possible values of b
                for transformation in transforms:  # transforms
= [[1,1],[-1,1],[-1,-1],[1,-1]]
                    w_t = w*transformation # transforming w
                    correctly_classified = True #setting correct
classifier flag as true
                    # every data point should be correct
```

```
                                    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: # set optimised when w is negaitive
                optimized = True
            else: # else update w
                w = w - lrate

        norms = sorted([n for n in length_Wvector]); # array of
norms of
#          print(norms);
        minimum_wlength = length_Wvector[norms[0]];
        w = minimum_wlength[0];
        b = minimum_wlength[1];
    return w,b
```

Training

In [5]:

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

```
[0.47747112 0.47747112]
0.8681293004993362
```

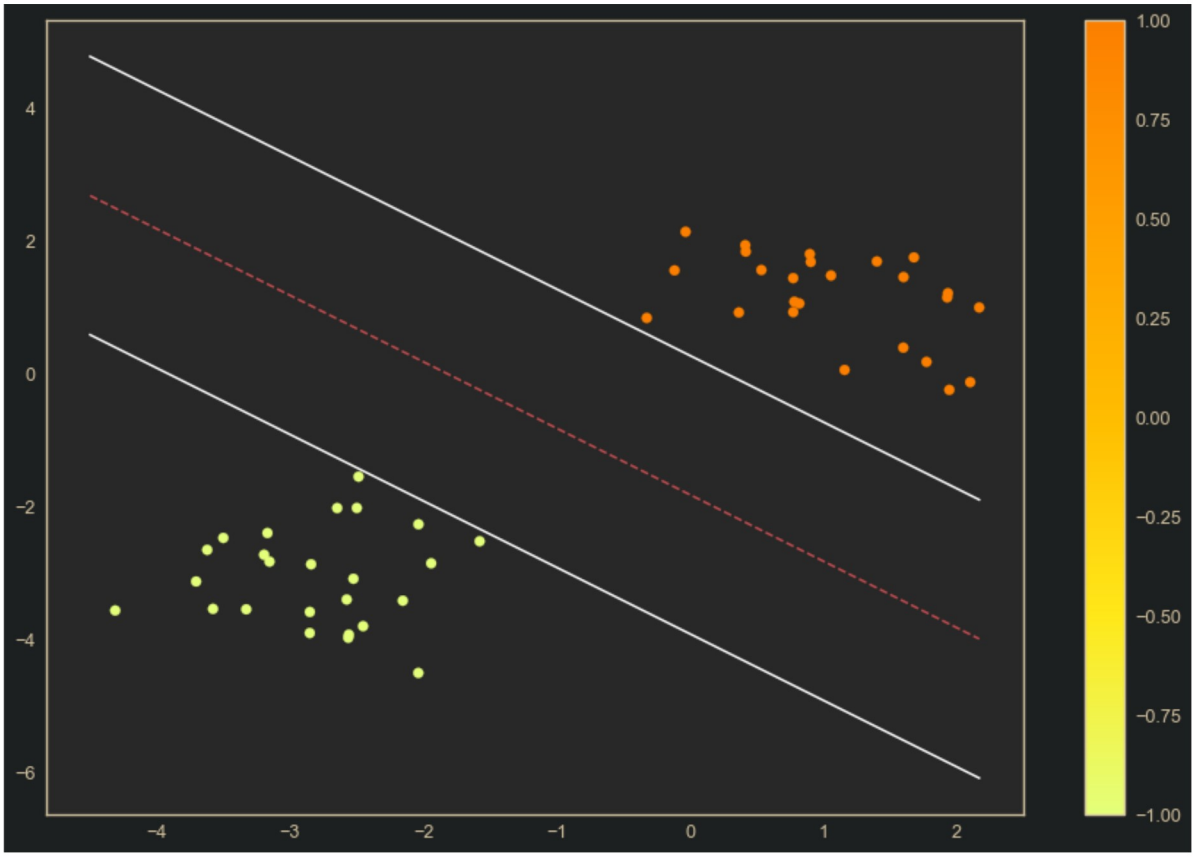
Visualization of the SVM separating hyperplanes (after training)

In [6]:

```
def visualize(data_dict,w,b):

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

        # 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], 'white')

        # (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], 'white')

        # (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], 'r--')
```

In [7]:

```
fig = plt.figure(figsize=(15,10))
visualize(data_dict,w,b)
plt.colorbar();
```



Testing

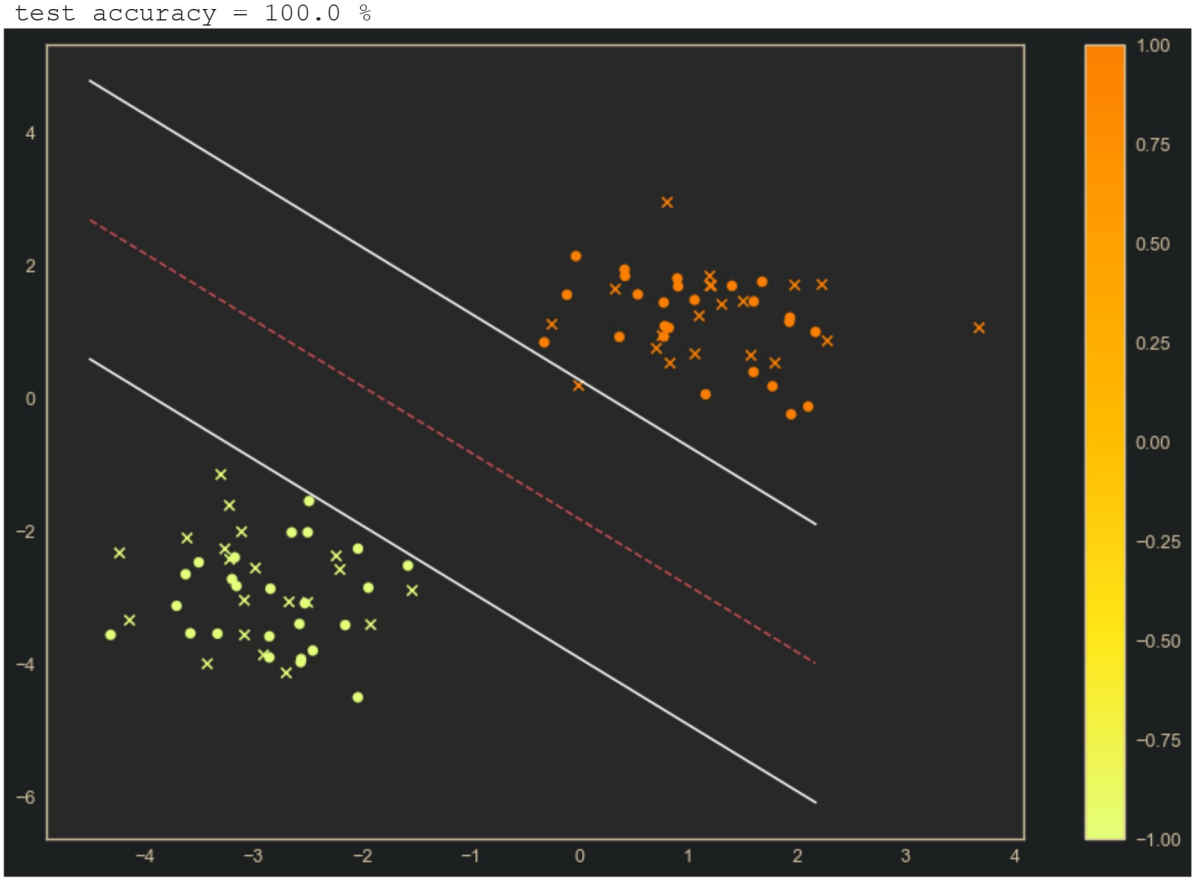
In [8]:

```
def predict(data,w,b):
    # predict 1 if w.x+b>0 and -1 if w.x+b<0
    y_pred = np.array(1*(data@w+b>0)-1*(data@w+b<0))## write
your code here
#     y_pred = np.sign(np.dot(np.array(data),w)+b).astype(int)
    return y_pred
```

In [9]:

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

# evaluate with the trained model
y_pred = predict(test_data,w,b)
accuracy = 100*np.sum(y_pred==y_gr)/len(y_gr);
print('test accuracy =',accuracy,'%');

#  Visualization
plt.figure(figsize=(15,10))
visualize(data_dict,w,b);
plt.scatter(test_data[:,0],test_data[:,1],marker='x',c=y_gr,cmap='
plt.colorbar();
```

```
test accuracy = 100.0 %
```

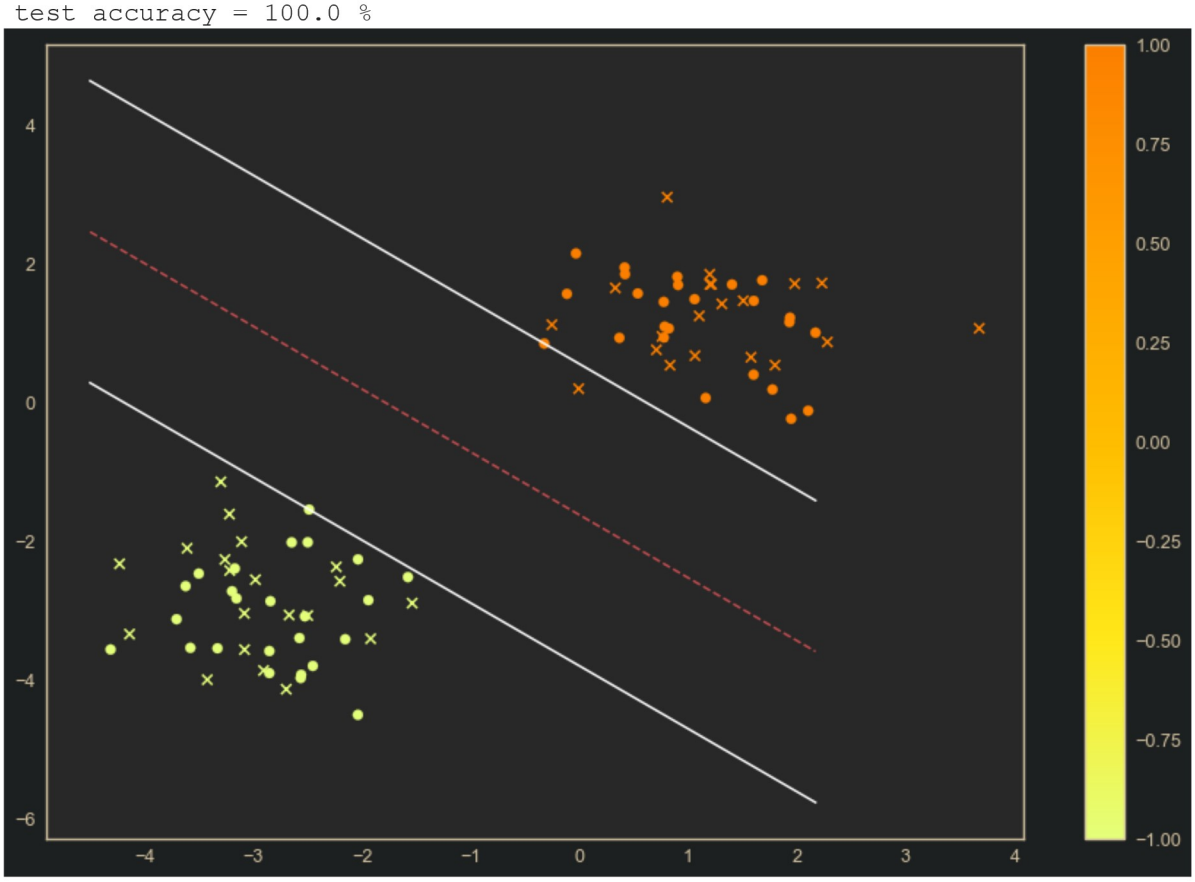


Use the Sci-kit Learn Package and perform Classification on the above dataset using the SVM algorithm

In [10]:

```
## Write your code here
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X, y)
y_pred = svclassifier.predict(test_data)
w_pred = svclassifier.coef_[0];
b_pred = svclassifier.intercept_[0];

# evaluate with the trained model
y_pred = predict(test_data,w,b)
accuracy = 100*np.sum(y_pred==y_gr)/len(y_gr);
print('test accuracy =',accuracy,'%');

#  Visualization
plt.figure(figsize=(15,10))
visualize(data_dict,w_pred,b_pred);
plt.scatter(test_data[:,0],test_data[:,1],marker='x',c=y_pred,cmap
plt.colorbar();
```

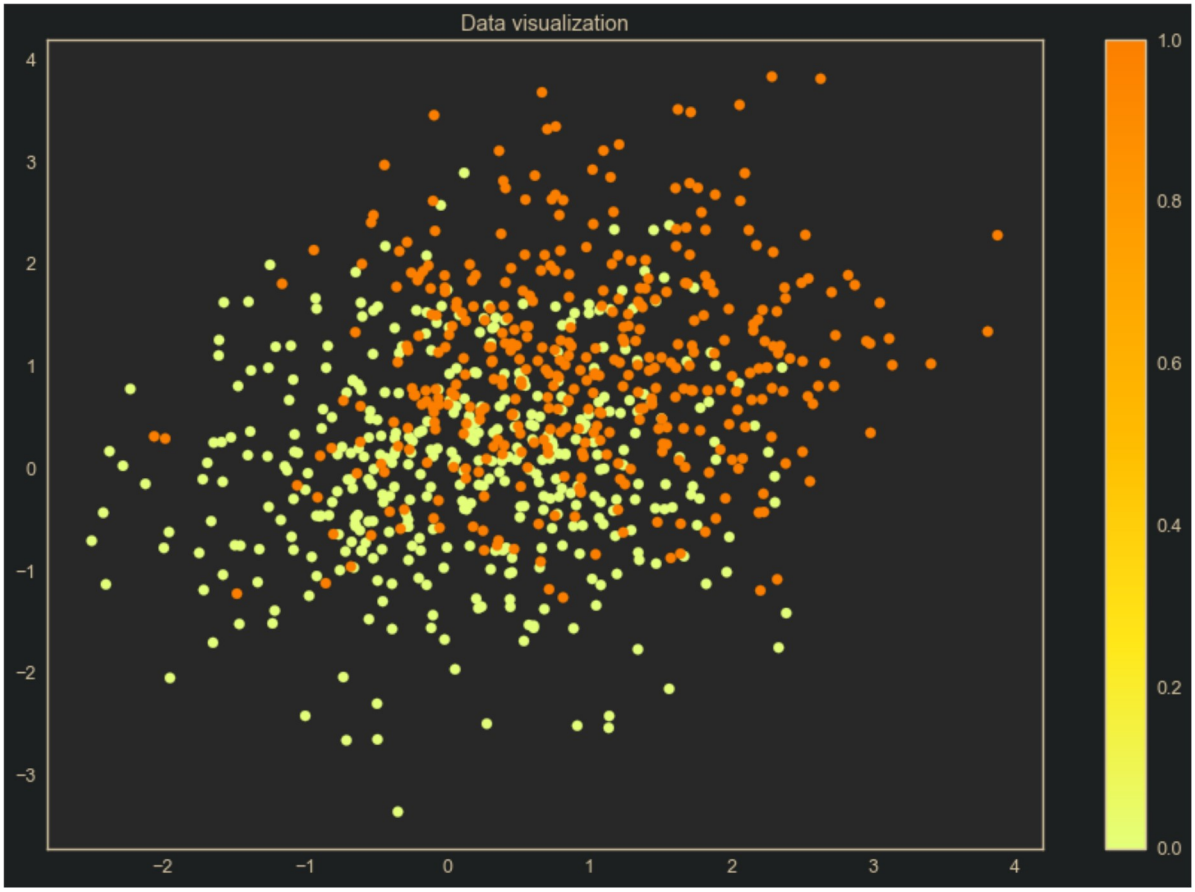
```
test accuracy = 100.0 %
```



K-Nearest Neighbours (KNN)

In [11]:

```
import numpy as np
import matplotlib.pyplot as plt
# generating points
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)))
# print(data_train.shape)
# print(label.shape)

# visualising points
plt.figure(figsize=(15,10))
plt.scatter(data_train[:,0],data_train[:,1],c=label,cmap='Wistia')
plt.title('Data visualization');
plt.colorbar();
```



In [12]:

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

In [13]:

```
def get_neighbors(train,label_train, test_row, num_neighbors):
    ## write your code here
    # finding distances of points from all training points
    distances = []
    for i in range(train.shape[0]):
        distances.append(euclidean_distance(test_row,
train[i,:]))
    #print(distances)

    # storing k nearest neighbors labels
    neighbors = []
    for k in range(num_neighbors):
    # find minimum distance
        index = distances.index(min(distances))
        neighbors.append(label[index])
        distances[index] = max(distances)
    #print(neighbors.shape)

    return neighbors
```

In [14]:

```
def predict_classification(neigbors):
#     write your code here
    # counting labels and prediction label with max frequency
    labels,count = np.unique(neigbors,return_counts=True);
    prediction = labels[np.argmax(count)];
    #print(prediction)
    return prediction
```

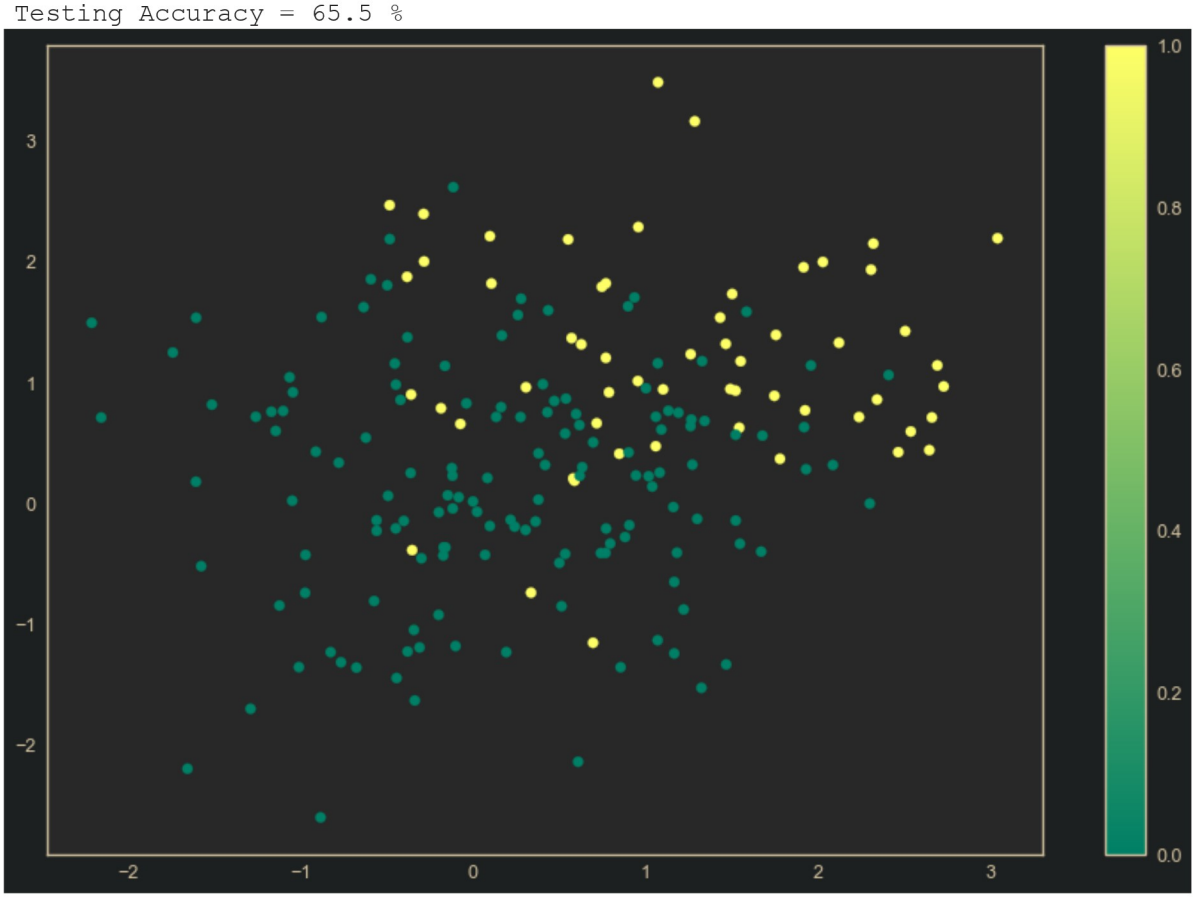
In [15]:

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

In [16]:

```
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;
plt.figure(figsize=(15,10));
plt.scatter(data_test[:,0],data_test[:,1],c=pred_label,cmap='summe
plt.colorbar();

print('Testing Accuracy =',accuracy,'%');
```

```
Testing Accuracy = 65.5 %
```

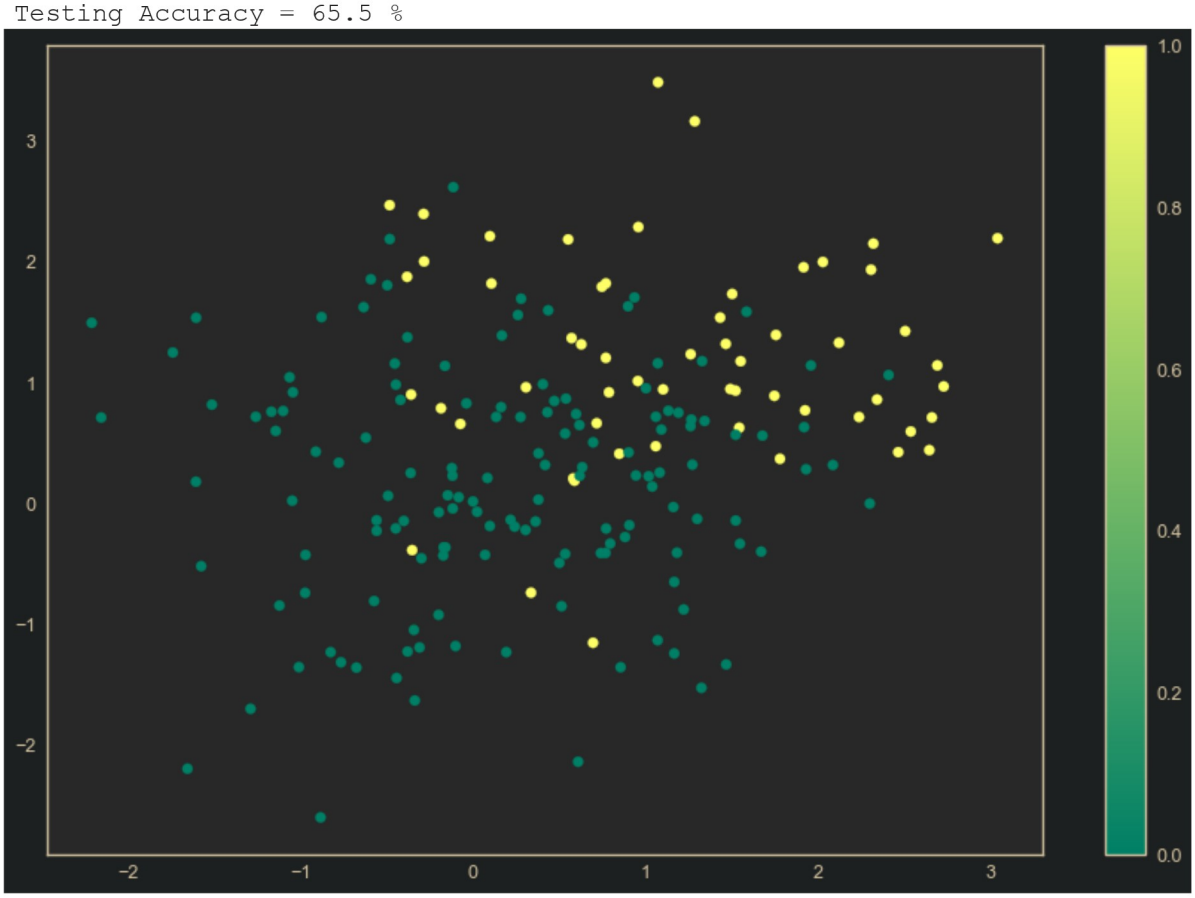


Use the Sci-kit Learn Package and perform Classification on the above dataset using the K-Nearest Neighbour algorithm

In [17]:

```
## Write your code here
from sklearn.neighbors import KNeighborsClassifier as KNN
# training and predicting
model = KNN(n_neighbors=2)
model.fit(data_train,label);
pred= model.predict(data_test);
# accuracy
accuracy=(len(np.where(pred_label==label_test)
[0])/len(label_test))*100;
print('Testing Accuracy =',accuracy,'%');
# plotting
plt.figure(figsize=(15,10));
plt.scatter(data_test[:,0],data_test[:,1],c=pred,cmap='summer');
plt.colorbar();
```

```
Testing Accuracy = 65.5 %
```



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 [18]:

```
## Write your code here
import idx2numpy
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
# paths to files
img_path = 'C:/Users/R3M0/Documents/EE 413 Pattern Recognition
and Machine Learning Laboratory/8/t10k-images-idx3-ubyte'
label_path = 'C:/Users/R3M0/Documents/EE 413 Pattern Recognition
and Machine Learning Laboratory/8/t10k-labels-idx1-ubyte'
# importing data
X= idx2numpy.convert_from_file(img_path)
y= idx2numpy.convert_from_file(label_path)
# print(X.shape)
# print(y.shape)

# reshaping 2d images to 1d features
X = np.reshape(X,(X.shape[0],-1));
# print(X.shape)

# slicing classes
X2 = X[np.where(np.logical_or(y==0,y==1))];
y2 = y[np.where(np.logical_or(y==0,y==1))];

# splitting into train test
data_train, data_test, labels_train, labels_test =
train_test_split(X2, y2, test_size=0.20, random_state=42)

print('\n== Classes 0 and Classes 1 ==\n')
#KNN
print("=====KNN=====")
model = KNN(n_neighbors=10)
model.fit(data_train,labels_train);
pred_tr = model.predict(data_train)
pred= model.predict(data_test)

cm = confusion_matrix(labels_train, pred_tr)
acc = accuracy_score(labels_train, pred_tr)
print('Train Accuracy:',acc*100)
```

```
print('Test Accuracy: ',acc*100)
print('Confusion Matrix:\n',cm)

#SVM
print("\n\n=====SVM=====")
model = SVC().fit(data_train, labels_train)

pred_test = model.predict(data_test)
pred_train = model.predict(data_train)
cm = confusion_matrix(labels_train, pred_train)
acc = accuracy_score(labels_train, pred_train)
print('Train Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

cm = confusion_matrix(labels_test, pred_test)
acc = accuracy_score(labels_test, pred_test)
print('Test Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

# LOGISTIC REGRESSION
print("\n\n=====Logistic Regression=====")
scaler = preprocessing.StandardScaler().fit(data_train)
data_scaled = scaler.transform(data_train)

logisticRegr = LogisticRegression(max_iter=10000)

logisticRegr.fit(data_scaled, labels_train)
pred_test = logisticRegr.predict(data_test)
pred_train = logisticRegr.predict(data_train)
cm = confusion_matrix(labels_train, pred_train)
acc = accuracy_score(labels_train, pred_train)
print('Train Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

cm = confusion_matrix(labels_test, pred_test)
acc = accuracy_score(labels_test, pred_test)
print('Test Accuracy:',acc*100)
```

```
== Classes 0 and Classes 1 ==

=====KNN=====
Train Accuracy: 99.822695035461
Confusion Matrix:
 [[765   3]
 [  0 924]]
Test Accuracy: 100.0
Confusion Matrix:
 [[212   0]
 [  0 211]]


=====SVM=====
Train Accuracy: 100.0
Confusion Matrix:
 [[768   0]
 [  0 924]]
Test Accuracy: 100.0
Confusion Matrix:
 [[212   0]
 [  0 211]]


=====Logistic Regression=====
Train Accuracy: 99.94089834515366
Confusion Matrix:
 [[768   0]
 [  1 923]]
Test Accuracy: 100.0
Confusion Matrix:
 [[212   0]
 [  0 211]]
```

In [19]:

```
# for all pairs of classes
y_unique = np.unique(y);
# print(y_unique)
train_knn = np.zeros((len(y_unique),len(y_unique)));
test_knn = np.zeros((len(y_unique),len(y_unique)));
train_svm = np.zeros((len(y_unique),len(y_unique)));
test_svm = np.zeros((len(y_unique),len(y_unique)));
train_log = np.zeros((len(y_unique),len(y_unique)));
test_log = np.zeros((len(y_unique),len(y_unique)));
# looping over all classes
for i in range(len(y_unique)):
    for j in np.arange(i+1,len(y_unique)):
        # slicing classes
        X2 = X[np.where(np.logical_or(y==i,y==j))];
        y2 = y[np.where(np.logical_or(y==i,y==j))];
        # train test split
        data_train, data_test, labels_train, labels_test =
train_test_split(X2, y2, test_size=0.20, random_state=42)

        #KNN
        model = KNN(n_neighbors=10)
        model.fit(data_train,labels_train);
        pred_tr = model.predict(data_train)
        pred_test= model.predict(data_test)

        acc = accuracy_score(labels_train, pred_tr)
        train_knn[i,j] = acc;
        acc = accuracy_score(labels_test, pred_test)
        test_knn[i,j] = acc;
        #SVM
        model = SVC().fit(data_train, labels_train)

        pred_test = model.predict(data_test)
        pred_train = model.predict(data_train)

        acc = accuracy_score(labels_train, pred_tr)
        train_svm[i,j] = acc;
        acc = accuracy_score(labels_test, pred_test)
        test_svm[i,j] = acc;

        # LOGISTIC REGRESSION
```

```
        logisticRegr.fit(data_scaled, labels_train)
        pred_test = logisticRegr.predict(data_test)
        pred_train = logisticRegr.predict(data_train)

        acc = accuracy_score(labels_train, pred_tr)
        train_log[i,j] = acc;
        acc = accuracy_score(labels_test, pred_test)
        test_log[i,j] = acc;
```

In [20]:

```
import pandas as pd
pd.DataFrame(train_knn+train_knn.T)
```

Out[20]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	0.998227	0.986948	0.998116	0.996813	0.993988	0.992258	0.997509	0.990403	0.993
1	0.998227	0.000000	0.976342	0.993590	0.990549	0.994448	0.995818	0.980347	0.989330	0.996
2	0.986948	0.976342	0.000000	0.993264	0.997517	0.995452	0.996231	0.986650	0.989401	0.995
3	0.998116	0.993590	0.993264	0.000000	0.998117	0.982906	0.999365	0.990798	0.981727	0.985
4	0.996813	0.990549	0.997517	0.998117	0.000000	0.995997	0.994201	0.995647	0.987852	0.975
5	0.993988	0.994448	0.995452	0.982906	0.995997	0.000000	0.989189	0.998047	0.985255	0.989
6	0.992258	0.995818	0.996231	0.999365	0.994201	0.989189	0.000000	1.000000	0.993528	0.997
7	0.997509	0.980347	0.986650	0.990798	0.995647	0.998047	1.000000	0.000000	0.994379	0.981
8	0.990403	0.989330	0.989401	0.981727	0.987852	0.985255	0.993528	0.994379	0.000000	0.989
9	0.993715	0.996501	0.995098	0.985759	0.975503	0.989474	0.997457	0.981584	0.989281	0.000

In [21]:

```
pd.DataFrame(test_knn+test_knn.T)
```

Out[21]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	1.000000	0.982630	0.994975	1.000000	0.997333	0.989691	0.997512	0.992327	0.992
1	1.000000	0.000000	0.981567	0.986014	0.985849	0.992611	0.990453	0.969977	0.985782	0.983
2	0.982630	0.981567	0.000000	0.985330	0.997519	0.997403	0.994975	0.985437	0.992537	0.995
3	0.994975	0.986014	0.985330	0.000000	0.989975	0.981627	0.994924	0.982843	0.972292	0.985
4	1.000000	0.985849	0.997519	0.989975	0.000000	0.989333	0.992268	0.987562	0.997449	0.982
5	0.997333	0.992611	0.997403	0.981627	0.989333	0.000000	0.994595	1.000000	0.970588	0.997

	0	1	2	3	4	5	6	7	8	
6	0.989691	0.990453	0.994975	0.994924	0.992268	0.994595	0.000000	0.994975	0.994832	1.000
7	0.997512	0.969977	0.985437	0.982843	0.987562	1.000000	0.994975	0.000000	0.975062	0.958

In [22]:
```
pd.DataFrame(train_svm+train_svm.T)
```

Out[22]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	0.998227	0.986948	0.998116	0.996813	0.993988	0.992258	0.997509	0.990403	0.993
1	0.998227	0.000000	0.976342	0.993590	0.990549	0.994448	0.995818	0.980347	0.989330	0.996
2	0.986948	0.976342	0.000000	0.993264	0.997517	0.995452	0.996231	0.986650	0.989401	0.995
3	0.998116	0.993590	0.993264	0.000000	0.998117	0.982906	0.999365	0.990798	0.981727	0.985
4	0.996813	0.990549	0.997517	0.998117	0.000000	0.995997	0.994201	0.995647	0.987852	0.975
5	0.993988	0.994448	0.995452	0.982906	0.995997	0.000000	0.989189	0.998047	0.985255	0.989
6	0.992258	0.995818	0.996231	0.999365	0.994201	0.989189	0.000000	1.000000	0.993528	0.997
7	0.997509	0.980347	0.986650	0.990798	0.995647	0.998047	1.000000	0.000000	0.994379	0.981
8	0.990403	0.989330	0.989401	0.981727	0.987852	0.985255	0.993528	0.994379	0.000000	0.989
9	0.993715	0.996501	0.995098	0.985759	0.975503	0.989474	0.997457	0.981584	0.989281	0.000

In [23]:
```
pd.DataFrame(test_svm+test_svm.T)
```

Out[23]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	1.000000	1.000000	0.994975	1.000000	0.992000	0.994845	0.997512	1.000000	0.997
1	1.000000	0.000000	0.997696	0.997669	1.000000	0.992611	0.990453	0.993072	0.995261	0.990
2	1.000000	0.997696	0.000000	0.985330	0.995037	0.994805	0.982412	0.978155	0.992537	0.997
3	0.994975	0.997669	0.985330	0.000000	0.989975	0.986877	0.994924	0.990196	0.974811	0.990
4	1.000000	1.000000	0.995037	0.989975	0.000000	0.997333	0.992268	0.995025	1.000000	0.982
5	0.992000	0.992611	0.994805	0.986877	0.997333	0.000000	0.989189	0.994792	0.986631	1.000
6	0.994845	0.990453	0.982412	0.994924	0.992268	0.989189	0.000000	0.994975	0.994832	1.000
7	0.997512	0.993072	0.978155	0.990196	0.995025	0.994792	0.994975	0.000000	0.985037	0.968
8	1.000000	0.995261	0.992537	0.974811	1.000000	0.986631	0.994832	0.985037	0.000000	0.987
9	0.997487	0.990676	0.997555	0.990099	0.982456	1.000000	1.000000	0.968137	0.987406	0.000

In [24]:
```
pd.DataFrame(train_log+train_log.T)
```

Out[24]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	0.998227	0.986948	0.998116	0.996813	0.993988	0.992258	0.997509	0.990403	0.993
1	0.998227	0.000000	0.976342	0.993590	0.990549	0.994448	0.995818	0.980347	0.989330	0.996
2	0.986948	0.976342	0.000000	0.993264	0.997517	0.995452	0.996231	0.986650	0.989401	0.995

	0	1	2	3	4	5	6	7	8	
3	0.998116	0.993590	0.993264	0.000000	0.998117	0.982906	0.999365	0.990798	0.981727	0.985
4	0.996813	0.990549	0.997517	0.998117	0.000000	0.995997	0.994201	0.995647	0.987852	0.975
5	0.993988	0.994448	0.995452	0.982906	0.995997	0.000000	0.989189	0.998047	0.985255	0.989
6	0.992258	0.995818	0.996231	0.999365	0.994201	0.989189	0.000000	1.000000	0.993528	0.997
7	0.997509	0.980347	0.986650	0.990798	0.995647	0.998047	1.000000	0.000000	0.994379	0.981
8	0.990403	0.989330	0.989401	0.981727	0.987852	0.985255	0.993528	0.994379	0.000000	0.989

In [25]:
```
pd.DataFrame(test_log+test_log.T)
```

Out[25]:

	0	1	2	3	4	5	6	7	8	
0	0.000000	1.000000	0.992556	0.989950	0.997455	0.973333	0.987113	0.990050	0.994885	0.992
1	1.000000	0.000000	0.993088	0.993007	0.997642	0.990148	0.990453	0.990762	0.954976	0.988
2	0.992556	0.993088	0.000000	0.953545	0.982630	0.974026	0.969849	0.961165	0.927861	0.987
3	0.989950	0.993007	0.953545	0.000000	0.987469	0.868766	0.989848	0.950980	0.954660	0.975
4	0.997455	0.997642	0.982630	0.987469	0.000000	0.981333	0.984536	0.982587	0.982143	0.934
5	0.973333	0.990148	0.974026	0.868766	0.981333	0.000000	0.981081	0.968750	0.759358	0.973
6	0.987113	0.990453	0.969849	0.989848	0.984536	0.981081	0.000000	0.994975	0.963824	1.000
7	0.990050	0.990762	0.961165	0.950980	0.982587	0.968750	0.994975	0.000000	0.855362	0.953
8	0.994885	0.954976	0.927861	0.954660	0.982143	0.759358	0.963824	0.855362	0.000000	0.846
9	0.992462	0.988345	0.987775	0.975248	0.934837	0.973753	1.000000	0.953431	0.846348	0.000

In [26]:

```
data_train, data_test, labels_train, labels_test =
train_test_split(X, y, test_size=0.20, random_state=42)

print('\n== Classes 0 and Classes 1 ==\n')
#KNN
print("=====KNN=====")
model = KNN(n_neighbors=10)
model.fit(data_train,labels_train);
pred_tr = model.predict(data_train)
pred= model.predict(data_test)

cm = confusion_matrix(labels_train, pred_tr)
acc = accuracy_score(labels_train, pred_tr)
print('Train Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

cm = confusion_matrix(labels_test, pred)
acc = accuracy_score(labels_test, pred)
print('Test Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

#SVM
print("\n\n=====SVM=====")
model = SVC().fit(data_train, labels_train)

pred_test = model.predict(data_test)
pred_train = model.predict(data_train)
cm = confusion_matrix(labels_train, pred_train)
acc = accuracy_score(labels_train, pred_train)
print('Train Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

cm = confusion_matrix(labels_test, pred_test)
acc = accuracy_score(labels_test, pred_test)
print('Test Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

# LOGISTIC REGRESSION
print("\n\n=====Logistic Regression=====")
scaler = preprocessing.StandardScaler().fit(data_train)
data_scaled = scaler.transform(data_train)
```

```
pred_test = logisticRegr.predict(data_test)
pred_train = logisticRegr.predict(data_train)
cm = confusion_matrix(labels_train, pred_train)
acc = accuracy_score(labels_train, pred_train)
print('Train Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)

cm = confusion_matrix(labels_test, pred_test)
acc = accuracy_score(labels_test, pred_test)
print('Test Accuracy:',acc*100)
print('Confusion Matrix:\n',cm)
```

```
== Classes 0 and Classes 1 ==

=====KNN=====
Train Accuracy: 95.42500000000001
Confusion Matrix:
 [[767   1   0   0   0   3   5   1   0   0]
 [  0 914   2   2   0   0   1   0   0   0]
 [ 11  23 757   3   1   1   2  16   5   0]
 [  0   6   3 774   0   5   0   8   2   4]
 [  0  17   0   0 723   0   2   2   1  22]
 [  4   5   0  16   2 680   8   1   0   2]
 [  4   6   0   0   0   2 746   0   0   0]
 [  0  35   2   0   2   1   0 791   0  10]
 [  5  10   3  17   7  16   1   6 717   6]
 [  3   7   2   5   6   1   1  18   3 765]]
Test Accuracy: 94.3
Confusion Matrix:
 [[199   0   0   0   0   1   3   0   0   0]
 [  0 216   0   0   0   0   0   0   0   0]
 [  6   7 189   2   0   0   1   7   1   0]
 [  0   0   1 202   0   2   0   3   0   0]
 [  0   1   0   0 201   1   3   1   0   8]
 [  1   5   1   4   1 160   1   0   0   1]
 [  4   0   0   0   1   0 195   0   0   0]
 [  0   4   0   0   0   0   0 181   0   2]
 [  2   1   4   3   2   4   2   5 162   1]
 [  3   2   1   1   4   0   0   6   0 181]]


=====SVM=====
Train Accuracy: 98.88749999999999
Confusion Matrix:
 [[773   0   0   0   0   1   0   1   2   0]
 [  0 913   2   2   0   1   0   1   0   0]
 [  0   0 813   1   0   0   0   3   1   1]
 [  0   0   1 791   0   3   0   5   1   1]
 [  0   0   0   0 758   0   1   1   1   6]
 [  1   0   0   1   0 713   1   0   2   0]
 [  1   3   0   0   0   1 753   0   0   0]
 [  0   9   2   1   2   0   0 822   0   5]
 [  0   0   0   1   2   4   1   0 779   1]
 [  3   0   0   2   4   1   0   3   2 796]]
Test Accuracy: 96.35000000000001
```

```
Confusion Matrix:
 [[200   0   0   1   1   0   1   0   0   0]
 [  0 214   1   0   0   0   1   0   0   0]
 [  1   0 206   1   1   0   2   1   0   1]
 [  0   0   1 201   0   3   0   3   0   0]
 [  1   0   2   0 204   0   3   1   1   3]
 [  0   0   0   1   1 167   3   0   1   1]
 [  3   0   2   0   2   2 191   0   0   0]
 [  1   0   1   0   0   0   0 182   0   3]
 [  0   0   2   3   1   0   0   2 178   0]
 [  2   2   0   2   6   0   0   2   0 184]]


=====Logistic Regression=====
Train Accuracy: 82.825
Confusion Matrix:
 [[706   0   6  18   3   3   3   2  29   7]
 [  0 804   0  18   0   0   0   0  97   0]
 [  0   4 663  30  14   0   4   4  94   6]
 [  0   0   2 764   1   0   0   0  35   0]
 [  1   0   0   1 718   0   0   0  33  14]
 [  5   4   8  92  10 195   6   5 378  15]
 [  0   1   7   5  22   0 611   2 105   5]
 [  1   2  15  55  11   0   1 666  44  46]
 [  0   0   1   2   1   0   0   0 783   1]
 [  1   0   2  10  18   0   1   2  61 716]]
Test Accuracy: 81.15
Confusion Matrix:
 [[180   0   5   6   0   0   2   1   7   2]
 [  0 182   1   5   0   0   0   1  27   0]
 [  1   1 166   4   2   0   3   0  31   5]
 [  0   0   1 193   1   1   0   1   9   2]
 [  0   0   1   1 183   0   2   0  12  16]
 [  1   2   2  20   4  63   4   1  69   8]
 [  2   0   2   1   7   1 163   0  23   1]
 [  1   1   5  10   3   0   0 145  11  11]
 [  0   0   0   6   1   0   0   1 177   1]
 [  0   1   0   4   4   0   0   0  18 171]]
```

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