Python Notebook Viewer

Name - Niraj Kumar

Roll No - 160455

Assignment 1 Submission Solution

- Please go through the whole notebook as the details regarding the question from assignment and explanation is given below. Thank You

Out [2]:

```
import numpy as np
import pandas as pd
import scipy
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifie
#class A and class B
#define seed

seed =4
# setting the seed

np.random.seed(seed)

A = np.random.multivariate_normal([1,0],[[1,0],[
B = np.random.multivariate_normal([0,1],[[1,0],[


plt.plot(A)
plt.show()
plt.plot(B)
```

```
plt.show()
#covariance for generating training and test data

identity_2 = [[1/5,0],[0,1/5]]
#training data variable train_A and corresponding
train_A = []
for i in range(100):
    index = np.random.choice([0,1,2,3,4,5,6,7,8,9
    m_k = A[index]
    train_A.append(np.random.multivariate_normal(

train_B = []
for i in range(100):
    index = np.random.choice([0,1,2,3,4,5,6,7,8,9
    m_k = B[index]
    train_B.append(np.random.multivariate_normal(

test_A = []
for i in range(5000):
    index = np.random.choice([0,1,2,3,4,5,6,7,8,9
    m_k = A[index]
    test_A.append(np.random.multivariate_normal(m
test_B = []
for i in range(5000):
    index = np.random.choice([0,1,2,3,4,5,6,7,8,9
    m_k = B[index]
    test_B.append(np.random.multivariate_normal(m

#so now we have the train_A,train_B,test_A,test_B
#with 100,100,5000,5000 data points respectively

#taking only the numerical rows from the array ty

new_train_A = []
for i in train_A:
    for j in i:
        new_train_A.append(j)

#converting the new_train_A data ponints into the
```

```
df_train_A = pd.DataFrame(new_train_A)
df_train_A.columns = ['train_A_f1','train_A_f2']

#similarly for train_B
new_train_B = []
for i in train_B:
    for j in i:
        new_train_B.append(j)

df_train_B = pd.DataFrame(new_train_B)
df_train_B.columns = ['train_B_f1','train_B_f2']

new_test_A = []
for i in test_A:
    for j in i:
        new_test_A.append(j)

df_test_A = pd.DataFrame(new_test_A)

df_test_A.columns = ['test_A_f1','test_A_f2']

new_test_B = []
for i in test_B:
    for j in i:
        new_test_B.append(j)


df_test_B = pd.DataFrame(new_test_B)

df_test_B.columns = ['test_B_f1','test_B_f2']
df_test_A['label'] = 0
df_test_B['label'] = 1
df_train_A['label'] = 0
df_train_B['label'] = 1

#making copies to the above varibles such that tl

df_test_A.columns = ['f1','f2','Label']
df_test_B.columns = ['f1','f2','Label']
```

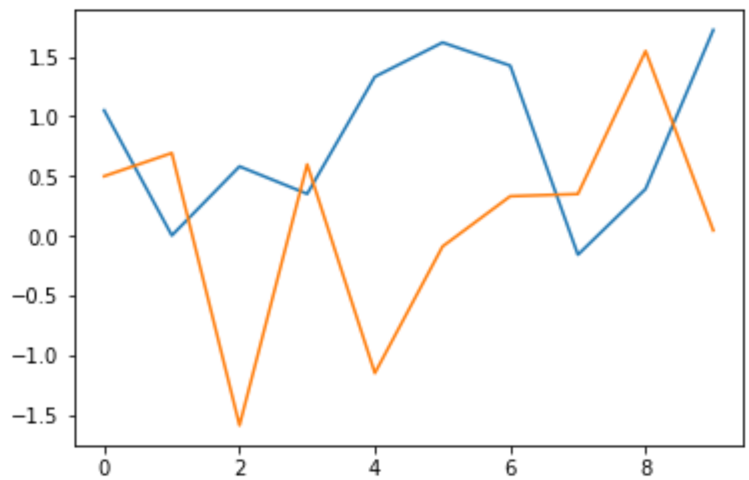
```
df_train_A.columns = ['f1','f2','Label']
df_train_B.columns = ['f1','f2','Label']
df_testA_copy = df_test_A
test_data = df_testA_copy.append(df_test_B)
train_data = df_train_A.append(df_train_B)

#taking the labels

train_labels = train_data.Label
test_labels = test_data.Label

#our training data
train_data_c = train_data[['f1','f2']]
test_data_c = test_data[['f1','f2']]

from sklearn.metrics import accuracy_score,f1_sc

#running the algorithm
acc_vec_train= []
k_points_train = []
for w in range(1,41):
    if (w==0):
        model = KNeighborsClassifier(n_neighbors
        k_points_train.append(1)
    else:
        model = KNeighborsClassifier(n_neighbors
        k_points_train.append(w)
    model.fit(train_data_c,train_labels)
    y_pred = model.predict(train_data_c)
    #print(accuracy_score(train_labels,y_pred))
    acc_vec_train.append(accuracy_score(train_la


error_train =[]
for i in acc_vec_train:
    #print(i)
    error_train.append(100-i*100)
```

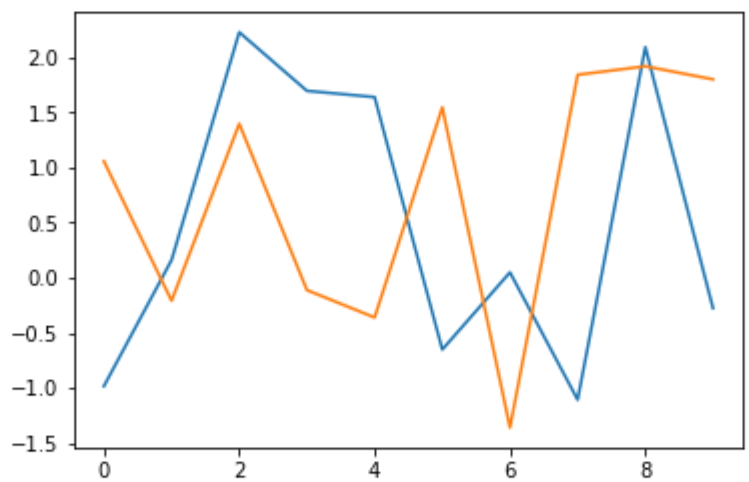
```
#plotting the training data k_values vs misclass:
plt.figure(figsize=(12,7))
plt.plot(error_train,color='black', marker='.', :
plt.title('Training Data')
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error_training')
plt.show()


#for test data

acc_vec_ver2= []
f1_acc = []
k_points_ver2 = []
for w in range(1,41):
    if (w==0):
        model = KNeighborsClassifier(n_neighbors=
        k_points_ver2.append(1)
    else:
        model = KNeighborsClassifier(n_neighbors=
        k_points_ver2.append(w)
    model.fit(train_data_c,train_labels)
    y_pred = model.predict(test_data_c)
    #print(accuracy_score(test_labels,y_pred))
    acc_vec_ver2.append(accuracy_score(test_labe:
    f1_acc.append(f1_score(test_labels,y_pred))
    #print("f1_score_values")
    #print(f1_score(test_labels,y_pred))
    #print("\n")

error_vec_ver2 = []
error_f1 =[]
for i in acc_vec_ver2:
    print(i)
    error_vec_ver2.append(100-i*100)

error_f1 =[]
for i in f1_acc:
    #print(i)
```

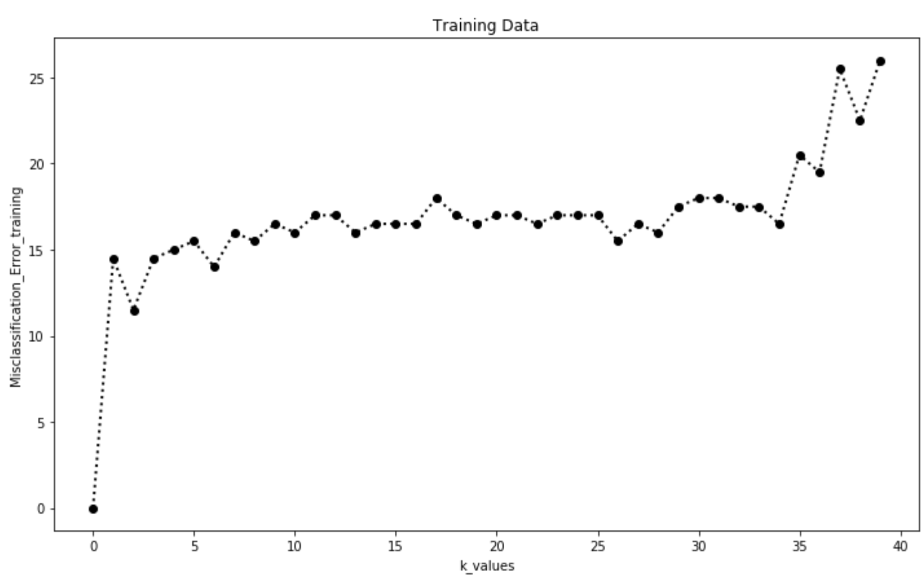
```
    error_f1.append(100-i*100)

#plotting the misclassification rate with test da

plt.figure(figsize=(12,7))
plt.plot(error_vec_ver2,color='m', marker='.', l:
plt.title('Accuracy_score (Test_Data)')
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.show()

#combining the plots

#combining all_plots
plt.figure(figsize=(12,7))
plt.plot(error_train,color='black', marker='.', :
#plt.figure(figsize=(12,7))
plt.plot(error_vec_ver2,color='red', marker='.',
plt.title('Accuracy_score (Test_Data & Training_[
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.legend()
plt.show()

#in bigger version

#combining all_plots
plt.figure(figsize=(24,14))
plt.plot(error_train,color='black', marker='.', :
#plt.figure(figsize=(12,7))
plt.plot(error_vec_ver2,color='green', marker='.
plt.title('Accuracy_score (Test_Data & Training_[
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.legend()
plt.show()


#plotting in ratios
```

```
k_points_train_2 = [1/i for i in k_points_train]
#plotting in ration

#combining all_plots
plt.figure(figsize=(24,14))
plt.plot(k_points_train_2,error_train,color='bla
#plt.figure(figsize=(12,7))
plt.plot(k_points_train_2,error_vec_ver2,color='
plt.title('Accuracy_score (Test_Data & Training_[
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.legend()
plt.show()


#plotting the regression line in misclassificati
plt.figure(figsize=(14,7))
X1 = k_points_ver2[:]
Y1 = error_train[:]
X2 = k_points_ver2[:]
Y2 = error_vec_ver2[:]

# solve for a and b
def best_fit(X, Y):

    xbar = sum(X)/len(X)
    ybar = sum(Y)/len(Y)
    n = len(X) # or len(Y)

    numer = sum([xi*yi for xi,yi in zip(X, Y)])
    denum = sum([xi**2 for xi in X]) - n * xbar*

    b = numer / denum
    a = ybar - b * xbar

    print('best fit line:\ny = {:.2f} + {:.2f}x'

    return a, b
```

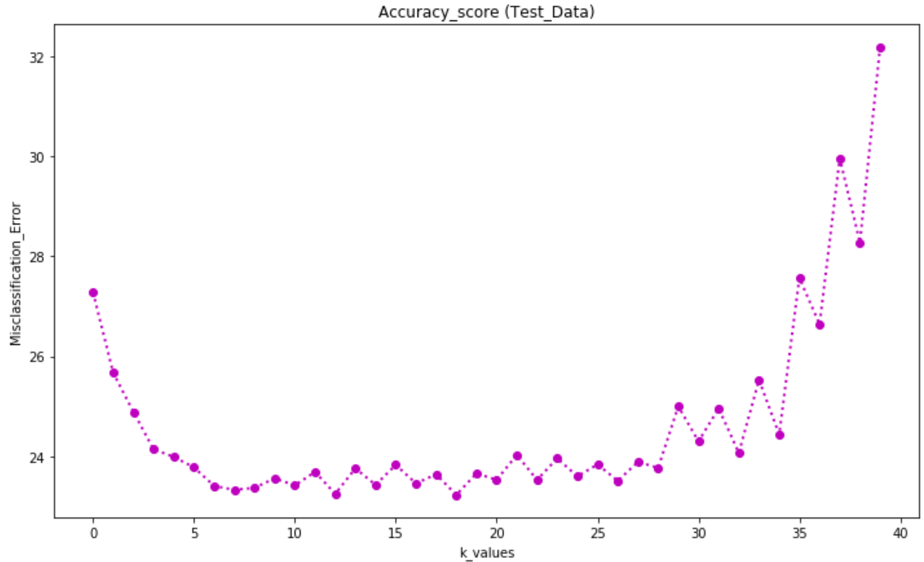
```
#A_A,B_B = best_fit(X1,Y1)

plt.figure(figsize=(12,6))
a, b = best_fit(X1, Y1)

plt.scatter(X1, Y1)
yfit1 = [a + b * xi for xi in X1]
plt.plot(X1, yfit1,'r',label = 'Training')


# help(plt.scatter)
a, b = best_fit(X2, Y2)

plt.scatter(X2, Y2)
yfit2 = [a + b * xi for xi in X2]
plt.plot(X2, yfit2,'m',label = 'test')
plt.legend()
plt.show()


#now using the gaussian Naives bayes classifier

# Using the bayes classification
from sklearn.naive_bayes import GaussianNB
mo = GaussianNB()
mo.fit(train_data_c,train_labels)
y_pr= mo.predict(test_data_c)
print("naive_bayes error is:")
print(100-accuracy_score(test_labels,y_pr)*100)
tr = mo.predict(train_data_c)
print("naive_bayes error on testing data is:")
print(100-accuracy_score(train_labels,tr)*100)

print("The best possible accuracy on the test dat
```

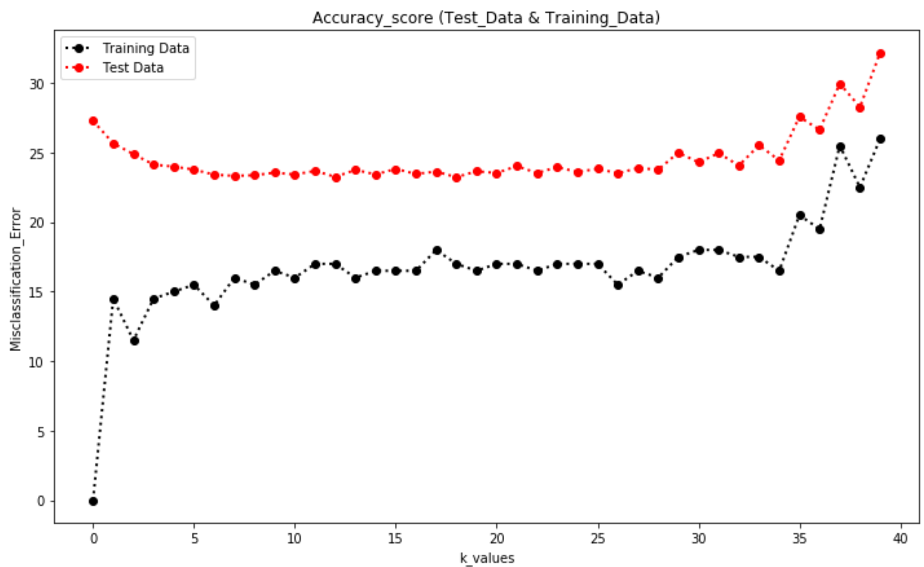


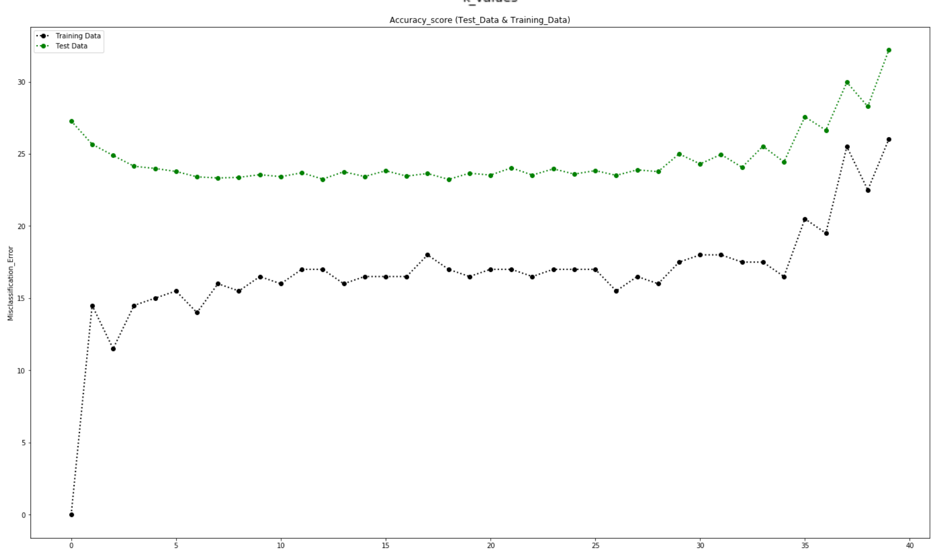


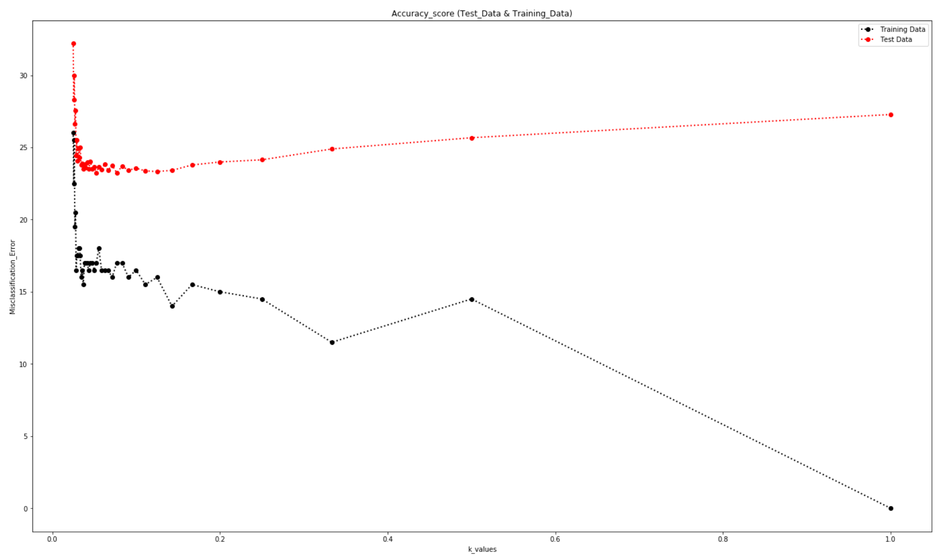

Training Data
Misclassification_Error_training
k_values

```
0.7272
0.7433
0.7511
0.7586
0.7601
0.7622
0.7659
0.7667
0.7663
0.7644
0.7658
0.7631
0.7675
0.7624
0.7657
0.7617
0.7654
0.7636
0.7677
0.7634
0.7647
0.7597
0.7647
0.7604
0.764
0.7616
0.7649
0.7611
0.7623
0.75
0.757
0.7504
0.7593
0.7447
0.7556
0.7243
0.7336
0.7004
0.7172
0.6781
```


Accuracy_score (Test_Data)
Misclassification_Error
k_values

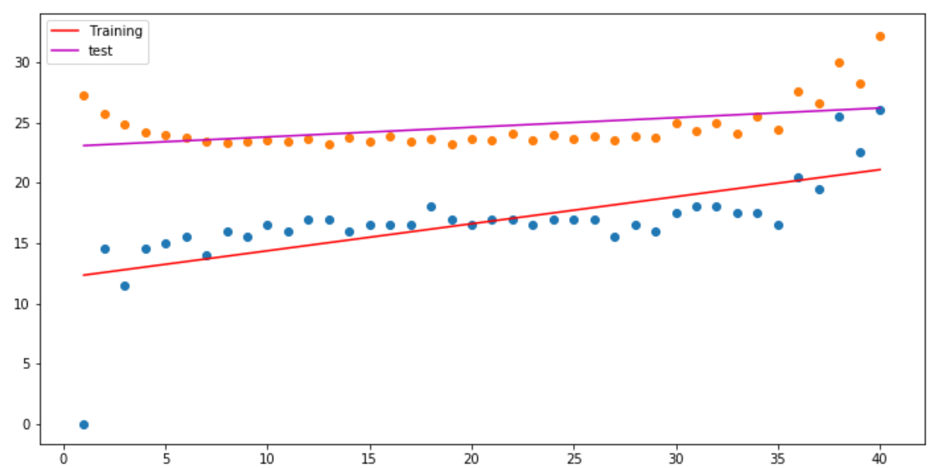

Accuracy_score (Test_Data & Training_Data)
Training Data
Test Data
Misclassification_Error
k_values


Accuracy_score (Test_Data & Training_Data)
Training Data
Test Data
k_values




```
best fit line:
y = 12.12 + 0.22x
best fit line:
y = 23.00 + 0.08x
```

```
&lt;Figure size 1008x504 with 0 Axes&gt;
```




```
naive_bayes error is:
31.39999999999999
naive_bayes error on testing data is:
31.5
The best possible accuracy on the test data form kNN
that I got is 77% at the k value = 5
```

#Solution 1.a

The Solution Starts From Here

Defining the Packages

- import numpy as np
- import pandas as pd
- import scipy
- import sklearn
- import matplotlib.pyplot as plt
- import seaborn as sns
- from sklearn.neighbors import KNeighborsClassifier

Solution 1.a

I generated 10 samples of mk for class A as

- A = np.random.multivariate_normal([1,0],[[1,0],[0,1]],10)

Solution 1.b

and for B

- B = np.random.multivariate_normal([0,1],[[1,0],[0,1]],10)

showing the plots for A and B

- plt.plot(A)
- plt.show()
- plt.plot(B)
- plt.show()

covariance for generating training and test data

- Covariance = [[1/5,0],[0,1/5]]

Solution 1.c

- for i in range(100):
    - index = np.random.choice([0,1,2,3,4,5,6,7,8,9],p=

[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]) m_k = A[index] train_A.append(np.random.multivariate_normal(m_k,identity_2,1))

this will create 100 points for class A training data

Similarly for training data B

train_B = []

for i in range(100): index = np.random.choice([0,1,2,3,4,5,6,7,8,9],p=[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1]) m_k = B[index] train_B.append(np.random.multivariate_normal(m_k,identity_2,1))

Solution 2

Similary for test_A and test_B generating 5000 points each

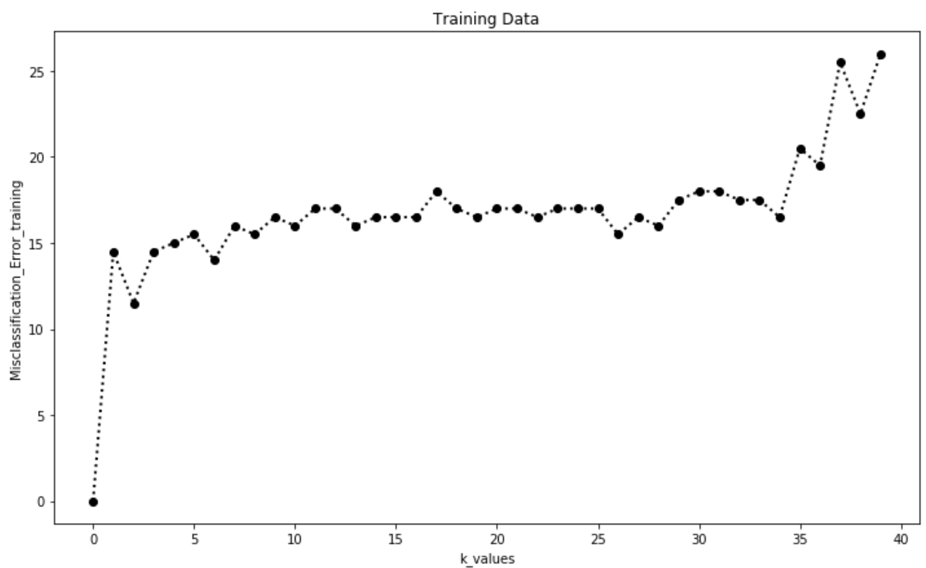
- test_A = []
- for i in range(5000):
    - index = np.random.choice([0,1,2,3,4,5,6,7,8,9],p=[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1])
        - m_k = A[index]
    - test_A.append(np.random.multivariate_normal(m_k,identity_2,1))
- test_B = []
- for i in range(5000):
    - index = np.random.choice([0,1,2,3,4,5,6,7,8,9],p=[0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1])
    - m_k = B[index]
    - test_B.append(np.random.multivariate_normal(m_k,identity_2,1))

Solution 3.a

In [3]:

```
plt.figure(figsize=(12,7))
```

```
plt.plot(error_train,color='black', marker='.', l
plt.title('Training Data')
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error_training')
plt.show()
```



from above it can be clearly seen that as the value of k increases even on the training data the error rate or percentage increases

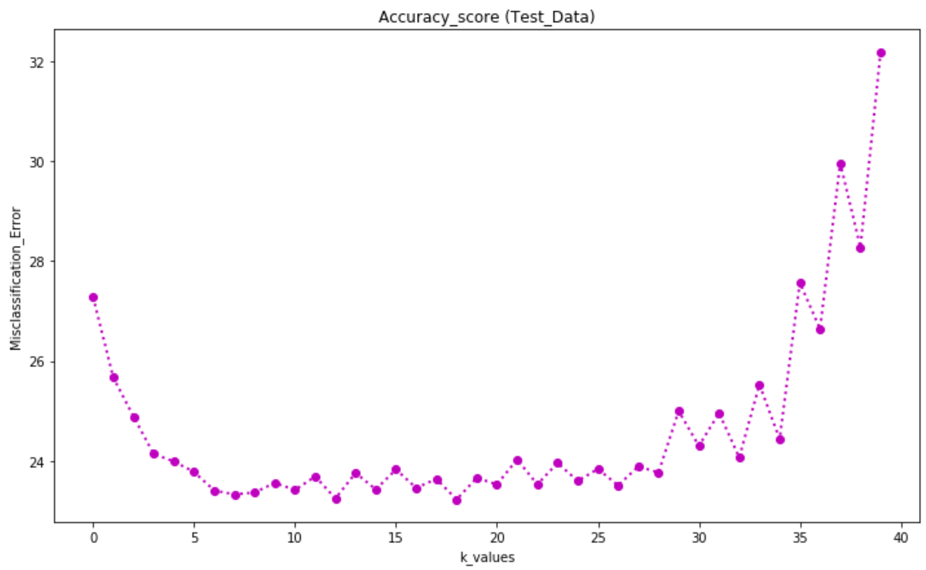
these clearly states that for higher value of the k we should not use the algo

Solution 3.b

on the test data

In [4]:

```
plt.figure(figsize=(12,7))
plt.plot(error_vec_ver2,color='m', marker='.', l
plt.title('Accuracy_score (Test_Data)')
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.show()
```



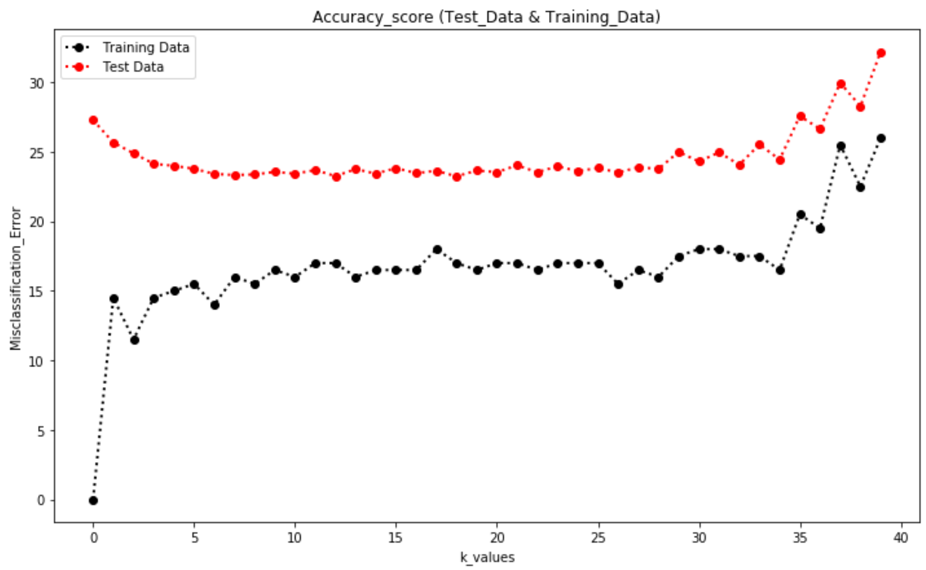
*** initially for k=1 the error was around 27% but as the k goes to 3,4,5,6,7**

*** the error decreases and thus we obtain our least error at K= 7 which is 77%**

Combining both the plots

In [5]:

```
#combining all_plots
plt.figure(figsize=(12,7))
plt.plot(error_train,color='black', marker='.', l
#plt.figure(figsize=(12,7))
plt.plot(error_vec_ver2,color='red', marker='.',
plt.title('Accuracy_score (Test_Data & Training_D
plt.xlabel('k_values')
plt.ylabel('Misclassification_Error')
plt.legend()
plt.show()
```



the above plot is just the combination of both the plots

Now generating the regression line on our obtained error data on both training data and test data

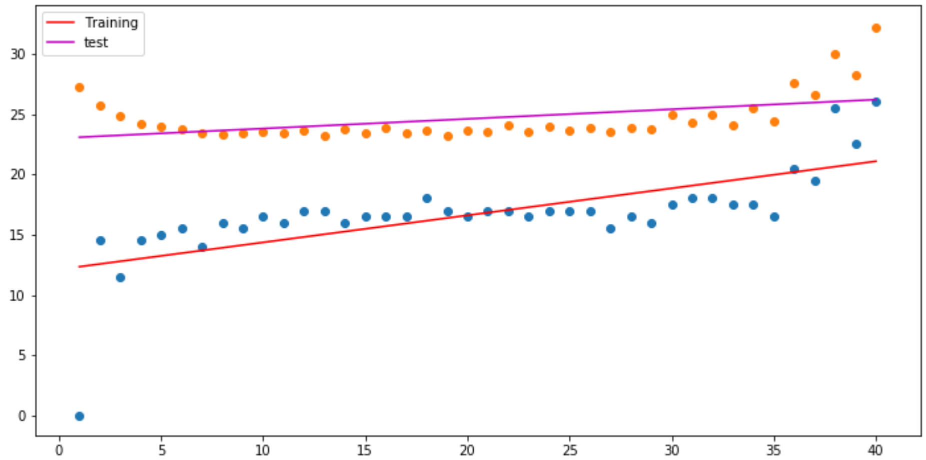
In [6]:

```
plt.figure(figsize=(14,7))
X1 = k_points_ver2[:]
Y1 = error_train[:]
X2 = k_points_ver2[:]
Y2 = error_vec_ver2[:]

# solve for a and b
def best_fit(X, Y):

    xbar = sum(X)/len(X)
    ybar = sum(Y)/len(Y)
    n = len(X) # or len(Y)

    numer = sum([xi*yi for xi,yi in zip(X, Y)]) -
    denum = sum([xi**2 for xi in X]) - n * xbar*

    b = numer / denum
    a = ybar - b * xbar
```

```
    print('best fit line:\ny = {:.2f} + {:.2f}x'

    return a, b



#A_A,B_B = best_fit(X1,Y1)

plt.figure(figsize=(12,6))
a, b = best_fit(X1, Y1)

plt.scatter(X1, Y1)
yfit1 = [a + b * xi for xi in X1]
plt.plot(X1, yfit1,'r',label = 'Training')


# help(plt.scatter)
a, b = best_fit(X2, Y2)

plt.scatter(X2, Y2)
yfit2 = [a + b * xi for xi in X2]
plt.plot(X2, yfit2,'m',label = 'test')
plt.legend()
plt.show()
```

```
best fit line:
y = 12.12 + 0.22x
best fit line:
y = 23.00 + 0.08x
```

```
<Figure size 1008x504 with 0 Axes>
```



- we created our regression line

In [ ]:

Solution 4

- I used Gaussion Naive Bayes to find out the accuracy on the test data that we generated earlier
- The result which I got was 69% on test data adn 73% on the training data
- below is the given code

In [8]:

```
#now using the gaussian Naives bayes classifier

# Using the bayes classification
from sklearn.naive_bayes import GaussianNB
mo = GaussianNB()
mo.fit(train_data_c,train_labels)
y_pr= mo.predict(test_data_c)
print("naive_bayes error is:")
print(100-accuracy_score(test_labels,y_pr)*100)
tr = mo.predict(train_data_c)
print("naive_bayes error on testing data is:")
print(100-accuracy_score(train_labels,tr)*100)

#print("The best possible accuracy on the test da
```



```
naive_bayes error is:
31.39999999999999
naive_bayes error on testing data is:
31.5
```

- "The best possible accuracy on the test data form kNN that I got is 77% at the k value = 7"

Solution 5

- Some of the Insights:
    - The error rate trend on of KNN decreased as we increased the the value of K but in between it fluctuated between increasing and decreasing an after K = 30 the error increased as higher value causes disambiguity
    - Since the data generated cannot be for my random seed = 4 cannot be difined using a linear classifier as it was clear when I scatter plotted thus my accuracy did not go up much above 77%
    - The Overall trend line(regression line) got increased on both the training data and the test data as higher value of K in KNN predicts disambiguity results
    - The max accuracy on test data is 77%(KNN)
    - The max accuracy on train data is 100% as expected (KNN)
    - The max accuracy on train data is 73% on Gaussian Naive Bayes
    - the max accuracy on test data is 69% on Gaussian Naive Bayes
    - the Value of K in KNN corresponding to the Gaussina Naive Bayes error of 69% is K = 20

```
In []:
```