1a)

The below part of the code reads the file (iris.txt) with Iris data and gets the num data points &features.

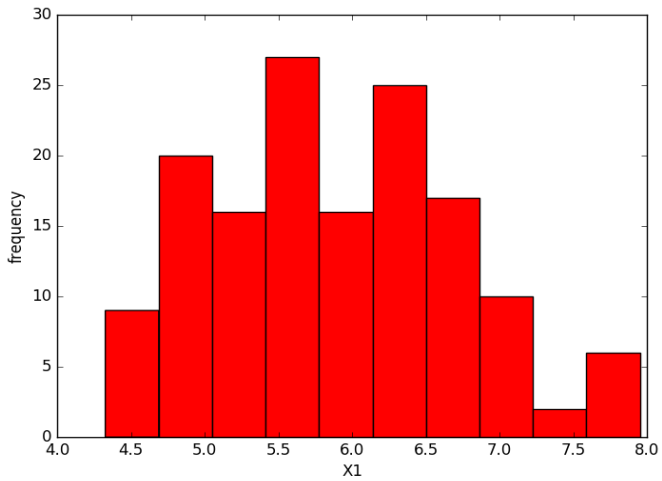
```
iris = np.genfromtxt("data/iris.txt",delimiter=None) # load the text file
Y = iris[:,-1] # target value is the last column
X = iris[:,0:-1]


num_pts =(len(X))
features= (len(X[0]))
```

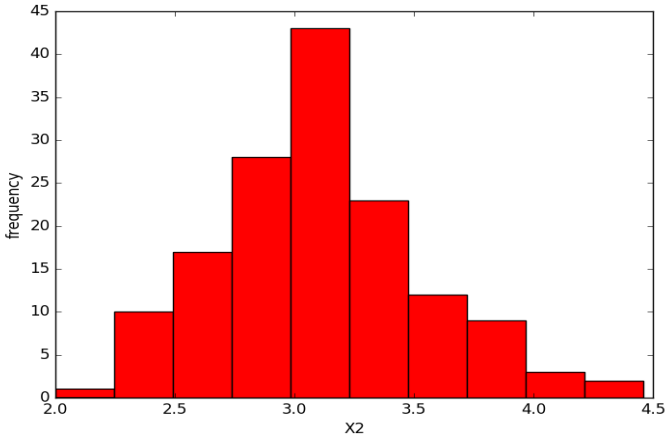
1b)

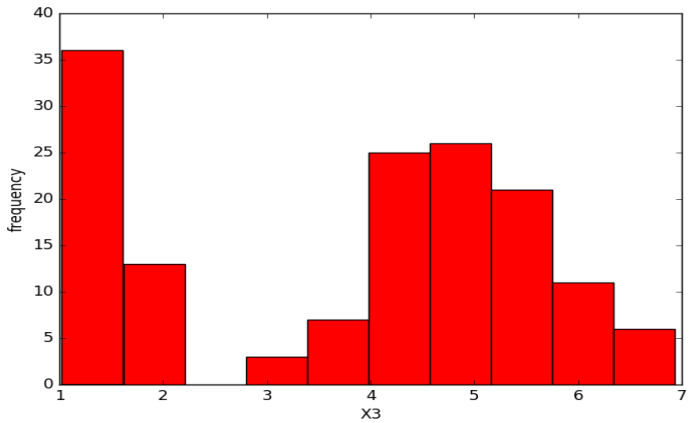
The below code plots the histogram for each feature with feature name in X-axis & frequency on Yaxis.

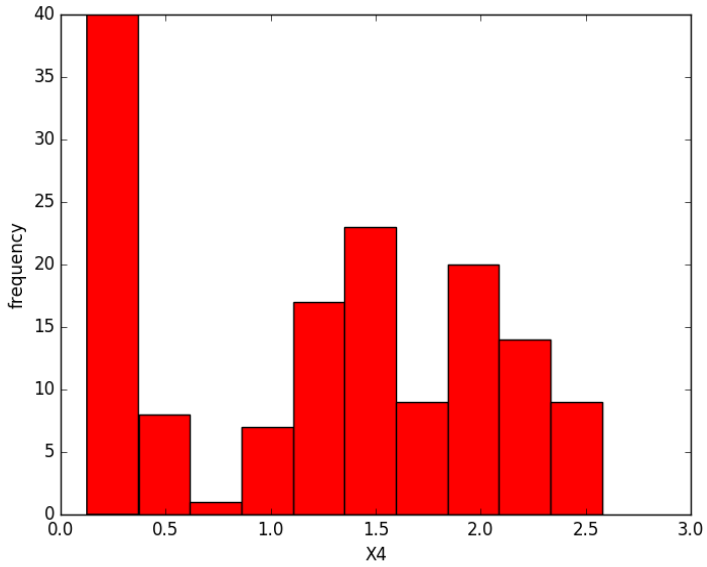
```
plt.hist(X[:,0],color='red')
plt.xlabel('X1')
plt.ylabel('frequency')
plt.show()

plt.hist(X[:,1],color='red')
plt.xlabel('X2')
plt.ylabel('frequency')
plt.show()


plt.hist(X[:,2],color='red')
plt.xlabel('X3')
plt.ylabel('frequency')
plt.show()

plt.hist(X[:,3],color='red')
plt.xlabel('X4')
plt.ylabel('frequency')
plt.show()
```

And the plots are attached below.




frequency
X2


frequency
X3


frequency
X4

1c) & 1d)

the following method finds out the mean, variance & std deviation for each feature

```
def mean_var_std(x) :
    result = [np.mean(x),np.var(x),np.std(x)]
    print("mean : %f  var : %f  std_dev : %f" % (result[0],result[1],result[2]))
    return result
```

for each feature of X the “mean,var,std” is found out by invoking the above method.

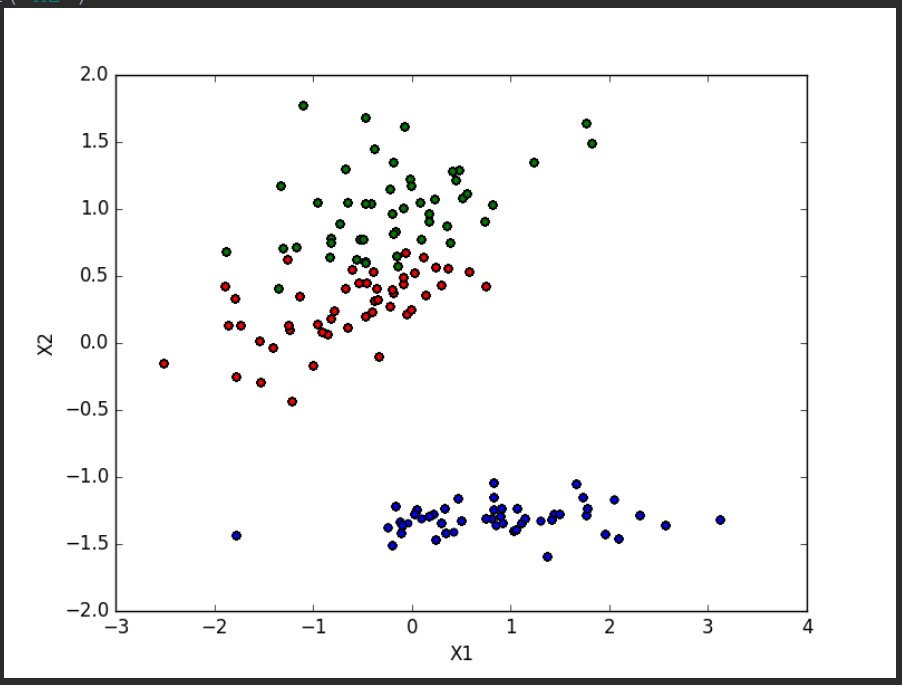
```
for i in range(len(X[0])):
    #print (mean_var_std(X[:,i]))
    mean_var_std(X[:, i])
```

1e) Normalization of the data is done using a for loop the code is below.

```
arr=[]
for i in range(0,len(X[0])):
    arr.append( mean_var_std(X[:,i]))
    print (i)
    X[:,i] -= arr[i][0]
    X[:,i] /= arr[i][2]
```

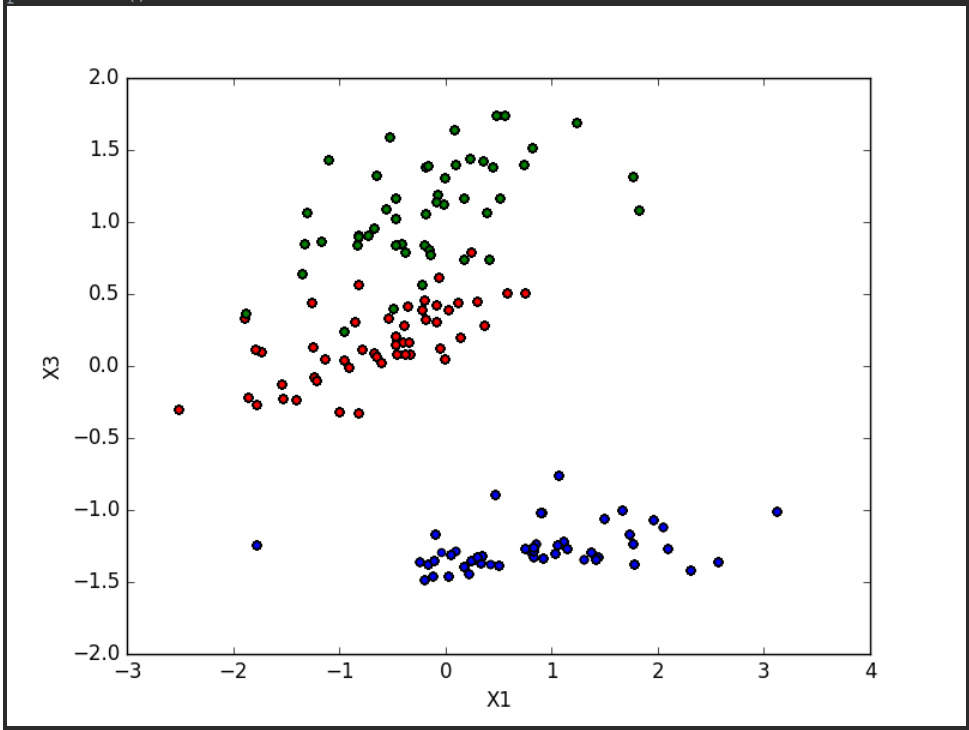
1f)

```
for i in range(len(Y)):
    #plt.subplot(100+i)
    if(Y[i] == 0):
        plt.scatter(X[i:,1],X[i:,2],c='b')
    elif (Y[i] == 1):
        plt.scatter(X[i:, 1], X[i:, 2], c='r')
    else :
        plt.scatter(X[i:, 1], X[i:, 2], c='g')
plt.xlabel('X1')
plt.ylabel('X2')
```

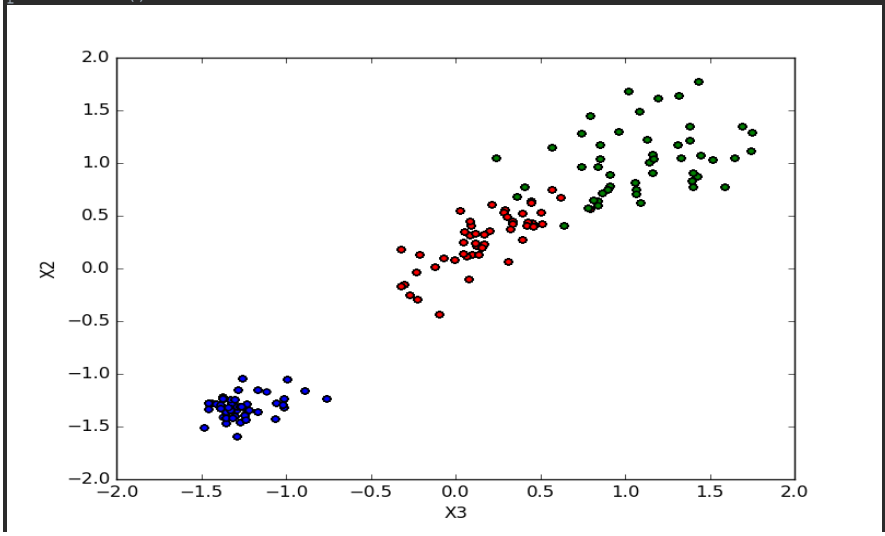



```
plt.show()
```

```
for i in range(len(Y)):
    #plt.subplot(100+i)
    if(Y[i] == 0):
        plt.scatter(X[i:,1],X[i:,3],c='b')
    elif (Y[i] == 1):
        plt.scatter(X[i:, 1], X[i:, 3], c='r')
    else :
        plt.scatter(X[i:, 1], X[i:, 3], c='g')
plt.xlabel('X1')
plt.ylabel('X3')
plt.show()
```



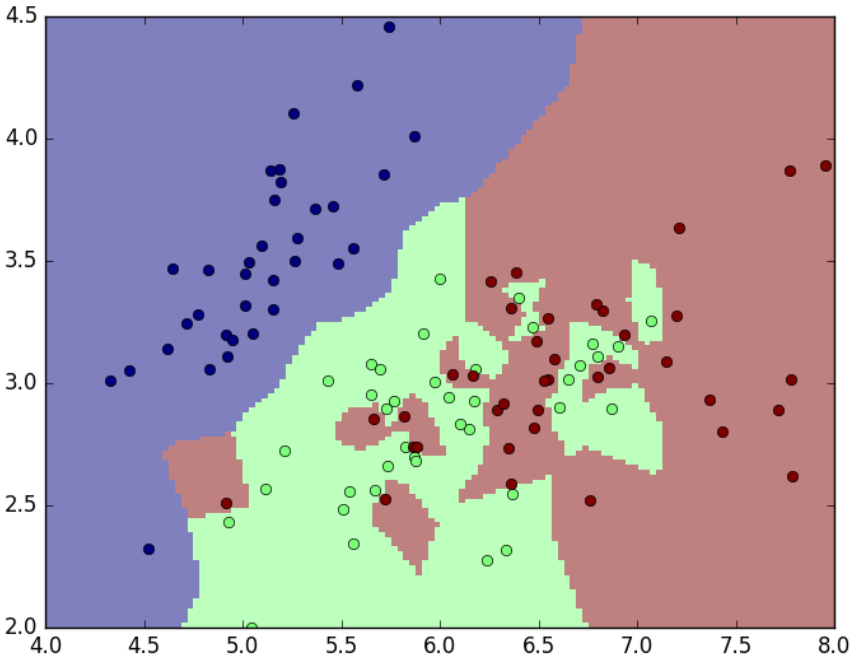

```
for i in range(len(Y)):
    #plt.subplot(100+i)
    if(Y[i] == 0):
        plt.scatter(X[i:,3],X[i:,2],c='b')
    elif (Y[i] == 1):
        plt.scatter(X[i:, 3], X[i:, 2], c='r')
    else :
        plt.scatter(X[i:, 3], X[i:, 2], c='g')
plt.xlabel('X3')
plt.ylabel('X2')
plt.show()
```

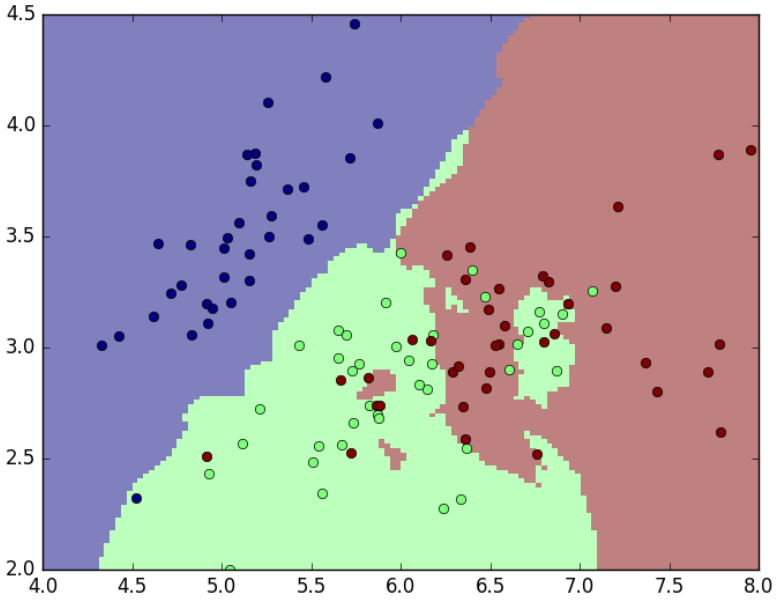


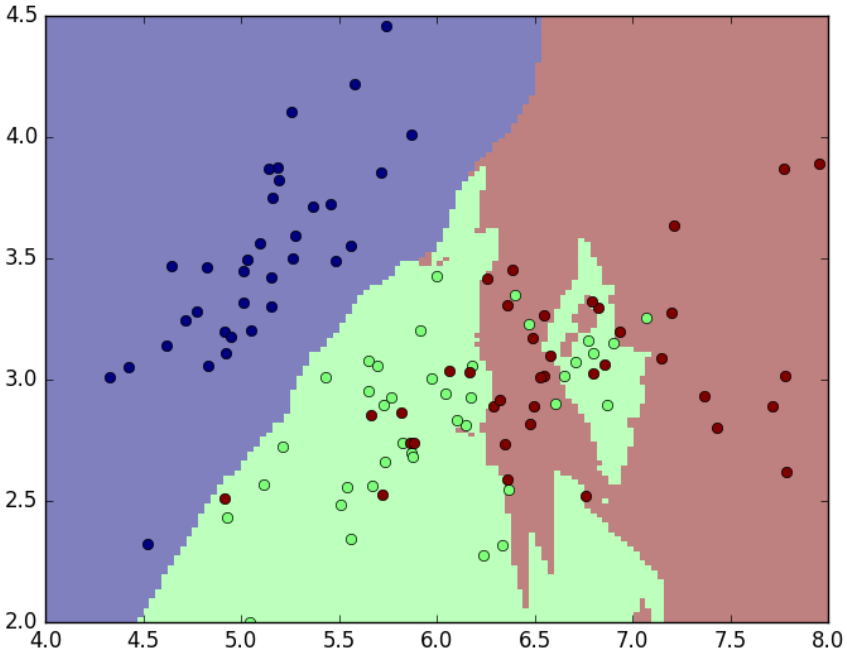
2a)

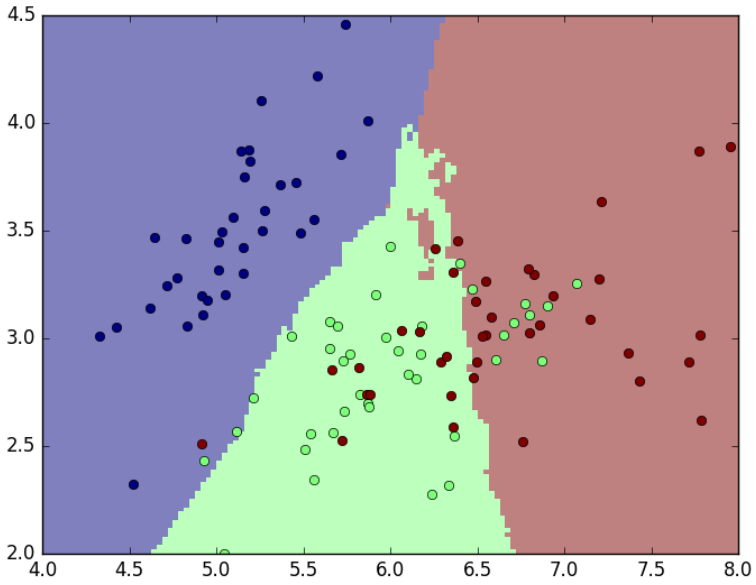
```
import mltools as ml
import numpy as np
import matplotlib.pyplot as plt
#from pylab import figure, clf, plot, xlabel, ylabel

iris = np.genfromtxt("data/iris.txt",delimiter=None) # load the text file
Y = iris[:,-1] # target value is the last column
X = iris[:,0:-1]

print (len(Y))
X,Y = ml.shuffleData(X,Y)
Xtr,Xte,Ytr,Yte = ml.splitData(X,Y, 0.75)
print (len(Ytr))
print (len(Yte))
knn = ml.knn.knnClassify() # create the object and train it
for K in [1,5,10,50]:
    knn.train(Xtr[:,0:2],Ytr,K) # where K is an integer, e.g. 1 for nearest
neighbor prediction
    YteHat = knn.predict(Xte[:,0:2])
    ml.plotClassify2D(knn,Xtr[:,0:2],Ytr)
```

Below are the plots for the K values 1 5 10 50 respectively




4.5
4.0
3.5
3.0
2.5
2.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


4.5
4.0
3.5
3.0
2.5
2.0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0



2b)

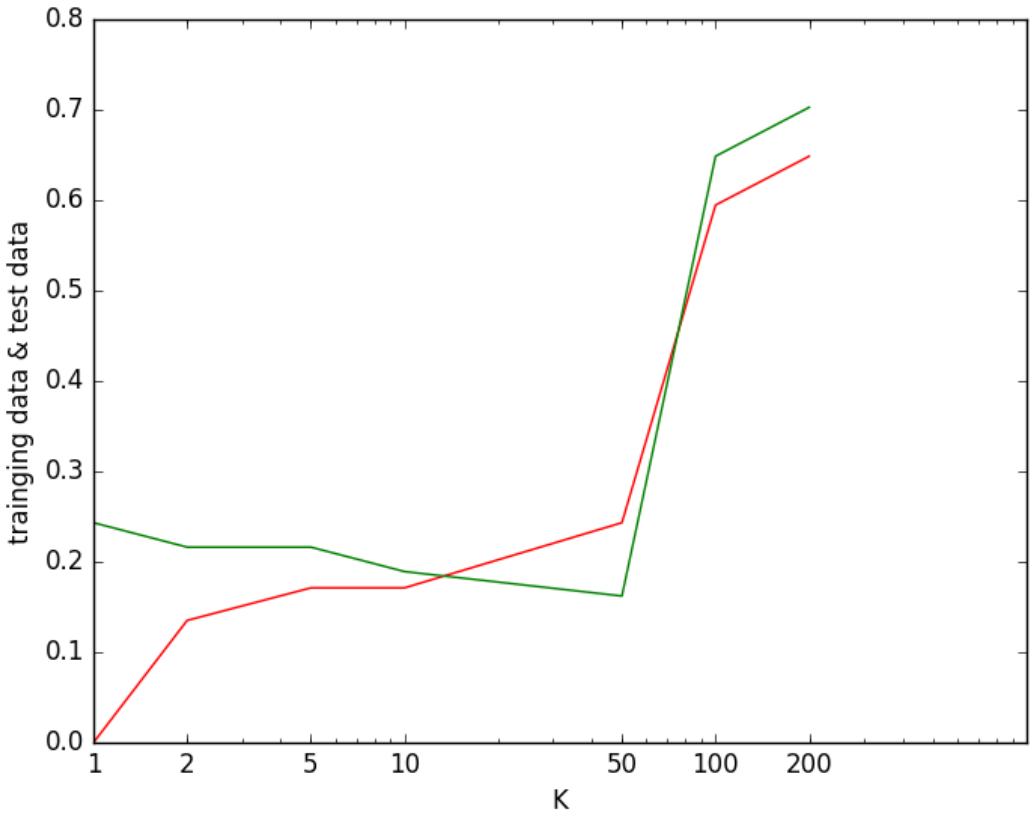
fraction of pred wrong for Training is 0.262548

fraction of pred wrong for Testing is 0.416988

The value of K recommended is 50 based on the plot as the test data error is minimum for that case.

```
K=[1,2,5,10,50,100,200]

errTrain1=[]
errTest1 =[]
K=[1,2,5,10,50,100,200]
count1 =0
count2 =0
Yhattra=[]
Yhatte=[]

for i,k in enumerate(K):
    learner = ml.knn.knnClassify()
    learner.train(Xtr[:, 0:2], Ytr, k)
    Yhattra = learner.predict(Xtr[:,0:2])
    errTrain = []
    for i in range(len(Ytr)):
        errTrain.append (abs(Ytr[i] - Yhattra[i]))
        if ((Ytr[i] - Yhattra[i]) !=0 ):
            count1+=1
    errTrain1.append(np.mean(errTrain))
    Yhatte = learner.predict(Xte[:, 0:2])
    errTest = []
    for i in range(len(Yte)):
        errTest.append(abs(Yte[i] - Yhatte[i]))
        if ((Yte[i] - Yhatte[i]) != 0):
            count2+= 1
    errTest1.append(np.mean(errTest))


print (" fraction of Training pred wrong is %f"%(count1/(7*len(Ytr))))
```

```
print (" fraction of Testing pred wrong is %f"%(count2/(7*len(Yte))))

plt.clf()
plt.semilogx(K,errTrain1,color='red')
plt.semilogx(K,errTest1,color='green')
plt.xticks(K,K)
plt.xlabel('K')
plt.ylabel('trainging data & test data')
plt.show()
```



3a)

First the given data is copied to a file Training_data.txt and the data is extracted using the numpy as below. Then the probability of y (prob_y is a list with keys of possible class values (1,-1)) is calculated as below:

the values are prob_y={1: 0.4, -1: 0.6}

```
import numpy as np


iris = np.genfromtxt("Training_data.txt",delimiter=None) # load the text file
Y = iris[:,-1] # target value is the last column
X = iris[:,0:-1]
```

```
prob_y = {1: list(Y).count(1)/len(Y) , -1: list(Y).count(-1)/len(Y), }
print (prob_y)
```

Now the individual feature probabilities are calculated using the below method cond_prob:

```
        def cond_prob(list1,list0,j):
    for i in range(0,len(Y)):
        if (list(Y)[i] == 1 ):
            if (X[i][j] ==1):
                list1[1] += 1
            else:
                list1[0] += 1
        else:
            if (X[i][j] ==1):
                list0[1] += 1
            else:
                list0[0] += 1
    else:
        list1[0]/=list(Y).count(1)
        list1[1] /= list(Y).count(1)
        list0[0] /= list(Y).count(-1)
        list0[1] /= list(Y).count(-1)
```

In the loop, it calculates the number of times for each feature(say X1) was 1 or 0 when the Y was +1 in the outer if and when Y was -1 in the outer else blocks. And in the end it divides each entry by the length of the observations (which is len of Y) to get the respective probabilities.

The above method is invoked for each feature as below:

prob_x1_yis1 is a list with two keys 0,1(possible values of x1 when y is 1) and the values at these keys represent their probabilities when y is 1.

prob_x1_yis0 is a list with two keys 0,1(possible values of x1 when y is -1) and the values at these keys represent their probabilities when y is -1.

Similarly for all other features x2,x3,x4,x5.

```
cond_prob(prob_x1_yis1,prob_x1_yis0,0)
cond_prob(prob_x2_yis1,prob_x2_yis0,1)
cond_prob(prob_x3_yis1,prob_x3_yis0,2)
cond_prob(prob_x4_yis1,prob_x4_yis0,3)
cond_prob(prob_x5_yis1,prob_x5_yis0,4)
```

The values are

prob_x1_yis1 {0: 0.25, 1: 0.75} prob_x1_yis-1 {0: 0.5, 1: 0.5}

prob_x2_yis1 {0: 1.0, 1: 0.0} prob_x2_yis-1 {0: 0.16666666666666666, 1: 0.8333333333333334}

prob_x3_yis1 {0: 0.25, 1: 0.75} prob_x3_yis-1{0: 0.3333333333333333, 1: 0.6666666666666666}

prob_x4_yis1 {0: 0.5, 1: 0.5} prob_x4_yis-1 {0: 0.16666666666666666, 1: 0.8333333333333334}

prob_x5_yis1 {0: 0.75, 1: 0.25} prob_x5_yis-1{0: 0.6666666666666666, 1: 0.3333333333333333}

3b)

Y is 1 for:X [0, 0, 0, 0, 0]

The below method predictor takes an input observation like X[0 0 0 0 0] and uses the Naïve Bayes formula to calculate the probability of the class 1 and checks if that is >= 0.5 if yes then predicts the value as 1 for that observation else it predicts -1 for that observation.

```
def predictor(inp):
    prob_yis1 = ((prob_y[1] * prob_x1_yis1[inp[0]] * prob_x2_yis1[inp[1]] *
                  prob_x3_yis1[inp[2]] * prob_x4_yis1[inp[3]] *
prob_x5_yis1[inp[4]])
                 / (p_of_X1[inp[0]] *
p_of_X2[inp[1]]*p_of_X3[inp[2]]*p_of_X4[inp[3]]*p_of_X5[inp[4]]))
    if( prob_yis1 >= 0.5):
        print (" Y is 1 for:X %s" % str(inp))
        #print (inp)
    else:
        print(" Y is -1 for X:%s" % str(inp))
        #print (inp)
    print(" pofy=1 %f" % prob_yis1)
    return prob_yis1

predictor([0,0,0,0,0])
```

3c)

posterior prob of Y=1 is 0.000000 for [1, 1, 0, 1, 0]

Again the above method is used to calculate the probability for that.

```
print("posterior prob of Y=1 is %f for %s" % ((predictor([1, 1, 0, 1, 0])), [1, 1,
0, 1, 0]))
```

3d) OverFitting effect

For 5 features, in case of the joint Bayes classifier then the number of entries are exponential 32 (2 power n). But the data that we have is only 10 entries. As we have finite data set and we are computing the probabilities using the empirical probabilities and for some observations we might predict that a particular class never occurs.

3e)

As knowing the author of the email is one of the features based on which we predict to read an email or not. If the address book is not there, then the predicting the class should be done without this feature information. We have to retrain the model. Only the features x2 x3 x4 x5 are used in the prediction and the naïve Bayes model becomes:

P(Y=k|X) = P(y) * **(** p(x2|Y=k) * p(x3|Y=k) * p(x4|Y=k) * p(x5|Y=k) **) / (** p(x2) * p(x3) * p(x4) *p(x5) **)**

148

111

37

214

fraction of Training pred wrong is 0.275418

94

fraction of Testing pred wrong is 0.362934