HW1 Bryan Oliande 13179240 CS 178

January 14, 2017

Problem 1: Python and Data Exploration

```
In [84]: import numpy as np

         import matplotlib.pyplot as plt
         %pylab inline


         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] # features are the other columns
```

```
Populating the interactive namespace from numpy and matplotlib
```

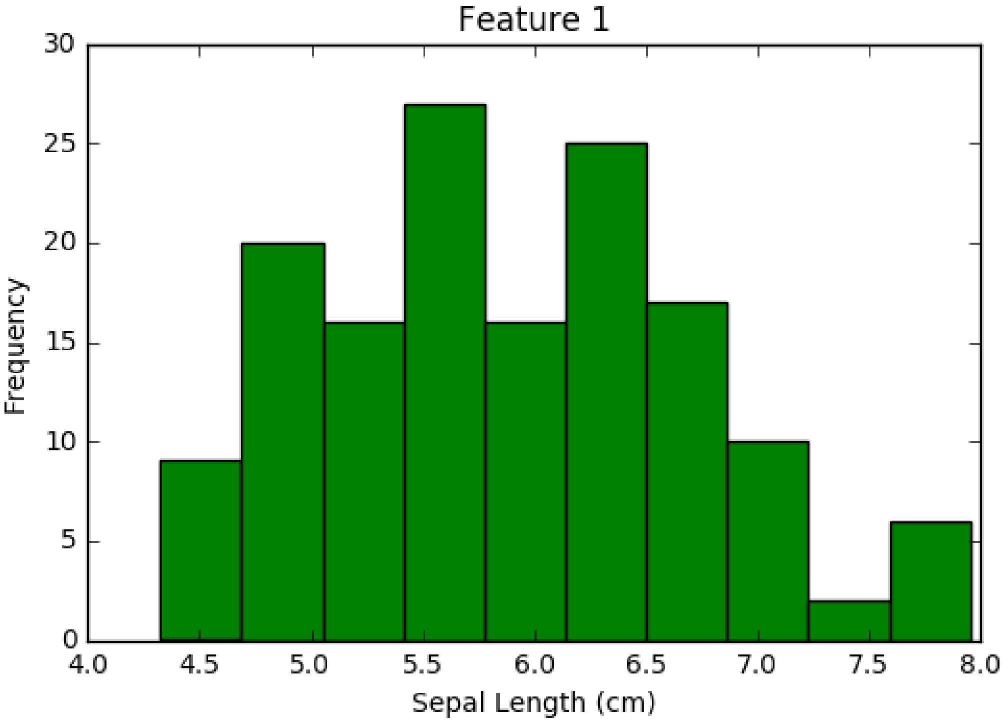
(a) Use X.shape[1] to get the number of features, and X.shape[0] to get the number of data points.

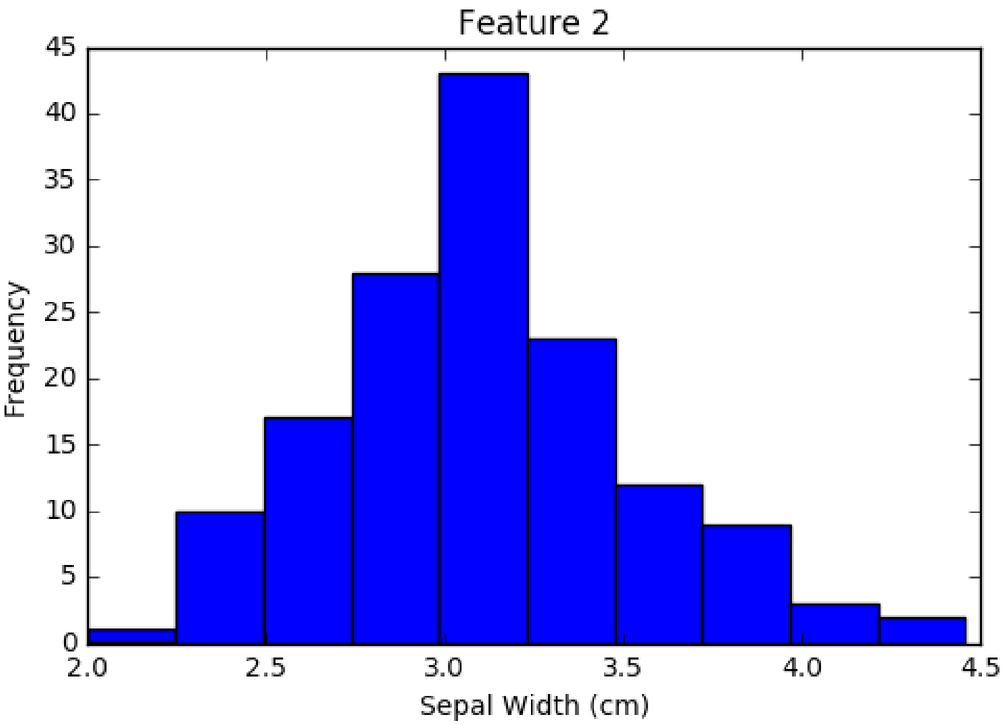
```
In [85]: print(X.shape[1])
         print(X.shape[0])
```

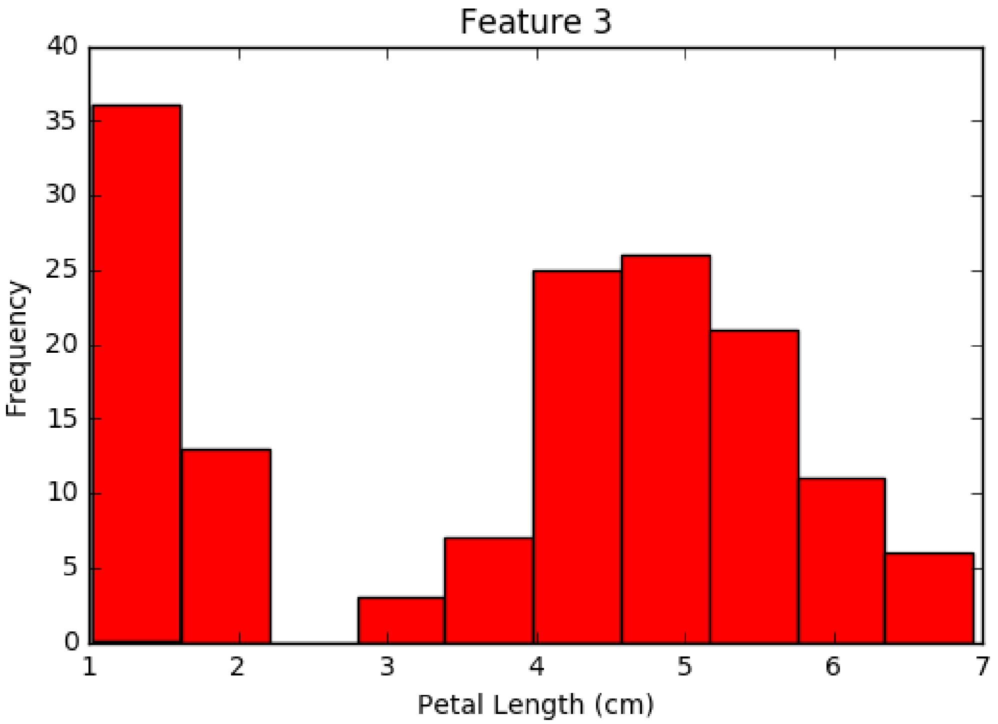
```
4
148
```

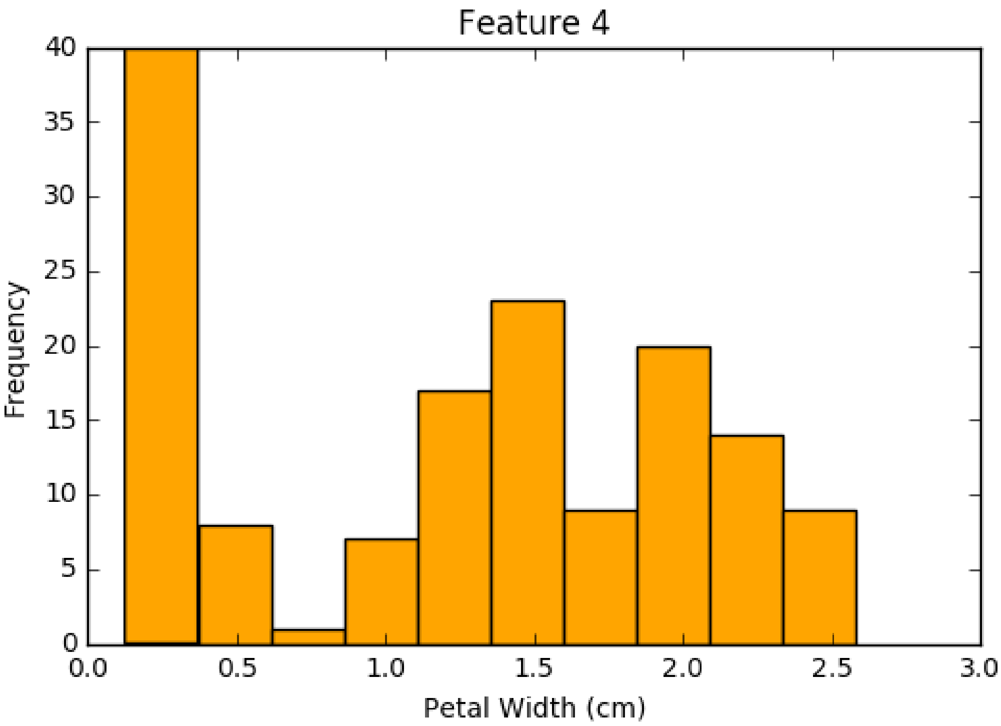
(b) For each feature, plot a histogram ("plt.hist") of the data values

```
In [86]: plt.hist(X[:,0],color='green')
         plt.title("Feature 1")
         plt.xlabel("Sepal Length (cm)")
         plt.ylabel("Frequency")
         plt.show()

         plt.hist(X[:,1])
         plt.title("Feature 2")
         plt.xlabel("Sepal Width (cm)")
plt.ylabel("Frequency")
plt.show()

plt.hist(X[:,2], color='red')
plt.title("Feature 3")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Frequency")
plt.show()

plt.hist(X[:,3], color='orange')
plt.title("Feature 4")
plt.xlabel("Petal Width (cm)")
plt.ylabel("Frequency")
plt.show()
```




Feature 2
Frequency
Sepal Width (cm)
0
5
10
15
20
25
30
35
40
45
2.0
2.5
3.0
3.5
4.0
4.5


Feature 3
Frequency
Petal Length (cm)
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
7



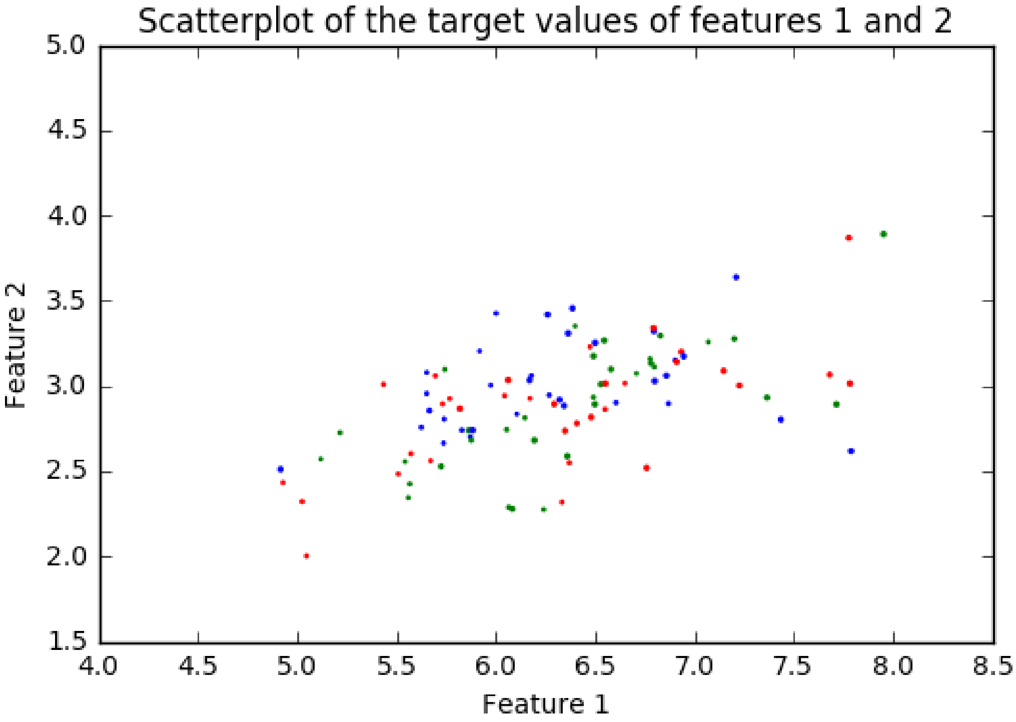
(c) Compute the mean & standard deviation of the data points for each feature (np.mean, np.std)

```
In [87]: f1_mean = np.mean(X[:,0])
         f2_mean = np.mean(X[:,1])
         f3_mean = np.mean(X[:,2])
         f4_mean = np.mean(X[:,3])

         f1_std = np.std(X[:,0])
         f2_std = np.std(X[:,1])
         f3_std = np.std(X[:,2])
         f4_std = np.std(X[:,3])

         print('feature 1 mean:',f1_mean)
         print('feature 1 standard deviation:',f1_std)
         print('feature 2 mean:',f2_mean)
         print('feature 2 standard deviation:',f2_std)
         print('feature 3 mean:',f3_mean)
         print('feature 3 standard deviation:',f3_std)
         print('feature 4 mean:',f4_mean)
         print('feature 4 standard deviation:',f4_std)
```

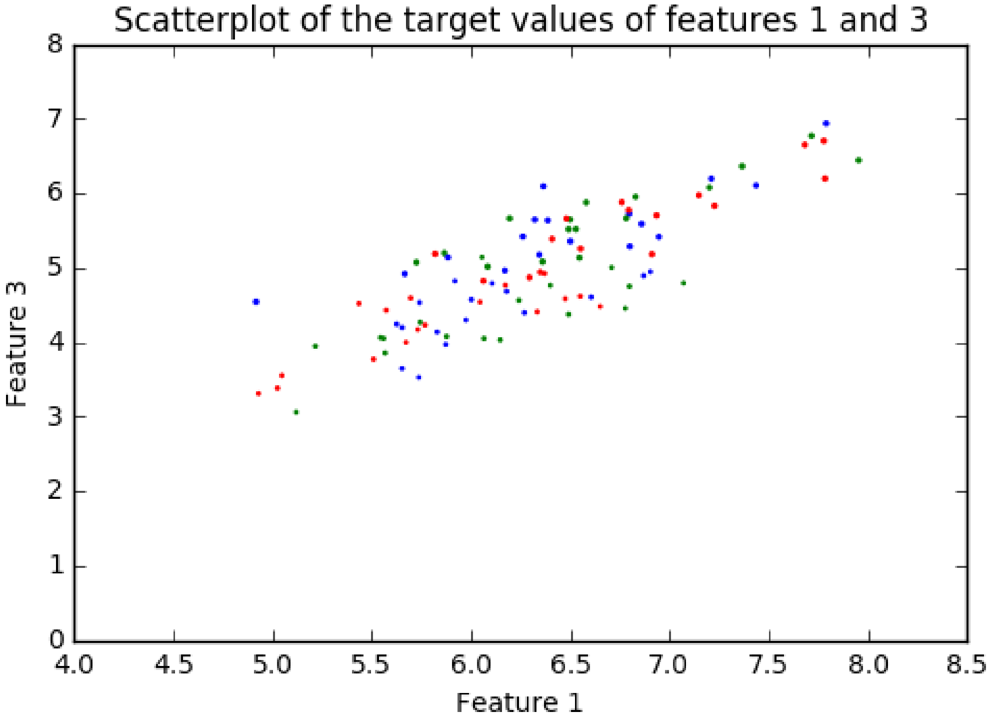
```
feature 1 mean: 5.90010376419
feature 1 standard deviation: 0.833402066775
feature 2 mean: 3.09893091689
feature 2 standard deviation: 0.436291838001
feature 3 mean: 3.81955484054
feature 3 standard deviation: 1.75405710934
feature 4 mean: 1.25255548459
feature 4 standard deviation: 0.758772457026
```

1 (f) For each pair of features (1,2), (1,3), and (1,4), plot a scatterplot (see ‘plt.plot” or “plt.scatter”) of the feature values, colored according to their target value (class). (For example, plot all data points with y = 0 as blue, y = 1 as green, etc.)

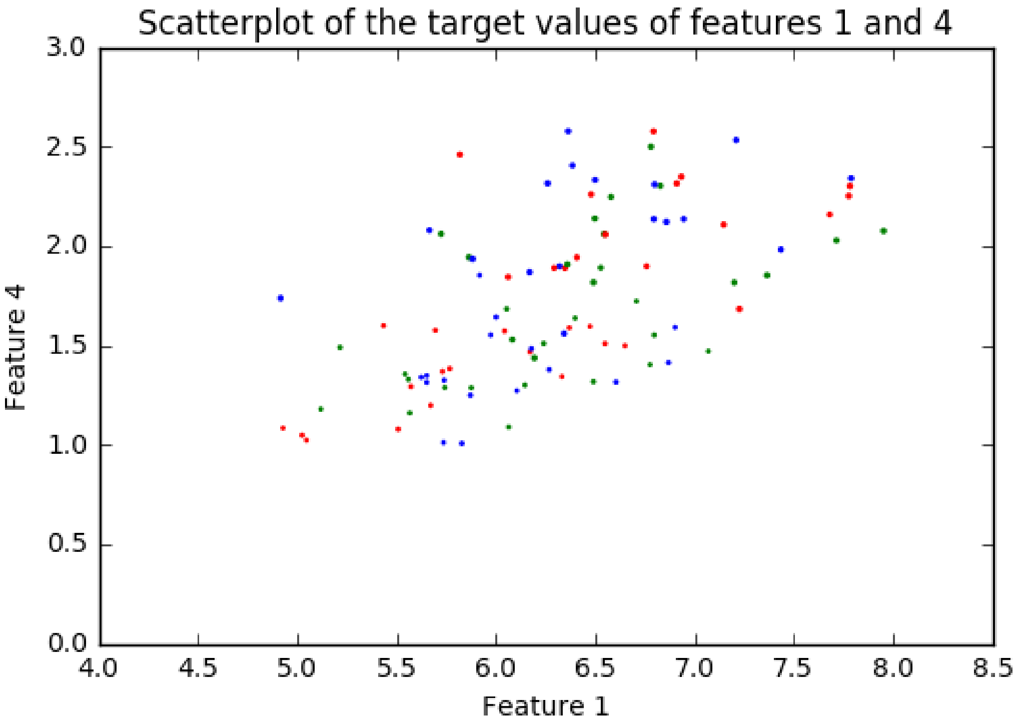
```
In [88]: plt.title("Scatterplot of the target values of features 1 and 2")
         plt.scatter(X[:,0],X[:,1], Y, color = ['blue', 'green', 'red'])
         plt.xlabel("Feature 1")
         plt.ylabel("Feature 2")
         plt.show()

         plt.title("Scatterplot of the target values of features 1 and 3")
         plt.scatter(X[:,0],X[:,2], Y, color = ['blue', 'green', 'red'])
         plt.xlabel("Feature 1")
         plt.ylabel("Feature 3")
         plt.show()

         plt.title("Scatterplot of the target values of features 1 and 4")
         plt.scatter(X[:,0],X[:,3], Y, color = ['blue', 'green', 'red'])
         plt.xlabel("Feature 1")
         plt.ylabel("Feature 4")
         plt.show()
```



Scatterplot of the target values of features 1 and 2
Feature 2
Feature 1
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5



Scatterplot of the target values of features 1 and 3
Feature 3
Feature 1
8
7
6
5
4
3
2
1
0
4.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5

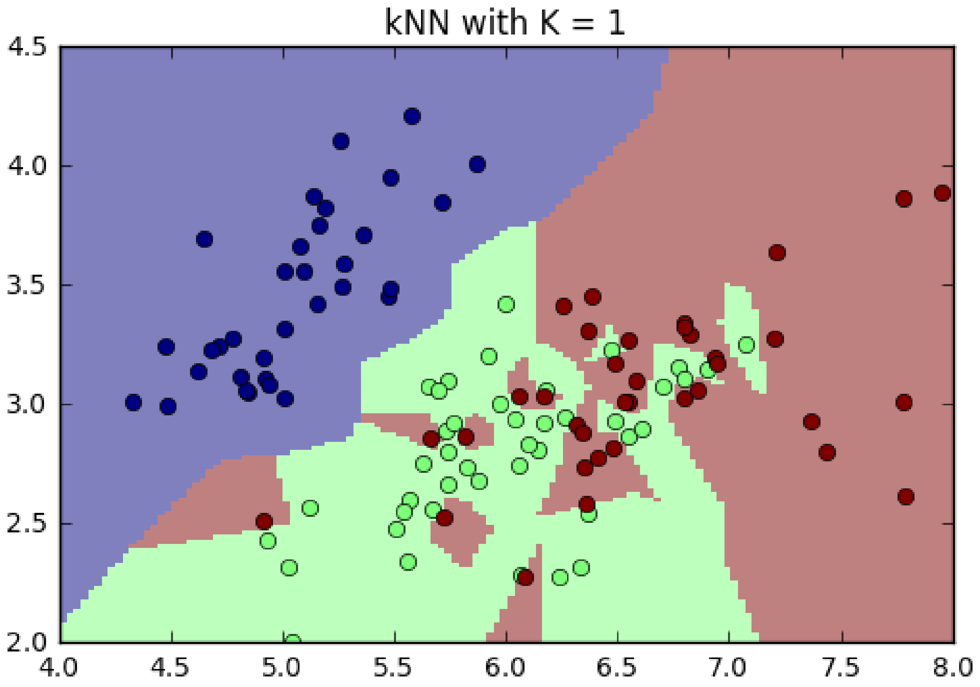


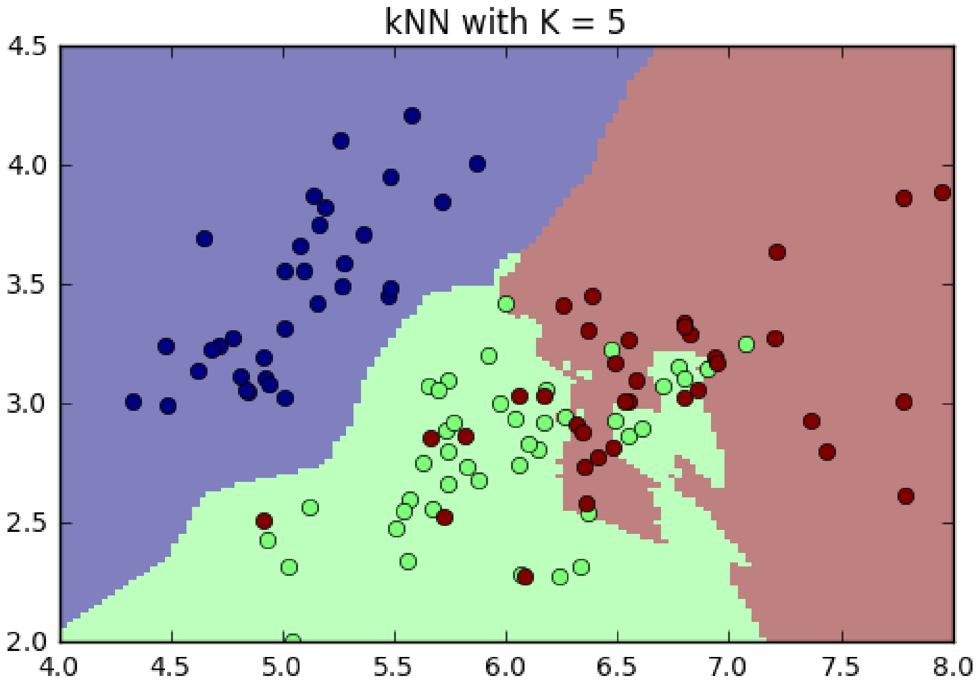
Problem 2: kNN Predictions

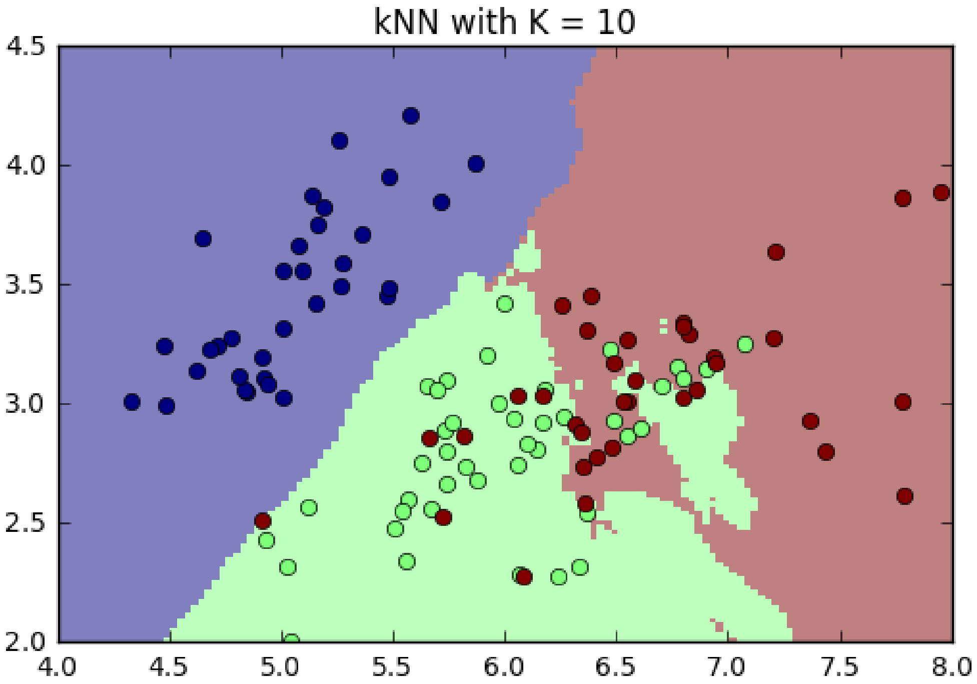
(a)Modify the code listed above (Not included in this writeup) to use only the first two features of X (e.g., let X be only the first two columns of iris, instead of the first four), and visualize (plot) the classification boundary for varying values of K = [1, 5, 10, 50] using plotClassify2D.

```
In [49]: import numpy as np
         import matplotlib.pyplot as plt
         iris = np.genfromtxt("data/iris.txt",delimiter=None)


         # load the data
         Y = iris[:,-1]
         X = iris[:,0:2]
         import mltools as ml

         # We'll use some data manipulation routines in the provided class code
         # (This is a good idea in case your data are ordered in some pathological
         # as the Iris data are)
         X,Y = ml.shuffleData(X,Y); # shuffle data randomly
         Xtr,Xte,Ytr,Yte = ml.splitData(X,Y, 0.75); # split data into 75/25 train/t

         K = [1, 5, 10, 50];

         for k in K:
```

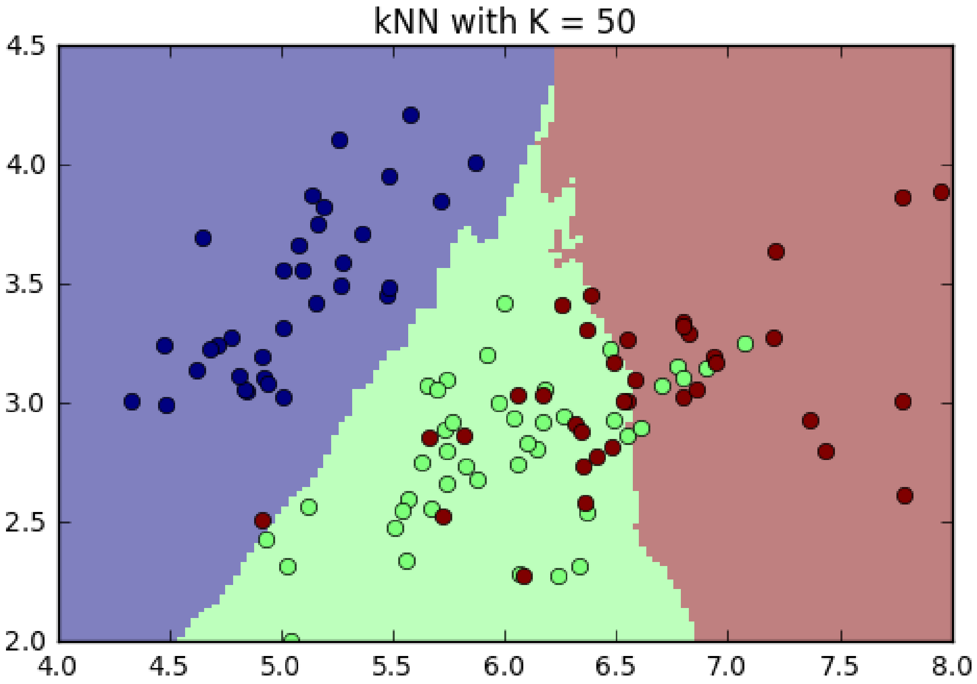
```
knn = ml.knn.knnClassify() # create the object and train it
knn.train(Xtr, Ytr, k) # where K is an integer, e.g. 1 for nearest nei
ml.plotClassify2D(knn, Xtr, Ytr ) #make 2D classification plot with da
plt.title("kNN with K = " + str(k))
plt.show()
```


kNN with K = 1


kNN with K = 5



kNN with K = 10

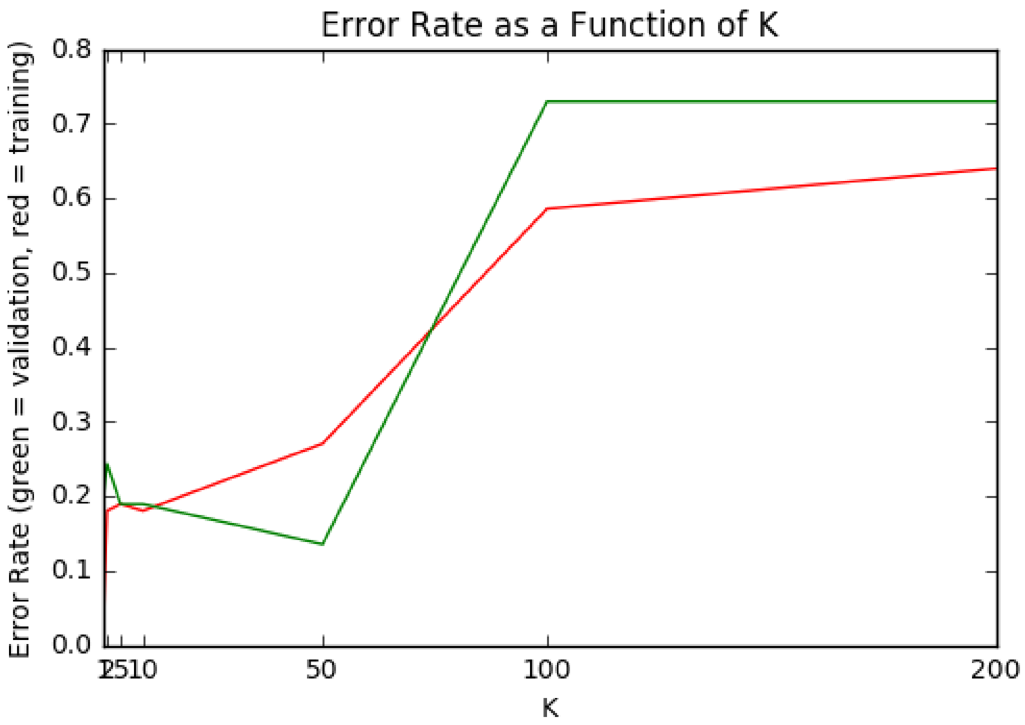


kNN with K = 50

2 (b) Again using only the first two features, compute the error rate (number of misclassifications) on both the training and test data as a function of K = [1, 2, 5, 10, 50, 100, 200]. You can do this most easily with a for-loop.

Plot the resulting error rate function using the semi-log plot(“semilogx”), with training error in red and validation error in green. Based on these plots what value of K would you recommend?

```
In [78]: import numpy as np
         import matplotlib.pyplot as plt
         iris = np.genfromtxt("data/iris.txt",delimiter=None)


         # load the data
         Y = iris[:,-1]
         X = iris[:,0:2]
         import mltools as ml

         # We'll use some data manipulation routines in the provided class code
         # (This is a good idea in case your data are ordered in some pathological
         # as the Iris data are)
         X,Y = ml.shuffleData(X,Y); # shuffle data randomly
         Xtr,Xte,Ytr,Yte = ml.splitData(X,Y, 0.75); # split data into 75/25 train/t


         K=[1,2,5,10,50,100,200];
         errTrain = [None] * len(K)
         errVal = [None] * len(K)

         for i,k in enumerate(K):

             learner = ml.knn.knnClassify() # create the object and train it
             learner.train(Xtr, Ytr, k) # where K is an integer, e.g. 1 for nearest
             errTrain[i] = learner.err(Xtr, Ytr)
             errVal[i] = learner.err(Xte, Yte)



         fig = plt.figure()
         ax = fig.add_subplot(1,1,1)
         major_ticks = [1,2,5,10,50,100,200]
         ax.set_xticks(major_ticks)


         plt.title("Error Rate as a Function of K")
         plt.xlabel("K")
         plt.ylabel("Error Rate (green = validation, red = training)")
         plt.plot(K, errTrain, color = 'red')
         plt.plot(K, errTest, color = 'green')
         plt.show()
```



Based on the above plot, I would choose K = 50 because it is the value of K that causes the minimum error rate on the validation data.

Problem 3: Naive Bayes Classifiers

a)

P(y = 1) = 4/10 P(y = -1) = 6/10
P(x1 = 0 | y = -1) = 3/6 P(x1 = 1 | y = -1) = 3/6 P(x1 = 0 | y = 1) = 1/4 P(x1 = 1 | y = 1) = 3/4
P(x2 = 0 | y = -1) = 1/6 P(x2 = 1 | y = -1) = 5/6 P(x2 = 0 | y = 1) = 4/4 P(x2 = 1 | y = 1) = 0/4
P(x3 = 0 | y = -1) = 2/6 P(x3 = 1 | y = -1) = 4/6 P(x3 = 0 | y = 1) = 1/4 P(x3 = 1 | y = 1) = 3/4
P(x4 = 0 | y = -1) = 1/6 P(x4 = 1 | y = -1) = 5/6 P(x4 = 0 | y = 1) = 2/4 P(x4 = 1 | y = 1) = 2/4
P(x5 = 0 | y = -1) = 4/6 P(x5 = 1 | y = -1) = 2/6 P(x5 = 0 | y = 1) = 3/4 P(x5 = 1 | y = 1) = 1/4

b)

For class X = (0 0 0 0 0):

We calculate P(y = 1)P(X = [0 0 0 0 0] | y = 1) and compare it to P(y = -1)P(X = [0 0 0 0 0] | y = -1) and then predict the more likely outcome.

P(y = 1)P(X = [0 0 0 0 0] | y = 1) = P(y = 1) P(x1 = 0 | y =1) P(x2 = 0 | y =1) P(x3 = 0 | y =1) P(x4 = 0 | y = 1) P(x5 = 0 | y = 1) = (4/10) * (1/4) * (4/4) * (1/4) * (2/4) * (3/4) = 0.009375

P(y = -1)P(X = [0 0 0 0 0] | y = -1) P(y = -1) P(x1 = 0 | y = -1) P(x2 = 0 | y = -1) P(x3 = 0 | y = -1) P(x4 = 0 | y = -1) P(x5 = 0 | y = -1) = (6/10) * (3/6) * (1/6) * (2/6) * (1/6) * (4/6) = 0.00185185

We predict y = 1

For class X = (1 1 0 1 0):

We calculate P(y = 1)P(X = [1 1 0 1 0] | y = 1) and compare it to P(y = -1)P(X = [1 1 0 1 0] | y = -1) and then predict the more likely outcome

P(y = 1)P(X = [1 1 0 1 0] | y = 1) = P(y = 1) P(x1 = 1 | y =1) P(x2 = 1 | y =1) P(x3 = 0 | y =1) P(x4 = 1 | y = 1) P(x5 = 0 | y = 1) = (4/10) * (3/4) * (0/4) * doesn't matter = 0

P(y = -1)P(X = [1 1 0 1 0] | y = -1) = P(y = -1) P(x1 = 1 | y = -1) P(x2 = 1 | y = -1) P(x3 = 0 | y = -1) P(x4 = 1 | y = -1) P(x5 = 0 | y = -1) = (6/10) * (3/6) * (5/6) * (2/6) * (5/6) * (4/6) = 0.04629629

We predict y = -1

c)

The posterior probability that y = +1 given the observation X = (1 1 0 1 0) is

P(y = 1 | X = [1 1 0 1 0]) =

[P(X = [1 1 0 1 0] | y = 1) P(y =1)]/ P(X = [1 1 0 1 0] = 0/doesn't matter = 0

d) We should probably not use a "joint" Bayes classifier for these data because we will need 2ˆ5 parameters. We would need to make an estimate for every possible combination of features. If we used Naive Bayes, we only need to estimate each feature individually and can then compute the probability by multiplying out the estimates.

e) Because we are using a Naive Bayes classifier, we only need to eliminate all occurences of:
P(x1 = 0 | y = -1) = 3/6
P(x1 = 1 | y = -1) = 3/6 P(x1 = 0 | y = 1) = 1/4
P(x1 = 1 | y = 1) = 3/4

in our calculations above if we wanted to make new predictions over our new model. In Naive Bayes, the features are independent. Our new X is X = [x2 x3 x4 x5] and our Y remains the same. All of the calculations will be the same except for the eliminations. So, re-training our model is fairly simple.