Homework1

January 18, 2018

```
In [1]: # Student: John Lu
        # StudentID: 21796269

        import numpy as np
        import matplotlib.pyplot as plt

        %matplotlib inline

In [2]: iris = np.genfromtxt("data/iris.txt", delimiter=None)
        Y = iris[:, -1]
        X = iris[:, 0:-1]
```

1 Problem 1: Python & Data Exploration

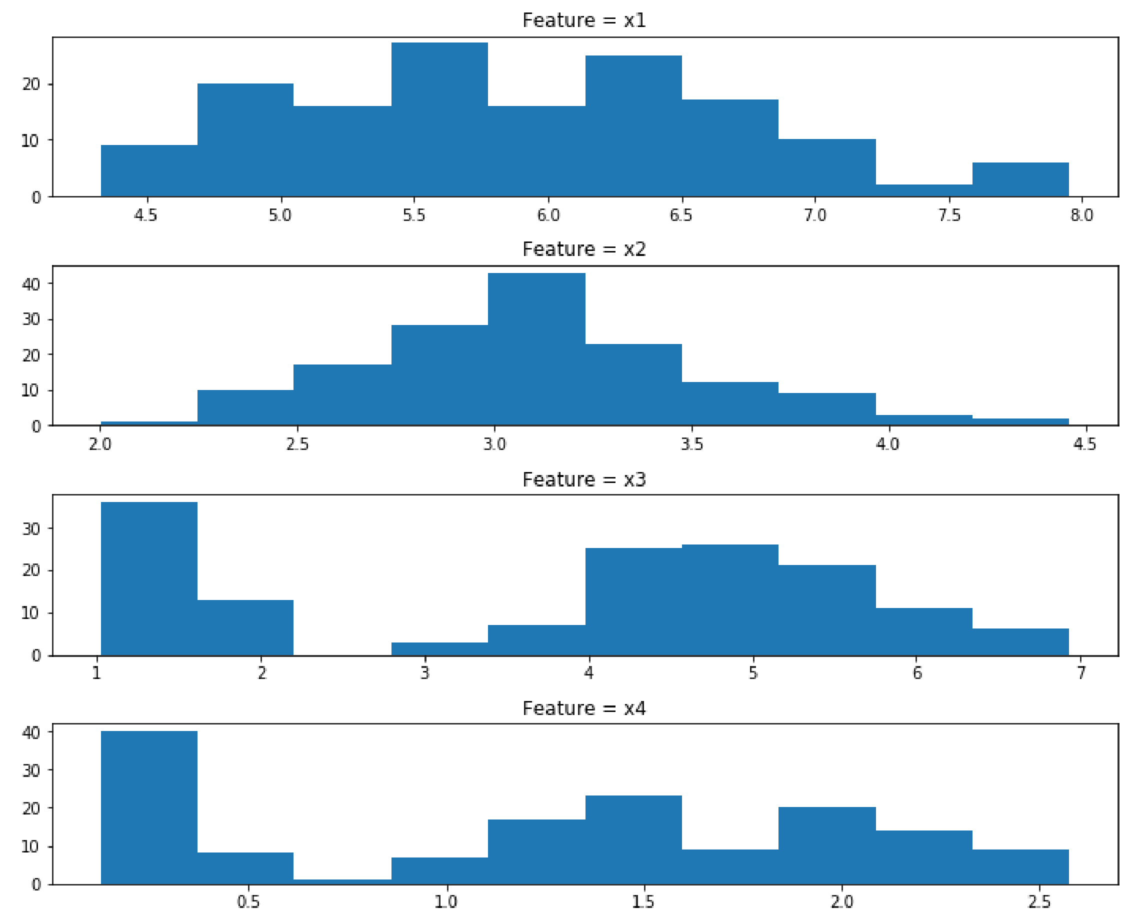
1.1 Part 1

```
In [3]: num_data = X.shape[0]
        num_features = X.shape[1]
        print('Number of Data Points: ' + str(num_data))
        print('Number of Features: ' + str(num_features))
```

```
Number of Data Points: 148
Number of Features: 4
```

1.2 Part 2

```
In [4]: fig = plt.figure(figsize=(10,10))
        for i in range(num_features):
            ax = plt.subplot(5,1,i+1)
            plt.hist(x=X[:, i])
            ax.set_title('Feature = ' + str("x") + str(i+1))
        plt.tight_layout()
```



1.3 Part 3

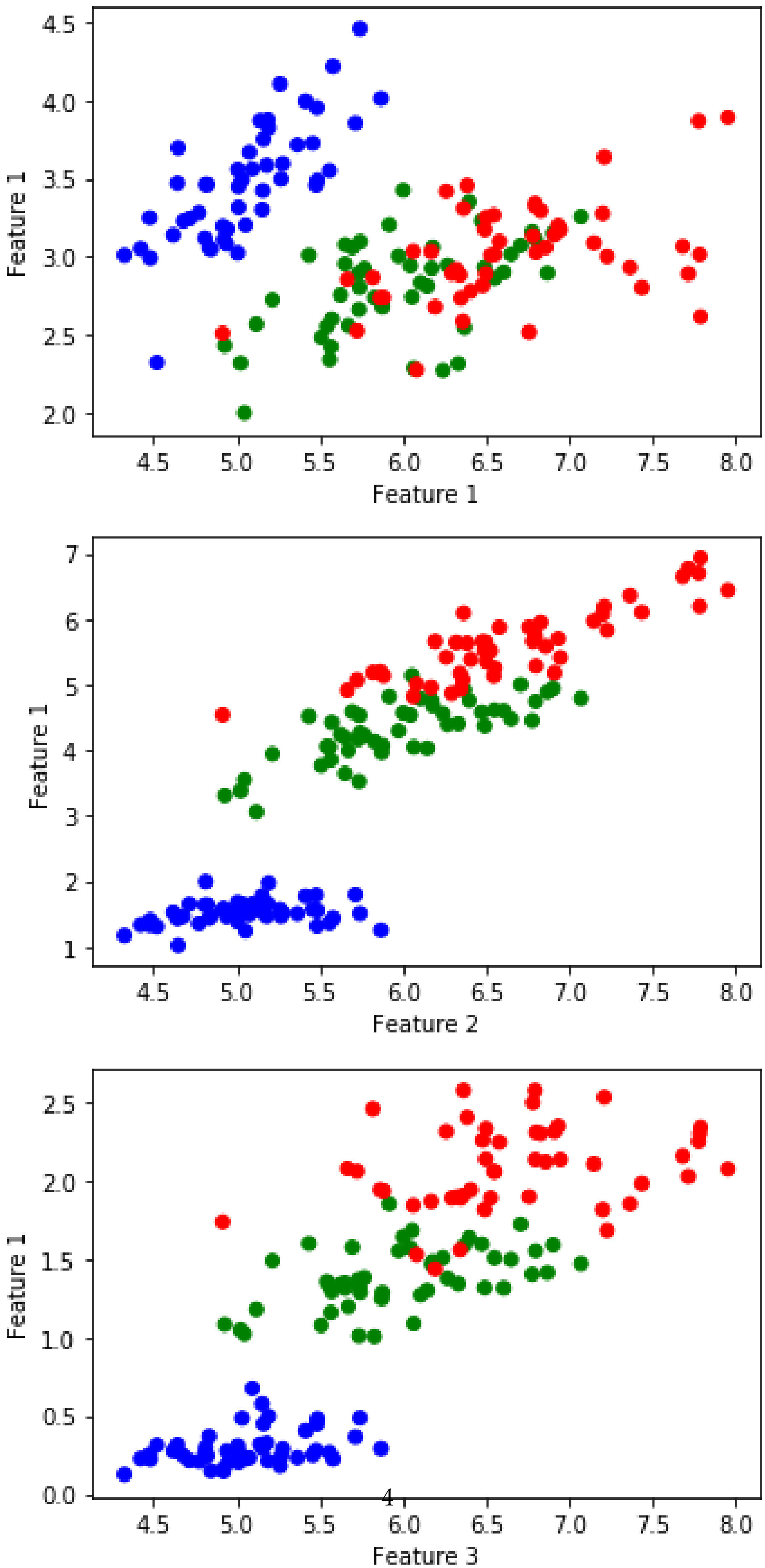
```
In [5]: mean = X.mean(axis = 0)
        std = X.std(axis = 0)

        for i in range(num_features):
            print('Feature {:d} has Mean = {:f} and Std. Dev = {:f}' \
                  .format(i+1,mean[i], std[i]))
```

```
Feature 1 has Mean = 5.900104 and Std. Dev = 0.833402
Feature 2 has Mean = 3.098931 and Std. Dev = 0.436292
Feature 3 has Mean = 3.819555 and Std. Dev = 1.754057
Feature 4 has Mean = 1.252555 and Std. Dev = 0.758772
```

Plot scatter plots of features 2, 3, and 4 against feature 1. All data points with y = 0 are blue, y = 1 are green, and y = 2 are red

```
In [6]: # Our color scheme
        colors = ['blue', 'green', 'red']
```

```
color_arr = [colors[int(Y[i])] for i in range(num_data)]

feat_one = X[:,0]

plt.subplots(3,1, figsize=(5,10))
for i in range(num_features-1):
    ax = plt.subplot(3,1,i+1)
    ax.set_xlabel('Feature ' + str(i+1))
    ax.set_ylabel('Feature 1')
    plt.scatter(x=feat_one, y=X[:,i+1], c=color_arr)

plt.tight_layout()
```


4.5
4.0
3.5
3.0
2.5
2.0
Feature 1
4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
Feature 1
7
6
5
4
3
2
1
Feature 1
4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
Feature 2
2.5
2.0
1.5
1.0
0.5
0.0
Feature 1
4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
Feature 3

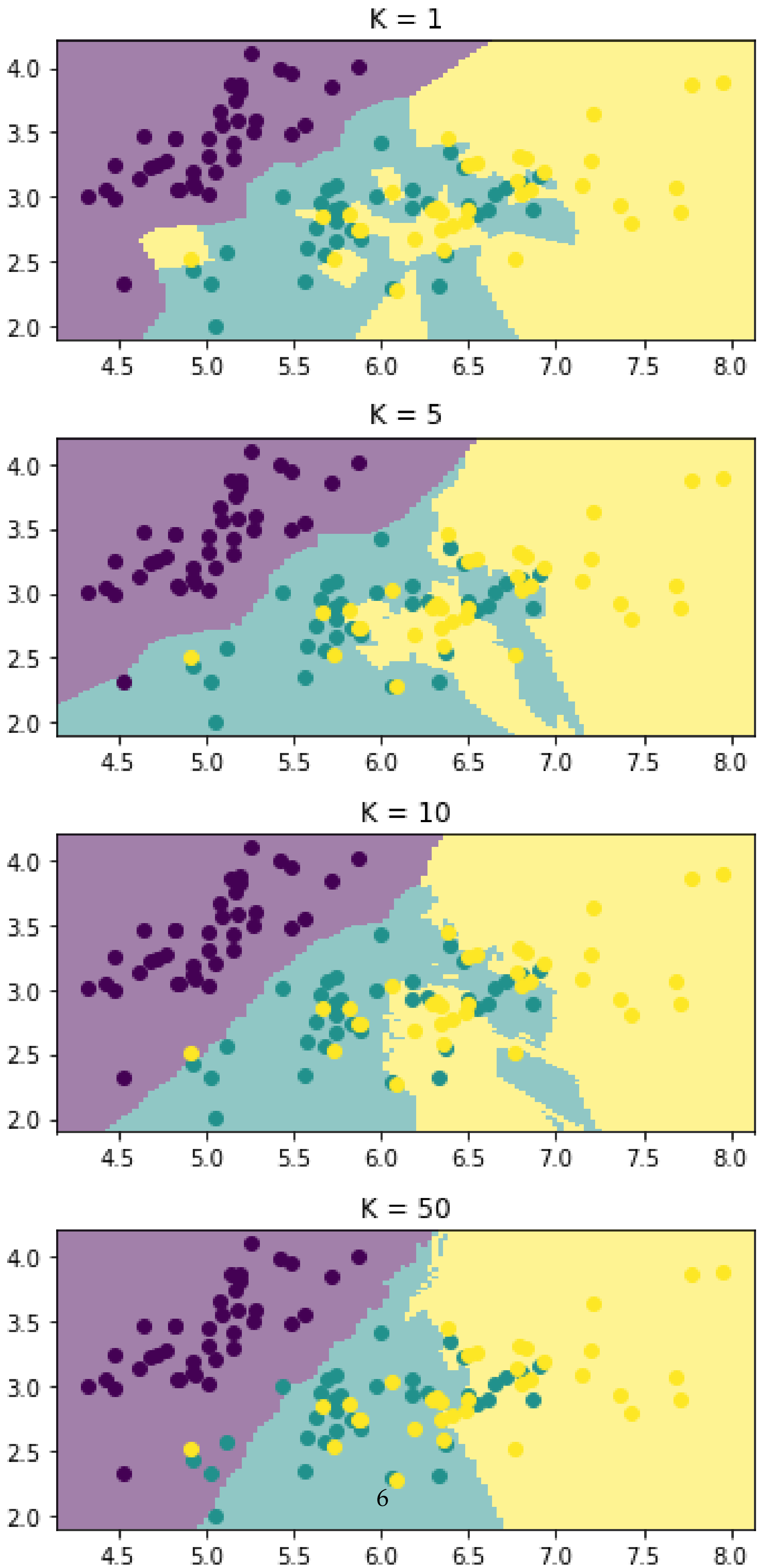
2 Problem 2: k-nearest-neighbor predictions

2.1 Part 1

In this section, we use the first two features to of the iris dataset to train a knn classifier and plot the decision boundary for K = [1, 5, 10, 50].

In [7]:
```
import mltools as ml

iris = np.genfromtxt("data/iris.txt", delimiter=None)
Y = iris[:, -1]
X = iris[:, 0:2]

np.random.seed(0)
X, Y = ml.shuffleData(X, Y)
X_train, X_val, Y_train, Y_val = ml.splitData(X, Y, 0.75)
```

In [8]:
```
K = [1,5,10,50]

plt.subplots(figsize=(5,10))
for i, k in enumerate(K):
    ax = plt.subplot(len(K), 1, i+1)
    ax.set_title('K = ' + str(k))

    knn = ml.knn.knnClassify()
    knn.train(X_train, Y_train, k)
    ml.plotClassify2D(knn, X_train, Y_train)

plt.tight_layout()
```

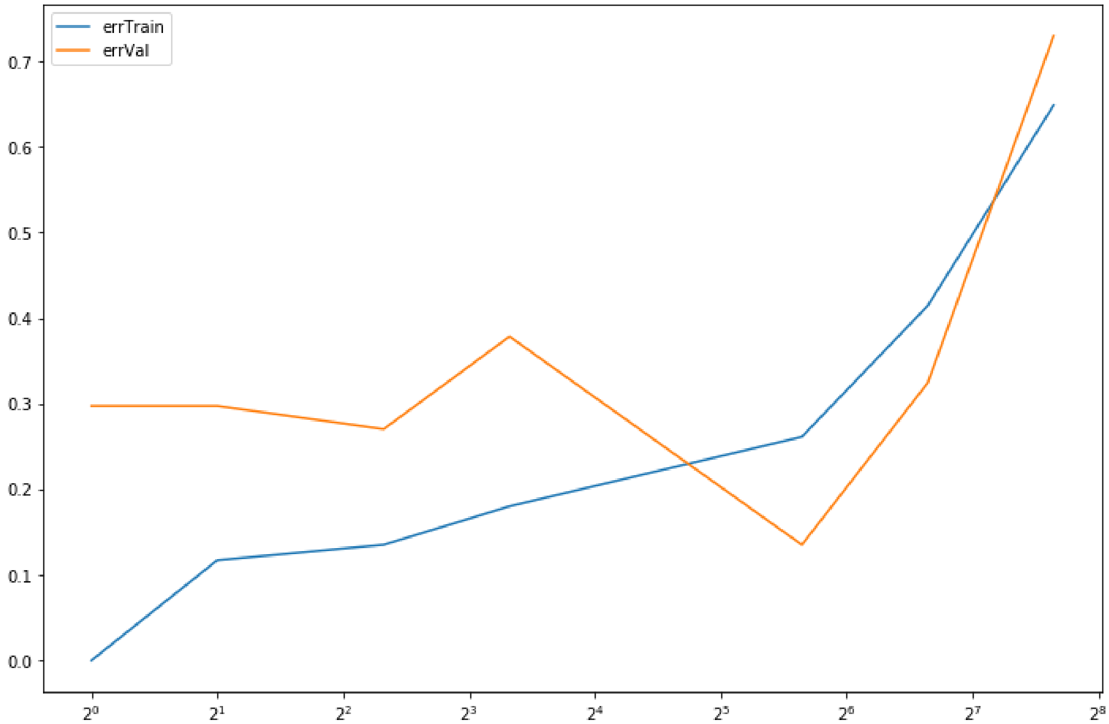

K = 1
K = 5
K = 10
K = 50
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

2.2 Part 2

We now train on all features and compute the error rate on both the training and validation sets for K = [1, 2, 5, 10, 50 , 100, 200]

Error is defined by: $\frac{\textit{\# correct predictions}}{\textit{\# data in set}}$

```
In [9]: def plot_errors(X_train, Y_train, X_val, Y_val, K):
            """
            This function takes as input the training and validation data
            and an array, K, of integers and plots the knn_classifier
            error against K.
            """

            errTrain = []
            errVal = []
            plt.figure(figsize=(12,8))
            for i, k in enumerate(K):
                knn = ml.knn.knnClassify()
                knn.train(X_train, Y_train, k)

                # save the training error
                Yhat_tr = knn.predict(X_train)
                errTrain.append((Yhat_tr != Y_train).sum() / len(Y_train))

                # save the validation error
                Yhat_val = knn.predict(X_val)
                errVal.append((Yhat_val != Y_val).sum() / len(Y_val))
            plt.semilogx(K, errTrain, label="errTrain", basex=2)
            plt.semilogx(K, errVal, label="errVal", basex=2)

            # Create a legend
            ax = plt.gca()
            handles, labels = ax.get_legend_handles_labels()
            ax.legend(handles, labels)

            # In case further modification is needed
            return ax

In [10]: K = [1,2,5,10,50,100,200]
         plot_errors(X_train, Y_train, X_val, Y_val, K);
```

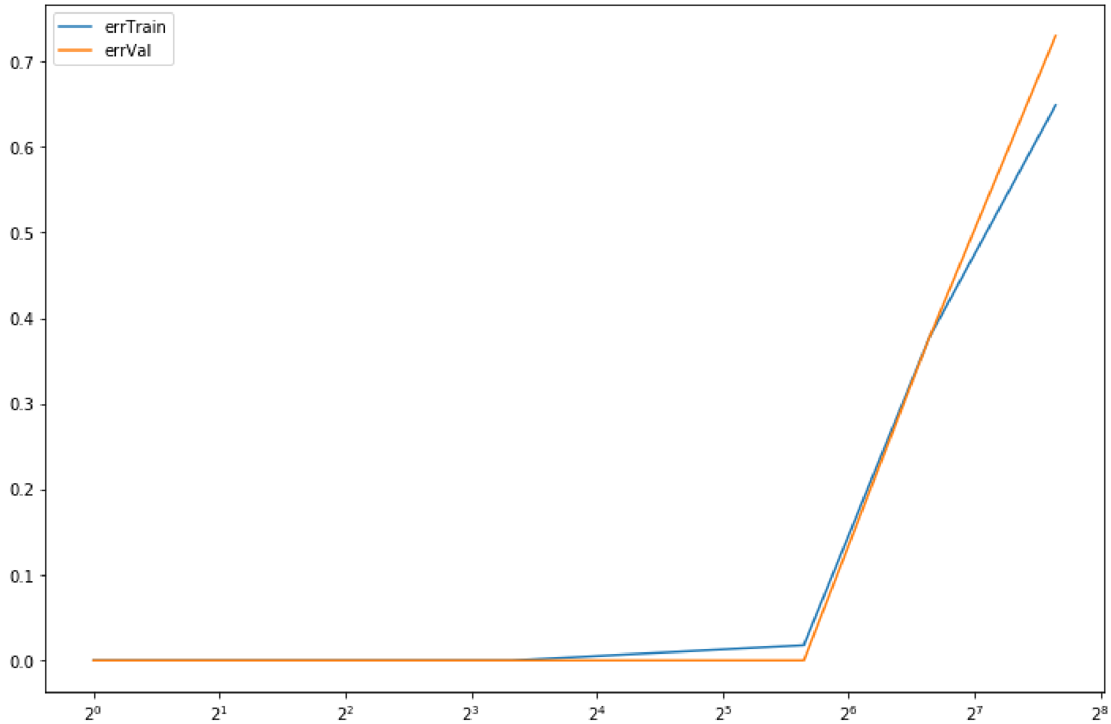


Based on the plots, I would recommend experiementing with k values between 32 and 64, where the validation error is lowest.

2.3 Part 3

Create the same error plots but using all features this time instead of just the first two.

```
In [11]: iris = np.genfromtxt("data/iris.txt", delimiter=None)
         Y = iris[:, -1]
         X = iris[:]

         np.random.seed(0)
         X, Y = ml.shuffleData(X, Y)
         X_train, X_val, Y_train, Y_val = ml.splitData(X, Y, 0.75)
         ax = plot_errors(X_train, Y_train, X_val, Y_val, K)
```



The plot is quite different from before. I would recommend experimenting with k between 1 and 8 (small k) as both the validation and training error are low in that region.

3 Problem 3: Naive Bayes Classifiers

3.1 Part 1

We will first load the data into a numpy array for easier processing.

```
In [12]: emails = np.array([[0,0,1,1,0,-1],
                            [1,1,0,1,0,-1],
                            [0,1,1,1,1,-1],
                            [1,1,1,1,0,-1],
                            [0,1,0,0,0,-1],
                            [1,0,1,1,1,1],
                            [0,0,1,0,0,1],
                            [1,0,0,0,0,1],
                            [1,0,1,1,0,1],
                            [1,1,1,1,1,-1]])
         emails
         num_email = len(emails)
         print(emails)
         print(num_email)
```

```
[[ 0  0  1  1  0 -1]
 [ 1  1  0  1  0 -1]
 [ 0  1  1  1  1 -1]
 [ 1  1  1  1  0 -1]
 [ 0  1  0  0  0 -1]
 [ 1  0  1  1  1  1]
 [ 0  0  1  0  0  1]
 [ 1  0  0  0  0  1]
 [ 1  0  1  1  0  1]
 [ 1  1  1  1  1 -1]]
10
```

Let us define some useful probability functions that we will use later.

```
In [13]: def cond_prob(feat, feat_val, given, given_val):
             """
             This function calculates the conditional probability of a feature
             taking any specified value given the value of another specified variable.
             """
             num_email_cond = (emails[:, given] == given_val).sum()
             temp = np.logical_and(emails[:,given] == given_val,
                                   emails[:,feat-1] == feat_val).sum()
             return temp / num_email_cond
```

```
In [14]: def prob_y(y=1):
             """
             This function returns the marginal probabilty of y being a specified value.
             """
             y_eq_1 = (emails[:,5] == 1).sum()
             y_eq_minus_1 = num_email - y_eq_1
             prob = y_eq_1 / num_email
             return prob if y == 1 else 1-prob
```

Using the above functions, we calculate the required probabilities.

```
In [15]: print('-----P(Y)-----')
         print('p(y=1)  = ' + str(prob_y(1)))
         print('p(y=-1)  = ' + str(prob_y(-1)))

         print('-----P(X_1 | Y)-----')
         print('p(x_1=1 | y=1)  = ' \
               + str(cond_prob(feat=1, feat_val=1, given=5, given_val=1)))
         print('p(x_1=0 | y=1)  = ' \
               + str(cond_prob(feat=1, feat_val=0, given=5, given_val=1)))
         print('p(x_1=1 | y=-1)  = ' \
               + str(cond_prob(feat=1, feat_val=1, given=5, given_val=-1)))
         print('p(x_1=0 | y=-1)  = ' \
               + str(cond_prob(feat=1, feat_val=0, given=5, given_val=-1)))
```

```
        print('-----P(X_2 | Y)-----')
        print('p(x_2=1 | y=1) = ' \
              + str(cond_prob(feat=2, feat_val=1, given=5, given_val=1)))
        print('p(x_2=0 | y=1) = ' \
              + str(cond_prob(feat=2, feat_val=0, given=5, given_val=1)))
        print('p(x_2=1 | y=-1) = ' \
              + str(cond_prob(feat=2, feat_val=1, given=5, given_val=-1)))
        print('p(x_2=0 | y=-1) = ' \
              + str(cond_prob(feat=2, feat_val=0, given=5, given_val=-1)))

        print('-----P(X_3 | Y)-----')
        print('p(x_3=1 | y=1) = ' \
              + str(cond_prob(feat=3, feat_val=1, given=5, given_val=1)))
        print('p(x_3=0 | y=1) = ' \
              + str(cond_prob(feat=3, feat_val=0, given=5, given_val=1)))
        print('p(x_3=1 | y=-1) = ' \
              + str(cond_prob(feat=3, feat_val=1, given=5, given_val=-1)))
        print('p(x_3=0 | y=-1) = ' \
              + str(cond_prob(feat=3, feat_val=0, given=5, given_val=-1)))

        print('-----P(X_4 | Y)-----')
        print('p(x_4=1 | y=1) = ' \
              + str(cond_prob(feat=4, feat_val=1, given=5, given_val=1)))
        print('p(x_4=0 | y=1) = ' \
              + str(cond_prob(feat=4, feat_val=0, given=5, given_val=1)))
        print('p(x_4=1 | y=-1) = ' \
              + str(cond_prob(feat=4, feat_val=1, given=5, given_val=-1)))
        print('p(x_4=0 | y=-1) = ' \
              + str(cond_prob(feat=4, feat_val=0, given=5, given_val=-1)))

        print('-----P(X_5 | Y)-----')
        print('p(x_5=1 | y=1) = ' \
              + str(cond_prob(feat=5, feat_val=1, given=5, given_val=1)))
        print('p(x_5=0 | y=1) = ' \
              + str(cond_prob(feat=5, feat_val=0, given=5, given_val=1)))
        print('p(x_5=1 | y=-1) = ' \
              + str(cond_prob(feat=5, feat_val=1, given=5, given_val=-1)))
        print('p(x_5=0 | y=-1) = ' \
              + str(cond_prob(feat=5, feat_val=0, given=5, given_val=-1)))
```

```
-----P(Y)-----
p(y=1) = 0.4
p(y=-1) = 0.6
-----P(X_1 | Y)-----
p(x_1=1 | y=1) = 0.75
p(x_1=0 | y=1) = 0.25
p(x_1=1 | y=-1) = 0.5
```

```
p(x_1=0 | y=-1) = 0.5
-----P(X_2 | Y)-----
p(x_2=1 | y=1) = 0.0
p(x_2=0 | y=1) = 1.0
p(x_2=1 | y=-1) = 0.833333333333
p(x_2=0 | y=-1) = 0.166666666667
-----P(X_3 | Y)-----
p(x_3=1 | y=1) = 0.75
p(x_3=0 | y=1) = 0.25
p(x_3=1 | y=-1) = 0.666666666667
p(x_3=0 | y=-1) = 0.333333333333
-----P(X_4 | Y)-----
p(x_4=1 | y=1) = 0.5
p(x_4=0 | y=1) = 0.5
p(x_4=1 | y=-1) = 0.833333333333
p(x_4=0 | y=-1) = 0.166666666667
-----P(X_5 | Y)-----
p(x_5=1 | y=1) = 0.25
p(x_5=0 | y=1) = 0.75
p(x_5=1 | y=-1) = 0.333333333333
p(x_5=0 | y=-1) = 0.666666666667
```

3.2 Part 2

By Bayes' Rule, $p(y|x) = \frac{p(x|y)*p(y)}{p(x)}$. We use the the conditional probabilities above to computer our answer.

Now, by applying the law of total probability and considering that all x_1 conditionally independent given y, we have $p(y|x) = \frac{p(x_1|y)p(x_2|y)...p(x_5|y)p(y)}{\alpha}$ where $\alpha = p(x|y = 1)p(y = 1) + p(x|y = -1)p(y = -1)$

```
In [16]: def prob_x_vec_given_y(x, y):
             """
             Returns the probability of an observation given a specified value of y
             """
             prob = 1
             for i in range(len(x)):
                 feat_num = i+1
                 prob = prob*cond_prob(feat=feat_num, feat_val=x[i], given=5, given_val=y)
             return prob
```

```
In [17]: def prob_y_given_x(y, given_x):
             """
             Returns the probability of y being a specified value given an observation x
             """
             alpha = prob_x_vec_given_y(given_x, 1)*prob_y(1) \
                 + prob_x_vec_given_y(given_x, -1)*prob_y(-1)
```

```
        numerator = prob_x_vec_given_y(given_x, y)*prob_y(y)

        return numerator / alpha
```

```
In [18]: x = (0,0,0,0,0)
         print("prob y=1 given x=%s is %s" % (x,prob_y_given_x(1, x)))
         print("prob y=-1 given x=%s is %s" % (x,prob_y_given_x(-1, x)))

         x = (1,1,0,1,0)
         print("prob y=1 given x=%s is %s" % (x,prob_y_given_x(1, x)))
         print("prob y=-1 given x=%s is %s" % (x,prob_y_given_x(-1, x)))
```

```
prob y=1 given x=(0, 0, 0, 0, 0) is 0.835051546392
prob y=-1 given x=(0, 0, 0, 0, 0) is 0.164948453608
prob y=1 given x=(1, 1, 0, 1, 0) is 0.0
prob y=-1 given x=(1, 1, 0, 1, 0) is 1.0
```

Answer: Both of the observations would predict class y=1

3.3 Part 3

```
In [19]: x = (0,0,0,0,0)
         print("prob y=1 given x=%s is %s" % (x,prob_y_given_x(y=1, given_x=x)))

         x = (1,1,0,1,0)
         print("prob y=1 given x=%s is %s" % (x,prob_y_given_x(y=1, given_x=x)))
```

```
prob y=1 given x=(0, 0, 0, 0, 0) is 0.835051546392
prob y=1 given x=(1, 1, 0, 1, 0) is 0.0
```

3.4 Part 4

To specify a full joint distribution would require finding $2^5 = 32$ probabilities. The conditional independence assumption allows us to simplify the model and only specify 2 probabilities per each variable (technically, we'd only need to specify 1 probability per feature since all features are binary) for a total of $2 * 5 = 10$ probabilities.

Further, since we have very few data points, we would get several 0 probabilities in the full joint distribution.

3.5 Part 5

Supposing that the information for feature x_1 is lost, we make predictions by using the formula:

$$p(y|x) = \frac{p(x_2|y)...p(x_5|y)p(y)}{p(x_2|y)...p(x_5|y)p(y)p(x|y=1) + p(x_2|y=1)...p(x_5|y)p(y=-1)p(y=-1)}$$

(the same formula as before, but with $p(x_1|y)$ ommited from both the numerator and denominator.

Why is simply ommitting the conditional probability of the missing feature (given y) the right thing to do? Well, consider the joint distribution

$$
\begin{aligned}
p(x_2, ..., x_5|y) &= \sum_{x_1} p(x_1, x_2, ...x_5|y) \\
&= \sum_{x_1} p(x_1|y)p(x_2|y)...p(x_5|y) \\
&= \left(\sum_{x_1} p(x_1|y)\right) p(x_2|y)...p(x_5|y) \\
&= (1)p(x_2|y)...p(x_5|y) \\
&= p(x_2|y)...p(x_5|y)
\end{aligned}
$$

Thus, if our address book is missing, we can make predictions in the same manner as before (simply pretend the x_1 feature was never there in the first place).

4 Problem 4: Statement of Collaboration

John Lu | Student ID: 21796268

I did not collaborate with any other student on this homework assignment. All work in this notebook is my own.