HW3_problem1

October 1, 2018

```
In [1]: import numpy as np
        spambase = np.loadtxt('spambase.data', delimiter=',')
        X = spambase[:, :-1]
        y = spambase[:, -1].astype(int)
        num_sample = 2000
```

```
In [2]: def shuffle(X, y, num_sample):
            #Xy = np.hstack((X, y))
            Xy = np.zeros((len(X),len(X[0,:])+1))
            Xy[:,:-1] = X
            Xy[:,-1] = y
            np.random.shuffle(Xy)
            X_training = Xy[0:num_sample,:-1]
            X_test = Xy[num_sample:,:-1]
            y_training = Xy[0:num_sample,-1]
            y_test = Xy[num_sample:,-1]
            return X_training, X_test, y_training, y_test
```

```
In [3]: def quantize1(X_training, X_test):
            medians = np.median(X_training, axis=0)
            X_temp_training = np.zeros(len(X_training))
            X_temp_test = np.zeros(len(X_test))
            X_training_quant = np.zeros((np.shape(X_training)))
            X_test_quant = np.zeros((np.shape(X_test)))
            for i in range(len(X_training[0])):
                # >=
                X_temp_training[:] = X_training[:,i]
                X_temp_training[X_training[:,i] >= medians[i]] = 1
                X_temp_training[X_training[:,i] < medians[i]] = 0
                X_training_quant[:,i] = X_temp_training

                X_temp_test[:] = X_test[:,i]
                X_temp_test[X_test[:,i] >= medians[i]] = 1
                X_temp_test[X_test[:,i] < medians[i]] = 0
                X_test_quant[:,i] = X_temp_test
            return X_training_quant, X_test_quant
```

```
In [4]: def quantize2(X_training, X_test):
            medians = np.median(X_training, axis=0)
            X_temp_training = np.zeros(len(X_training))
            X_temp_test = np.zeros(len(X_test))
            X_training_quant = np.zeros((np.shape(X_training)))
            X_test_quant = np.zeros((np.shape(X_test)))
            for i in range(len(X_training[0])):
                # <=
                X_temp_training[:] = X_training[:,i]
                X_temp_training[X_training[:,i] > medians[i]] = 1
                X_temp_training[X_training[:,i] <= medians[i]] = 0
                X_training_quant[:,i] = X_temp_training

                X_temp_test[:] = X_test[:,i]
                X_temp_test[X_test[:,i] > medians[i]] = 1
                X_temp_test[X_test[:,i] <= medians[i]] = 0
                X_test_quant[:,i] = X_temp_test
            return X_training_quant, X_test_quant

In [5]: def training(X_training_quant, y_training, num_y):
            P_y0 = sum(1-y_training) / num_y
            P_y1 = sum(y_training) / num_y
            P_xy0 = []
            P_xy1 = []
            for i in range(len(X_training_quant[0])):
                P_xi0y0 = np.dot(1-X_training_quant[:,i], 1-y_training) / num_y
                P_xi1y0 = np.dot(X_training_quant[:,i], 1-y_training) / num_y
                P_xi0y1 = np.dot(1-X_training_quant[:,i], y_training) / num_y
                P_xi1y1 = np.dot(X_training_quant[:,i], y_training) / num_y
                P_temp = [P_xi0y0, P_xi1y0] / P_y0
                P_xy0.append(P_temp)
                P_temp = [P_xi0y1, P_xi1y1] / P_y1
                P_xy1.append(P_temp)
            return P_xy0, P_xy1

In [6]: def testing(X_test_quant, y_test, num_y, P_xy0, P_xy1):
            P_y0 = sum(1-y_test) / num_y
            P_y1 = sum(y_test) / num_y
            y_hat = []
            for i in range(len(X_test_quant)):
                P_y0x = P_y0
                P_y1x = P_y1
                for j in range(len(X_test_quant[0])):
                    P_y0x *= P_xy0[j][int(X_test_quant[i][j])]
                    P_y1x *= P_xy1[j][int(X_test_quant[i][j])]
                y_hat.append(int(P_y1x >= P_y0x))
            return y_hat

In [16]: test_error1 = []
         test_error2 = []
         for trial in range(100):
             # shuffle the data
             [X_training, X_test, y_training, y_test] = shuffle(X, y, num_sample)

             # quantize the data
             # >=
             [X_training_quant1, X_test_quant1] = quantize1(X_training, X_test)
             # <=
             [X_training_quant2, X_test_quant2] = quantize2(X_training, X_test)

             # Naive Bayes
             # training
             num_y = len(y_training)
             [P_xy0_1, P_xy1_1] = training(X_training_quant1, y_training, num_y)
             [P_xy0_2, P_xy1_2] = training(X_training_quant2, y_training, num_y)

             # testing
             y_hat1 = testing(X_test_quant1, y_test, len(y), P_xy0_1, P_xy1_1)
             y_hat2 = testing(X_test_quant2, y_test, len(y), P_xy0_2, P_xy1_2)

             test_error1.append(sum(abs(y_hat1 - y_test)) / len(y_test))
             test_error2.append(sum(abs(y_hat2 - y_test)) / len(y_test))
```

```
In [17]: avg_test_error1 = np.mean(test_error1)
         avg_test_error2 = np.mean(test_error2)
         print("Is '>=' median better than '<='? ", avg_test_error1<avg_test_error2)
         print([avg_test_error1, avg_test_error2])
         print(min(avg_test_error1, avg_test_error2))
```

```
Is '>=' median better than '<='?  False
[0.24091503267973852, 0.11144175317185699]
0.11144175317185699
```

```
In [18]: # sanity check
         test_error_scheck = []
         for trial in range(100):
             # shuffle the data
             [X_training, X_test, y_training, y_test] = shuffle(X, y, num_sample)
             P_y0 = sum(1-y_training) / num_y
             P_y1 = sum(y_training) / num_y
             y_hat = [int(P_y1 >= P_y0)]*len(y_test)
             test_error_scheck.append(abs(sum(y_hat - y_test) / len(y_test)))
         print(np.mean(test_error_scheck))
```

```
0.3943675509419455
```

```
In [ ]:
```