```
 1 import matplotlib.pyplot as plt
 2 import numpy as np
 3 import pandas as pd
 4
 5 from sklearn.model_selection import KFold
 6
 7 data = pd.read_csv('MNIST_CV.csv', skiprows=[0], header=None)
 8
 9 def sigmoid(x):
10     """
11     Sigmoid functio as defined in the slides given an input x
12     From Andrew Ng Machine learning course on Coursera
13     """
14
15     return (1.0 / (1.0 + np.exp(-x)))
16
17 def predict(X, weights):
18     return sigmoid(np.dot(X, weights))
19
20 def cost(y, X, weights):
21     """
22     Grabbed this cost function from Andrew Ng course
23     Cost(h_theta(x), y) =
24             (-ylog(h_theta(x)) + (1-y)log(1-h_theta(x)))/m
25     """
26     m = y.size
27     predictions = predict(X, weights)
28     cost = np.sum((-y * np.log(predictions)) + ((1-y) * np.log
   (1-predictions)))
29     return  (cost / float(m))
30
31 def log_regression(y, X, n, lr):
32     weights = np.zeros(X.shape[1])
33     costs = []
34     m = y.size
35
36     for i in range(n):
37         descent = (np.dot((predict(X, weights) - y).T, X)).T
38         weights = weights - lr * descent
39         costs.append(cost(y, X, weights))
40     return weights, costs
41
42 def accuracy(X, y):
43     return (sum(X == y) / float(len(y))) * 100
44
45 kfold = KFold(n_splits=10, shuffle=True)
46 all_costs = []
```

```
47 fprs = []
48 tprs = []
49
50 for train_index, test_index in kfold.split(data):
51     y_training = data.iloc[train_index, 0]
52     y_test = data.iloc[test_index,0]
53     X_test = data.iloc[test_index,1:]
54     X_training = data.iloc[train_index,1:]
55
56     # We should normalize our data because it's the "right"
   thing to do but
57     # given we're using the MNIST data set and everypoint is a
    grayscale int
58     # between 0 and 255 it's not really needed
59     X_training = X_training / 255.0
60     X_test = X_test / 255.0
61
62     # Since logistic regression needs everything between 0 and
    1 I tried doing what we did last time
63     # which converts the array to an boolean. From there I
   tried using the .astype method to convert it to
64     # an int array but it didn't work. Not sure why but since
   I know each piece of data in here is 6 or 8
65     # I can just mod 3 and if it's 0 then it's 0(6) and if it'
   s 2(8) then
66
67     y_training = np.where(y_training % 3 == 0, 0.0, 1.0)
68     y_test = np.where(y_test % 3 == 0, 0.0, 1.0)
69
70     weight, costs = log_regression(y_training, X_training,
   1000, 5e-5)
71     all_costs.append(costs)
72     p = predict(X_test, weight)
73     # Convert the predictions to 1 or 0
74     p = np.where(p > .5, 1.0, 0.0)
75
76     print "Accuracy: {}".format(accuracy(p, y_test))
77
78     # Calculate the true positive rate and false positive rate
79     # I stole the logical_and part from Linda in class, it's
   genius!
80     tpr = np.sum(np.logical_and(p, y_test)) / float(np.sum(
   y_test == 1))
81     fpr = np.sum(np.logical_and(p == 0, y_test == 1)) / float(
   np.sum(y_test == 0))
82
83     tprs.append(tpr)
```

```
 84     fprs.append(fpr)
 85
 86 print fprs
 87 print tprs
 88
 89 # Add in the beginning and end so we have a 0 and 1 and then
    sort them
 90 # Not really sure why it works since we're plotting the false
     positive and true positive rate as if they don't have
 91 # any correlation when the obviously do.
 92 fprs += [0,1]
 93 fprs.sort()
 94 tprs += [0,1]
 95 tprs.sort()
 96 plt.plot(fprs, tprs)
 97 plt.show()
 98
```