```
# Sean McGlincy
# HW 3: Logistic Regression

#  Environment: Centos 7, Python 3.6 with Pycharm
#  PIP requires numpy, sklearn, scipy
import numpy as np
import math
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold


# Websites used:
# Equations from here:  https://beckernick.github.io/logistic-regression-from-scratch/
# K Folding  http://scikit-learn.org/stable/modules/cross_validation.html#k-fold

# Other websites
#
http://ksuweb.kennesaw.edu/~mkang9/teaching/CS7267/code4/Maximum_Likelihood_Estimation.html
# https://github.com/michelucci/Logistic-Regression-Explained/blob/master/MNIST%20with
%20Logistic%20Regression%20from%20scratch.ipynb
# http://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#introduction

##########################################################################################
##################
####################################  Functions
####################################################
##########################################################################################
##################

# Normalizes data between 0-1
def NormalizeData(data):
    return data / 255.0

# Normalizes labels to binary data {0,1}
def NormalizeLabels(lables):
    arr = np.zeros(len(lables))
    m = np.amax(lables)
    for i in range(len(lables)):
        if lables[i] == m:
            arr[i] = 1
    return arr

# Sigmoid for calculating Y-axis value
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

# Calculates prediction value
```

```
def prediction(X, weights):
    return sigmoid( np.dot(X, weights))

# def likelihood(X, y, weights):
#     return  np.sum( y * np.dot(X, weights) - np.log( 1 + np.exp(np.dot(X, weights))))


# Gradent
def GradAscent(X, y, weights):
    return np.dot( X.transpose(), y -  prediction(X, weights))

# Convex cost function
def Cost(X, y, weights):
    # If the label is 1
    label_1 = -y* np.log(prediction(X, weights))           # If y=1 use this equation
    label_0 = -(1 - y)* np.log(1 - prediction(X, weights))  # If y=0 use this equation
    return sum(label_1 + label_0 ) / len(y)                # Sum and divide by len of y

# Calculates True Positive Rates
def TPR(predict, y, threshold):
    true_pos = 0
    false_neg = 0
    for i in range(len(y)):
        if y[i] == 1:
            if predict[i] > threshold:
                true_pos += 1
            else:
                false_neg +=1
    return true_pos / float(true_pos + false_neg)



# Calculates False Positive Rates
def FPR(predict, y, threshold):
    false_pos = 0
    true_neg = 0
    for i in range(len(y)):
        if y[i] == 0:
            if predict[i] <= threshold:
                true_neg+= 1
            else:
                false_pos +=1
    return false_pos / float(false_pos + true_neg)


# Displays ROC Curve
def DisplayLearningCurve(plot):
    plt.plot(plot)
    plt.interactive(False)
```

```
    plt.show(block=True)

# Displays ROC Curve
def DisplayGraph(tpr, fpr):
    # Output Graph.  Flags for PyCharm
    avg_tpr = sum(tpr) / len(tpr)
    avg_fpr = sum(fpr) / len(fpr)

    # Print Values
    print("TPR: ", tpr)
    print("FPR: ", fpr)
    print("Avg TPR: ", avg_tpr )
    print("Avg FPR: ", avg_fpr )

    # Append 0 and 1 to array to produce a line
    tpr = [0] + tpr
    fpr = [0] + fpr
    tpr.append(1)
    fpr.append(1)
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.plot(fpr, tpr)
    plt.plot(avg_fpr, avg_tpr, 'X')
    plt.plot([0, 1], [0, 1])
    plt.ylabel("TPR")
    plt.xlabel("FPR")
    plt.interactive(False)
    plt.show(block=True)

###################################################################################
##################
#######################################  Main
#####################################################
###################################################################################
##################
#  Main Start Program HERE
# Import data as Integers and remove labels

k_fold = 10
threshold = 0.5
data = np.genfromtxt('MNIST_CV.csv', delimiter=',', dtype=int, skip_header=1)

# Data Stats We Want
tpr = []  # True Positive Rates
fpr = []  # False Positive Rates
display_curve = True
counter = 1

# Start K Folding
```

```
kf = KFold(n_splits=k_fold)
kf.get_n_splits(data)
d = kf.split(data)
for train_index, test_index in d:
    print("Processing K-Fold: ", counter)
    counter += 1
    # KFold returns array positions
    # Make new arrays to work with
    train_data = np.array(data[train_index])
    test_data = np.array(data[test_index])

    # Make sub arrays of labels and data
    train_labels = np.array(train_data[:,0])
    train_data = np.array(train_data[:, 1:-1])
    test_labels = np.array(test_data[:,0])
    test_data = np.array(test_data[:, 1:-1])



    # Normalize Data and Labels
    train_data = NormalizeData(train_data)
    test_data = NormalizeData(test_data)
    train_labels = NormalizeLabels(train_labels)
    test_labels = NormalizeLabels(test_labels)

    # Gradent

    plot = []
    learn_rate = 1e-5
    r = 1000
    weights = np.zeros(len(train_data[0]))  # Array of zeros for each feature
    for i in range(0, r):
        weights = weights + learn_rate * GradAscent(train_data, train_labels, weights)
        plot.append(Cost(train_data, train_labels, weights))

    # Don't need to display all 10 curves
    if display_curve:
        DisplayLearningCurve(plot)
        display_curve = False

    # Statistical stuff
    predict = np.array(prediction(test_data, weights))
    tpr.append(TPR(predict, test_labels, threshold))
    fpr.append(FPR(predict, test_labels, threshold))
    # End of KFold Loop


DisplayGraph(tpr, fpr)
exit(0)
```