Machine learning Project 2 – Logistic Regression

In this project, you are going to implement logistic regression. The project is of two parts. The first part is about binary classification using logistic regression and the second part is about multi-class classification.

1. **Part 1: Binary classification:**

The data file is called diabetes.csv, it has 8 medical features: pregnancies in month, patient Glucose concentration, Diastolic Blood Pressure, Skin Thickness, Insulin level, Body mass index, diabetes pedigree function, and age of patient.

The data file is a survey collection of patients in some hospital. The hospital wants to implement a learning program that classify new patients based on their 8 features to classify them as diabetes or healthy.

The first thing to note about this data file is that the target is the first column. To read the file, you need to write something like:

data = pd.read\_csv('path to file name\\binary.csv', header = 0,

name = ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'y'])

The above statement will read the data file and rename the columns to 'x1', 'x2', 'x3', …, 'x8' as feature columns and 'y' as target. This way you will not need to deal with every data file column name and use the column names x1, x2, x3, …, x8 and y as target.

Look at the file logisiticRegreesionDiabetes.py and complete the '…' with code/codes that satisfy the comment above it.

Use your learning algorithm to predict if a patient has diabetes where his data is: [8, 155, 66, 0, 21.6, 0.627, 34].

Answer I get was: has diabetes with probability of 54.82 %.

Accuracy of prediction is: 78.39 %

**2. Part 2: Multi-class classification:**

The second part of the project is about digit recognition. Automated handwritten digit recognition is widely used today - from recognizing zip codes (postal codes) on mail envelopes to recognizing amounts written on bank checks. This exercise will show you how the methods you've learned can be used for this classification task.

**2.1 Dataset:**

You are given a data set in ex3data1.mat that contains 5000 training examples of handwritten digits. The '.mat' format means that that the data has been saved in a native MATLAB matrix format, instead of a text (ASCII) format like a csv-file. These matrices can be read directly into your program by using the load command. After loading, matrices of the correct dimensions and values will appear in your program's memory. The matrix will already be named, so you do not need to assign names to them.

**MATLAB** is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-**use** environment where problems and solutions are expressed in familiar mathematical notation.

You can read the MATLAB file as follows:

Import scipy.io as sp

data = sp.loadmat('path to file name\\binary.csv')

**# Extract features:**

X = data['X']

Y = data['Y']

There are 5000 training examples in ex3data1.mat, where each training example is a 20 pixel by 20 pixel gray scale image of the digit. Each pixel is represented by a floating point number indicating the gray scale intensity at that location. The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image.

The second part of the training set is a 5000-dimensional vector y that contains labels for the training set. To make things more compatible with MATLAB indexing, where there is no zero index, we have mapped the digit zero to the value ten. Therefore, a "0" digit is labeled as "10", while the digits "1" to "9" are labeled as "1" to "9" in their natural order.

**2.2. Visualization:**

You can visualize some of the images in the file by typing the following code:

\_, axarr = plt.subplots(10,10,figsize=(10,10))

for i in range(10):

for j in range(10):

**## get an int random number from 5000 examples**

r = np.random.randint(X.shape[0])

**## get that image from X**

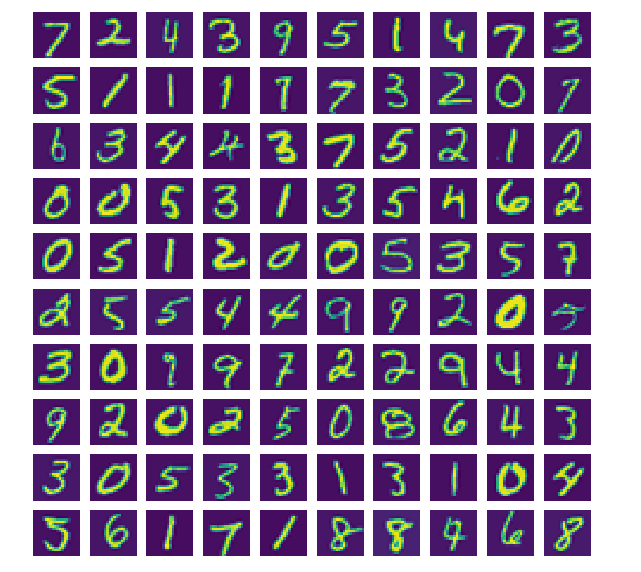
XA = X[r].reshape((20, 20), order='F')

axarr[i,j].imshow(XA) **## show in sub-figure (i, j)**

axarr[i,j].axis('off') #**# turn off axis...**

plt.show()

Result is:



In this data file you have images for digits from 0 to 9. The target is a multi-class {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. You have to implement a multi-class learning algorithm that can classify images to their corresponding digit. Look at the digitRecognition.py Python program and fill the '…' with Python code/codes.

**2.3 Vectored Logistic Regression:**

You will be using multiple one-vs-all logistic regression models to build a multi-class classifier. Since there are 10 classes, you will need to train 10 separate logistic regression classifiers. To make this training efficient, it is important to ensure that your code is well vectored. In this section, you will implement a vectored version of logistic regression that does not employ any for loops. You can use your code in the last exercise as a starting point for this exercise.

**2.4 One-vs-all Prediction**

After training your one-vs-all classifier, you can now use it to predict the digit contained in a given image. For each input, you should compute the "probability" that it belongs to each class using the trained logistic regression classifiers. Your one-vs-all prediction function will pick the class for which the corresponding logistic regression classifier outputs the highest probability and return the class label (1, 2,..., or K) as the prediction for the input example.

You should now complete the code in predictOneVsAll function to use the one-vs-all classifier to make predictions. Once you are done, you will call your predictOneVsAll function using the learned value of Θ. You should see that the training set accuracy is about 93.4% (i.e., it classifies 93.4% of the examples in the training set correctly).