Dalhousie University Faculty of Computer Science
CSCI 1120 Project II Due 15 Dec 2022

****Please submit Lab F with this project****

Background:
In Lab 15, you have trained a classifier to recognize a single digit. Now the classifier is to be extended to recognize any digit. You will need to import the training and testing data as you did for your *single-digit classifier*. The classifier you built for the lab worked for a *single* class. Each data point (an image, encoded as a single row of a $785 \times 60K$ matrix), is classified as the digit in question, or not. The classifier uses the columns of the $785 \times 60K$ as *features* (the X_is in the **sum** below) to learn the labels using the labels for the training set: (Y). The process took the images (rows of the X matrix), computed a *regression prediction* by computing the regression coefficients, βs, by minimizing the *log-loss* using *gradient descent*. The sum:

$$b_0 + b_1 \cdot X_1 + \ldots + b_{784} \cdot X_{784}$$

is then put through a *Logistic* (*i.e.* the Sigmoid) function to produce a prediction: $\hat{Y}$ for the label (Y).

The Extended Classifier:
The basic idea is to have 10 *single-digit classifiers*, each built to recognize a single digit, working in *parallel*. Recall that *all* the features of *all* the images are presented at the same time to the classifier using the X matrix. Then, each image, put through the classifier built above, to obtain the regression prediction followed by being put through the Sigmoid (before using `np.round()`), delivers a value between 0 & 1. The prediction value giving the *confidence* in the prediction.

A similar approach will be used for the extended classifier. Each of the 10 single digit classifiers will return a value between 0 & 1 for each image presented to it. Matrix algebra offers a particularly simple way to put the single-digit classifiers in parallel:

- Write a function to create and return the encoded-Y *matrix*, that as before, has 60K rows (one per training sample) **&** has number of columns equal to the *classes* of images (you have 10 classes: representing digits from 0 to 9). Each row of this matrix will have exactly one "1".
- Remember, the labels are in Y_{train}. In the function, and the label number corresponds to the column number in the encoded-Y *matrix*. Use these to create the encoded-Y *matrix* which is `numImages` $\times$ `numClasses = m` $\times$ `c` .

- These being matrices, the right hand side of the regression equation; $X \cdot \beta$ must have the **same** dimensions as the encoded-Y.

$$\begin{aligned} \text{encoded} - Y \quad &= X \cdot \beta \\ (m \times c) \quad &= (m \times f) \cdot (f \times c) \end{aligned}$$

  Where f are the number of *features* (*i.e.* columns) of X. You will need to initialize a β matrix of appropriate size.
- The above changes in the regression equation dimensions means that the predicted-Y, *i.e.* $\hat{Y}$, is a matrix of dimensions $(m \times c)$. Each row of $\hat{Y}$ contains $c (= 10)$ numbers.
  Modify the `classify(`X`,` β`)` function you wrote earlier, to return an $(m \times 1)$ array L. Each row of L contains is **indexed** by a label (*i.e.* one of: 0, 1, . . . , 9), that corresponds to the **column number** of the largest value in to each row of $\hat{Y}$. The content of the array should be the *largest value* in that particular row of the $\hat{Y}$ matrix.
  The `numpy` function `argmax()` can with find the largest column (axis = 1) value in each row.
  To transform the array into a one dimensional array as required for L, the `numpy` function `reshape(-1,1)` will flatten an array to a 1-D array that is compatible to the original array. For examples, see:

  `https://stackoverflow.com/questions/18691084/`
  `what-does-1-mean-in-numpy-reshape`
- Create a report that show the iteration number, the percentage of correct matches and the loss, for the test examples. This is best done by a function that takes: `iteration number, X_train, Y_train, X_test, Y_test,` β as arguments. This function should be called from your `train(...)` function.