Programming Assignment 1

Due Date: 23:55, May 02, 2019

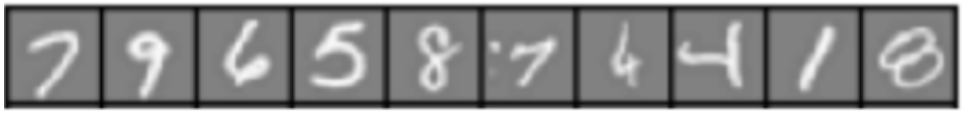
Please note your name + id number on the first line of code (comment format)

Requirements:

- You can use Python only (version 3.6/3.7).
- Use library packages for importing the dataset and NumPy only. You can't use prepared models.
- The running time of your algorithm should be reasonable, up to few minutes.
- Full points will be given on computational efficiency, runtime, vectorized implementation (without unnecessary 'for' loops if possible).

Introduction:

In this exercise you will develop a classifier for the MNIST dataset . This dataset consists of 28x28 grayscale images of hand-written digits (0-9) and their corresponding labels (the actual digits). The following figure depicts 10 examples from the dataset:



The corresponding labels vector is: [1,9,6,5,8,1,4,4,1,8].

Instructions:

- At the end of the document you will find appendix for 'How-To' fetch MNIST dataset.
- The original MNIST dataset should be partitioned to Training-set/ Test-set (60K images for Training and 10K images for Testing). You can use 'train_test_split' of the sklearn module.
- The test-set should be used for testing the accuracy of your model.
- Plot the test loss and the training loss (both curves on the same plot) as a function of the iteration index.
- For ease of calculation, you will transform the labels to one-hot vector. For example, for the label '5', the corresponding vector is [0,0,0,0,0,1,0,0,0,0] and label '0' corresponding vector is [1,0,0,0,0,0,0,0,0,0].
- The input images should be flattened so that each image of size 28x28 pixels will be represented by a vector of size 785x1 whose first component corresponds to the bias term and is equal to 1.
- [Wikipedia] A **confusion matrix** is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the

fact that it makes it easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).
Example: If a classification system has been trained to distinguish between cats, dogs and rabbits, a confusion matrix will summarize the results of testing the algorithm for further inspection. Assuming a sample of 27 animals — 8 cats, 6 dogs, and 13 rabbits, the resulting confusion matrix could look like the table below:

		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	**5**	2	0
	Dog	3	**3**	2
	Rabbit	0	1	**11**

In this confusion matrix, of the 8 actual cats, the system predicted that three were dogs, and of the six dogs, it predicted that one was a rabbit and two were cats. We can see from the matrix that the system in question has trouble distinguishing between cats and dogs, but can make the distinction between rabbits and other types of animals pretty well. All correct predictions are located in the diagonal of the table (highlighted in bold), so it is easy to visually inspect the table for prediction errors, as they will be represented by values outside the diagonal.

- [Wikipedia] A **table of confusion** is a table with two rows and two columns that reports the number of *false positives (FP)*, *false negatives (FN)*, *true positives (TP)*, and *true negatives (TN)*. This allows more detailed analysis than mere proportion of correct classifications (accuracy). Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the data set is unbalanced (that is, when the numbers of observations in different classes vary greatly). For example, if there were 95 cats and only 5 dogs in the data, a particular classifier might classify all the observations as cats. The overall accuracy would be 95%, but in more detail the classifier would have a 100% recognition rate for the cat class but a 0% recognition rate for the dog class.
  Assuming the confusion matrix above, its corresponding table of confusion, for the cat class, would be:

		Actual class	
		Cat	**Non-cat**
Predicted class	**Cat**	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

TP means observation is positive, and is predicted to be positive (e.g observation is the digit '7' and the predicted digit is '7').
FP means observation is negative, but is predicted positive (e.g observation is digit!='7' and the predicted digit is '7').
FN means observation is positive, but is predicted negative (e.g observation is digit='7' and the predicted digit is digit!='7').

TN means observation is negative, and is predicted to be negative (e.g observation is digit!='7' and the predicted digit is digit!='7').

The Accuracy is defined as ACC=(TP+TN)/(TP+TN+FP+FN)
The sensitivity (True positive rate) is defined as TPR = TP/(TP+FN)
The selectivity (True negative rate) is defined as TNR = TN/(TN+FP)

PART A

1. The Perceptron Learning Algorithm (PLA) for binary classification was developed in class. Extension to the multi-class perceptron classifier can be obtained using multiple binary PLA classifiers trained with one-vs-all strategy. The one-vs-all strategy is based on a reduction of the multi-class problem into K binary problems, each of which discriminates between one class to all the rest of the classes. Specifically, in the first stage we assign the label +1 to all examples labeled '0' and the label -1 to all examples labeled '1','2',…,'9'. The resulting weight vector will be denoted by w^0 . In the second binary problem we assign the label +1 to all examples labeled '1' and the label −1 to all examples labeled '0', '2', '3', ..., '9'. The resulting vector will be denoted by w^1 , and so on. We end up with K=10 weight vectors.
   Given a new instance x, we predict the label that gets the highest confidence:

$$\hat{y} = argmax_{y\in\{0,\dots,9\}} w^y x \ .$$

   The Perceptron Learning Algorithm for binary classification in the linearly separable case:

   - Initialization: $w = w_0$

   - Pick a misclassified example and denote it as $(x(t), y(t))$

   i.e., $y(t) \neq h\big(x(t)\big) = sign\big(w^T(t)x(t)\big)$

   - Update the weights

$$w(t) + y(t)x(t) \rightarrow w(t+1)$$

   -Continue iterating until there are no misclassified examples in the training set.

   When the data is not linearly separable, PLA will never terminate, and its behavior can become quite unstable, and can jump from a good perceptron to a very bad one within one update. One approach for getting an approximate solution is to extend PLA through a simple modification into what is called the pocket algorithm. This algorithm keeps 'in its pocket' the best weight vector encountered up to iteration t in PLA. At the end, the best weight vector will be reported as the final hypothesis. Specifically,

- Set the pocket weight vector $\hat{w}$ to $w(0)$ of PLA.
- For t=0,…,T-1 do:
  - Run PLA for one update to obtain $w(t+1)$.
  - Evaluate $E_{in}(w(t+1))$.
  - If $w(t+1)$ is better than $\hat{w}$ in terms of E_{in}, set $\hat{w}$ to $w(t+1)$.
- Return $\hat{w}$.

Where $E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}[sign(w^T x_n) \neq y_n]$

Updates for both parts:

1. A total of 3 files should be submitted:

- A python script for part A

- A Python script for part B

- A pdf summary that includes analysis and figures.

2. In order to reduce the training time of the Pocket algorithm in part A, we suggest to compute the training loss every L=1,000 updates of the perceptron learning algorithm (and not after each iteration). You should run between 1-5 epochs, where an epoch is one complete presentation of the dataset to the learning algorithm. A plot of both the train loss and the test loss sampled every 1,000 iterations should be included.

Apply the multi-class perceptron algorithm on the MNIST classification problem.

- Calculate the confusion matrix for the multi-class classification problem on the test-data and compute the accuracy (ACC).

- Calculate the table of confusion for each of the digits and compute the sensitivity (TPR) for each class.

- Discuss your results.

PART B

In the following part you will apply Softmax Regression on the MNIST dataset.

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes, i.e., $y_n \in \{1, \dots, K\}$ where K is the number of classes.

Given a test input $\boldsymbol{x}$, we want our hypothesis to estimate the probability that $P(y = k|\underline{x})$ for each value of k=1,…,K, i.e., we want to estimate the probability of the class label taking on each of the K different possible values. Thus, our hypothesis will output a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities:

$$\underline{h}(\underline{x}) = \begin{pmatrix} P(y=1|\underline{x}) \\ \dots\dots \\ P(y=K|\underline{x}) \end{pmatrix} = \frac{1}{\sum_{j=1}^{K} \exp\left(\underline{w}^{(j)^T}\underline{x}\right)} \begin{pmatrix} \exp(\underline{w}^{(1)^T}\underline{x}) \\ \dots\dots \\ \exp(\underline{w}^{(K)^T}\underline{x}) \end{pmatrix}.$$

Minimize (using the gradient descent method) the following softmax cost function on the training data with respect to the vectors $\underline{w}_j$ j=0,1,…,9:

$$E_{in}(\underline{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} 1\{y_n = k\} \log \frac{e^{w_k^T x_n}}{\sum_j e^{w_j^T x_n}}.$$

The gradient is calculated by:

$$\nabla_{w_k} E_{in}(\underline{w}) = -\frac{1}{N}\sum_{n=1}^{N}\left(1\{y_n = k\} - y_n^{(k)}\right)x_n$$

where $y_n^{(k)} = \frac{e^{w_k^T x_n}}{\sum_j e^{w_j^T x_n}}$.

- Calculate the confusion matrix for the multi-class classification problem on the test-data and compute the accuracy (ACC).

- Calculate the table of confusion for each of the digits and compute the sensitivity (TPR) for each class.

- Discuss your results.

Note:

You should use the following code block to import the MNIST dataset:

```
from scipy.io import loadmat
import urllib
mnist_alternative_url = "https://github.com/amplab/datascience-
sp14/raw/master/lab7/mldata/mnist-original.mat"
mnist_path = "./mnist-original.mat"
response = urllib.request.urlopen(mnist_alternative_url)
with open(mnist_path, "wb") as f:
    content = response.read()
    f.write(content)
mnist_raw = loadmat(mnist_path)
mnist = {
    "data": mnist_raw["data"].T,
    "target": mnist_raw["label"][0],
    "COL_NAMES": ["label", "data"],
    "DESCR": "mldata.org dataset: mnist-original",
}
print("Success!")
```