Data Engineering 414 ~ Practical 1: Python Introduction

Johan Neethling 24739286

Introduction

We were given a template project including a Python script with skeleton code, two datasets and a SoftMaxRegression model.

With this project we followed a relatively detailed tutorial which took us through the process of importing and using the SoftMaxRegression model.

The data provided was the MNIST data set which consisted of 28x28 pixel images of handwritten digits.

Report Overview

This practical was very introductory and the course admin did a complete job of holding our hand through it all. We had a crash-course on Python commands and logic, demis to assist in the two-hour practical period and ample opportunity to email queries to the lecturer or teaching-assistant. Working together amongst each other was also supported.

On a more personal level:

At first glances, the practical was quite chaotic – trying to work in Ubuntu on the lab's computers only to be lagged out into using Anaconda only to be recommended (and thankfully so) to use Pycharm (a very useful IDE for working in Python in Windows). The two hours flew by, but I got the backbone of my project done with the help of our notes and a sprinkle of ChatGPT (please let me know if this is frowned upon, I just feel that if we can google coding practice – we should be able to Chat it) in the practical session. I completed the last few questions over the next few evenings (question 7 and on).

After completion I feel excited to learn more about this language and the course. I feel we will be filled with knowledge and skills that will make us very useful and educated on the topic of machine learning in the near future.

Report

Very simple request, not much to say here…

```
########################################################################
#               (1) Import numpy and matplotlib                        #
########################################################################

#************************* YOUR CODE HERE ************************#

import numpy as np
import matplotlib.pyplot as plt

#*****************************************************************#
```

Figure 1: Code for q1

The train csv file contained all the training data for the model. It was rather strange to read that in, as we did not need to train the model's weights. The test csv file contained the data that we were to test our model on, the first column was the true outputs and the remaining were the inputs for the model.

I used a simple function from the numpy library to read the data into the variables as seen below. I separate the targets from the general data by slicing the dataset (I do not know if it is bad practice to reuse variables as I did here…), the " 1: " represents everything from the 1st (actually second) index.

```
######################################################################
#   (2) Read in the MNIST dataset from files data/train.csv and      #
#       data/test.csv and store as numpy variables with shapes:      #
#                   train : (60000,784)                              #
#                   test : (10000,784)                               #
#                   train targets : (60000,1)                        #
#                   test targets : (10000,1)                         #
#                                                                    #
#    Note: * open data/train.csv to see how the data is stored:      #
#          target,pixel_0_0,pixel_0_1,...,pixel_28_27,pixel_28_28    #
#                                                                    #
#          * The MNIST data set consists of 28 x 28 pixel            #
#          handwritten digits where each pixel has a value           #
#          between 0 and 255                                         #
######################################################################

#************************ YOUR CODE HERE ************************#

train = np.genfromtxt( fname: 'data/train.csv', delimiter=',')
train_targets = train[:, 0]
train = train[:, 1:]

test = np.genfromtxt( fname: 'data/test.csv', delimiter=',')
test_targets = test[:, 0]
test = test[:, 1:]

#************************************************************************#
```

Figure 2: Code for q2

Simple printing function, not much to say here…

```
#########################################################################
#       (3) Print the shapes of the training and test sets              #
#########################################################################

#************************* YOUR CODE HERE ************************#

print("train shape: ", train.shape)
print("test shape: ", test.shape)
print("train targets shape: ", train_targets.shape)
print("test targets shape: ", test_targets.shape)

#*****************************************************************#
```

Figure 3: Code for q3

To normalise the data to the range 0,1 we take the maximum and minimum of the dataset and then scale all of the observations to be a fraction representative of their proportion of that interval. The formula I used was " normalised = (observation – min) / (max – min) ". I found that I had to double np.max and np.min the datasets to get around the 2-D dimensionality of the dataset. " axis=1 " means that we first look at each row.

```
#########################################################################
#           (4) Normalise the dataset to the range 0,1                  #
#########################################################################

#************************* YOUR CODE HERE ************************#

min_train = np.min(np.min(train))
max_train = np.max(np.max(train))
print("Min train: ", min_train)
print("Max train: ", max_train)

min_test = np.min(np.min(test, axis=1))
max_test = np.max(np.max(test))
print("Min test: ", min_test)
print("Max test: ", max_test)

train = (train - min_train) / (max_train - min_train)
test = (test - min_test) / (max_test - min_test)

print("Normalised train min: ", np.min(np.min(train, axis=1)))
print("Normalised train max: ", np.max(np.max(train, axis=1)))
print("Normalised test shape: ", test.shape)
print("Normalised train shape: ", train.shape)

#*****************************************************************#
```

Figure 4: Code for q4

I took row 1 (index 0) and all the observations (bar the expected outcome), reshaped it to suit the output we want to use – a 28x28 pixel image.

```
#########################################################################
#            (5) Plot an example from the loaded datset                 #
#########################################################################

#************************** YOUR CODE HERE ***************************#

example = train[0, :]
example = np.reshape(example, newshape: (28, 28))
plt.imshow(example, cmap = 'gray')
plt.title('Example at index 0 from Training data')
plt.axis('off')
plt.show()

#*************************************************************************#
```

Figure 5: Code for q5

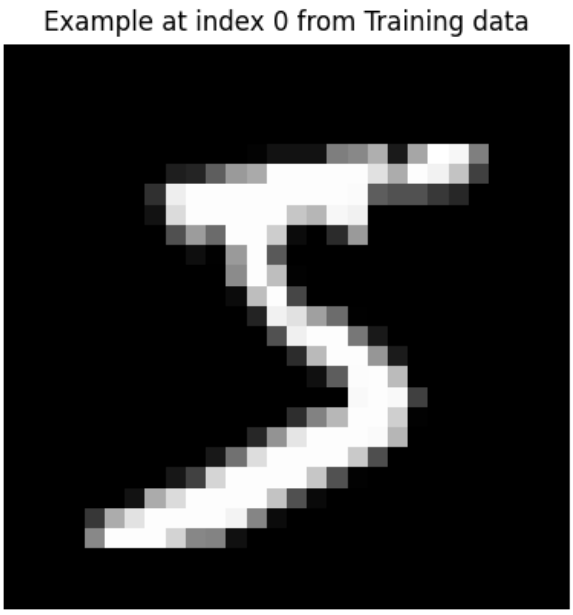

Figure 6: Output for code from q5

Once again, quite a simply reuse of the genfromtxt function from the numpy library.

```
#########################################################################
#      (6) Load in weights from a logistic regression model from        #
#           the file data/weights.txt                                   #
#                                                                       #
#          * Store the weights as a numpy array called weights          #
#            the shape of the array should be (785, 10)                 #
#########################################################################

#************************* YOUR CODE HERE *************************#

weights = np.genfromtxt( fname: 'data/weights.csv', delimiter=',')
print("weights shape:", weights.shape)

#******************************************************************#
```

Figure 7: Code for q6

Now is where we get to the juicy bits, we load the weights int our model as described by that lovely help function. We then format our single test input – I actually struggled with this question as I thought we would have to train the model, the thought of the weights being trained beforehand missed me completely – and find out output, y. I do believe I got a bit confused between this and the next question (I did these two during and between classes so I was on the move and not super settled) and started working on my bar chart here.

```
#************************* YOUR CODE HERE *************************#

model.load(weights)
input = np.reshape(test[7], newshape: (1,784))
y = model(input).reshape(-1)
print("Prediction: ")
x = np.arange(0,10,1)

for num in x:
    element = y[num]
    print(f"Prob({num}): {element}")

#******************************************************************#
```

Figure 8: Code for q7

Here I just plot the probabilities for the output at index 7 on a bar chart. The tallest bar is the most likely value and will there for be our model output – so in this case the model has predicted 9.

The weights have a shape of 785x**10** and so we can assume that we can have outputs of the range 1,10.

```
#########################################################################
#   (8) Plot a bar graph of the model probabilities                     #
#                                                                       #
#       * Also plot which number you are trying predict                 #
#########################################################################

#************************** YOUR CODE HERE **************************#

plt.bar(x, y)
plt.title("Probabilities for index 7")
plt.show()

#************************************************************************#
```

Figure 9: Code for q8

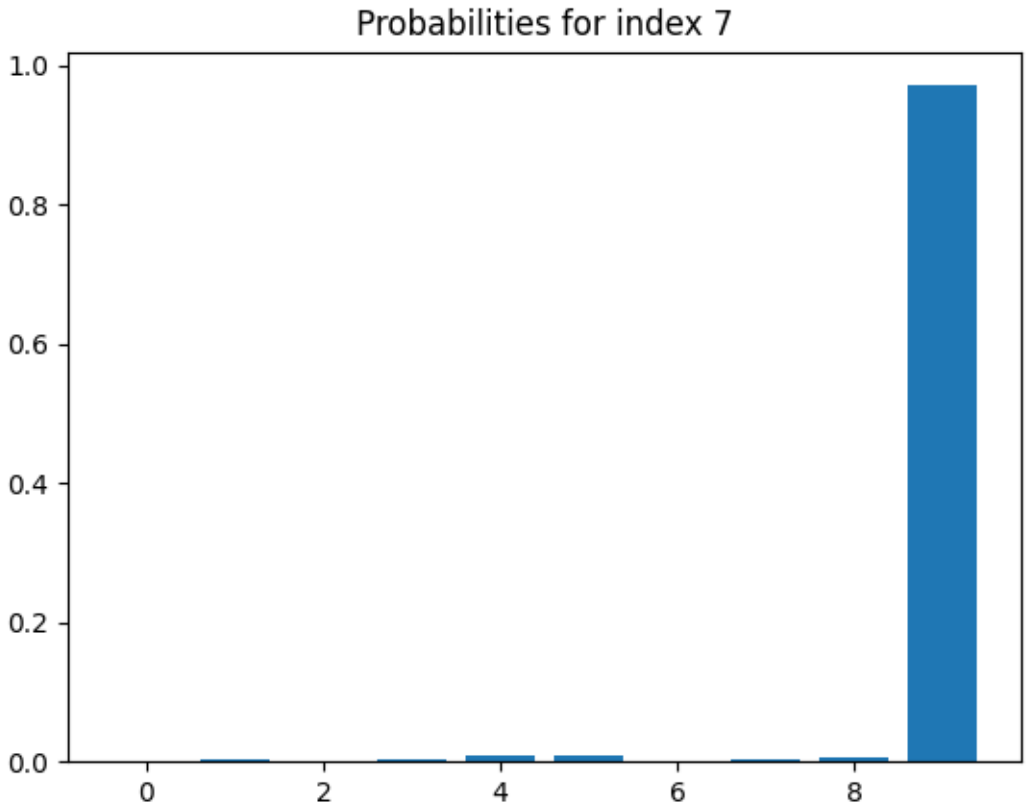

Figure 10: Output for code from q8

We were tasked to define a Python formula to calculate model accuracy and then test the model's accuracy.

The formula for accuracy is " accuracy = sum_of_correct_predictions / total_predictions ".

The argmax function returns the index of the maximum observation on the row dimension in this case and the equal function returns true if the two compared values (per index between the two vectors) are equal, else false.

```
#************************** YOUR CODE HERE **************************#
1 usage
def accuracy(true_data, pred_data):
    """
    The function will read in true data and predicted data from a mod
    and compare them to test the model's accuray.
    :param true_data: True outputs provided in the data
    :param pred_data: Outputs predicted by the model
    :return: Accuracy of the model
    """
    pred_out = np.argmax(pred_data, axis=1)
    eq = np.equal(pred_out, true_data)
    correct = np.sum(eq)
    return correct/len(true_data)

acc = accuracy(test_targets, model.predict(test))
print(f"Accuracy of model: {acc*100}%")

#********************************************************************#
```

Figure 11: Code for q9

Conclusion

With this model I found that the main takeaways were more the produced plots and less the actual "terminal" outputs. Here they are for reference:

```
train shape:  (60000, 784)
test shape:  (10000, 784)
train targets shape:  (60000,)
test targets shape:  (10000,)
Min train:  0.0
Max train:  255.0
Min test:  0.0
Max test:  255.0
Normalised train min:  0.0
Normalised train max:  1.0
Normalised test shape:  (10000, 784)
Normalised train shape:  (60000, 784)
weights shape: (785, 10)
Prediction:
Prob(0): 3.1478514332591343e-07
Prob(1): 0.003203920992598947
Prob(2): 0.0004390825263143842
Prob(3): 0.001960671792449701
Prob(4): 0.00796483631025206
Prob(5): 0.007659250088267349
Prob(6): 0.0001723562097913326
Prob(7): 0.0020711333941906473
Prob(8): 0.00526661506505905
Prob(9): 0.9712618188359332
Accuracy of model: 92.29%

Process finished with exit code 0
```

Figure 12: Code run output

This practical was a pleasant experience; I feel more confident in my Python skills and am excited to learn and start my own little Python snakelet projects.