Data Engineering 414 ~ Practical 3: Logarithmic Regression

Johan Neethling 24739286

2024 - 04 - 15

Introduction

This practical tested our understanding and implementation skills of multinomial logistic regression models. We were given the skeleton code of the model and tasked to finish implementing it, we had to design a softmax function, the forward pass and training algorithm for our model. Alongside the model aspects, we were also tasked with developing a log-likelihood calculating function, a batching function and code to initialise, train and test the model while recording the results in a csv file.

This practical took particular interest in testing the following ECSA Graduate Attributes: Engineering Design; Investigation, Experiments, and Data Analysis; Engineering Methods, Skills and Tools, including Information Technology; and Professional and Technical Communication.

Report Overview

We implemented a multinomial logistic regression model. This model takes inputs (**X**) of shape NxD, where N is the number of inputs and D is the number of datapoints per input, and put it through a layer of linear regression where we multiply X by the model's weights (**W**) of shape DxK, where K is the number of classes the output can fall under, to get our logits (**Z**) of shape NxK. This is then taken per observation (or row) and put through the softmax function to transform each input on the row into relative likelihoods (normalised to 0-1). These are outputted as y_hat which is a NxK matrix.

We trained our model through mini-batch gradient descent.

Hyperparameters are parameters that cannot be adjusted through back propagation or forward passes, the are set before training occurs and remain constant. We develop our algorithm to find the best combination of hyperparameters in the question 5 section.

Report

Question 1

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

X_exp = np.exp(X)
sum_exp = np.sum(X_exp, axis=axis, keepdims=True)                    # decrease dimension of input and sum values acro
return X_exp/sum_exp

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

To softmax an input matrix we transform each value into the e^{value} of that value, then we divide each input by the sum of the values in their respective rows (for vectors we just take each value over the sum of the vector).

$$\sigma_k(\mathbf{Z}) = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}}$$

Question 2

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

ll = np.log(Y_hat)
ll = Y*ll

return np.sum(ll, axis=-1, keepdims=True)

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

We find the logarithms of the model outputs and then dot product those with the ground truth matrix. We get the log likelihood by taking the sum per row. The dot product of the two matrices gives us the normalised likelihood of the model predicting the ground truth output, by setting each other false output likelihood to zero. By summing the rows, we can find the likelihood of each row, we can find the likelihood of each observation being predicted correctly, which can in turn be used to test the accuracy of the model.

We can use the log likelihood to test the "goodness" of the fit of our model.

$$LL = \sum_{n=1}^{N}\sum_{k=1}^{K} y_k^{(n)} ln(\hat{y}_k^{(n)})$$

Question 3

```
def get_minibatch(X,Y,i,batch_size=60):

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

    start = i*batch_size                                              # we start at batch 0
    end = batch_size*(i+1)
    x_batch = X[start:end, :]
    y_batch = Y[start:end, :]

    return x_batch, y_batch

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

At first we had set the start as " = (i-1)*batch_size"; but, under advice from a demi, we adjusted the code to start batch zero at zero and batch one at one.

We store the (in this case) 60 observations in mini batches (inputs and ground truth outputs) and return them.

Question 4

```
#********************** 4.a YOUR CODE HERE ********************#

X_with_bias = np.insert(X, 0, 1, axis=1)
return softmax(X_with_bias@self.weights)

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

We insert a bias term to our data matrix to match the shape of the given weights. We then softmax the product of this new data matrix and the models weights as the forward method of our model.

```
#********************** 4.b YOUR CODE HERE **********************#

y_hat = self.forward(train_X)
m_weights = np.insert(train_X, 0, 1, axis=1).T@(train_Y-y_hat)
self.weights = self.weights + learning_rate*m_weights

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

We train the models through gradient descent. The gradient of our model's error is found by multiplying the transpose of the input data with the different between the ground truth of the training data and our model's predictions. We update the weights by adding a proportion (the learning rate) of the gradient to the current weights.

```
#******************** 4.c and 4.d YOUR CODE HERE *****************#

mlr = MultinomialLogisticRegression(D=784, K=10)

with open('log.csv', 'w', newline='') as csvfile:
    fieldnames = ['Epoch', 'New Accuracy']

    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    file_empty = csvfile.tell() == 0

    if file_empty:
        writer.writeheader()

    for i in range(0, 32):
        mini_X, mini_Y = get_minibatch(train_X, train_Y, i)
        mlr.train_minibatch(mini_X, mini_Y)
        test_accuracy = accuracy(test_Y, mlr.forward(test_X))
        writer.writerow({'Epoch': i+1, 'New Accuracy': test_accuracy})

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

Epoch	New Accur
1	0.2944
2	0.4126
3	0.3608
4	0.4828
5	0.6166
6	0.6707
7	0.7207
8	0.7118
9	0.6385
10	0.6831
11	0.7015
12	0.6969
13	0.6325
14	0.7158
15	0.6823
16	0.7482
17	0.7734
18	0.8205
19	0.8059
20	0.8104
21	0.7351
22	0.6843
23	0.6479
24	0.7719
25	0.8261
26	0.8327
27	0.8318
28	0.8202
29	0.8345
30	0.8057
31	0.8316
32	0.8451

We initialise our model and record the accuracies as we train the model to compare how they change in a csv file. We set the column names to Epoch and New Accuracy, respectively. For this question we iterate over 32 epochs, from batch 0 to batch 31, and train our model using our predefined functions. We then keep track of the accuracy of our model per training iteration in the log file. On the right is the log file produced.

Question 5

```
#******************* Question 5 CODE ****************************#

batch_sizes = [10, 50, 100, 200, 500, 1000, 1500, 2000]
epochs = [10, 50, 100, 200, 500, 1000, 1500, 2000]
best = [0, 0, 0]                                          # epoch batch_size accuracy

with open('Hyperparameters_report.csv', 'w', newline='') as csvfile:
    fieldnames = ['Epochs', 'Batch Size', 'Test Accuracy']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    mlr2 = MultinomialLogisticRegression(D=784, K=10)

    writer.writeheader()

    for batch_size in batch_sizes:
        for epoch in epochs:
            mlr2.weights = np.random.normal(0,1e-2,(mlr2.D,mlr2.K))
            for i in range(epoch):
                mini_X, mini_Y = get_minibatch(train_X, train_Y, i)
                mlr2.train_minibatch(mini_X, mini_Y)
            test_accuracy = accuracy(test_Y, mlr2.forward(test_X))
            if test_accuracy > best[2]:
                best[0] = epoch
                best[1] = batch_size
                best[2] = test_accuracy
            print(f'Epochs: {epoch}, Batch Size: {batch_size}, Test Accuracy: {test_accuracy}')
            writer.writerow({'Epochs': epoch, 'Batch Size': batch_size, 'Test Accuracy': test_accuracy})

    print('Best: \nEpochs:', best[0], 'Batch Size:', best[1], 'Test Accuracy:', best[2])
#************************************************************#
```

We were tasked to test various combinations of hyperparameters to find an optimal relationship between batch size and epochs. We begin by deciding on batch sizes and epochs. I decided to move exponentially as wanted to see the effect of the different sizes of the hyperparameters on the model's accuracy, not so much as find the exact optimum for it.

We also set up a benchmark best hyperparameter relationship and the related accuracy array.

We initialise the model before the for loops as to be more efficient, but reset the weights before each combination begins training the model as to maintain a fair test.

The for loops move per iteration, per epoch size, per batch size. We train the model then print it's accuracy to the terminal and then record it in a csv file. If the accuracy of a particular combination of hyperparameters is better than the bench mark, update the benchmark and print the combination after the for loops run.

Conclusion

We conclude that according to the results of our experiments, the accuracy of our model increases per larger number of epochs regardless of batch size. The batch size does not seem to have too much of an effect on the accuracy of the model. We see that there is a recurring pattern per number of epochs which supports this statement.

Epochs	Batch Size	Test Accuracy	Avg per batch size
10	10	0.6844	0.8653
50	10	0.8686	
100	10	0.8831	
200	10	0.8709	
500	10	0.8999	
1000	10	0.9052	
1500	10	0.905	
2000	10	0.9053	
10	50	0.6912	0.865825
50	50	0.8678	
100	50	0.8833	
200	50	0.8694	
500	50	0.8999	
1000	50	0.9049	
1500	50	0.905	
2000	50	0.9051	
10	100	0.6876	0.865625
50	100	0.8679	
100	100	0.8842	
200	100	0.8705	
500	100	0.8996	
1000	100	0.9046	
1500	100	0.9053	
2000	100	0.9053	
10	200	0.6807	0.8648125
50	200	0.869	
100	200	0.883	
200	200	0.871	
500	200	0.8998	
1000	200	0.9054	
1500	200	0.9049	
2000	200	0.9047	
10	500	0.6813	0.864725
50	500	0.8683	
100	500	0.8827	
200	500	0.8706	
500	500	0.9002	
1000	500	0.9051	
1500	500	0.9051	
2000	500	0.9045	
10	1000	0.6914	0.8660375
50	1000	0.8685	
100	1000	0.8828	
200	1000	0.8707	
500	1000	0.8997	
1000	1000	0.9051	
1500	1000	0.9045	
2000	1000	0.9056	
10	1500	0.6853	0.8653
50	1500	0.8686	
100	1500	0.8826	
200	1500	0.8703	
500	1500	0.8998	
1000	1500	0.9051	
1500	1500	0.9053	
2000	1500	0.9054	
10	2000	0.686	0.86555
50	2000	0.8685	
100	2000	0.884	
200	2000	0.8707	
500	2000	0.9005	
1000	2000	0.9044	
1500	2000	0.905	
2000	2000	0.9053	

Figure 2: Augmented results from question 5

Epochs	Batch Size	Test Accur
10	10	0.6844
50	10	0.8686
100	10	0.8831
200	10	0.8709
500	10	0.8999
1000	10	0.9052
1500	10	0.905
2000	10	0.9053
10	50	0.6912
50	50	0.8678
100	50	0.8833
200	50	0.8694
500	50	0.8999
1000	50	0.9049
1500	50	0.905
2000	50	0.9051
10	100	0.6876
50	100	0.8679
100	100	0.8842
200	100	0.8705
500	100	0.8996
1000	100	0.9046
1500	100	0.9053
2000	100	0.9053
10	200	0.6807
50	200	0.869
100	200	0.883
200	200	0.871
500	200	0.8998
1000	200	0.9054
1500	200	0.9049
2000	200	0.9047
10	500	0.6813
50	500	0.8683
100	500	0.8827
200	500	0.8706
500	500	0.9002
1000	500	0.9051
1500	500	0.9051
2000	500	0.9045
10	1000	0.6914
50	1000	0.8685
100	1000	0.8828
200	1000	0.8707
500	1000	0.8997
1000	1000	0.9051
1500	1000	0.9045
2000	1000	0.9056
10	1500	0.6853
50	1500	0.8686
100	1500	0.8826
200	1500	0.8703
500	1500	0.8998
1000	1500	0.9051
1500	1500	0.9053
2000	1500	0.9054
10	2000	0.686
50	2000	0.8685
100	2000	0.884
200	2000	0.8707
500	2000	0.9005
1000	2000	0.9044
1500	2000	0.905
2000	2000	0.9053

Figure 1: CSV output file from question 5

We addressed GA 3 in question 3, and 4 by implementing successful and correct softmax, log-likelihood functions; working for loops; and a functioning logistic regression model.

We addressed GA 4 by comparing adjusted hyperparameters for our logistic regression model through experimentation and reporting on the results in the above report.

We addressed GA 5 throughout the practical. The practical was based in Python and so we had to be sufficient in this coding language and the Numpy library to complete it, as we did.

We addressed GA 6 through our reporting on the practical.