COSI 134 PA1: Logistic Regression

Elena Álvarez Mellado

1 Implementation

The file `maxent.py` contains the implementation for the logistic regression classifier. The `main` method in the file `maxent.py` should be able to run all the experiments and print the results (please note that the experiments were done in several series by commenting and uncommenting the code, not everything in one go).

The method `train_sgd()` sets the different parameters needed for the classifier: training instances, lambda value (will be 0 if none is specified), learning rate, batch size, etc. This method will also call the `featurize()` method on the training set and the method for computing the gradient descent.

The `minibatch_grad_desc()` splits the training set into minibatches of the size as specified. The gradient will be computed by `gradient_instance()`. The field `parameters` (the weights) will be updated with each minibatch. The process will repeat for at least 50 full iterations on the training set (50 epochs) and until the difference between the last and the previous negative log likelihood becomes smaller than 30 (I came up with these 2 conditions experimentally and they ensure that the process both finishes and reaches decent results).

2 Experiment 1

For experiment 1, five different training sizes were set: 1000, 10000, 50000, 100000 and all, while keeping the development and test sets fixed. In all sizes, the batch size was fixed to 30, the learning rate was 0.0001 and no regularization was applied (lambda = 0).

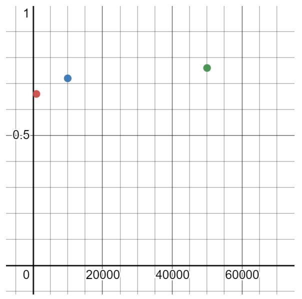

Figure 1: Training set size and accuracy

Training set size	Accuracy
1000 instances	0.66
10000 instances	0.72
50000 instances	0.76

Table 1: Training set size and accuracy

However, it was only possible to run sizes 1000, 10000 and 50000 successfully. Higher training sets produced constant memory errors, even running it on Lycastus server. In fact, it took several times to train using the 50000 instances and it took more than 12 hours to finish. Comparing these three training sets, it seems that the bigger the training set, the better the results, with 50000 training set producing the best result, 0.76 (see table and figure 1).

3 Experiment 2

Due to the difficulties to train the model using 50000 instances (more than 12 hours to run, constant memory errors), experiments 2 and 3 were run using only 10000 instances, even when the 50000 training set had produced a higher accuracy in experiment 1.

Batch size	**Accuracy**
1 instance	0.68
10 instances	0.74
50 instances	0.72
100 instances	0.62
1000 instances	0.17

Table 2: Batch size and accuracy (Training size = 10000)

For experiment 2, different batch sizes were tested: 1, 10, 50, 100 and 1000. The learning rate was the same as in experiment 1 and again no regularization was applied. The batch size 10 produced the best accuracy results, 0.74 (see table and figure 2). The accuracy obtained with batch size 1000 was remarkably low (see table and figure 2).

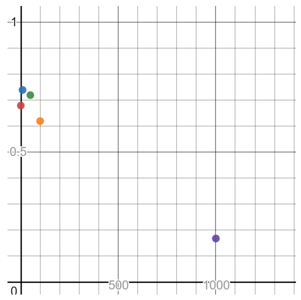

Figure 2: Batch size and accuracy

4 Experiment 3

For experiment 3, different values were tried for regularization. The lower the lambda value, the better the accuracy (see table and figure 3). The value 0.1 resulted in the best accuracy (0.70), although that result is lower than the accuracy previously obtained in experiment 2, when no regularization was applied. The lambda value 0.5 produced a very strange 0.12 accuracy.

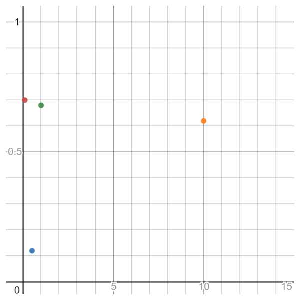

Figure 3: Lambda values and accuracy

Lambda values	**Accuracy**
0.1	0.70
0.5	0.12
1	0.68
10	0.62

Table 3: Lambda values and accuracy (Training size=10000; batch size=10)

5 Experiment 4

In order to try to speed up the process and avoid memory errors, a different featurization approach was explored with the hope that it could both make the process run faster and perhaps boost accuracy: I tried creating a bag-of-words featurization using only the words contained on the last sentence of each review. This approach was based on the intuition that normally the last sentence of a review will contain the overall sentiment of the entire review. However, it lowered a bit the accuracy results and it did not improve the speed. Therefore, I went back to the bow approach. The code for this experiment has been left as comment on the `test_maxent.py` file.