CSE 5525 Text Classification (part 2)

Alan Ritter

In the second part of this assignment, you will implement the perceptron algorithm (with parameter averaging) and a simple feed-forward neural network, using the same IMDB sentiment analysis dataset you experimented with in part 1. This assignment will ask you to inspect perceptron model's parameters and compare the performance of the three algorithms.

As before, the experiments required to complete the assignment may take some time to run - so the efficiency of your code will be an important consideration, and it is a good idea to start early.

Gradient of Logistic Regression (written question - 2 points)

Derive the gradient of the conditional log likelihood objective for multi-class logistic regression:

$$\frac{\partial}{\partial w_i} \sum_j \log P(y_j|x_j) \tag{1}$$

Perceptron

Implement the perceptron classification algorithm (use the starter code provided in `Perceptron.py`). The only hyperparameter is the number of iterations. Run the classifier and report results on the dev set with various numbers of iterations (for example: 1, 10, 50, 100, 1000).

Parameter Averaging

Modify the perceptron code to implement parameter averaging. Instead of using parameters from the final iteration, w_n, to classify test examples, use the average of the parameters from every iteration, $\sum_{i=1}^{N} w_i$. A nice trick

for doing this efficiently is described in section 2.1.1 of Hal Daume's thesis (`http://www.umiacs.umd.edu/~hal/docs/daume06thesis.pdf`).

Compare the performance of your implementation with and without parameter averaging. Also compare to the results of your Naive Bayes classifier.

Features

Print out the 20 most positive and 20 most negative words in the vocabulary sorted by their weight according to your model This will require a bit of thought how to do because the words in each document have been converted to IDs (see `Vocab.py`). The output should look something like so:

```
word1_pos weight1
word2_pos weight2
word3_pos weight3
...

word1_neg weightk
word2_neg weightk+1
word3_neg weightk+2
...
```

Where `wordn_pos` and `wordn_neg` are the top 20 positive and negative words. (Hint: you might find the `numpy.argsort` method useful - `http://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html`). Please include this output in your report.

Feed Forward Neural Network

First, we provide sample code for using Pytorch to implement a simple neural network that is capable of learning the XOR function. Try running the following command:

```
python3 xor.py
```

Read through the code for `xor.py`, which provides an example of how to use Pytorch to define a computation graph, initialize the parameters of a neural network, compute gradients and optimize a loss function.

Your next task is to implement a simple feed-forward neural network to classify the IMDB sentiment dataset. We recommend starting with an

Embedding Layer[1], a ReLU nonlinearity[2], a single hidden layer and Adam optimizer.[3] Please refer to the Pytorch Documentation for more information.

Evaluation

Once your implementation is working, spend some time tuning hyperparameters (learning rate, optimizer, number of epochs, number of hidden units, etc.) on the provided development set. Once you are satisfied with the results, switch your code to evaluate on the test data (see the main functions in `Perceptron.py`, `FFNN.py` and report performance of these two methods (you can also compare to Naive Bayes as well. You should only evaluate performance on the test data once, and report whatever numbers that you get (do *not* tune your models on the test set).

What to Turn In

Please turn in the following to the dropbox on Carmen:

1. Your code.
2. Your answer to the written question.
3. A brief writeup that includes the numbers/evaluation requested above (text file format is fine).

[1] `https://pytorch.org/docs/stable/nn.html#embedding`
[2] `https://pytorch.org/docs/stable/nn.html#relu`
[3] `https://pytorch.org/docs/stable/optim.html#torch.optim.Adam`