CS134 (Fall 2016) Programming Assignment 2

September 28, 2016

1 Maximum Entropy Model

Due October ??, 2015

1.1 Overview

Maximum Entropy classifier is the most used classifier in natural language processing. It's a strong statement but it's true. It's also used very widely in other fields. The model has many strong points. It handles the case where you have much more features than the number of data points. It also handles the correlation between features really well. In addition, the time and space complexity for the model is not too taxing. All of these reasons make MaxEnt model a very versatile classifier for most NLP tasks.

It is hard to go through an NLP class these days and not build a classifier for sentiment analysis. So we are going to use MaxEnt model for sentiment analysis.

1.2 Getting ready

Download the starter code along with the datasets using the link on latte. If you don't have the overall idea of how to implement MaxEnt, look at the slides from the office hours (will upload later on latte). There's quite a bit of disconnect between theory and practice for MaxEnt, so we highly recommend reviewing the slides before starting implementing.

1.3 Datasets

- **Sentiment analysis on Yelp dataset.** We processed the data from Yelp dataset challenge and put it in the json format. Review class provided in corpus.py reads in the input for you. Each restaurant review is tagged with 'negative' (1-2 stars), 'neutral' (3 stars), or 'positive' (4-5 stars). We are building a classifier that tags a review as one of the three types of sentiments based on the review text.
- **Name classification.** Just like the last assignment. This should be used as your test case. You can just use the first character and the last character as your features.

1.4 What needs to be done

Maximum Entropy model has a lot of moving parts. The list below guides you through what needs to be done. It comes down to The Three Questions that we learn about in class: representation, learning, and inference.

1.4.1 Representation

- Choose the feature set. Use something reasonable and make sure the feature set is reasonably small or the learning algorithm will take a long time.
- Choose the data structures that hold the features. We recommend sparse feature vectors. Regardless of your choice, cache the features internally within each Document object as the algorithm is iterative. Featurization must be done only once.

- Choose the data structures that hold the parameters. We recommend using a kxp matrix where k is the number of labels, and p is the number of linguistic features.

1.4.2 Learning

- Compute the negative log-likelihood function given a minibatch of data. You will need this function to track progress of parameter fitting.
- Compute the gradient with respect to the parameters. You will need the gradient for updating the parameters given a minibatch of data.
- Implement mini-batch gradient descent algorithm to train the model to obtain the best parameters.

1.4.3 Classification/Inference.

- Apply the model on the unseen data. The provided test cases will evaluate the model on both datasets, and you must pass those.

1.4.4 Experiments

In addition to implementing mini-batch gradient descent and MaxEnt model, you are asked to do the following experiments to understand the algorithms a little more. For both experiments, use the Yelp review dataset as it is a more realistic one (Real People. Real Reviews. or so they claim).

Experiment 1:

Why are we allowed to use mini-batch instead of the whole training set when updating the parameters? This is indeed the dark art of this optimization algorithm, which works well for many complicated models, including neural networks. Computing gradient is always expensive, so we want to know how much we gain from each gradient computation.

In this experiment, try minibatch size = {1, 10, 30, 50, 100, 1000}. For each mini-batch size, plot the number of datapoints that you compute the gradient for so far (x-axis) against the accuracy of the dev set (y-axis). Analyze and write up a short paragraph on what you learn or observe from this experiment.

Experiment 2 :

Does size really matter? The mantra of machine learning tells us that the bigger the training set the better the performance. We will investigate how true this is.

In this experiment, fix the feature set to something reasonable and fix the dev set and the test set. Vary the size of the training set {1000, 10000, 20000, 50000, 100000, or more} and compare the (peak) accuracy from each training set size. Make a plot size vs accuracy. Analyze and write up a short paragraph on what you learn or observe from this experiment.

1.5 Submission

Submit the following on Latte:

All your code. But don't include the datasets as we already have those.

Report. The report should include all the results from the experiments. And it should explain your experiment setting so that other people can replicate it. It should explain clearly what feature set is being used and how you set up mini-batch gradient descent because there can be quite a bit of variation. This should not be more than two pages single-spaced including graphs.

1.6 More on Mini-batch Gradient Descent

In this assignment, we will train MaxEnt models using mini-batch gradient descent. Gradient descent learns the parameter by iterative updates given a chunk of data and its gradient.

If a chunk of data is the entire training set, we call it batch gradient descent.

```
In [ ]: while not converged:
            gradient = compute_gradient(parameters, training_set)
            parameters += gradient * learning_rate
```

Batch gradient descent is much slower. Each update requires us to compute gradient from the entire dataset. This is usually not necessary. Computing gradient from the smaller subset of dataset at a time usually gives the same results if done repeatedly.

If the data is subset of the training set, we call it mini-batch gradient descent. This approximates the gradient of batch gradient descent. Each update only requires the computation of gradient of a small subset of the data (called mini-batch of data)

```
In [ ]: while not converged:
            minibatches = chop_up(training_set)
            for each minibatch in all minibatches:
                gradient = compute_gradient(parameters, minibatch)
                parameters += gradient * learning_rate
```

If a chunk of data is just one instance from the training set, we call it stochastic gradient descent (SGD). Each update only requires the computation of gradient of one instance of data.

```
In [ ]: while not converged:
            for each datapoint in training_set:
                gradient = compute_gradient(parameters, datapoint)
                parameters += gradient * learning_rate
```

1.6.1 Practical issues with mini-batch gradient descent

- How should I initialize the parameters at the first iteration?

  Set them all to zero. This is generally not advisable for more complicated model. But for the Maximum Entropy model, zero initialization works perfectly.

- How do I introduce the bias term?

  Include a feature that exists in ALL data instances. And treat it as a normal feature and proceed as usual.

- Why do the posterior P(Y|X) become NaN?

  It is very likely that you exponentiate some big number and divide by the same amount i.e. if unnoramlized_score is a vector of unnormalized scores (the sum of lambdas), then:

  ```
  posterior = exp(unnormalized_score) / sum(exp(unnormalized_score))
  ```

  This is no good. We have to get around by using some math tricks:

  ```
  posterior = exp(unnormalized_score - scipy.misc.logsumexp(unnormalized_score))
  ```

  If this confuses you or you are not sure why this is correct, think about it more or ask Chuan or Kahlil. But we are quite confident that you will need to use logsumexp function.

- How do you know that it converges?

  It is extremely difficult to know. If you stop too early, the model has not reached its peak yet i.e. underfitting. If you stop too late, the model will fit too well to the training set and not generalize to the unseen data i.e. overfitting. But there are multiple ways to guess the convergence. We suggest this method called early stopping.

  Every once in a while evaluate the model on the development set during gradient descent.

  – If the performance is better than last evaluation, then save this set of parameters and keep going for a little more.

  - If the performance stops going up after a few updates, stop and use the last saved parameters. (How many is a few? Up to you)

- How often should I run evaluation on the dev set during training?

  Up to you. It is actually OK to run the evaluation on the dev at every update you make to the parameters.

- How do I know that my implementation is correct?

  Look at the average negative log-likelihood. It should keep going down monotonically i.e. at every single update. You should also see that the gradient should get closer and closer to zero.

```
In [ ]:
```