COSI134A FALL 2019

Project1: Maximum Entropy Classifier *Jiawei Zhang*

This project is my implementation of applying multinomial logistic regression on Yelp dataset to classify reviews' sentiment. On average, the program takes 90s to train on 10,000 randomly selected training set, including loading data, extracting features, training and evaluating. The classifier's accuracy is between 70-75%.

Code Structure

maxent.py: In training function, we first get labels of data using *set_labels()*. Then we get whole vocabulary with *set_whole_vocabulary(),* then create feature matrix with *create_feature_mat()*, initialize the weights with *initialize_weights()*. In the gradient descent, we use *compute_gradient_reg()* to compute the regularized gradient of minibatch data and then update the weights until converge. We use *compute_neg_logliklihood()* to compute loss of development data. The converge condition is set by either reaching max iteration times 100 or percentage change for loss is between (-0.05%, 0) for 5 times in a row. We can also change the parameters including *training set size, learning rate, batch size, lambda*.

exp_and_plot.py: do all the four experiments and plot the results, it might take hours to run

test_maxent.py: rewrite the class BagOfWords and Name and move it to Corpus.py.

Corpus.py: add BagOfWords, Names, BagOfWordsBigram, BagOfWordsTrigram class

How to run the classifier

First we load corpus and assign its' document class (ex. BagOfWords), then train and classify the Maximum Entropy Classifier.

```
reviews = ReviewCorpus('yelp_reviews.json', document_class = document_class)
classifier = MaxEnt()          #call the classifier
classifier.train(train, dev)   #train the classifier
classifier.classify(dev[0])    #predict the label
```

The *classifier.train()* function will output loss on development set of each iteration and accuracy of development set.

Experiment settings

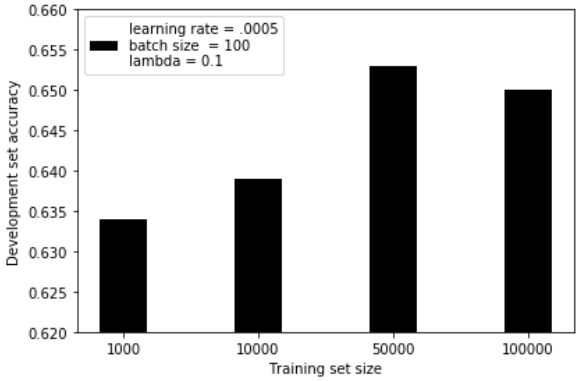
a) Feature engineering

Pre-process: get rid of all the punctuations and turn words into lowercase.

The whole vocabulary consists of all words except some stopwords (from NLTK) in 20,000 randomly selected samples. The vocabulary is stored in a dictionary with values denoting presence sequence. Weights is stored is a $k \times (p+1)$ matrix. K denotes k categories and p+1 denotes p features and 1 bias term. Data matrix is stored is a $n \times (p+2)$ matrix. N denotes n observations and p+2 denotes p features, 1 bias term and 1 label term.

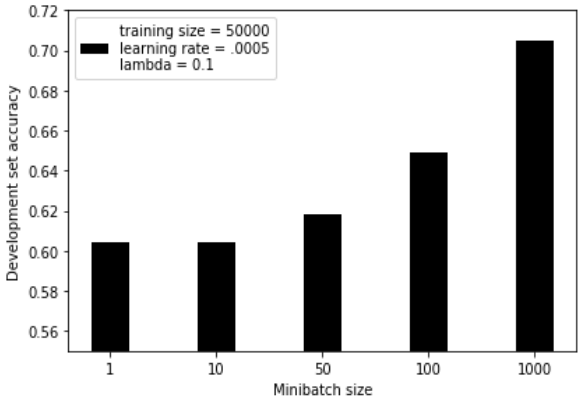
b) development set and test set are fixed (1,000 randomly selected reviews)

Exp1: training set size



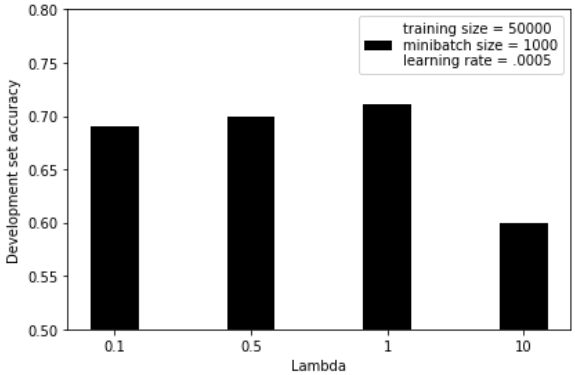
It seems as training set size get larger, the accuracy of development first increases and then decreases. Within certain limits, bigger training set contains more information and can provide better prediction. However, when the number goes too large, the training set might contain some noise which leads to overfitting. The best training size is 50,000 in this experiment. It's also worth noticing that only 1,000 sample can also train a pretty good model, with accuracy only 0.1 less than best model. If we are constrained by time and space complexity, we can choose smaller sample size as well.

Exp2: minibatch size



We can tell the larger the minibatch size, the higher the development set accuracy. While the program run time does increase as minibatch size goes up. Run time of 1000 minibatch size is nearly twice as much as 1 minibatch size (04:23 vs 02:21). In practical, we should make tradeoff between accuracy and computation time/power. In this project, we choose 1000 as the best minibatch size. Also, one possible guess for lower accuracy associated with smaller minibatch size is that it might need larger learning rate.

Exp3: lambda



We can learn from the plot that when lambda is small, the logistic classifier may fit the train data well but lose prediction power in dev/test data – overfitting; when lambda is large, models are too general -- underfitting, also leading accuracy goes down. Apparently, 10 is too large here. lambda = 1 is the best for yelp data.

Exp4: extra features

We add bigram and trigram to the features. The accuracy (bigram: 0.716, trigram: 0.725) does increase a little as we are adding new features. However, the time increase dramatically. The trigram model takes more than an hour to run. We should also consider the time complexity problem when in application.

Also, I learn a lot about learning rate, first I set a small rate, the accuracy on development set is too small; when the learning rate is too high, the loss might jump back and forth.