CS378 assignment1

Qinxin Wang

February 6, 2020

1 Perceptron

Q1 I implemented unigram perceptron algorithm here. My highest accuracy on dev set is 76.14%. It is achieved by training on the full training set for 20 epochs, with the step size initialized at 1, updated by $\alpha = \alpha * 0.8$ every epoch. The training and evaluation time is 2.23 seconds.

My unigram feature extractor creates a feature vector with the same size as the whole vocabulary, and use the occurrence count of every word in the sentence as its feature. I did lowercase before counting them.

In order to save training time, I used a list of words and their conuts as feature vector, instead of returning a $|V|$-d vector for every word. I also did cache the features in a dictionary after calculating them. Therefore, after all the features are calculated in the first epoch, training would be really fast.

Q2 After training the model for 20 epochs with a constant step size $\alpha = 1$, I can get an accuracy of 75.11%. I tried three different schedules, all trained for 20 epochs:

a) Decrease the step size by 0.05 every epoch. Accuracy is 75.80%.

b) Multiply the step size by 0.8 every epoch. Accuracy is 76.14%.

c) Multiply the step size by $1/t$ every epoch, t is the number of epochs. Accuracy is 75.11%.

Q3 10 words with highest positive weight: ('eyes', '7.28'), ('enjoyable', '7.2992256'), ('refreshing', '7.312'), ('wonderfully', '7.4'), ('manages', '7.427456'), ('human', '7.449856'), ('IMAX', '7.928'), ('appealing', '8.1472'), ('rare', '8.35232'), ('remarkable', '8.752')

10 words with lowest negative weight: ('flat', '-7.337856'), ('depressing', '-8.1824'), ('instead', '-8.19008'), ('mess', '-8.5696'), ('failure', '-8.6992'), ('suffers', '-8.752'), ('stupid', '-8.752'), ('worst', '-9.01433748836'), ('TV', '-9.1091712'), ('devoid', '-9.4')

I found many of them are words with a strong emotional tendency, like 'remarkable', 'appealing', 'wonderfully', 'manages' and 'stupid', 'worst', 'suffers', 'depressing'. But there are also nouns that seems not so positive or negative, like 'TV' and 'IMAX'.

Q4 Training accuracy is 99.91%, while development accuracy is 76.16%, which is much lower. That's because after training for many epochs, the weights converges to a point that it can predict all the data in training set correctly. But the model has never seen test data, so it will predict the test set using the same weights on the training set, therefore causing some mistakes. What fits the training set may not always fits the test set.

2 Logistic Regression

Q5 I get 77.64% accuracy on the development set with the same unigram feature as **Q1**. I trained with a fixed step size 0.5 for 30 epochs. The training and evaluation time is 5.78 seconds.

Q6 I tried five different step size: $\alpha = 0.1$, $\alpha = 0.5$, $\alpha = 1$, $\alpha = 1/t$, $\alpha = 1/\sqrt{t}$. Their accuracy are 76.61%, 77.64%, 76.15%, 76.38%, 76.72% respectively.

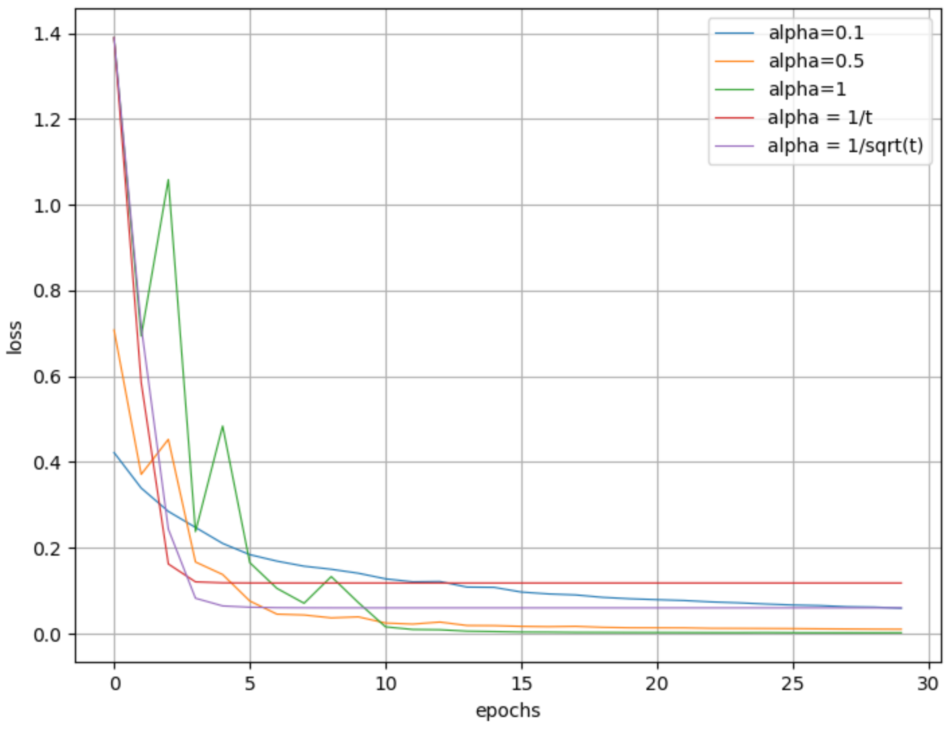

Figure 1: Loss functions with different step size

We can see that with a large step size (e.g., $\alpha = 1$), the loss function decreases slower, with more fluctuations. With a small step size, the loss function is smooth and decreases quickly. However, a smaller step size is likely to ends up with a larger loss value, while large alpha can have a loss closer to 0.

3 Features

Q8 I implemented Bigram features with every adjacent pairs of words in the sentence. I did experiments with perceptron and logistic regression algorithm with the same parameters as before. The accuracy is 68.12% for perceptron and 71.67% for LR.

Q9 Here is a table showing the performance of different feature extractors. Except for unigram and bigram features, I tried several other features listed below. For each algorithm (LR or perceptron), I used the same parameters as before.

a) Term frequency(TF).

$$TF_t = \frac{c_{t,d}}{\sum_{t' \in d} c_{t',d}} \tag{1}$$

$c_{t,d}$ denotes the count of term t in a document d. $c_{t',d}$ denotes the count of all the terms in d.

b) Log normalized term frequency(NTF).

$$NTF_t = log(TF_t + 1) \tag{2}$$

c) Term frequency–Inverse document frequency(TF-IDF).

$$IDF_t = \log \frac{|D|}{|d : t \in d|} \tag{3}$$

I tried two different term frequency expression with the same IDF. They used the term frequency in a) and b) respectively.

Table 1: Comparison of different feature extractors.

	Perceptron(%)	LR(%)
unigram	76.14	**77.64**
bigram	68.12	71.67
unigram + bigram	75.34	77.29
TF	70.76	75.92
NTF	70.64	75.69
TF-IDF-1	75.11	**77.41**
TF-IDF-2	**76.95**	77.40

I used TF-IDF-1 in my BetterFeatureExtractor. In order to achieve the best result, I adjusted the parameters according to with feature extractor is used. I used step size $\alpha = 1, \alpha = \alpha * 1/t$ in my better feature implementation for LR and achieved 77.75% accuracy.