Ningqian Zhang

CS2731

Assignment1 Report

Implementation of Language Model

For all three models, I first store all the vocabularies appear in the training corpus in a dictionary (vocab\_dict). I set the vocabularies as keys and the number of occurrences as values. After all the vocabularies are stored, I go through the whole dictionary to check for OOVs. I remove all the (key, value) pairs with value equals to 1 and at the same time add 1 to the value of key “<unk>”. Then I store this dictionary in vocabulary.pkl and store the model type as well as whether smoothed in model\_type.txt.

For the unsmoothed unigram model, I do not need to store anything else for rebuilding the language model, because I can obtain the context size which equals to the count of bigrams by adding up all the values in the vocabulary dictionary.

For the unsmoothed trigram model, I fist insert two <s> at the beginning of and one </s> at the end of each sentence before storing the vocabularies in these sentences, so that these two characters will be included in the vocabularies. When I need the vocabularies size later, I will calculate it with the number of keys in the vocabulary dictionary minus 1, since <s> should not be counted as one vocabulary. Then I create another dictionary (trigram\_dict) to store all the trigrams and bigrams as well as their number of occurrences for later use. For this trigram\_dict, I replace all the tokens that are not included in the vocab\_dict with <unk>. The files I store are model\_type.txt, trigram\_counts.pkl and vocabulary.pkl.

For the smoothed trigram model, the train.py is exactly the same with the unsmoothed trigram model. I do not need to store the vocabulary size separately in a file. I can obtain it by the simple calculation described above.

Perplexity Scores:

1. unsmoothed unigram model:

wsj: 219.061827761

sb: 123.175339826

2. unsmoothed trigram model:

wsj: NaN

sb: NaN

3. smoothed trigram model:

wsj: 1815.37932587

sb: 558.142633127

In genre detection, the smoothed trigram model has the best performance. The accuracy of unigram model is 359/1000 (35.00%), the accuracy of unsmoothed trigram model is 155/291 (53.00%), the accuracy of smoothed trigram model is 612/1000 (61.00%).

My run-task.py decides whether a sentence is from wsj or sb by adding up the probabilities of ngrams based on the language model. Since the unsmoothed trigram has the problem that it could not deal with unseen trigrams, calculating the perplexity or log probabilities will result in error. Also to avoid overflow caused by multiplying all the probabilities together, I instead do the summation of all the probabilities a model run on one corpora. If an unseen trigram appears while running the unsmoothed trigram model, I simply assign probability 0 to that trigram. When the test sentences are all processed, I compare the two probabilities and decide the sentence is from the corpora with a larger probability. Since I use this method for the unsmoothed trigram model, it would be better to use the same method for other language model as well.

The scores I get combine the information of how likely the smaller parts of the sentence would appear in those two corpus respectively. If most of the smaller parts, no matter it is a unigram or a trigram, are very likely to appear in a corpora, then the whole sentence would be likely to appear in that corpora. Since a larger probability means something being more likely to happen, the larger the score of a corpora I get would mean the sentence is more likely to be written in that genre.

The OOV handling strategy assigns the number of occurrences of all the tokens that appear only once to one single <unk> token, which makes the <unk> seem to appear a lot in that corpora. For example when running unigram model, when an unseen unigram appears, we map this unseen unigram to <unk> regardless of whether this unseen unigram may in fact have very small number of occurrences. In this way, we are actually overestimating this unseen unigram. Based on our strategy of mapping all the tokens with one occurrence to <unk>, this problem will be increasingly worse if we have a small corpora and there happen to be a lot of tokens with only one occurrence, because the number of occurrence of <unk> will be unreasonably large. If this is the case, then what makes a genre have the higher scores would probably be a larger amount of unseen unigram rather than the sentence being more similar with the training sentences the language model trained on.

The comparison of my unigram and trigram perplexity scores indicate that the unigram model has lower perplexity scores which is better than the trigram model. According to the analysis above, this probably tells me that the unigram model can use the occurrences of all the unseen unigram in training data to represent any new unseen unigram it encounters during testing, which help it produce a higher score. One the other hand, the smoothed trigram is less likely to map an new unseen trigram to an old one, because there are more possible combinations of tokens when it comes to trigram and more importantly, it uses a smoothing method which smooth away some part of the impact of OOV by adding the vocabulary size to the denominator. Thus, even though the trigram model also use the OOV strategy, it still has very large perplexity scores

The comparison between unsmoothed and smoothed tells me that by smoothing the data, we can avoid facing the trouble of not being able to calculating the probabilities or perplexity because of the occurrences of many zeros. Both the log probability and the perplexity require non zero values, so having too many ngrams with zero occurrences will result in very biased performance of language models, especially when the corpora is relatively small.