E1-246 Natural Language Understanding Assignment-1

Sayambhu Sen
M.Tech (1st Year)
SE (Department of EE)
SR No. 14351
sayambhusen@iisc.ac.in

Abstract

This is a report for the first assignment of Natural Language Understanding. The objective is to create language models using a given corpus and understanding it's behaviour when parameters are changed.

1 Language Modelling tools

The language model implemented in this assignment by me is Bigrams and Trigrams using *Kneyser Ney Smoothing*. The Corpus used for training are brown corpus and gutenberg corpus. Python3 was the programming language used for this program in the environement "spyder". The libraries used for the programs are 'random', 'collections','numpy','ngrams from nltk' and 'train,test,split from sklearn' Three separate files are given. The first file named 'perplexcalc.py' calculates the perplexities. The file named 'Senes.py' generates the Bigrams and Trigrams from a given corpus. For now it is using only the gutenberg corpus since that produces the most natural sentences. The file 'Quicksent.py' is used to generate a sentence from the created Bigram and Trigram Probabilities created by the file 'Senes.py'.

2 Introduction

Two types of language modelling methods have been attempted for this assignment. The +1 smoothing or 'Laplace smoothing' technique and the Kneyser Ney Smoothing technique. Ultimately, it was decided to use only Kneyser Ney smoothed probabilities for calculating perplexity as well as for sentence generation.

3 Laplace Smoothing

For any given text document, unigram probability of a particular words is calculated using $Count(Word_j)/\sum_i Count(Word_i)$

This is the unigram probability of the word indexed by j and for a given corpus, this need not change. The size of the Vocabulary *V* is the number of unique words in the corpus given for training. Now for the test data we may have words that are not seen in the training data. Hence, we randomly sample some words with unigram count =1. And convert to a token ' <UNKN >'. Thus, any unseen word is accounted for in the model. Now, the main problem is the bigram and Trigram calculation.

$$P(Word_i|Word_{i-1}) = \frac{Count(Word_iWord_{i-1}) + 1}{CountWord_{i-1} + V}$$

where *V* is the size of the vocabulary.

For the trigram model, the probabilities are calculated as :

$$P(Word_i|Word_{i-1}Word_{i-2}) = \frac{Count(Word_iWord_{i-1}Word_{i-2}) + 1}{CountWord_{i-1} + V^2}$$

Thus, we find that even for medium size corpus, the Vocabulary size goes upwards of 50,000 causing the probabilities to be very small. This is a very crude method of calculating probabilities and causes the perplexity values to be very high. Perplexity reaches above 1000 just for bigrams itself. Hence, this model was discarded in favour of Kneyser Ney Smoothing since it is reported to have the best probability values.

3.1 Kneyser Ney Smoothing

Kneyser Ney smoothing is a method which takes into account not only the frequency of number of counts of a given word but it also takes into account, the continuation probabilities of a word. The continuation probability of a word is given by

$$P_{continuation}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_i |v : C(vw_i) > 0|}$$

This means this is the unique number of words in the bigram that follow a given word divided by the sum of all such possible values.

Also, while calculating the probabilities of bigrams a discount factor is used which distributes the Counts to other bigrams which may not appear in the test set. It is to be noted that the total number of bigrams possible is the square of the size of the vocabulary *V*. Hence, the bigram count matrix would be extremely sparse to begin with. Hence, bigrams and trigrams are stored in a dictionary with the values of the keys being the Count of the occurances of such bigrams.

Now, the bigram probabilities are calculated as

$$P_{KN}(w_i|w_{i-1}) = \frac{max((Count(w_i w_{i-1}) - d), 0)}{Count(w_i)}$$

$$+\lambda(w_{i-1})P_{continuation}(w_i)$$

where the λ is given by:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v Count(w_{i-1}v)}|\{w : Count(w_{i-1}w) > 0\}|$$

In general, for an N-gram model, the Kneyser Ney Smoothed Probabilities are given by

$$P_{KN}(w_i|w_{i-n+1}^{i-1}) = \frac{max((Count(w_{i-n+1}^{i}) - d), 0)}{Count(w_{i-n+1}^{i-1})}$$

$$+\lambda(w_{i-n+1}^{i-1})P_{KN}(w_i|w_{i-n+2}^{i-1})$$

$$\lambda(w_{i-n+1}^{i-1}) = \frac{d}{\sum_v Count(w_{i-n+1}^{i-1}v)}|\{w : Count(w_{i-n+1}^{i}) > 0\}|$$

Thus, for each case, we use lower order probabilities to calculate the higher order probabilities. In fact, For those trigrams whose Counts are zero. The discounting second makes up for the unknown words and makes the probabilities of higher orders non-zero.

3.2 Kneyser Ney Implementation

The following steps are taken for the Language modelling:

1. The data being chosen is first being divided int *80 %* training data and *20 %* test data.
2. It's unigram, bigram and trigram counts are stored in a dictionary.
3. The unigrams are calculated in such a way that some portion of the unigrams whose counts are 1 are replaced by a token called '<UNKN >'
4. Then for each unique word in the unigram, we find it's continuation count
5. Similarly, We find the Follow Count for use in calculating the values of Lambda.
6. Finally, we use the formula given above to calculate the Bigram probabilities of the test set.
7. Now for each bigram (non-unique) in the test set ,we calculate the probabilities and calculate the perplexity

   $$Perplexity = \frac{1}{(\Pi_i P_{KN}(w_i|w_{i-1}))^{1/n}}$$

   For trigram,

   $$Perplexity = \frac{1}{(\Pi_i P_{KN}(w_i|w_{i-1}w_{i-2}))^{1/n}}$$

   where n is the number of bigrams or trigrams in the corpus respectively
8. Now the discount factor d can be different for bigrams and trigrams respectively. So we have tested out the data for them.

Table 1: Effect of discounting(Gutenberg (train and test) Px: Perplexity)

$d1$	$d2$	$Px(bigram)$	$Px(trigram)$
0.75	0.75	156.66	46.75
0.5	0.5	146.94	42.48
0.5	0.75	147.30	45.29
0.75	0.5	173.73	46.76

Table 2: Effect of discounting(Brown(train and test) Px: Perplexity)

$d1$	$d2$	$Px(bigram)$	$Px(trigram)$
0.75	0.75	255.56	37.894
0.5	0.5	236.7896	35.046
0.5	0.75	236.7849	36.475
0.75	0.5	299.554	39.2153

4 Sentence Generation

Now that we have the different Bigram and Trigram probabilities for the different corpus (Brown, Gutenberg and it's mixture), we can now try to generate different sentences using these probabilities.

The following steps are used in Generating sentences:

Table 3: Effect of discounting(train: Gutenberg +Brown, test: brown Px: Perplexity)

$d1$	$d2$	$Px(bigram)$	$Px(trigram)$
0.75	0.75	170.496	51.724
0.5	0.5	160.954	47.176
0.5	0.75	160.85	50.179
0.75	0.5	188.17	51.61

Table 4: Effect of discounting(train: Gutenberg +Brown, test: Gutenberg Px: Perplexity)

$d1$	$d2$	$Px(bigram)$	$Px(trigram)$
0.75	0.75	344.504	56.5319
0.5	0.5	319.33	52.158
0.5	0.75	320.90	54.631
0.75	0.5	398.276	58.338

1. We start with "<s >" and look for all bigrams whose first word is "<s >".

2. Among them we randomly pick a bigram. Now we choose a random number uniformly between 0 and 1. If that number is lesser than the probability of that bigram, we choose that bigram as our first word.

3. Now to choose the next word, we look for all trigrams whose last two words are the last two words in our sentence.

4. We use the same random number methods described in a previous step to select it and move ahead.

5. If we find that the number of possible trigrams of a certain last two tokens is 0, then we reset and start generating the sentence from the begining.

6. For the ending, we choose all possible trigrams whose ending is </s >and then choose a trigram whose first word is our current last word and last word is our any of the first words of the ending sentence trigrams.

4.1 Example Sentences

Some example sentences generated are shown below:

1. With Gutenberg only ['<s >', 'You', 'must', 'be', 'made', 'by', 'the', 'way', 'that', 'his', 'heart', 'to', 'pray', 'before', 'to', 'Mrs', '.', "</ s >']

2. With brown only ["<s >', 'When', 'he', 'was', 'the', 'millions', 'of', 'the', 'theme', 'of', 'the', 'various', 'types', 'of', 'powers', 'a', 'year', "</s >']

5 Discussion

We find that the brown corpus contains a far greater number of texts than does gutenberg. Hence, for large and diverse corpora, the perplexity becomes quite large in comparison.

A possible way for varying discounts could be that the discounts will be tested for held out set only on certain corpus. Thus, a sort of lexical similarity for deciding the values of discounts can be used. This is true because we find the discounts to give a lower perplexity when the same corpus is used for testing and training.

References

- http://smithamilli.com/blog/kneser-ney/
- Speech and Language Processing Dan Jurafsky and James H. Martin