E1246 - Natural Language Understanding
Assignment1 : Language Models

Sachin Mittal (SR Number 14539)

Abstract

We designed a language model on **Brown**(D1) corpus and **Gutenberg**(D2) corpus using ngrams. Following are two tasks that we performed -

- **Task 1:** Divide both dataset into train, dev, and test and build the best LM in the following settings and evaluate.
  - **S1**: Train: D1-Train, Test: D1-Test
  - **S2**: Train: D2-Train, Test: D2-Test
  - **S3**: Train: D1-Train + D2-Train, Test: D1-Test
  - **S4**: Train: D1-Train + D2-Train, Test: D2-Test
- **Task 2:** Generate few sentences of 10 tokens.

1 Handling Unknown Words

It may so happen that some of the words in the test set will not appear **even once** in the training set . These words are called unknown words, and need to be handled in some way. We have used following technique to handle-
we can remove **some** of the words that occurred only once (rare words) from our train set, and replace them with a special ”unk” symbol, representing unknown words. And at the time of finding probabilities in test corpus, we replaced out of vocabulary (OOV) words with ”unk”.
Doing this we make sure that there is no unigram in test set that has count zero. (Either that unigram belong to train vocab or that is ”unk”)

2 Smoothing Techniques

We have used two smoothing techniques -

- Interpolated Kneser-Ney:
  We have implemented this for bigrams, below is the formula for **bigrams:**
  $P_{KN}(w_n|w_{n-1}) = \frac{\max\{C(w_{n-1},w_n)-D,0\}}{\sum_{w'} C(w_{n-1},w')} + \lambda(w_{n-1}) \times P_{cont}(w_n)$
  with the normalizing factor $\lambda(w_{n-1})$ given as
  $\lambda(w_{n-1}) = \frac{D}{\sum_{w'} C(w_{n-1},w')} \times N_{1+}(w_{n-1}\bullet)$
  and the continuation probability $P_{cont}(w_n)$ of a word w_n
  $P_{cont}(w_n) = \frac{N_{1+}(\bullet w_n)}{\sum_{w'} N_{1+}(\bullet w')}$
  where $N_{1+}(\bullet w)$ is the number of contexts w was seen in or, simplier, the number of distinct words $\bullet$ that precede the given word w.
  here D is hyper-parameter that needs to be tuned using dev corpus
- linear interpolation
  We have implemented this for **trirams:**
  In this, we combine different order N-grams by linearly interpolating all the models. Thus, we estimate the trigram probability $P(w_n|w_{n-2}w_{n-1})$ by mixing together the unigram, bigram, and trigram probabilities, each weighted by λ: $P(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$
  here $\lambda_1, \lambda_2, \lambda_3$ are hyper-parameters that needs to be tuned using dev corpus.

3 Evaluation Metric

We have used **perplexity** as metric. Perplexity of the whole corpus C that contains m sentences and N words, is given by

$$Perplexity(C) = \sqrt[N]{\frac{1}{P(s_1, s_2, ..., s_m)}}$$

The probability of all those sentences being together in the corpus C (if we consider them as independent) is:

$$P(s_1, ..., s_m) = \prod_{i=1}^{m} p(s_i)$$

In, simple terms It can we written like this - $Perplexity(C) = \sqrt[N]{\frac{1}{\prod_{i=1}^{m} p(s_i)}}$
$= 2^{\log_2 [\prod_{i=1}^{m} p(s_i)]^{-N}}$
$= 2^{-\frac{1}{N} \log_2 [\prod_{i=1}^{m} p(s_i)]}$
$= 2^{-\frac{1}{N} \sum_{i=1}^{m} \log_2 p(s_i)}$
We have used above formula to measure Perplexity.

4 Result

S1 : train = D1 train and test = D1 test	
Kneser-Ney for Bigrams	412.73
linear interpolation for trigrams	433.53
S2 : train = D2 train and test = D2 test	
Kneser-Ney for Bigrams	249.24
linear interpolation for trigrams	177.79
S3 : train = D1 train+D2 train and test = D1 test	
Kneser-Ney for Bigrams	525.68
linear interpolation for trigrams	370.58
S4 : train = D1 train+D2 train and test = D2 test	
Kneser-Ney for Bigrams	262.90
linear interpolation for trigrams	169.30

5 Sentence Genration

We generated sentences using trigrams, that takes previous two words into account. we pick a random word, but **no** uniformly - with the probability in the model. Suppose, We have 2 words in a model: ”yes” and ”no”, and the probability distribution is 2/3 ”yes”, 1/3 ”no”, than the generated text may look like this:
yes no no yes yes no yes yes yes no yes yes yes I.e., we’ll have approximately 2/3 ”yes” in the text and 1/3 ”no”. Below is Algo that we implemented-

Algo for generating text
Step 1. take initial two words i.e.<s ><s >
// now we want to pick word that follows probability distribution of model
Step 2. Generate a random number from 0 to 1.
Step 3. Iterate over all words that appears after our two previous words, summing their probability weights. As soon as the sum is larger than the generated number, emit the current word w.
Step 4. Finish sentence when encounter <\ s >
Step 5. go to Step 1 to generate more sentences