Intro to NLP Assignment 1

Vamshi Krishna Bonagiri

2020114011

Average Perplexity Scores

Language Model 1: Europarl Corpus, Kneser-Ney Smoothing

Avergage Perplexity on train: **3.9624327923463794**

Avergage Perplexity on test: **201.13697335549458**

Language Model 2: Europarl Corpus, Witten-Bell Smoothing

Avergage Perplexity on train: **2.777360375636144**

Avergage Perplexity on test: **188.7050823460798**

Language Model 3: Medical Abstracts Corpus, Kneser-Ney Smoothing

Avergage Perplexity on train: **3.4748307371587956**

Avergage Perplexity on test: **389.4713023244258**

Language Model 4: Medical Abstracts Corpus, Witten-Bell Smoothing

Avergage Perplexity on train: **2.347575168078692**

Avergage Perplexity on test: **357.4075441755355**

Observations

Kneser-Ney Smoothing performed almost the same in both the corpii during testing but gave a bigger perplexity on the Medical Abstract corpus compared to the Europarl corpus, meaning it works very effectively on text that it has already seen (average perplexity < 3) compared to new words and ngrams being introduced, which was more well observed in the Medical Abstracts Corpus. Witten-Bell performs in a very similar manner with similar perplexity results as compared to Kneyser-Ney in all the comparable situations. This could mean that both the language models are good when it comes to generalization, as they both give similar results in both the corpii, with witten-bell being slightly better in all of them.

One inconsistency occuring in both the models is when there are unseen ngrams, this leads to their $N1+$ counts and normal counts becoming 0, but these are used in the denominators of the formulae provided for both the smoothing cases. This was avoided by returning very small values instead of zero to ensure that the interpolation takes place, however, it still poses a problem when the numerator is large and denominator is close to zero as it halts the backoff process by giving all the probability mass to one term.

Tokenization

Apart from handling the cases of punctuations, URLs, Hashtags and Mentions, many other ways to clean the data were implemented. They are as follows:

- Making the text lowercase to make sure the ngram counts are on-point.
- Replacing
  - Percentages in the form of x% as `<PERCENTAGE>`
  - All date format to `<DATE>`
  - All time formats to `<TIME>`
  - other values with numbers as `<NUMBER>`
- Any repeated punctuations such as ..... or !!!!!!!!!!! to `.` and `!`
- Any letter repeated more than twice such as tiiiiiiiired -> tired and oooooops to oops
- Replaced:
  - can't -> cannot
  - n't -> not
  - 'm -> am
  - 's -> is
  - 're -> are
  - 'll -> will
  - 'd -> would
  - 've -> have
- Splitting punctuations before and after a word, such as "good." -> "good ." or "[Dang]" -> "[Dang]" to make sure the words are counted.
- Splitting hyphenated words to include them in word counts

Code Optimization

The Code for both Kneser-Ney and Witten-Bell were optimized by precomputing all the values necessary while calling the smoothing functions. All the N_+ values were counted using Dynamic Programming while making the dictionaries for each ngram, instead of calculating them for every ngram with $n * V$ time complexity every time needed, Which is very costly compared to the O[1] complexity provided by precomputing. This has significantly improved the runtime of the algorithm allowing it to run on the whole corpus in just few seconds.

Handling Unknown Words

Unknown words were handled by replacing every word appearing only once in the corpus(frequency=1) to `"<UNK>"`. (as mentioned in Jurafsky Martin Chapter 3.)

This would make the count of `"<UNK>"` = no.of.unigrams with frequency = 1

The specific frequency of 1 was chosen because the corpus had 20,000 words, a lower value is usually preferred to prevent underfitting (as more unknowns lead to more perplexity scores regardless of the corpus).

Any new word appearing in the test corpus will be treated as this Unknown token and evaluated accordingly.

Kneser-Ney Smoothing

Formula used:

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \frac{\max\{c(w_{i-n+1}^{i}) - D, 0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} + \frac{D}{\sum_{w_i} c(w_{i-n+1}^{i})} N_{1+}(w_{i-n+1}^{i-1}\bullet)\, p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1})$$

with the base case:

$$p_{\mathrm{KN}}(w_i) = \frac{N_{1+}(\bullet w_i)}{N_{1+}(\bullet\bullet)}$$

where

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : c(w_{i-1}w_i) > 0\}|$$

is the number of different words w_{i-1} that precede w_i in the training data and where

$$N_{1+}(\bullet\bullet) = \sum_{w_{i-1}} N_{1+}(w_{i-1}\bullet) = |\{(w_{i-1}, w_i) : c(w_{i-1}w_i) > 0\}| = \sum_{w_i} N_{1+}(\bullet w_i)$$

Witten-Bell Smoothing

Forumla used:

$$p_{\mathrm{WB}}(w_i|w_{i-n+1}^{i-1}) = \lambda_{w_{i-n+1}^{i-1}}\, p_{\mathrm{ML}}(w_i|w_{i-n+1}^{i-1}) + (1 - \lambda_{w_{i-n+1}^{i-1}})\, p_{\mathrm{WB}}(w_i|w_{i-n+2}^{i-1})$$

$$1 - \lambda_{w_{i-n+1}^{i-1}} = \frac{N_{1+}(w_{i-n+1}^{i-1}\bullet)}{N_{1+}(w_{i-n+1}^{i-1}\bullet) + \sum_{w_i} c(w_{i-n+1}^{i})}$$

where:

$$N_{1+}(w_{i-n+1}^{i-1}\bullet) = |\{w_i : c(w_{i-n+1}^{i-1}w_i) > 0\}|$$

THE END