Assignment 1: Language Modeling

28th July, 2018

1 Installing Anaconda and NLTK

The assignment is aimed to familiarize one with programming in python and using some aspects of the NLTK library.

To ensure we are all on the same page, the coding environment will be in **python3**. For those, who are on Windows or those on Unix who do not wish to mess up their environment variables, I'd advise downloading anaconda3 and make that your default coding environment for NLP.

The link to anaconda3 for Windows and Linux is available here.

The steps to install NLTK is available on the link:

```
sudo pip3 install nltk
python3
nltk.download()
```

2 Language Modelling without smoothing

The first part of the assignment involves creating simple language models on the training corpus. The training data includes only the first 40,000 sentences of **brown corpus** already available in nltk. The following preprocessing on the training data are carried out:

- Case-folding: Convert all the letters to lowercase
- Retain only alphabets and spaces after removing numerals and special characters.

Assignment Task 1:

In this phase you are expected to carry out the following tasks:

- Create the following language models on the training corpus:
  - Unigram
  - Bigram (with padding)
  - Trigram (with padding)
- Verify Zipf's Law on the aforementioned 3 language models.

  Zipf's law states that the frequency of a word (f) is inversely related to its position in the list or rank (r).

$$f \propto \frac{1}{r} \tag{1}$$

- List the top 10 unigram, bigram and trigrams in each case.

- Evaluate the log-likelihood and perplexity scores for the following sentences. For each of these sentences, please insert the appropriate padding at the beginning and end of the sentences for the bigram and trigram language model.
  - he lived a good life
  - the man was happy
  - the person was good
  - the girl was sad
  - he won the war

These sentences will be submitted in an input text file along with this assignment.

3 Language Modelling with Smoothing

As emphasized heavily, smoothing is essential to account for unforeseen words or ngrams. We will explore some simple ones in this particular assignment:

3.1 Laplacian Smoothing/ Additive smoothing

The key idea behind Additive Smoothing is that each n-gram occurs k times more than its actual count. Consequently, the new probability is

$$p_{add}(w_i|w_{i-n+1}..w_{i-1}) = \frac{k + count(w_{i-n+1}..w_i)}{k|N| + count(w_{i-n+1}..w_{i-1})} \tag{2}$$

Here, k ϵ (0,1] and $|N|$ denotes the number of unique n-grams.

Assignment Part 2:

Implement the additive smoothing model for the unigram, bigram and trigram language model.

Also report the log-likelihood and perplexity scores of the above 5 sentences for different values of k where k ϵ $10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1$.

3.2 Good Turing Smoothing

The key notion behind Good Turing estimate is to 'reassign the probability mass of n-grams that occurs r+1 in the training corpus to those n-grams that occurs occurs r times'. Simply, put we compute an adjusted count r^* for those n-grams that occurs r times which is

$$r^* = (r+1)\frac{n_{r+1}}{n_r} \tag{3}$$

where, n_r are the n-grams that are seen exactly r-times.

Consequently, the new probability mass of a n-gram than initially occurs r times is now:

$$P_{GT}(x : count(x) = r) = \frac{r^*}{\sum_{r=1}^{\infty} rn_r} \tag{4}$$

Let's illustrate the working of Good Turing via an example:

Question: In a corpus it was found that the number of unique unigrams is 1446 and the total number of bigrams is 9420. Suppose out of 9420, 5315 bigrams occur once in the corpus. Use Good Turing smoothing to estimate the effective count for the bigrams not seen in the corpus.

Answer: The total possible set of bigrams is 1446*1446 = 2090916

The number of bigrams that occur only once = n_1 = 5315

The total number of bigrams that occur uniquely= N= 9420

The number of bigrams that occur 0 times = n_0= 2090916- 9420= 2081496

Thus the effectively count of bigrams that occur 0 times is $0^*= 1\frac{5315}{2081496} = 0.002553$

Thus, Good Turing enables us to account for tokens or n-grams that have not been seen at all.

Assignment Task 3:

Implement the Good Turing smoothing for the bigram and trigram language model.[1] Suggest why it is not possible to do the same for the unigram model.

Compute the log-likelihood and perplexity scores for the 5 sentences using the modified Good Turing smoothing models.

4 Interpolation Method:

An important observation for the above smoothing methods is that it assigns the same probability mass to unforeseen ngrams, irrespective of the frequency of the lower order grams. Consequently we, interpolate between higher and lower order models.

The interpolation models for the bigram and trigram models are given below:

$$P\ (w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i) \tag{5}$$

$$P\ (w_i|w_{i-2}w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + (1-\lambda_1-\lambda_2)P(w_i) \tag{6}$$

Assignment Task 4: Implement the interpolation models for the bigram model

Evaluate the log-likelihood probabilities and perplexity scores for the above five sentences for different coefficient values. For bigram model, compute for $\lambda\ \epsilon\ 0.2, 0.5, 0.8$

Bonus: No MARKS Implement the interpolation models for the trigram model. Evaluate the log-likelihood probabilities and perplexity scores for the above five sentences for the best value of λ_1 and λ_2. You can estimate the best value of these λ_1 and λ_2 values by testing on a held-out set . The held-out set is the remaining sentences of the Brown corpus.

5 Deliverables:

Submit the codes for each of the following language models in a single Assignment_1_YourRollNo.py file.

This .py file should take in an input text file (similar to the one provided along with the assignment) and give the scores of the sentences in the text file.

Write a single consolidated report highlighting the observations and findings of these experiments and submit it as Assignment_1_YourRollNo.pdf .

[1] Also since we consider padding in bigram and trigram model, please include the tokens $< s >$ and $< /s >$ inside the unigram vocabulary.