Exercise Sheet VI

Submission Deadline: June 19th, 23:59

Long-Range Dependencies

1) **Correlation Function (2 points)**
Words in natural languages exhibit both short-range local dependencies as well as long-range global dependencies to various degrees. To analyse these dependencies, the temporal correlation function can be used. The correlation between two words w_1 and w_2 is defined as

$$Corr_D(w_1, w_2) = \frac{P_D(w_1, w_2)}{P(w_1)P(w_2)} \tag{1}$$

where $P_D(w_1, w_2)$ is the probability of observing w_2 after w_1 separated by a distance D, estimated from a (continuous) text corpus, and $P(w_1)$ and $P(w_2)$ are the unigram probabilities of w_1 and w_2, respectively, estimated from the same text corpus. The term $P_D(w_1, w_2)$ can be defined as

$$P_D(w_1, w_2) = \frac{N_D(w_1, w_2)}{N} \tag{2}$$

where $N_D(w_1, w_2)$ is the total number of times w_2 was observed after w_1 at distance D and N is the total number of word tokens in the text corpus.

(a) Pre-process the corpus English_train.txt by applying white-space tokenization and removing punctuation. You should ignore sentence boundaries. You can use the `tokenize` function provided in the script.

(b) Implement a function `correlation` that takes two words, w_1 and w_2, and distance D as inputs and returns as an output the correlation value.

(c) Use the correlation function and compute it for the word pairs ("he","his"), ("he","her"), ("she","her"), ("she","his") with different distances of $D \in \{1, 2, .., 50\}$. Plot the correlation vs. distance with the values obtained. Explain your findings.

Your solution should contain the plot for (c), a sufficient explanation of the findings in this exercise, as well as the source code to reproduce the results.

Language Models

1) **Linear Interpolation (2 points)**
In this exercise, you will explore simple linear interpolation in language models as an extension of the Lidstone smoothing you already implemented in assignment 3.

In the case of a higher-order ngram language model, there is always the problem of data sparseness. One way to solve this problem is by mixing the higher-order ngram estimates with lower-order probability estimates. This interaction is called interpolation. A simple way to do this is as follows:

$$P_I(w_2|w_1) = \lambda_1 P(w_2) + \lambda_2 P(w_2|w_1) \tag{3}$$

and

$$P_I(w_3|w_1, w_2) = \lambda_1 P(w_3) + \lambda_2 P(w_3|w_2) + \lambda_3 P(w_3|w_1, w_2) \tag{4}$$

where $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$

(a) First, implement a function to estimate interpolated bigram probabilities. Your implementation of this exercise can be an extension of your code in assignment 3 according to formula (3). Use the Lidstone smoothing technique that you implemented in Assignment 3 to obtain $P(w_2)$ and $P(w_2|w_1)$ where

$$P(w) = \frac{N(w) + \alpha}{N + \alpha V} \tag{5}$$

and

$$P(w|w_{-1}) = \frac{N(w_{-1}, w) + \alpha}{N(w_{-1}) + \alpha V} \tag{6}$$

(b) Create a bigram language model using interpolation with $\lambda_{1,2} = 1/2, 1/2$ and $\alpha = 0.3$ on the English_train.txt corpus.

(c) Compute the perplexity of the model on the English_test.txt corpus and compare it with the perplexity of the original model with Lidstone smoothing ($\alpha = 0.3$) and without interpolation. How do you explain the differences?

Your solution should contain the perplexity values for part (c), a sufficient explanation of the findings in this exercise, as well as the source code to reproduce the results.

2) Absolute Discounting Smoothing (4 points)

In this exercise, you will implement absolute discounting smoothing for a bigram language model with a back-off strategy. In contrast to interpolation, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram. When the higher-order n-gram model is a bigram LM, absolute discounting is defined as

$$P_{abs}(w_i|w_{i-1}) = \frac{max\{N(w_{i-1}, w_i) - d, 0\}}{\sum_{w' \in V} N(w_{i-1}w')} + \lambda(w_{i-1}) P_{abs}(w_i) \tag{7}$$

$$P_{abs}(w_i) = \frac{max\{N(w_i) - d, 0\}}{\sum_{w' \in V} N(w')} + \lambda(.) P_{unif}(w_i) \tag{8}$$

where $0 <$ d < 1 is the discounting parameter, P_{unif} is a uniform distribution over the vocabulary V, $\lambda(w_{i-1})$ and $\lambda(.)$ are the backing-off factors defined as

$$\lambda(w_{i-1}) = \frac{d}{\sum_{w' \in V} N(w_{i-1}w')} N_{1+}(w_{i-1}\bullet) \tag{9}$$

where

$$N_{1+}(w_{i-1}\bullet) = |\{w : N(w_{i-1}w) > 0\}| \tag{10}$$

and

$$\lambda(.) = \frac{d}{\sum_{w' \in V} N(w')} N_{1+} \tag{11}$$

where

$$N_{1+} = |\{w : N(w) > 0\}| \tag{12}$$

(a) Given the equations above, implement absolute discounting smoothing and build a bigram LM using the English_train.txt corpus. Your implementation of this exercise can be an extension of the `ngram_LM` class or your could design your own code from scratch if that is more convenient for you. The main adaptation should be on the `estimate_smoothed_prob` function. For a unigram LM, consider the empty string as the n-gram history. You can use the test function to make sure that your probabilities sum up to 1.

(b) Compute and report the perplexity of the backing-off LM for the test corpus with $d = 0.7$.

(c) Compare the perplexity values of the model with the best-performing model you implemented with Lidstone smoothing. Explain the difference in performance by comparing the two methods.

(d) Investigate the effect of the discounting parameter d on the perplexity of the test corpus. For each $d \in \{0.1, 0.2, ..., 0.9\}$ compute the perplexity and plot a line chart where the x-axis corresponds to the discounting parameter and the y-axis corresponds to the perplexity value.

Your solution should contain the plot for (d), a sufficient explanation of the findings in this exercise, as well as the source code to reproduce the results.

3) **Kneser-Ney Smoothing (2 points)**
In this exercise, you will explore Kneser-Ney smoothing. Consider the following notation

- V - LM Vocabulary
- N(x) - Count of the n-gram x in the training corpus
- $N_{1+}(\bullet w)$ where $|\{u : N(u, w) > 0\}|$ - number of bigrams ending in w
- $N_{1+}(w\bullet)$ where $|\{u : N(w, u) > 0\}|$ - number of bigrams starting with w
- $N_{1+}(\bullet w\bullet)$ where $|\{(u, v) : N(w, w, v) > 0\}|$ - number of trigram types with w in the middle

For a trigram Kneser-Ney LM, the probability of a word w_3 is estimated as

$$P_{KN}(w_3|w_1w_2) = \frac{max\{N(w_1w_2w_3) - d, 0\}}{\sum_{v \in V} N(w_1w_2v)} + \lambda(w_1w_2)P_{KN}(w_3|w_2) \tag{13}$$

$$= \frac{max\{N(w_1w_2w_3) - d, 0\}}{N(w_1w_2)} + \lambda(w_1w_2)P_{KN}(w_3|w_2) \tag{14}$$

As shown above, the definition of the highest order probability in Kneser-Ney LM is identical to that in absolute discounting. However, the main difference is in estimating the lower-order back-off probabilities

$$P_{KN}(w_3|w_2) = \frac{max\{N_{1+}(\bullet w_2w_3) - d, 0\}}{\sum_{v \in V} N_{1+}(\bullet w_2)} + \lambda(w_2)P_{KN}(w_3) \tag{15}$$

$$= \frac{max\{N_{1+}(\bullet w_2 w_3) - d, 0\}}{N_{1+}(\bullet w_2 \bullet)} + \lambda(w_2) P_{KN}(w_3) \tag{16}$$

The base case of this recursive definition is the unigram probability which is estimated as

$$P_{KN}(w_3) = \frac{N_{1+}(\bullet w_3)}{\sum_{v \in V} N_{1+}(\bullet v)} = \frac{N_{1+}(\bullet w_3)}{N_{1+}(\bullet\bullet)} \tag{17}$$

where $N_{1+}(\bullet\bullet)$ is the number of bigram types. Contrary to absolute discounting, a Kneser-Ney LM does not interpolate with the zerogram probability. In case w3 was not in the vocabulary of the LM (i.e., OOV), then $P_{KN}(w_3|w_1 w_2) = P_{KN}(w_3) = \frac{1}{|V|}$

(a) From the English_train.txt corpus, collect the necessary statistics and fill the table below

	w = "longbourn"	w = "pleasure"
N(w)		
$N_{1+}(\bullet w)$		
$-log_2 P_{Lid}(w)$		
$-log_2 P_{KN}(w)$		

Discuss the differences.

(b) Describe the idea behind the Kneser Ney smoothing technique. How does it differ from other backing-off language models?

Your solution should include the table and the source code of part (a), and a sufficient explanation of part (b).

Submission Instructions

The following instructions are mandatory. Please read them carefully. If you do not follow these instructions, the tutors can decide not to correct your exercise solutions.

- You have to submit the solutions of this exercise sheet as a team of 2 students.
- If you submit source code along with your assignment, please use Python unless otherwise agreed upon with your tutor.
- NLTK modules are not allowed, and not necessary, for the assignments unless otherwise specified.
- Make a single `ZIP` archive file of your solution with the following structure
  - A `source_code` directory that contains your well-documented source code and a `README` file with instructions to run the code and reproduce the results.
  - A `PDF` report with your solutions, figures, and discussions on the questions that you would like to include. You may also upload scans or photos of high quality.
  - A `README` file with group member names, matriculation numbers and emails.
- Rename your `ZIP` submission file in the format

`exercise02_id#1_id#2.zip`

where `id#n` is the matriculation number of every member in the team.

- Your exercise solution must be uploaded by only one of your team members under *Assignments* in the *General* channel on Microsoft Teams.
- If you have any problems with the submission, contact your tutor before the deadline.