Statistical Natural Language Processing - Assignment 06

Rayyan Mohammad Minhaj (7074982/rami00002@stud.uni-saarland.de)
Abdullah Abdul Wahid (7075730/abyy00002@stud.uni-saarland.de)

May 23, 2025

Exercise 6.1 - Smoothing and Interpolation

(a)

When we apply additive smoothing to a language model, we basically adjust the observed word frequency by adding a constant (α) to **every word's count**. This helps prevent zero probabilities for unseen words but **it also pulls the probability distribution towards a more uniform one**.
Now if we apply this smoothing, not once, but again and again and again, we would be treating each smoothed probability as normal, and smoothing them again! With each repetition, we're adding more and more equal weight to every word in the vocabulary, regardless of how often it appeared in the original data.
Overtime, the difference between high-frequence words, and low-frequency words would become smaller because theyre being pushed together. Eventually, in the limit, the model forgets the original frequencies and converges to a **uniform distribution**, **where every word is equally likely**.

(b)

Linear Discounting (Interpolation) works by interpolating the higher-order N-grams probability with a lower-order (backed-off) probability.

$$P(w|h) = (1-\epsilon)\frac{N(w,h)}{N(h)} + \epsilon\beta(w|h)$$

Where,

- ϵ is an interpolation weight (determines how much to trust the lower-order model).
- $\beta(w|h_{short})$ is the backed-off (e.g., (N-1)-gram) distribution.

It combines multiple language models of different orders (e.g., unigram, bigram, trigram) by assigning each a fixed weight and averaging their probabilities. This helps address data sparsity by backing off to lower-order models when higher-order data is unreliable or unavailable.

Exercise 6.2 - Kneser-Ney Smoothing

(a)

- $N(w_1w_2w_3)$ - Count of how many times the trigram occurs.
- $N(w_1w_2)$ - Count of how many times the bigram w_1w_2 occurs
- $N_+(\cdot w_2w_3)$ - Counts how many **unique** words **preceed** the bigram w_2w_3 (further info is for our own clarity - feel free to ignore)
  - EXAMPLE - ["I want to"], ["they want to"], ["you want to"] then $N(\cdot"want\ to") = 3$
- $N_+(\cdot w_2\cdot)$ - Count of how many **unique** trigrams have the middle word as w_2

- $N_+(\cdot w_3)$ - Count of **unique** bigram words that end on w_3.
- $N_+(\cdot\cdot)$ - Count of all the unique bigrams in training data.
- $N_+(w_1 w_2 \cdot)$ - Count of all **unique** trigrams that start with $w_1 w_2$.
- $N_+(w_2 \cdot)$ - Count of all **unique** bigrams that start with w_2
- $\lambda(w_1 w_2)$ - Discounting factor proportional to number of words that follow the bigram $w_1 w_2$
- $\lambda(w_2)$ - Is the leftover probability mass from bigram model $P(w_3|w_2)$ that is passed to the unigram model. and it is proportional to the number of distinct words that follow w_2 in the training data. (Apologies for the long explanation for this one).

(b)

Kneser-Ney smoothing will assign high probabilities to bigrams like "Abu Dhabi" or "Game Over" because the second words ("Dhabi" and "Over") rarely appear in many different contexts.
In standard models, rare bigrams might get low probabilities, but Kneser-Ney rewards such fixed phrases when the second word typically follows only a few specific words. Since "Dhabi" almost always follows "Abu" and "Over" often follows "Game," their continuation counts are low, making the bigram more predictive.
PS: This is one of the strengths of Kneser-Ney: **it captures and favors collocations and fixed expressions.**

(c)

Kneser-Ney smoothing is better than Good-Turing because it looks at how often words appear in new contexts, not just how often they appear overall. It does this by using continuation counts to assign more accurate probabilities to lower-order n-grams.
Unlike Good-Turing, which mainly adjusts counts based on frequency, Kneser-Ney effectively models how likely a word is to appear as a novel continuation, resulting in improved estimates for rare and unseen n-grams. This helps it give more accurate chances to rare word combinations.

Exercise 6.3 - Linear Interpolation and cross-validation

Done in provided notebook/python files.

Exercise 6.4 - Kneser-Ney vs the World

Done in provided notebook/python files.