INLP Assignment 1 Report

Penumalla Aditya Pavani
2021101133

Tokenizer.py:

- Processes sentence boundaries and prevents splitting on common abbreviations such as "Mr.", "Ms.", and "Dr." by making use of negative lookbehinds.
- Separates individual words, allowing alphanumeric characters and hyphens, and preserves the boundaries of words in the tokenized output.
- Adds placeholders ("<NUM>", "<MAILID>", "<URL>", "<HASHTAG>", "<MENTION>") to mask certain types of information (such as numbers, email addresses, URLs, hashtags, and mentions) within the tokenized text.
- Separates punctuation marks from words and maintains the punctuation information in the tokenized output.
- The ultimate result is a nested list structure where each inner list corresponds to a sentence. Within these inner lists, tokens (including words and punctuation marks) are present, and identified patterns are replaced with placeholders.

N-grams:

Preprocessing :

- Substitutes newline characters ('\n') with spaces (' '), tokenizes the preprocessed corpus, and includes n-1 start tokens ("<s>") at the sentence beginning, along with 1 end token ("<\s>") at the sentence end.
- Substitutes punctuation marks with spaces (' '). Any tokens occurring only once are replaced with '<UNK>'.

Good Turing Smoothing:

$$\text{Count}^*(w_1 w_2 w_3) = r^* = (r+1) * \frac{S(N_{r+1})}{S(N_r)} (r = \text{Count}(w_1 w_2 w_3))$$

$$S(N_0) = 1$$

$$P(w_3 | w_1 w_2) = \frac{\text{Count}^*(w_1 w_2 w_3)}{\sum_{w_i \in V} \text{Count}^*(w_1 w_2 w_i)}$$

Nr for unknown values is estimated from

$$log(N_r) = a + b\ log(r)$$

a, b are intercept and slope of log(Zr) - log(r) regression line.

Interpolation:

$$P(t_3 | t_1, t_2) = \lambda_1 \hat{P}(t_3) + \lambda_2 \hat{P}(t_3 | t_2) + \lambda_3 \hat{P}(t_3 | t_1, t_2) \quad (6)$$

$\hat{P}$ are maximum likelihood estimates of the probabilities, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$, so P again represent probability distributions.

Unigrams: $\hat{P}(t_3) = \frac{f(t_3)}{N}$

Bigrams: $\hat{P}(t_3 | t_2) = \frac{f(t_2, t_3)}{f(t_2)}$

Trigrams: $\hat{P}(t_3 | t_1, t_2) = \frac{f(t_1, t_2, t_3)}{f(t_1, t_2)}$

To estimate lambda values :

set $\lambda_1 = \lambda_2 = \lambda_3 = 0$
foreach trigram t_1, t_2, t_3 with $f(t_1, t_2, t_3) > 0$
    depending on the maximum of the following three values:
        case $\frac{f(t_1,t_2,t_3)-1}{f(t_1,t_2)-1}$: increment λ_3 by $f(t_1, t_2, t_3)$
        case $\frac{f(t_2,t_3)-1}{f(t_2)-1}$: increment λ_2 by $f(t_1, t_2, t_3)$
        case $\frac{f(t_3)-1}{N-1}$: increment λ_1 by $f(t_1, t_2, t_3)$
    end
end
normalize $\lambda_1, \lambda_2, \lambda_3$

Average Perplexity Scores:

LM TYPE	TRAIN CORPUS	TEST CORPUS
LM-1: pride and prejudice - gt	1707565037434.78	13876046339.851923
LM-2: pride and prejudice - i	67.50689434624802	2094.0110266635556
LM-3: ulysses - gt	22705317028015.008	506249449492.0035
LM-4: ulysses - i	223.89212951153698	8770.014430487638

Generator.py:

- In the N-gram model (without smoothing), an increase in the value of N results in more accurate word generation due to a longer historical context. Consequently, the number of guesses decreases.

- N-gram models face challenges in contexts beyond their training data as they heavily depend on the provided training set, resulting in suboptimal generations.
- N-gram models might encounter difficulties in capturing extensive dependencies between words, particularly when the context extends over a significant distance.
- Utilizing these smoothing techniques improves the flexibility of N-gram models in handling contexts beyond their training data. This is achieved by addressing challenges related to zero probabilities and fostering a more resilient approach to language modeling.