Introduction to NLP
Assignment 1

REPORT

February 5, 2024

Ashna Dua
2021101072
ashna.dua@students.iiit.ac.in

1 Tokenization

The first part of the assignment includes designing a tokenizer using regex. The **Tokenizer** class provides several methods for preprocessing the corpus, replacing certain entities with special tokens, and splitting the text into sentences and words.

The following entitiy replacements have been implemented:

- URL
- MAILID
- DATE
- TIME
- AGE
- PERCENTAGE
- CURRENCY
- NUM
- MENTION
- HASHTAG

2 N-Grams

This part of the assignment includes designing a class that takes a value of n, and the corpus path to generate n-sized ngram model.

To obtain the probability of the model, the following formula is used:

$$\mathrm{P}(ngram|\mathrm{context}) = \frac{\mathrm{count}(\mathrm{context} + \mathrm{ngram})}{\mathrm{count}(\mathrm{context})} \tag{1}$$

3 Good Turing Smoothing

In this part of the assignment, smoothing technique, i.e. **Good Turing Smoothing** is implemented with a **3-Gram** model.

3.1 Explanation

Good Turing Smoothing adjusts the distribution of given frequencies to estimate missing probabilities. This involves multiple steps, which are as follows:

3.1.1 Frequency of Frequency

In this step, a structure of frequency of frequency is defined, which is as follows:

$$\mathrm{N}_r = \mathrm{count}(w_1, w_2, \ldots, w_n \,|\, \mathrm{freq}(w_1, w_2, \ldots, w_n) = r) \tag{2}$$

3.1.2 Calculation of Z

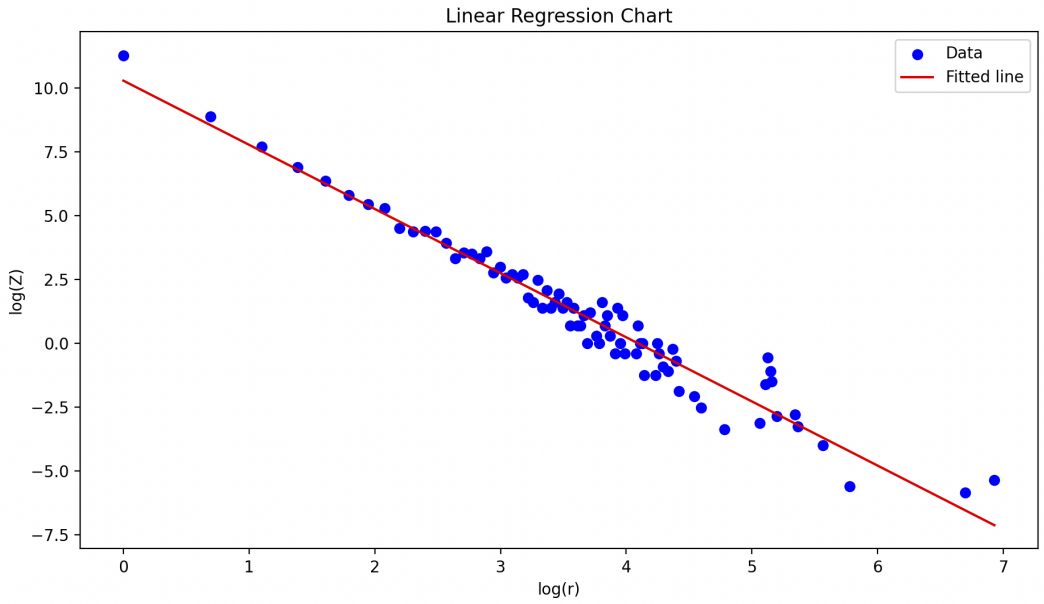
For smoothing the erratic values in N_r for large r, we make a log-log plot of N_r vs r. This causes an issue because for large r, many N_r will be 0. Therefore, we define Z_r.

$$\mathrm{Z}_r = \frac{\mathrm{N}_r}{0.5 \cdot (t - q)} \tag{3}$$

For where q, r, and t are three consecutive subscripts with non-zero counts N_q, N_r, N_t. For the special case when r is 1, take q to be 0. In the opposite special case, when $r = r_{\text{last}}$ is the index of the last non-zero count, replace the divisor $\frac{1}{2}(t - q)$ with $r_{\text{last}} - q$, so $Z_{r_{\text{last}}} = \frac{N_{r_{\text{last}}}}{r_{\text{last}} - q}$.

$$Z_{r_{\text{last}}} = \frac{N_{r_{\text{last}}}}{r_{\text{last}} - q} \tag{4}$$

Linear Regression is performed over the log-log domain of Z_r and r.



3.1.3 Calculation of S

To calculate the probabilities by assuming that expected occurrences for r, i.e. count*(r) is similar to the expected occurences for r+1, i.e. count*(r+1).

$$S_r = \frac{(r+1) \cdot S_{n_{r+1}}}{S_{n_r}} \tag{5}$$

Here, S_{nr} is taken from the regression line, and S_r represents the adjusted count.

3.1.4 Calculation of Probability

To calculate the probability of a token given its context, we use the following formula:

$$\mathrm{P(token|context)} = \frac{\mathrm{S(context + token)}}{\sum_{t \in V} \mathrm{S(context} + t)} \tag{6}$$

4 Linear Interpolation

In this part of the assignment, smoothing technique, i.e. **Linear Interpolation** is implemented with a **3-Gram** model.

4.1 Explanation

To calculate the P(token given context), Linear interpolation uses backoff techniques to give a weighted probability using lower degree models.

$$\mathrm{P(w3|w1w2)} = \lambda_1 \hat{P}(\mathrm{w3}) + \lambda_2 \hat{P}(\mathrm{w3|w2}) + \lambda_3 \hat{P}(\mathrm{w3|w1w2}) \tag{7}$$

Also,

$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{8}$$

To find the values of lamdas, we used the following algorithm:

```
set λ1 = λ2 = λ3 = 0
foreach trigram t1, t2, t3 with f(t1, t2, t3) > 0
    depending on the maximum of the following three values:
        case (f(t1,t2,t3)−1)/(f(t1,t2)−1): increment λ3 by f(t1, t2, t3)
        case (f(t2,t3)−1)/(f(t2)−1): increment λ2 by f(t1, t2, t3)
        case (f(t3)−1)/(N−1): increment λ1 by f(t1, t2, t3)
    end
end
normalize λ1, λ2, λ3
```

5 Generation

In this part of the assignment, generation was done for Simple N-Gram models, and Linear Interpolation only, because the probability assigned to unseen n-grams in Good Turing is very, causing the model to generate random values.

The first run is to obtain non-zero probabilities from the n degree model. If no such non-zero probability is received, the model is backed off to a smaller degree model. This is done only in the case of NGram model, as Linear Interpolation handles OOV tokens on it's own.

6 Results

6.1 Smoothing Results

The following perplexities are recieved for LM1, LM2, LM3 and LM4 on the train and test corpus:

	Train	Test
LM1	389.19	998.40
LM2	57.43	600.26
LM3	1023.28	2826.79
LM4	234.23	4357.01

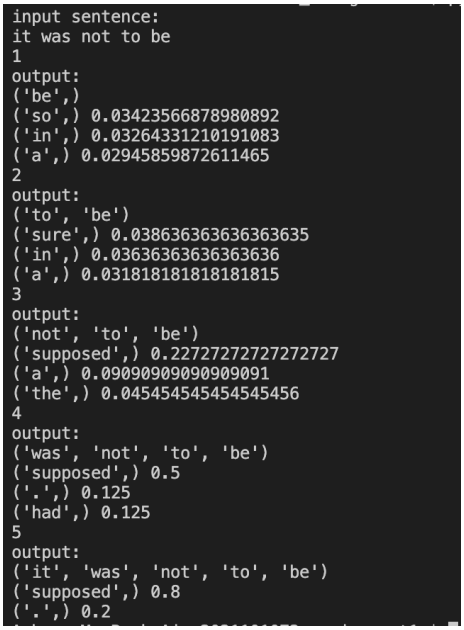
6.2 Generation Results

6.2.1 N-Gram Model

In the n-gram model, the probability of a word depends on the previous n words. The generator starts with the highest order n-gram (the context) and falls back to lower order n-grams if it can't find a suitable word to follow the context.

Generally, higher order-ngrams (larger n) can capture more context and produce more fluent text, but also require more data and are prone to overfitting.

The following image shows the result of 5 n-gram models, i.e. 1, 2, 3, 4, and 5. Unigram and Bigram models produce results that are random in nature and not fluent, where trigram, onwards the results make sense and are more fluent in comparison to the lower n-gram models.



In an Out-of-Data (OOD) scenario, where the context or some words in the context have not been seen in the training data, n-gram models can struggle. This is because n-gram models base their predictions on the frequency of occurrence of n-grams in the training data. If an n-gram has not been seen before, the model will assign it a probability of zero, which can cause issues.

To handle OOD scenarios, n-gram models often use smoothing techniques. These techniques assign a small amount of probability mass to unseen n-grams to ensure that every possible n-gram has a non-zero probability.

For the case of generation, back off technique has been used, where the model falls back to a lower-order n-gram model when it encounters an unseen n-gram.

Also, if no such ngrams are found the EOS token is given a very low probability to mark the end of generation.

6.2.2 Linear Interpolation Model

In the Linear Interpolation model, the probability of a word is calculated as a weighted sum of the probabilities from the unigram, bigram, and trigram models. This approach helps to balance the strengths of different n-gram models.

Linear interpolation can help to mitigate the data sparsity problem that often occurs with higher-order n-gram models. By giving some weight to lower-order n-grams, it ensures that every word in the vocabulary has a non-zero probability, even if it doesnât appear in the context in the training data.

This provides a balance between the context captured by higher-order n-grams and the robustness of lower-order n-grams, leading to more fluent and sensible text generation.

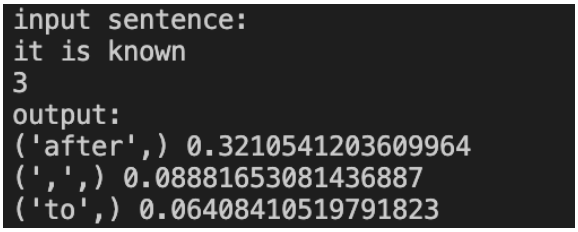


Figure 1: Output of Linear Interpolation - Pride and Prejudice

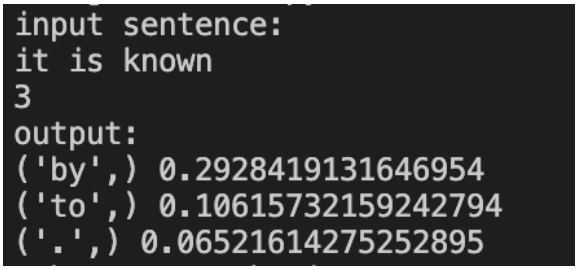


Figure 2: Output of Linear Interpolation - Ulysses

Both the corpus provide a different top prediction with different probabilities. Following are the observations:

- Corpus Influence: The âUlyssesâ model predicts âbyâ as the most likely next word, while the âPride and Prejudiceâ model predicts âafterâ. This reflects the differences in the writing styles of the two books.
- Probability Distributions: The probabilities assigned to the top predictions are also different in the two models. This is due to differences in the frequency of these word sequences in the respective corpora.