Interpolation Model for Language Modeling

1. Introduction

This report details the implementation of an interpolation model for language modeling. The goal is to improve language model performance by combining different n-gram probabilities using optimized interpolation weights. We also analyze execution time and efficiency improvements.

2. Problem Description

Given a dataset, we construct an n-gram language model and employ linear interpolation to combine probabilities from unigram, bigram, and trigram models. The interpolation technique optimizes the weight values (λ) to minimize perplexity on a validation set.

3. Implementation Details

3.1 Dataset Processing

- Tokenized text into unigrams, bigrams, and trigrams.
- Constructed frequency distributions to estimate probabilities:
    - **Unigram Model**
    - **Bigram Model**
    - **Trigram Model**

3.2 Interpolation & Perplexity Calculation

- Used linear interpolation to compute smoothed probabilities.

- Evaluated performance using perplexity.

4. Results

4.1 Perplexity Scores

Model	**Average Perplexity**
Unigram Model	5964.11
Bigram Model	678.35
Trigram Model	1692.21
Interpolation Model	430.85

The interpolation model significantly reduces the perplexity compared to individual n-gram models, demonstrating its effectiveness in improving language modeling performance.

5. Execution Time Analysis

We measured execution time for each major component:

Tasks	**Time Taken (seconds)**
Tokenization and Preprocessing	83.26
N-Gram Frequency Calculation	28.88
Perplexity Calculation	0.16
Interpolation Optimization	9.57
Total Execution Time	121.87

6. Efficiency & Optimization Strategies

- **Text Normalization:** All text was converted to lowercase for consistency.
- **Replacing OOV words with <UNK>**: This helps in generalizing the model and avoiding data sparsity issues.
- **Early Discarding (`if l3 < 0: continue`)**: Avoids unnecessary computations for invalid lambda combinations.

6.1 Different Approaches Used

- **spaCy vs. NLTK:** Initially, spaCy was tested instead of NLTK, resulting in lower perplexity. However, preprocessing with spaCy was slower. Due to this significant time overhead, NLTK was preferred.
- **Stemming vs. Lemmatization:** Both stemming and lemmatization were evaluated. Lemmatization was chosen since stemming did not provide a notable improvement in processing speed.
- **Hyperparameter Tuning:** Instead of using a trial-and-error approach, **GridSearchCV** was employed to fine-tune the hyperparameter efficiently.

7. Conclusion

The interpolation model effectively combines different n-gram probabilities to improve language modeling. The results show that the interpolation model significantly reduces perplexity compared to individual unigram, bigram, and trigram models. Additionally, optimization strategies such as efficient text processing, hyperparameter tuning, and strategic algorithm selection contributed to improved efficiency and execution time.