N-Gram Language Model

Kumar Shresth

Roll Number: 24CS60R51

Contents

1 Introduction

This project implements an optimized n-gram language model using unigram, bigram, and trigram probabilities with interpolation. The goal is to preprocess text data, train n-gram models using MLE with Laplace smoothing, and optimize interpolation weights (λ values) to minimize perplexity. The final model is evaluated on a test set using perplexity as a metric.

2 Execution Time Breakdown

The table below provides the execution time for each step:

Step	**Task**	**Time (seconds)**
1	Loading & Splitting Data	10s
2	Text Cleaning (Punctuation, Non-ASCII)	7s
3	Stopword Removal	10s
4	Lemmatization	150s
5	Tokenization	60s
6	N-Gram Generation	$\sim$5s
7	Filtering N-Grams ($\geq$1% of Articles)	48s
8	MLE Probability Computation	40s
9	Interpolation Optimization	$\sim$30s
10	Test Set Evaluation	$\sim$5s
Total Execution Time	**$\sim$365s ($\sim$6 minutes)**	

Table 1: Time taken for each part of the code

3 Optimization and Design Choices

To improve efficiency and performance, we made the following choices:

3.1 Memory Optimization

- **Using Sets for Filtering:** Quick lookup operations were used for filtering n-grams appearing in $\geq$1% of articles.
- **Efficient Data Structures:** ‘Counter‘ dictionaries were used to reduce redundant operations.
- **In-Place String Operations:** Applied transformations like ‘str.replace()‘ and ‘str.strip()‘ directly to DataFrame columns.

3.2 Time Optimization

- **Vectorized Operations:** Used ‘apply()‘ in Pandas instead of explicit loops for better performance.
- **Optimized N-Gram Generation:** Used tuple slicing with ‘zip()‘ instead of manual loops.
- **Efficient Probability Lookup:** Stored probabilities in dictionaries for O(1) lookup.

3.3 Algorithmic Improvements

- **Laplace Smoothing:** Prevents zero probability for unseen n-grams.
- **Interpolation Model:** Combines unigram, bigram, and trigram probabilities for robustness.
- **Grid Search & Optimization:** Used ‘scipy.optimize.minimize()‘ to find optimal λ values.

4 Conclusion

The final model achieves a test set perplexity of **121.05**, indicating a well-trained language model. The optimizations in memory usage and time complexity ensure an efficient pipeline, making it scalable for larger datasets.