N-Gram Language Model Assignment Report

Summary

This report analyzes the implementation and performance of N-gram language models trained on a corpus of 6.2 million tokens from multiple sources in the **nltk** package. The experiments demonstrate that trigram models (n=3) achieve optimal performance with a perplexity of 36.20, representing a 57.4% improvement over bigram models. The implementation successfully addresses key challenges including out-of-vocabulary words through smoothing techniques and computational efficiency through optimized data structures.

1. Introduction

N-gram models serve as fundamental building blocks in natural language processing, providing probabilistic predictions of word sequences based on local context. This project implements and evaluates N-gram models of varying orders (2-5) on a diverse corpus combining Reuters, Brown, Gutenberg, and web text sources.

2. Methodology

2.1 Data Collection and Preprocessing

The experimental setup utilized a comprehensive corpus drawn from four distinct sources within the NLTK package, ensuring diverse linguistic patterns and domains. The data underwent systematic preprocessing to enhance model quality:

Corpus Statistics:

- Total tokens: 6,196,290
- Training set: 4,957,032 tokens (80%)
- Testing set: 1,239,258 tokens (20%)
- Vocabulary size: 106,731 unique tokens

The preprocessing pipeline removed punctuation marks, numerical values, and other non-linguistic noise to create a cleaner token stream. This standardization process proved crucial for reducing sparsity and improving model generalization.

2.2 Model Implementation

The N-gram model implementation centers on an efficient dictionary-based data structure for storing context-word frequency pairs. This design choice offers several advantages:

- Constant-time lookups: O(1) average case complexity for frequency retrieval
- Memory efficiency: Direct mapping from (context, word) tuples to frequencies
- Scalability: Handles the 106,731-word vocabulary without performance degradation

The probability calculation follows the maximum likelihood estimation approach:

```
P(w_i | w_{i-n+1}, ..., w_{i-1}) = count(w_{i-n+1}, ..., w_i) / count(w_{i-n+1}, ..., w_{i-1})
```

2.3 Smoothing Techniques

To address the zero-probability problem for unseen N-grams, three smoothing methods were implemented:

1. Laplace Smoothing: Adds a constant α to all counts, ensuring non-zero probabilities
2. Stupid Backoff: Recursively backs off to lower-order N-grams with a penalty factor
3. Kneser-Ney Smoothing: Employs sophisticated discounting based on continuation probabilities

The adjustable hyperparameter α allows fine-tuning the smoothing strength based on corpus characteristics.

3. Results and Analysis

3.1 Perplexity Evaluation

Perplexity measurements across different N-gram orders reveal compelling insights:

N-gram Order	Perplexity	Relative Change from previous
2 (Bigram)	85.09	Baseline
3 (Trigram)	**36.20**	**-57.4%**
4 (4-gram)	36.54	+0.9%
5 (5-gram)	40.55	+11.0%

The trigram model emerges as the optimal configuration, achieving the lowest perplexity of 36.20. This represents a dramatic 57.4% improvement over bigram models, indicating that two-word contexts capture significant linguistic dependencies.

3.2 Performance Analysis

The results demonstrate a clear pattern:

- **Sharp improvement** from bigram to trigram models suggests that two-word contexts provide substantial predictive power
- **Diminishing returns** beyond trigrams indicate that longer contexts introduce sparsity without proportional gains
- **Performance degradation** at n=5 likely stems from overfitting to training data patterns

3.3 Text Generation Quality

The sentence generation module, which iteratively predicts words based on context, produced coherent outputs for trigram models. The generation process maintains linguistic fluency by:

- Starting with high-probability seed contexts
- Applying smoothing to handle novel word combinations
- Terminating at specified lengths or natural boundaries

4. Challenges and Solutions

4.1 Computational Complexity

Challenge: Training time increases exponentially as n decreases, particularly problematic for unigram and bigram models processing 6M+ tokens.

Solution: Implemented optimization strategies including:

- Batch processing of N-gram extraction
- Efficient string manipulation using optimized Python libraries
- Pre-computation of frequently accessed statistics

4.2 Memory Management

Challenge: Storing frequency counts for all possible N-grams with a 106,731-word vocabulary threatens memory exhaustion.

Solution:

- Sparse representation storing only observed N-grams
- Dynamic pruning of low-frequency entries
- Efficient dictionary implementation with hash-based lookups

4.3 Data Sparsity

Challenge: Higher-order models suffer from severe sparsity, with many valid word sequences appearing zero times in training data.

Solution: The three-pronged smoothing approach ensures robust probability estimates even for unseen sequences, with Kneser-Ney smoothing providing particularly effective handling of rare events.

5. Conclusions

This implementation successfully demonstrates the practical application of N-gram language models on a substantial corpus. Key findings include:

1. Optimal Model Order: Trigram models provide the best balance between contextual information and data sparsity
2. Smoothing Importance: Proper smoothing techniques are essential for handling the long tail of language
3. Efficiency Considerations: Careful implementation choices enable processing of multi-million token corpora
4. Dataset Scale: Larger datasets can significantly improve prediction accuracy by providing more comprehensive coverage of language patterns and reducing the impact of data sparsity

The success of this implementation validates N-gram models as robust baselines for language modeling tasks, while highlighting areas where modern neural approaches might provide advantages. The achieved perplexity of 36.20 for trigram models represents strong performance given the model's simplicity and interpretability.