N-Gram Language Modeling and Evaluation Report

Project Code Github Link: https://github.com/AAV13/ngram-language-model

This report details the implementation and evaluation of various N-gram language models on the Penn Treebank dataset. The objective was to understand the impact of N-gram order, the necessity of smoothing, and the comparative performance of different smoothing and backoff techniques. All evaluations were performed using the **Perplexity (PP)** metric on the test set, where a lower score indicates a better model.

1. Pre-processing and Vocabulary Decisions

Before training, the data was systematically pre-processed to prepare it for the N-gram models.

- **Tokenization**: Each sentence (line) in the dataset was converted to lowercase. For an N-gram model of order N, N-1 start-of-sentence (<s>) tokens were prepended, and one end-of-sentence (</s>) token was appended to each sentence. This is crucial for the model to learn how to start and end sequences.
- **Vocabulary**: A vocabulary was constructed from the training data. Exploratory Data Analysis (EDA) revealed that the source dataset was already heavily pre-processed. The vocabulary size was a fixed 9,999 unique words, and the special token <unk> (for unknown words) was the second most frequent token in the entire corpus. This indicates that all words outside the top 9,999 had already been mapped to <unk>. Therefore, our pre-processing pipeline simply ensured that any words in the validation or test sets not present in the training vocabulary were also mapped to <unk>.

2. Impact of N-gram Order

To understand the trade-offs between model complexity and data sparsity, Maximum Likelihood Estimation (MLE) models of orders N=1 through N=4 were trained and evaluated.

Model	Perplexity on Test Set
Unigram (N=1)	637.70
Bigram (N=2)	INF
Trigram (N=3)	INF
4-gram (N=4)	INF

Discussion

The results clearly illustrate the fundamental challenge of N-gram modeling.

- The **Unigram** model, which assumes every word is independent, produced a finite but very high perplexity. It's a poor model but robust because it can assign a non-zero probability to any word in its vocabulary.
- For **N > 1**, the perplexity was **infinite**. This is a direct consequence of **data sparsity**. As N increases, the number of possible N-grams grows exponentially. The training data is simply too sparse to contain every possible bigram, trigram, or 4-gram that might appear in the test set. When the model encounters an unseen N-gram in the test data, its MLE probability is zero, causing the perplexity calculation (which involves a product of probabilities) to become infinite. Our EDA's Zipf's Law plot visually confirmed this "long tail" of rare word combinations, making zero-probability events inevitable.

3. Comparison of Smoothing/Backoff Strategies

To To address the zero-probability issue, several smoothing and backoff techniques were implemented on the Trigram (N=3) model. For the Linear Interpolation model, we performed manual testing of several λ combinations on the validation set to find the optimal weights. The combination that minimized perplexity on the validation data was ($\lambda 1=0.2$, $\lambda 2=0.5$, $\lambda 3=0.3$), which was then used for the final evaluation on the test set.

Trigram Model Strategy	**Perplexity on Test Set**
Maximum Likelihood Estimation (MLE)	INF
Add-1 (Laplace) Smoothing	3295.02
Linear Interpolation	196.49
Stupid Backoff	**188.11**

Discussion

- **Why Smoothing Corrects the Problem**: The MLE model failed because any unseen N-gram had a zero probability. Smoothing techniques solve this by "stealing" a small amount of probability mass from the N-grams that *were* seen and redistributing it to all the N-grams that were *not* seen. This ensures that no event has a probability of exactly zero.
- **Performance Comparison**:
    - **Add-1 Smoothing** performed extremely poorly, yielding a perplexity far worse than even the Unigram MLE model. This is because it is a naive, blunt approach. It gives away far too much probability mass to the massive number of unseen trigrams.
    - **Linear Interpolation** and **Stupid Backoff** performed significantly better. Their strength lies in their ability to intelligently fall back on more reliable, less sparse lower-order models (bigrams and unigrams) when data for a specific trigram is unavailable.

- The **Stupid Backoff** model emerged as the best performer with a perplexity of **188.11**. This demonstrates that even a simple, non-probabilistic backoff heuristic can be highly effective, in this case slightly outperforming the more complex, tuned interpolation model.

4. Qualitative Analysis (Generated Text)

The best-performing model (Trigram with Stupid Backoff) was used to generate five sample sentences to qualitatively assess its fluency.

Generated Sentences:

1. in its merchandise up on jan. to generate some way to drive which is in danger and high costs
2. in the millions mostly paid for until the takeover would find sufficient evidence of neutrons from $ n gain down from major structures that met
3. so great decisions of government employees
4. for more than overall net income could be a ads bush in september fell n to n n n
5. <s> <unk> doctors ' offices <unk> <s> <s> <s> <s> what 's premier said he <s> <s> <s> the charge however net in the treatment

Discussion

The generated text highlights both the strengths and weaknesses of the N-gram model.

- **Fluency**: The model demonstrates good **local fluency**. Short phrases like "sufficient evidence of" (Sentence 2) and the entirety of Sentence 3 ("so great decisions of government employees") are grammatically correct and coherent. The model is clearly effective at learning word-to-word transitions.
- **Limitations**: The model's primary weakness is its lack of **global coherence**. Because of the **Markov Assumption** (it only remembers the last 2 words), it cannot maintain a consistent topic. Sentences 1 and 2 start plausibly but quickly drift into nonsensical combinations. Furthermore, the model exhibits clear failure modes, such as getting stuck in a **repetitive loop** ("n to n n n" in Sentence 4) and even generating its own special tokens like <s> and <unk> (Sentence 5), exposing its internal mechanics.