LaTeX Homework 1 (CSCI 5541 NLP)

Gretchen Corcoran
gcorcora@umn.edu

Abstract

This paper describes the implementation of an n-gram language model implemented with stupid backoff smoothing. In order to evaluate the implementation, we trained four separate language models with training corpora sourced from four well-known authors - Jane Austen, Charles Dickens, Leo Tolstoy, and Oscar Wilde. We used a 90% train, 10% development set split as the main form of evaluation comparing the performance of 2-gram, 3-gram, 4-gram, and 5-gram models across the various authors. Additional functionality allows for the models to run on separate test sets included as a separate file.

1 Introduction

This paper describes an author classification task using n-gram language models. Using the training corpora from four well-known authors - Jane Austen, Charles Dickens, Leo Tolstoy, and Oscar Wilde - a separate model is trained for each author. The program classifies sentences by calculating the perplexity for each author's model and selecting the author with the model that produces the lowest perplexity, or highest probability.

2-gram, 3-gram, 4-gram, and 5-gram models are compared, using stupid backoff smoothing for n-grams that were not seen in the training corpus. We measure performance based on accuracy with a 10% held-out development set. Our analysis aims to identify which n-grams perform best at author identification under conditions where the training corpus is limited in size.

2 Experimental Setup

Our analysis focuses on the performance of a development set split off from a training set, however our program allows a user to prespecify a separate test set, in which case the entire training set is used to train the models. When a separate test set is not specified, the training corpus for each author will be split into a 90% training and 10% development set.

Prior to splitting the data into train and development sets, initial intake and cleaning for the sets include stripping headers and removing empty lines. Only one corpus in our training data required header stripping. The removal of empty lines ensures training and test splits contain actionable data. At this point, unless a test set is specified, the training corpus for each author is randomly split into a 90% train, 10% development set. This is done by randomly shuffling the lines in the training corpus, then assigning the last 10% of the lines to the development set. After the train/development split, we train the separate models for each author.

3 N-gram Language Models

3.1 Tokenization

The first step of creating a model for a given n-gram includes encoding the corpora using tiktoken encoding with o200k_base (OpenAI, 2024). Each line of the text is treated as a separate sentence, and an end-of-sentence indicator is added to each line. Negative numbers are used to indicate the end of the sentence to avoid collisions with the values used by the tiktoken encoder, which uses only non-negative numbers for encoding.

If a model's n-gram length is greater than 1, beginning of sentence indicators are included, represented with a negative number. The number of beginning of sentence indicators depends on the length of the n-gram, such that a model includes n-1 beginning of sentence indicators. For example, a tri-gram model would have [-1] + [-1] + [unigram] to start a sentence.

3.2 N-gram counts

After the text is tokenized, the ngram and context counts are generated. These are stored in a dictionary with associated frequencies as values. Each line of the text is processed with a sliding window, processing the context and ngram within the window. For example, an 3-gram would have a window of length 3. The full 3-gram will be added to the n-gram count, and the first two tokens of the n-gram will be added to the context count.

4 Perplexity and Text Classification

After the n-gram models are trained, the development set or the test set is classified, whichever is present. The first step in this classification tokenizes the development or test set, adding beginning of sentence and end of sentence tokens in the same manner as for the training set.

In order to predict the author for a given line, we calculate the perplexity for the line. Perplexity is defined by (Jurafsky and Martin, 2026) as

$$\text{perplexity}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{p(w_i \mid w_{i-n+1}^{i-1})}}.$$

Lower perplexity is given to sentences that the language model indicates have a higher probability of being written by a given author. The author which a model assigns the lowest perplexity score to is chosen as the predicted author.

When calculating the perplexity for each author, we check to see if a line is shorter than the specified n-gram. This was not an issue in either our development or test sets, so this was not dealt with in this n-gram implementation. Future work may need to address this specific edge case.

4.1 Stupid Backoff Smoothing

The models in our project utilize stupid backoff, as described by (Jurafsky and Martin, 2026). Based on this method, models and counts are generated for all lower-level n-grams. For example, if a trigram model is specified, the models and counts for bigram and unigram models are generated. The backoff factor λ is set at 0.4.

Each sentence is processed with a sliding window, starting at the beginning of sentence indicators (except for the unigram models). For each target token w_i and its context c, the conditional probability is calculated as

$$p(w_i \mid c) = \frac{\text{count}(c, w_i)}{\text{count}(c)}.$$

N-gram	Austen	Dickens	Tolstoy	Wilde	Avg
2-gram	77.34%	67.86%	74.02%	74.23%	73.36%
3-gram	76.51%	69.24%	74.57%	73.46%	73.45%
4-gram	76.51%	68.69%	74.57%	73.84%	73.40%
5-gram	76.24%	68.78%	74.11%	73.65%	73.195%

Table 1: For each n-gram model, per-author accuracy and average accuracy on development set.

If an n-gram has a count of zero, the program backs off to a lower level of n-gram by shortening the context, reducing down to unigrams if needed. If after reducing to unigrams, the token is still unseen, the token is assigned a small non-zero probability to avoid taking the log of zero in the perplexity calculation.

The backoff probability is calculated by using the $\lambda^{(n-k)}$ scaling, where n is the model's main n-gram order and k is the order after back off.

The log probabilities for all tokens in a sentence are combined and averaged, so that sentence length does not impact probability, and exponentiated to get th final perplexity calculation.

5 Experimental Results & Discussion

The results for comparing 2-gram, 3-gram, 4-gram, and 5-gram models for Austen, Dickens, Tolstoy, and Wilde using 10% test sets are shown in Table 1. Since our training sets were rather limited, the results do not reflect large increases in accuracy with higher number n-grams, as expected. Interestingly, increasing the length of the n-gram does not lead to monotonic improvements in performance across all authors. Austen sees the best performance with the 2-gram model, while Dickens and Wilde see the best performance with 3-gram models. For Tolstoy, best performance is achieved using either a 3-gram or 4-gram model. On average across all authors, the best results are seen using the 3-gram model. Further studies are needed to identify the cause for the performance differences across authors - this may be due to limited training corpus and the text chosen for a given author, but differences in writing style, average sentence length, quantity of dialogue, or even artefacts of Russian-to-English translation in the case of Tolstoy, may influence the optimal choice of n-gram for a given author.

6 Conclusion

Further studies with a larger training corpus are likely to produce higher average performance. Additional studies to compare the use of different smoothing approaches, including interpolation, Kneser-Ney, and add-k smoothing may further elucidate the best methods to use for author identification models.

References

Daniel Jurafsky and James H. Martin. 2026. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3 edition.

OpenAI. 2024. tiktoken: Fast bpe tokeniser for use with openai's models. https://github.com/openai/tiktoken.