Performance Analysis of N-gram Language Models

Mileva Van Tuyl
New College of Florida
Sarasota, FL

Marina Sanchez
New College of Florida
Sarasota, FL

ABSTRACT

N-gram language models are a simple approach used to compute the probabilities of word sequences and predict potential upcoming words. In this paper, we use the DUC 2005 dataset to assess the performance of three n-gram language models (where n = 2, 3, and 4) with and without Laplace-k smoothing. Using metrics like average log-likelihood and perplexity, the most effective model appears to be the 4-gram model without Laplace Add-1 Smoothing. However, there are numerous factors at play and further exploration must be conducted to assess the performance of these different n-gram language models.

CCS CONCEPTS

CCS Concepts: • **Computing methodologies** → *Natural Language Processing;*

KEYWORDS

natural language processing, language models

1 Introduction

Language models are important in the field of natural language processing as they provide a method by which we can predict upcoming words in sequences and determine the likelihood of sentences. These models are grounded in notions of probability and rely on the the chain rule of probability to compute the likelihood of a given word sequence $P(w_{1:n})$:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w_n|w_{1:n-1}) = \prod_{k=1}^{n} P(w_k|w_{1:k-1}). \quad (1)$$

In Equation 1, w_i is the i^{th} word in the word sequence and $w_{1:n}$ represents a word sequence of length N. We can then compute the probability of the entire sequence of words by taking the product of the probabilities of each word w_i given the preceding word sequences $w_{1:i-1}$. However, given the diversity and changing nature of language, it is not feasible to compute the probabilities of these word sequences without making additional assumptions.

Thus, we turn to n-gram language models. These language models utilize n-grams, a sequence of length *n*, and the Markov assumption. From this, we can approximate the probability of a word using only a portion of the preceding sequence. In the case of the bigram model, where n=2, we can restructure Equation 1 as follows:

$$P(w_{1:n}) \approx P(w_1)P(w_2|w_1)P(w_3|w_2)...P(w_n|w_{n-1}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}). \quad (2)$$

This notion can also be generalized to develop trigram, 4-gram, and n-gram models more generally.

Once these models are developed, we must assess their performance. For the purpose of this paper, we apply intrinsic evaluation which involves splitting the dataset into a training set and a testing set. We then compute the metrics including the average log-likelihood and perplexity on the withheld testing corpus. Ultimately, these models can be used across a wide variety of applications including speech recognition, spelling and grammar correction, and language translation tasks [1].

2 Methodology

2.1 Dataset

To assess the performance of different n-gram language models, we use data from the DUC 2005 dataset. The DUC 2005 dataset is a collection of documents associated with the Document Understanding Conference (DUC) and is generated by NIST [2]. The dataset contains a total of 1863 documents and 57573 sentences. Of these, 1593 documents and 48421 sentences are in the training set and 270 documents and 9152 sentences are in the testing set. All documents contained in both the training and testing sets require parsing, cleaning, and preprocessing before use in the n-gram language models.

2.2 Preprocessing

While the preprocessing steps for the training data and testing data are similar, the process does differ. Thus, we started by processing the training dataset before undertaking the processing for the testing dataset.

2.2.1 Training Set The preprocessing for the training data consists of three core steps:

1. Parsing and cleaning the sentences,
2. Defining the vocabulary and the unknown words,
3. Obtaining the frequencies of all the unigrams, bigrams, trigrams, and 4-grams within the training dataset.

Each file is in a form similar to that of an HTML page with tags to delineate different portions of the file contents. We handled the initial parsing with the Beautiful Soup Python library. We then applied some of the common cleaning techniques for natural language processing tasks: case folding, removing punctuation, and stripping trailing white spaces. In our case, we chose not to apply lemmatization because it could result in a loss of information about the form of a word or the tense of a verb. This information is pertinent to the task of predicting the next word in a sentence or assessing the probability of a certain word sequence.

To conclude the parsing and cleaning, we applied the tokenizer from the NLTK Python library to split the document into sentences and tokens. However, an initial exploration of the sentences revealed that some of the "sentences" were of length greater than 600 tokens and did not represent actual sentences in the documents, but rather represented tables or figures. To account for these outliers, we excluded sentences of length greater than 55. This threshold is based on the distribution of the sentence lengths in the training set, depicted in Figure 1.

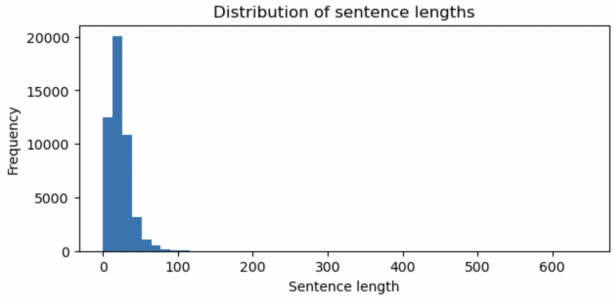

Figure 1: The distribution of sentence lengths in the training set.

Next, we defined the vocabulary to be used for the n-gram models. To do so, we retained all words that appeared in the training set at least five times within our vocabulary. Meanwhile, in the training sentences, we replaced all words that had a frequency below five with the <UNK> tag.

Last, we used the new sentences containing the <UNK> tags to obtain the unigrams, bigrams, trigrams, and 4-grams present in the training dataset. To do so, we had to apply padding in the form of the tokens <s> and </s> to each sentence in the corpus. For bigrams we applied one <s> and </s>, for trigrams we applied two <s> and </s>, and for 4-grams we applied three <s> and </s> to the beginning and ending of each sentence respectively. We then computed the frequencies of each unigram, bigram, trigram, and 4-gram and stored those frequencies within Python dictionaries to facilitate the computation of the probabilities during our analysis.

2.2.2 Testing Set The preprocessing for the testing data differed slightly from that of the training data. First, the testing data underwent the same parsing and cleaning process (case folding, removing punctuation, stripping trailing white spaces, tokenization, sentence handling). However, when replacing the words in the testing set with <UNK>, we applied the same vocabulary as was used for the training set. In a similar vein, we did not recompute additional unigram, bigram, trigram, and 4-gram dictionaries using the test set. This is because the unigram, bigram, trigram, and 4-gram dictionaries obtained from the training set will be used to compute the likelihood of the ngrams and sentences to ultimately assess the performance of the n-gram models.

3 Analysis

After the preprocessing phase and before evaluating the performance of the n-gram models on the testing dataset, we analyzed and compiled some statistics on our training and testing datasets as displayed in Figure 2. From these statistics we observe that 6.16% and 5.62% of all tokens in the training and testing corpora respectively are <UNK>. Figure 2 also provides context for the number of unique unigrams, bigrams, trigrams, and 4-grams present in each corpus.

	Training Corpus	Testing Corpus
# tokens	859703	199642
# unknown tokens	52997	11210
# unique unigrams	10247	10767
# unique bigrams	264407	90140
# unique trigrams	567807	154378
# unique 4-grams	711575	179327

Figure 2: Summary statistics of the training and testing corpora.

In addition to the summary statistics in Figure 2, we explored the distribution of various unigrams, bigrams, trigrams, and four-grams in the training dataset. Figures 3a and 3b display some of the top unigrams and bigrams in the training set.

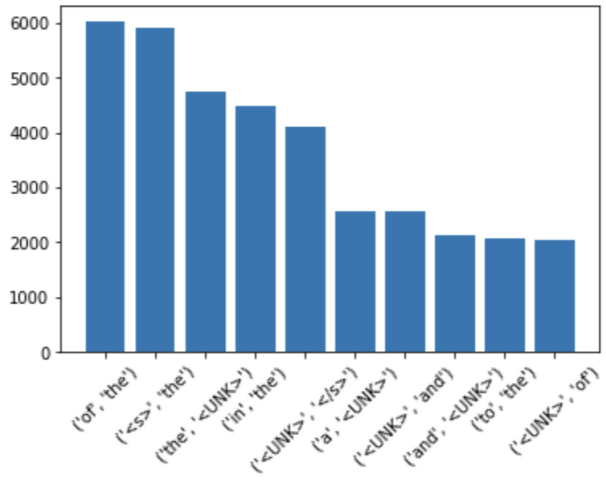

Figure 3a: Some top unigrams in the training set.

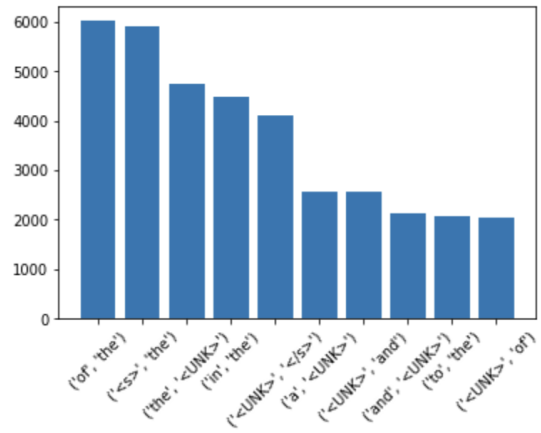

Figure 3b: Some top bigrams in the training set.

The final representation we considered when analyzing the n-grams in the training set is depicted in Figures 4 and 5. In these figures, the i^{th}, j^{th} cell represents the frequencies or probabilities of the bigram (w_i, w_j). This analysis will help provide intuition for the evaluation of the n-gram language models in Section 4.

	the	people	said	of	last	two	financial	years
the	13	76	0	0	212	263	445	27
people	5	0	4	25	0	1	0	0
said	424	2	0	17	24	3	0	0
of	6021	94	1	1	32	49	13	8
last	1	0	1	3	0	16	1	30
two	0	7	1	36	0	0	1	124
financial	0	0	0	0	0	0	0	1
years	33	1	11	93	0	0	0	0

	the	people	said	of	last	two	financial	years
the	0.00026	0.00151	0.00000	0.00000	0.00423	0.00524	0.00887	0.00054
people	0.00437	0.00000	0.00350	0.02185	0.00000	0.00087	0.00000	0.00000
said	0.10090	0.00048	0.00000	0.00405	0.00571	0.00071	0.00000	0.00000
of	0.25123	0.00392	0.00004	0.00004	0.00134	0.00204	0.00054	0.00033
last	0.00083	0.00000	0.00083	0.00248	0.00000	0.01322	0.00083	0.02479
two	0.00000	0.00554	0.00079	0.02850	0.00000	0.00000	0.00079	0.09818
financial	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00166
years	0.02322	0.00070	0.00774	0.06545	0.00000	0.00000	0.00000	0.00000

Figure 4: Bigram counts (left) and probabilities (right) with no smoothing for 8 words in the vocabulary.

	the	people	said	of	last	two	financial	years
the	14	77	1	1	213	264	446	28
people	6	1	5	26	1	2	1	1
said	425	3	1	18	25	4	1	1
of	6022	95	2	2	33	50	14	9
last	2	1	2	4	1	17	2	31
two	1	8	2	37	1	1	2	125
financial	1	1	1	1	1	1	1	2
years	34	2	12	94	1	1	1	1

	the	people	said	of	last	two	financial	years
the	0.00023	0.00127	0.00000	0.00000	0.00353	0.00437	0.00738	0.00046
people	0.00053	0.00000	0.00044	0.00228	0.00000	0.00018	0.00000	0.00000
said	0.02941	0.00021	0.00000	0.00125	0.00173	0.00028	0.00000	0.00000
of	0.17601	0.00278	0.00006	0.00006	0.00096	0.00146	0.00041	0.00026
last	0.00017	0.00000	0.00017	0.00035	0.00000	0.00148	0.00017	0.00271
two	0.00000	0.00070	0.00017	0.00321	0.00000	0.00000	0.00017	0.01086
financial	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00018
years	0.00291	0.00017	0.00103	0.00806	0.00000	0.00000	0.00000	0.00000

Figure 5: Bigram counts (left) and probabilities (right) with Laplace Add-1 smoothing for 8 words in the vocabulary.

Sentence	N-Gram Model Sentence Probability	Sentence Log Likelihood (log(P))
sentence 1 "these people are very vulnerable and often easily"	**Bigram Model** **Pb(<s> these people are very vulnerable and often easily <UNK> </s>) = P(these|<s>) * P(people|these) * P(are|people) *P(very|are)* P(vulnerable|very) * P(and|vulnerable) * P(often|and) * P(easily|often) * P(<UNK>|easily) * P(</s>|<UNK>) =** 0.004630 * 0.029325 * 0.046329 * 0.005097 * 0.004566 * 0.1071428 * 0.00087 * 0.00448 * 0.14063 * 0.07741	-41.857
	Trigram Model **Pt (<s><s> these people are very vulnerable and often easily <UNK> </s></s>)= P(these|<s><s>) * P(people|<s>,these) * P(are|these,people) * P(very|people,are) * P(vulnerable|are,very) * P(and|very,vulnerable) * P(often|vulnerable,and) * P(easily|and,often) * P(<UNK>|often,easily) * P(</s>|easily,<UNK>) * P(</s>|<UNK>,</s>) =** 0.004631 * 0.034091 * 0.25 * 0.037736 * 0.045455 * 0.5 0.33333 * 0.0625 * 1.0 * 0.22222 * 1.0	-22.576
	Fourgram Model **Pf (<s><s><s> these people are very vulnerable and often easily <UNK> </s></s></s>) = P(these|<s>,<s>,<s>) * P(people|<s>,<s>,these) * P(are|<s>,these,people) * P(very|these,people,are) * P(vulnerable|people,are,very) * P(and|are,very,vulnerable) * P(often|very,vulnerable,and) * P(easily|vulnerable,and,often) * P(<UNK>|and,often,easily) * P(</s>|often,easily,<UNK>) * P(</s>|easily,<UNK>,</s>) * P(</s>|<UNK>,</s>,</s>) =** 0.004631 * 0.034091 * 0.5 * 0.2 * 0.5 * 1.0 * 1.0 * 1.0 * 1.0 * 1.0 * 1.0 * 1.0	-11.749
sentence 2 "however the ifc has made no commitment on the next five dams"	**Bigram Model** **Pb(<s> however the ifc has made no commitment on the next five dams </s>) =** 0.00856 * 0.1447 * 0.0002989 * 0.1 * 0.01092 * 0.03030 * 0.00359 * 0.01282 * 0.2971 * 0.004604 * 0.024444 * 0.00314465 * 0.06383	-53.9320
	Trigram Model **Pt(<s><s> however the ifc has made no commitment on the next five dams </s></s>) =** 0.00856 * 0.144674 * 0.0002989 * 0.1 * 0.0109200 * 0.030303 * 0.0035906 * 0.012821 * 0.2971 * 0.004603 * 0.0244444 * 0.0031447 * 0.06383	-30.926
	Fourgram Model **Pf(<s><s><s> however the ifc has made no commitment on the next five dams </s></s></s>) =** 0.008551 * 0.218461 *, 0.0140846 *, 1.0 * 0.5 * 1.0 * 0.33333 * 0.33333 * 1.0 * 1.0 * 0.333333 * 0.0909091 * * 1.0 * 1.0 * 1.0	-16.932
sentence 3 "so you get the argument why not do biology?"	**Bigram Model** **Pb(<s>so you get the argument why not do biology?</s>) =** 0.004131 * 0.00444444 * 0.013062 * 0.109947 * 0.00018 * 0.0645161 * 0.0466321 * 0.00164745 * 0.0015773 * 0.04 * 0.99783	-47.965
	Trigram Model **Pt(<s><s>so you get the argument why not do biology?</s></s>) =** 0.004131 * 0.0063694 * 0.25 * 0.22222 * 0.023809523809523808, 0.111111 * 1.0 * 0.22222 * 0.2 * 1.0 * 1.0 * 1.0	-22.484
	Fourgram Model **Pf(<s><s><s>so you get the argument why not do biology?</s></s></s>) =** 0.00413071 * 0.0063694 * 1.0 * 1.0 * 0.5 * 1.0 * 1.0 * 1.0 * 0.5 * 1.0 * 1.0 * 1.0 * 1.0	-11.931

Figure 6: Likelihood computation for three randomly selected sentences from the training set, using the n-gram language model.

	2-gram		3-gram		4-gram	
	No Smoothing	Laplace-1 Smoothing	No Smoothing	Laplace-1 Smoothing	No Smoothing	Laplace-1 Smoothing
Average log-likelihood	-81.51	-234.22	-32.11	-265.26	-13.26	-280.58
PP	48.08	59551.84	4.67	131353.91	2.02	150665.82

Figure 7: Performance of the n-gram language models (where n = 2, 3, and 4) on the testing dataset.

4 Results

In the previous section, we focused on analyzing the training set to gain an intuition about the various factors that may influence the performance of the n-gram language models. To ultimately evaluate the performance of each model, however, we must compute the evaluation metrics on the withheld testing set.

Figure 6 demonstrates how these computations are obtained on three sentences within the training set using the bigram, trigram, and 4-gram models. Figure 7 then displays the final results that we obtain when applying these computations to the withheld testing corpora. The results in Figure 7 reveal three key findings:

- Finding 1: In the no smoothing case, as *n* increases, the average log-likelihood increases and the perplexity decreases.
- Finding 2: In the Laplace Add-1 smoothing case, as *n* increases, the average log likelihood decreases while the perplexity increases.
- Finding 3: Across all n-gram models, the magnitudes of both the average log-likelihoods and the perplexities are significantly larger when smoothing is applied compared to when smoothing is not applied.

Finding 1 corresponds with findings from prior work [1]. In other words, one would expect that as *n* increases, the average log-likelihood should increase while the perplexity decreases. However, for Finding 2 when smoothing is applied, we see the reverse of Finding 1. Additionally, Finding 3 is surprising as one would expect n-gram models with smoothing to perform better than models without smoothing.

5 Discussion

As noted in the previous section, some of the findings about the performance of the n-gram language models on the DUC 2005 dataset were unexpected. Thus, we conducted further investigation to understand the potential causes for these results.

Finding 1, an increase in the log-likelihood and a decrease in the perplexity as we progress from the bigram to 4-gram model, aligns with findings from other work. This result is expected because as *n* increases, the n-grams will contain more information about the preceding word sequences. Thus, we expect the 4-gram model to perform better than the trigram model. Likewise, we expect the trigram model to perform better than the bigram model.

Finding 2, a decrease in the log-likelihood and an increase in the perplexity as we progress from the bigram model with smoothing to the 4-gram model with smoothing is counterintuitive. This performance decrease indicates that smoothing results in either a loss of information or is more susceptible to overfitting.

Finding 3, that the magnitudes of the average log-likelihood and perplexities with smoothing are drastically larger than those without smoothing, sheds insight on the potential cause of Finding 2.

Laplace Add-k Smoothing fundamentally aims to redistribute probability mass to events that have an assigned probability of 0. In doing so, however, we can observe a significant decrease in probability away from meaningful word sequences. As shown in Figure 4, the bigram (‘people’, ‘the’) has a probability of 0.00437 without smoothing and a probability of only 0.00053 when smoothing is applied. Thus, the Laplace Add-1 smoothing can result in a loss of information. To address this issue, decreasing the value of *k* for Add-k smoothing or considering other forms of smoothing including Kneser-Ney smoothing would help improve the performance.

In our experiment, we observe that n-gram language models without smoothing and with larger values of *n* perform best. However, we must pursue further venues of research and explore additional approaches for preprocessing and smoothing to confirm these initial findings.

REFERENCES

[1] Daniel **Jurafsky** and James H. **Martin**. 2019. Speech and Language Processing: *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall.

[2] DUC 2005 Dataset: https://duc.nist.gov/pubs/2005papers/OVERVIEW05.pdf