Performance Analysis of N-gram Language Models

Thomas FitzGerald
New College of Florida
Sarasota, Florida USA
thomas.fitzgerald23@ncf.edu

Mei Maddox
New College of Florida
Sarasota, Florida USA
amelia.maddox16@ncf.edu

ABSTRACT

We perform an intrinsic evaluation of various n-gram language models with and without smoothing. We accomplish this by training various n-gram models on the same training corpus, and then comparing their performance on a test set.

We find that higher n-gram models outperform lower n-gram models on our test set when no smoothing is applied. Our tests suggest smoothing reduces model performance more on higher n-gram models than on lower n-gram models.

1. Introduction

An increasing amount of communication is done through text formats: text, e-mail, chatrooms, social media platforms, etc.. In addition to this, text is often an in-between step in speech recognition or translation tasks. Computer models which process human language quickly and efficiently are in high demand, which has resulted in the development of a variety of language models (LMs). LMs are a type of probabilistic model which assigns probabilities to an upcoming word and/or an entire sequence of words (e.g. a sentence). Our research focused on a particular type of LM: n-gram.

N-gram models calculate the probability of a word via a history of previous words. For example "word" is more probable to come after "guess the" than "cheese." The number of words comprising the history is equal to N-1: bigram model (N=2) has a one-word history; trigram model (N=3) has a two-word history. A unigram model (N=1) is a simpler version that takes each word as independent of the others, and calculates the probability of a word appearing based on its relative frequency in the model's training corpus.

In general, higher n-gram sequences are assumed to be better at predicting words, but encounter issues in terms of slower processing speeds, and require a large training corpus.

These models can be used for various tasks: for example, an auto-complete function may suggest high-probability words based on the immediately preceding text. Inversely, a low-probability phrase can be flagged for the user to check, either for typos or grammatical errors. These methods can also be a useful final step to translation tasks: machine translation, whether from language to language, or speech to text, no matter how sophisticated, benefits from a way to "double check" if a phrase or sentence makes sense. Lacking an intuitive sense of whether a translated sentence makes sense, computer translators can instead compare the probability of possible translations, and use that information to pick which translation to use.

2. Dataset

Our data comes from the DUC 2005 dataset, a collection of documents generally used for testing summarization tasks. The documents consist of articles from daily newspapers published in the late 1980's and early-mid 1990's: the Financial Times of London, a British newspaper focusing on current events from an economic perspective, along with a smaller number of articles from The Los Angeles Times, mostly on current events in California and environmental issues.

Each document is a text file made up of headers, including publication date, headline, author, etc. and article text denoted by <TEXT>...</TEXT> tags. Some documents had included tables to supplement idea portrayal.

Statistical Analysis of the Corpus		
	Training Set	*Test Set*
Total Files	1319	267
Sentences	37,694	8,976
Unknown Words	37,544 (4.2% of total unigrams)	13,892 (6.7% of total unigrams)
Unigrams	8,098	5,922
Bigrams	252,611	79,076
Trigrams	590,986	153,430
Quadrigrams	753,843	184,921

Figure 2.1: Comparison of training and test sets

3. Method

Most non-article text in the documents were not complete sentences. Therefore, we opted to only consider words encompassed by the "TEXT" tags. Text from tables was also ignored from analysis for the same reason. Documents without said text tags were excluded from the training corpus.

Paragraph tags and newline characters, which existed in some of the article texts, were removed. Further cleaning included case-folding all text to lowercase, removing all punctuation save for conjunctive apostrophes, separating conjoined numbers and letters (e.g. "2pm" to "2 pm"), separating hyphenated words (e.g. "mid-2000s" to "mid 2000s"), lemmatization, and tag conversions. Tag conversions are simply pseudo-word substitution for a particular type of token: <hyp> for hyperlinks and <email> for emails.

Articles were separated into sentences before preprocessing, and tokenized afterwards. The aggregation of each documents' preprocessed sentences composed the training corpus. Note, we treated semicolons as a sentence separator.

The tokenized sentences were then converted into a dictionary of n-gram (1:4) frequency counts. During this process, pseudo word tags (e.g. <s> or </s>) were prepended or appended when applicable. Tokens with a frequency occurrence below 5 were converted into <ukn> tags before the tally.

These combined dictionaries were then used in an intrinsic evaluation of 3 n-gram models: bigram, trigram, and qudrigram. Four separate tests were conducted: two with Laplace add-k smoothing ($k = 0.5, 1$), one with limited basic smoothing, and one without any smoothing.

Smoothing is a form of discounting which transfers a bit of probability mass to unseen events (n-grams). For the model with limited smoothing, a basic smoothing probability of $\frac{1}{V}$ was assigned for zero frequency n-grams.

For the model without smoothing, a stupid backoff strategy was implemented to handle zero frequency n-grams. This strategy uses the likelihood of the lower-order n-gram in cases where there is no evidence of a higher-order n-gram weighted by a constant factor. Termination occurs in the unigram, which has probability $S(w) = \frac{C(w)}{N}$.

For our experiment we used $\lambda = 0.4$ as recommended by Brants et al. (2007). Although interpolated back-off strategies provide probability distributions, implementations are more complex and we desired to establish a simple baseline comparison.

Preprocessing of the test corpus proceeded in a similar manner to the training corpus preprocessing procedure save the unknown token conversion: out of vocabulary words (unigrams) were converted into <ukn> tokens; there was no frequency thresholding.

4. Results

We used two metrics to evaluate a models' performance: maximum likelihood estimation (MLE) and perplexity (PP).

MLE uses normalized (from 0 to 1) counts for the probability calculation. Unfortunately, multiplying probabilities together, as dictated by the chain rule of probability, can quickly lead to issues of underflow. Therefore we used log probabilities when calculating the average sentence likelihood. Equation 4.1 shows the log likelihood calculation of a sequence of words, where N represents the n-gram size (e.g. N=2 for bigram). $P(w_k|w_{k-N+1:k-1})$ is simply the relative frequency of $w_{k-N+1:k}$ out of all n-grams beginning with $w_{k-N+1:k-1}$ (see equation 4.3).

$$P(w_{1:n}) = \sum_{k=1}^{n} ln\, P(w_k|w_{k-N+1:k-1}) \tag{4.1}$$

Perplexity is the inverse probability of the entire test corpus, normalized by the number of total words in said corpus, including pseudo tokens. Equation 4.2 shows the perplexity calculation of a sequence of words.

$$PP(w_{1:n}) = \prod_{k=1}^{n} P(w_k|w_{k-N+1:k-1})^{-\frac{1}{n}} = P(w_{1:n})^{-\frac{1}{n}} \tag{4.2}$$

We will now demonstrate the calculation of these metrics on the following sentences from our training set:

- "few will be above"
- "i do n't think it 's right"
- "i 've be here before"

Bigrams	**Trigrams**	**Quadrigrams**
<s> few will be above </s> ***<s> i do n't think it 's right </s>*** ***<s>i 've be here before </s>***	***<s><s> few will be above </s></s>*** ***<s><s> i do n't think it 's right </s></s>*** ***<s><s> i 've be here before </s></s>***	***<s> <s> <s> few will be above </s></s></s>*** ***<s> <s> <s> i do n't think it 's right </s></s></s>*** ***<s><s> <s> i 've be here before </s></s></s>***
Average Log Likelihood *(P†(W) ⇔ ln($P(W)$)		
P†(few|<s>)+P†(will|few)+ P†(be|will)+P†(above|be)+ P†(</s>|above)	P†(few|<s><s>)+P†(will|<s>few)+ P†(be|few will)+P†(above|will be)+ P†(</s>|be above)+P†(</s>|above </s>)	P†(few|<s><s><s>)+P†(will|<s><s>few)+ P†(be|<s>few will)+P†(above|few will be)+ P†(</s>|will be above)+P†(</s>|be above </s>)+ P†(</s>|<above></s></s>)

P†(i|<s>)+P†(do|i)+P†(n't|do)+ P†(think|n't)+P†(it|think)+P†('s|it)+ P†(right|'s)+P(</s>|right) P†(i|<s>)+P†('ve|i)+P†(be|'ve)+ P†(here|be)+P†(before|here)+ P†(</s>|before)	P†(i|<s><s>)+P†(do|<s>i)+P†(n't|i do)+ P†(think|do n't)+P†(it|n't think)+P†('s|think it)+ P†(right|it 's)+P†(</s>|'s right)+ P†(</s>|right</s>) P†(i|<s><s>)+P†('ve|<s>i)+P†(be|i 've)+ P†(here|'ve be)+P†(before|be here)+ P†(</s>|here before)+P†(</s>|before</s>)	P†(i|<s><s><s>)+P†(do|<s><s>i)+P†(n't|<s>i do)+ P†(think|i do n't)+P†(it|do n't think)+P†('s|n't think it)+ P†(right|think it 's)+P†(</s>|it 's right)+ P†(</s>|'s right</s>)+P†(</s>|right</s></s>) P†(i|<s><s><s>)+P†('ve|<s><s>i)+P†(be|<s>i 've)+ P†(here|i 've be)+P†(before|'ve be here)+ P†(</s>|be here before)+P†(</s>|here before</s>)+ P†(</s>|before</s></s>)
-7.8292 + -5.8805 + -1.3091 + -8.8283 + -3.7136 -4.2625 + -2.3758 + -1.5475 + -3.091 + -2.2172 + -3.0 + -5.7447 + -2.699 -4.2625 + -3.1611 + -1.667 + -7.2189 + -5.5872 + -3.0413	-7.8292 + -2.7081 + 0.0 + -6.6026 + -1.3863 + 0.0 -4.2625 + -2.2494 + -0.2957 + -2.5967 + -1.9459 + -1.2993 + -4.8777 + -3.1355 + 0.0 -4.2625 + -3.1393 + -1.8458 + -2.1972 + -2.9957 + 0.0 + 0.0	-7.8292 + -2.7081 + 0.0 + 0.0 + 0.0 + 0.0 + 0.0 -4.2625 + -2.2494 + -0.1542 + -1.2368 + -1.8245 + -0.5108 + -2.4849 + -1.0986 + 0.0 + 0.0 -4.2625 + -3.1393 + -1.5261 + -2.1972 + -1.0986 + 0.0 + 0.0 + 0.0
-27.5607 -24.9377 -24.938	-18.5262 -20.6627 -14.4405	-10.5373 -13.8217 -12.2237
(-27.5607 + -24.9377 + -24.938) / 3	(-18.5262 + -20.6627 + -14.4405) / 3	(-10.5373 + -13.8217 + -12.2237) / 3
-25.8121	-17.8765	-12.1942
Perplexity *(Tokens from all sentences are combined, thus additional probabilities must be calculated)		

P(few|<s>)*P(will|few)*P(be|will)*
P(above|be)*P(</s>|above)*****
P(<s>|</s>)*
P(i|<s>)*P(do|i)*P(n't|do)*
P(think|n't)*P(it|think)*P('s|it)*
P(right|'s)*P(</s>|right)*****
P(<s>|</s>)*
P(i|<s>)*P('ve|i)*P(be|'ve)*
P(here|be)*P(before|here)*
P(</s>|before)
	P(few|<s><s>)*P(will|<s>few)*
	P(be|few will)*P(above|will be)*
	P(</s>|be above)*P(</s>|above </s>)*****
	P(<s>|</s></s>)*P(<s>|</s><s>)*
	P(i|<s><s>)*P(do|<s>i)*P(n't|i do)*
	P(think|do n't)*P(it|n't think)*P('s|think it)*
	P(right|it 's)*P(</s>|'s right)*P(</s>|right</s>)*****
	P(<s>|</s></s>)*P(<s>|</s><s>)*
	P(i|<s><s>)*P('ve|<s>i)*P(be|i 've)*
	P(here|'ve be)*P(before|be here)*
	P(</s>|here before)*P(</s>|before</s>)
		P(few|<s><s><s>)*P(will|<s><s>few)*
		P(be|<s>few will)*P(above|few will be)*
		P(</s>|will be above)*P(</s>|be above </s>)*
		P(</s>|<above></s></s>)*****
		P(<s>|</s></s></s>)*P(<s>|</s></s><s>)*
		P(<s>|</s><s><s>)*
		P(i|<s><s><s>)*P(do|<s><s>i)*P(n't|<s>i do)*
		P(think|i do n't)*P(it|do n't think)*P('s|n't think it)*
		P(right|think it 's)*P(</s>|it 's right)*
		P(</s>|'s right</s>)*P(</s>|right</s></s>)*****
		P(<s>|</s></s></s>)*P(<s>|</s></s><s>)*
		P(<s>|</s><s><s>)*
		P(i|<s><s><s>)*P('ve|<s><s>i)*P(be|<s>i 've)*
		P(here|i 've be)*P(before|'ve be here)*
		P(</s>|be here before)*P(</s>|here before</s>)*
		P(</s>|before</s></s>)

$(e^{-25.8121} * \mathbf{0.99997} * \mathbf{0.99997})^{\mathbf{-1/22}}$	$(e^{-17.8765} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997})^{\mathbf{-1/22}}$	$(e^{-12.1942} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997} * \mathbf{0.99997})^{\mathbf{-1/22}}$

Figure 4.1: Demonstration of metric calculations

Laplace-1 smoothed probability calculation is similar save for an additional 1 unit in the numerator and additional units in the denominator equal to the vocabulary size (number of unique unigrams) in the training corpus (see equation 4.3).

$$P(w_k|w_{k-N+1:k-1}) = \frac{C(w_{k-N+1:k-1}w_k)}{C(w_{k-N+1:k-1})} \Rightarrow \frac{C(w_{k-N+1:k-1}w_k)+1}{C(w_{k-N+1:k-1})+V} \quad \mathbf{(4.3)}$$

Equation 4.3 : See figure 4.1 for an example of its application

Unigrams			Bigrams		
Token	*Raw Probability*	*Add-one Smoothed*	*Token*	*Raw Probability*	*Add-one Smoothed*
the	0.0585	0.0580	of the	0.00709	0.00533
<ukn>	0.0417	0.0413	<s> the	0.00666	0.00544
be	0.0303	0.0300	in the	0.00532	0.00364
of	0.0281	0.0278	the <ukn>	0.00404	0.00364
to	0.0245	0.0243	<ukn> <ukn>	0.00376	0.00307

Figure 4.2: Probabilities for the top 5 n-grams (out of $V = 8098$)

	Bigrams		Trigrams		Quadrigram	
	Stupid Backoff	*Basic Smoothing*	*Stupid Backoff*	*Basic Smoothing*	*Stupid Backoff*	*Basic Smoothing*
Average log likelihood	-93.08	-121.10	-49.28	-151.42	-25.178	-182.92
PP	54.69	182.42	7.04	401.15	2.52	820.71

Figure 4.3: Metrics for models *not* implementing Laplace-k smoothing

	Bigrams		Trigrams		Quadrigram	
	Laplace-0.5	*Laplace-1*	*Laplace-0.5*	*Laplace-1*	*Laplace-0.5*	*Laplace-1*
Average log likelihood	-146.20	-151.50	-183.78	-187.91	-203.46	-205.63
PP	536.57	673.92	1444.47	1700.95	1743.42	1888.00

Figure 4.4: Metrics for models implementing Laplace-k smoothing

Unsurprisingly, the most frequent tokens in all models are common articles/prepositions. <unk> words and sentence start/end tokens are also common. The drop in relative frequency between the unsmoothed and add-one smoothed models is fairly consistent for each token, because the relative frequencies were similar to begin with, so the reduction in probability space of each was also similar.

When looking at the performance of each model on the test set, we see two consistent trends. First, within each n-gram model, as we apply more aggressive smoothing (none->basic->laplace .05->laplace 1) the perplexity increases, and the average log likelihood decreases. This indicates that the test set contains few novel patterns, so applying smoothing is generally detrimental to the model's performance.

The second trend is that, when no smoothing is applied, higher n-gram models outperform lower n-gram models. However, once smoothing is applied, this trend immediately reverses. so a smoothed trigram model is outperformed by a smoothed bigram model, and so on.

5. Discussion/Conclusion

There are two things that stand out about the results as being somewhat odd: first, the perplexity values of our model appear very low, and second, the unsmoothed models, using the stupid backoff method, have an opposite trend to the smoothed models.

One possible explanation for the low perplexity values is our use of lemmatization, which resulted in a very small vocabulary for the model to learn. The odd trend of higher n, unsmoothed models outperforming lower n, unsmoothed models may be related to the backoff method we used.

It should also be noted that this was a very limited experiment: each model was only run against a single test set, so the relationship between the training and test set was the same in each test, which means the results should not be generalized without further experimentation.

That said, we can draw two conclusions from this experiment. The first is that, when using this backoff method, higher n-gram models will outperform lower n-gram models on the same data when no smoothing is applied. This only accounts for the performance of each model in predicting the test data - higher n models have a number of issues, such as longer training time, and larger data requirements.

Secondly, smoothing can, as it did in this experiment, have a serious negative effect on the performance of a model. This negative effect can vary in degree between different n-gram models, and the size of the effect is not necessarily proportional to the perplexity score of the unsmoothed model. Although we assume the negative effect on performance we observed is due to a low proportion of novel patterns in the test data (partly due to the lemmatization process), since we did not confirm this by examining the test data (beyond noting that the proportion of unknown unigram tokens was higher in the test set, 6.7% vs 4.2% for the training set), or try our model against other data to see if the pattern persists, this finding warrants further examination, and experimentation.

What we can conclude from this study, therefore, is that smoothing should be applied judiciously, because it can have a serious negative impact, even on an otherwise good model.

Project code and instructions on how to replicate/run the experiments can be found at:

https://github.com/maddoxamei/n-gram-model

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. Retrieved from https://web.stanford.edu/~jurafsky/slp3/

[2] DUC 2005 Dataset. https://duc.nist.gov/duc2005/tasks.html

[3] Campagnola, Chiara, 2020, *Perplexity in Language Models*, Retrieved from https://towardsdatascience.com/perplexity-in-language-models-87a196019a94