A Deeper look at N-gram Language Models

Technology Review – CS410 Text Information Systems

Arnab KarSarkar
The Grainger College of Engineering, Department of Computer Science, University of Illinois – Urbana Champaign
201 North Goodwin Avenue, Urbana, Illinois 61801-2302, United States of America
arnabk2@illinois.edu

***Abstract*—Statistical language models assign probabilities to the sequence of words. In this review, we will look at how we can use a model to assign probabilities to a sentence or a sequence of words which is called n-gram model. N-gram model is now widely used in communication theory, speech recognition, spelling correction, grammatical error correction, etc. Hence, it's very important to understand how different n-gram models work and which one is better. We all know predicting is not easy especially when we want to know what happens next. Hence it is difficult to predict the next word of a sentence. An n-gram model can help making it easier for us to predict the next word, given a sequence of a words. We will assign probabilities to the most likely words based on the model.**

Keywords—Statistical language model, n-gram, bi-gram, uni-gram, tri-gram, likelihood, probabilities.

I. Introduction

What is a N-gram? – A N-gram is the sequence of N words, where N = 1, 2, 3… For example, a 2-gram(bi-gram) model is a two-word sequence of words, a 3-gram(tri-gram) model three-word sequence, and so on.. "I like" is an example of a bi-gram model whereas "I like to" is an example of a tri-gram model. So with an n-gram model we try to calculate the probability of the next item in a sequence of words in the form of a (n-1) order. Scalability and Simplicity are the two main benefits of using n-gram model. A n-gram model can take following inputs – 1) words in a sentence 2) Paragraph or a document for Text retrieval.
It then outputs a list of probabilities range from 0 to 1 where the sum of the probabilities become 1.
As we saw above, the model predicts the next word based on previous (n-1) words, we will look at this equation in the next section. But why do we need to set a probability to a sentence?

1. Machine Translation
    a. P (**high** temperature today) < P (**large** temperature today)
2. Spelling correction
    a. It takes 3 hours 25 **minuets** to reach New York.
        i. P (It takes 3 hours 25 **minuets**) < P(It takes 3 hours 25 **minutes)**
3. Speech recognition, Question – answering, etc.

II. N-Grams

A. *P(w|h)*

Let's begin with the task of calculating P(w|h), where w is a given word and h is some history ($w_1 \dots , w_{n-1}$). Suppose given a sequence of words, "*here the weather is so pleasant that*", we want to calculate the probability of the next word "*the*":

$$P(\text{the} \mid \textit{here the weather is so pleasant that})$$

One way to estimate this is to take the relative frequency counts. Take a very large corpus, count the number of times we see the sequence and that's it. We would answer this question – "how many times we saw the word w, given the history h", as follows

$$P(\text{the} \mid \textit{here the weather is so pleasant that}) = \frac{C(\text{here the weather is so pleasant that the})}{C(\text{here the weather is so pleasant that})}$$

With a large corpus such as the internet, we can calculate the probability from the above equation. But it tuns out that in many cases even the internet is not big enough to give us good estimates. That's because language evolves, and people create new sentences all the time. Even simple extensions of a sentence may have zero count on the internet.

Also, if we want to compute the joint probability of an entire sequence of the words "*here the weather is so pleasant that",* we could do it by taking all possible probabilities of these word sequences. We must take the count of "*here the weather is so pleasant that*" and then divide this by the sum of all possible 7-word sequences. That is a lot of calculations we need to do. This is not desirable.

For this reason, we need to look at the chain rule.

B. *Chain rule*

We know that the definition of conditional probabilities are as follows –

$$P(B \mid A) = P(A, B) / P(A),$$

Rewriting this $P(A, B) = P(A)\, P(B \mid A)$

If we add more variable,

$$P(A, B, C, D) = P(A)P(B|A)P(C|A, B)P(D|A,B,C)$$

Now, let's look at the n-gram probability which was P (w|h)

And we saw earlier that we can rewrite h as $w_1, w_2, \ldots, w_n$

So, the joint probabilities of all the words can be written as

$P(w_1, w_2, \ldots, w_n)$

Hence, the chain rule of probability –

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1)P(w_2|w_1)P(x3 \mid w_1, w_2)\ldots P(w_n \mid w_1, \ldots, w_{n-1})$$

$$= \prod_{k=1}^{n} P(X_k|X_{1:k-1})$$

This equation shows that there is a link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

So, we can estimate the joint probability of a sentence by just multiplying the conditional probabilities.
P("here the weather is so pleasant") =

P(here) × P(the | here) × P(weather| here the)
× P(is|here the weather)
× P(so |here the weather is)
× P(pleasant | here the weather is so)

But the problem is that it doesn't really help us! We don't have any way to compute the exact probability of a word given a long sequence of preceding words, hence calculating

Probability (the| here the weather is so pleasant that) isn't possible practically.

There are too many possible sentences, and we will never see enough data to calculate the probability of this. We will now look at the Markov equation.

III. Markov Assumption

In probability theory, Markov model is a stochastic model used to model pseudo-randomly changing system. It assumes that future states depend on the current state, not on the events happened before that. We can predict the probability of some future unit without looking too far into the past.

In n-gram model instead of computing the probability of a word given its entire history, we can assume the history by just the last few words.

Thus, we take the bi-gram model, tri-gram model, etc. into consideration.

A. Bigram Model

The bigram model approximates the probability of a word by only using the conditional probability of the preceding word $P(w_n|w_{n-1})$. So, in our case, instead of computing the probability like this –

P (the| *here the weather is so pleasant that)*

we will just calculate the probability –

P (the| *that)*

B. Trigram Model

The trigram model approximates the probability of a word by only using the conditional probability of two preceding words $P(w_n|w_{n-1}, w_{n-2})$. So, in our case, instead of computing the probability like this –

P (the| *here the weather is so pleasant that)*

we will just calculate the probability –

P (the| *pleasant that)*

C. Maximum Likelihood Estimation(MLE)

Thus, the general equation of the n-gram probability calculation-

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

Given the bigram assumption as we saw above this equation becomes like this –

$$P(w_{1:n}) \approx \prod^{n}_{k=1} P(w_k|w_{k-1})$$

So how do we estimate these bigram probabilities? One way is to estimate this by using maximum likelihood estimator or MLE.

We calculate the MLE by getting counts from a corpus and normalizing the counts so that lie between 0 and 1.

For example, to compute a bigram probability of a word w_2 given a previous word w_1, we can calculate the count of the bigram $C(w_1w_2)$ and normalize it by the sum of all the bigrams that share the same first-word w_1.

$$P(w_n|w_{n-1}) = C(w_{n-1}w_n) / \text{sum}(C(w_{n-1}w))$$

We can simplify this equation by converting $\text{Sum}(C(w_{n-1}w))$ into $C(w_{n-1})$.

D. *Examples*

Let's look at some examples using small corpuses and training data. We will use <s> to identify the beginning of the sentence and </s> to identify the end of the sentence.

<s> I am Arnab </s>
<s> I am a MS Grad Student</s>
<s> MS Grad student am I</s>

Here are the probabilities for some of the bigrams

P(I | <s>) = 2 / 3 = 0.67. P(This | <s>) = 1 / 3 = 0.33

P(am | I) = 2 / 3 = 0.67. P(I | am) = 1 / 3 = 0.33

P(</s> | I) = 1 / 3 = 0.33

E. *Bigrams vs trigrams*

Although we have only discussed about the bigram model above, in practice its more common to use trigram or 4-gram models. In case of trigrams, we get more accuracy because we consider previous two words instead of just one. However, if we don't have big enough training text. Hence the predicted probability for a trigram model becomes *zero*.

IV. Problems

Although, we have seen the benefits of this model. There are certain issues which face often.

Google Inc. (the company) released an article and a data set in the year of 2006, here is an excerpt from their blogpost

We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together.

Some of the examples from their huge training data set, here they used 4-gram model –

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72

As we can see, we only got a handful of number from the huge training corpus. This is because the data is sparse.

Let's look at the below example –

Let's say, we have below sentences from a training set

- ….. denied the claim
- ….. denied the request
- ….. denied the offer

Now, we want to test the sentence from our test set

- … denied the report.

In this case the probability would be

P("report" | "denied the") = 0

A. *Smoothing*

One way to solve the above problem with zero probability is to add smoothing. We will add probability mass to overcome the problem.

There is one method we can use is by just adding +1 to the count of the words. This is also called La Place smoothing. In this methodology, we pretend that we saw the word one more time. So instead of the saying the count is 0 for an unseen word, we will assume the count is 1.

$$P_{\text{Add-1}}(w_i \mid w_{i-1}) = c(w_{i-1}, w_i) + 1 \;/\; (c(w_{i-1}) + V)$$

V. Conclusions

The n-gram model is one of the most widely used tools in language processing. Language model offers a way to assign probabilities to a sentence. And it helps us predicts a word from preceding words. We also saw how Markov assumptions helps us estimate words from a fixed window of previous words. This can be achieved by using Maximum likelihood estimates. We also saw how smoothing provides a sophisticated way to estimate the probability of a n-gram model.

VI. Acknoledgements

I would like to take this opportunity to really appreciate and thank Prof ChengXiang Zhai, TAs of CS 410: Text Information Systems for providing me an opportunity to do this technology review. Also grateful to my classmates for the collaboration and discussions throughout the course.

References

[1] CS410 – Text Information Systems, Coursera course by UIUC, https://www.coursera.org/learn/cs-410/home/welcome

[2] Prof. Dr. ChengXiang ("Cheng") Zhai, http://czhai.cs.illinois.edu/

[3] https://web.stanford.edu/~jurafsky/slp3/3.pdf. (Daniel Jurafsky & James H. Martin)

[4] https://en.wikipedia.org/wiki/Markov_model

[5] http://text-analytics101.rxnlp.com/2014/11/what-are-n-grams.html

[6] https://jon.dehdari.org/tutorials/lm_overview.pdf

[7] https://people.cs.georgetown.edu/nschneid/cosc572/s19/a1/

[8] http://www.cs.umd.edu/class/fall2018/cmsc470/slides/slides_10.pdf

[9] https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html

[10] https://aclanthology.org/P11-1027.pdf

[11] https://towardsdatascience.com/introduction-to-language-models-n-gram-e323081503d9

[12] https://en.wikipedia.org/wiki/N-gram