Lyrics Generation Using N-Gram Technology

Shivaji Arjun Kakad [1], Pro. A. R. Babhulgaonkar [2], Pro. Y. N. Patil [3]
Department of Computer Engineering[1&3], Department of Information Technology[2],
Dr. BAT University, Lonere, MH, India

Abstract- In this paper we are going to use N-Gram model from the Natural Language Processing, for the prediction of next word. Here we have to give single word as input and producing Lyrics for same. By using this technology we are generating Lyrics of song. To do this we are going to use many algorithms like N-gram, smoothing, discounting, etc. We are using corpus of words and predicting a word from previous one word, two words, three words, etc. We have described other algorithms for assigning words to classes based on the frequency of their co-occurrence with other words.

Keywords- Corpus, Discounting, Lyrics, N-gram model, Natural Language Processing, Smoothing.

1 INTRODUCTION

Now a day, there is very instant increase in use of Natural Language Processing. Hopefully most of you concluded that a very likely word is call, or international or phone, but probably not the. We formalize this idea of WORD PREDICTION with probabilistic models called N-grams, which predict the next word from the previous N −1 Language Models words, similarly (N-1)-grams predict the next word from previous N-2 Language Models, and so on. Such statistical models of word sequences are also called language models or LMs. Computing the probability of the next word will turn out to be closely related to computing the probability of a sequence of words.

In a number of natural language processing tasks, we face the problem of recovering a string of English words after it has been garbled by passage through a noisy channel. To tackle this problem successfully, we must be able to estimate the probability with which any particular string of English words will be presented as input to the noisy channel. In this paper, we discuss a method for making such estimates. We also discuss the related topic of assigning words to classes according to statistical behavior in a large body of text. The concept of a language model and give a definition of n-gram models. In next Section, we look at the subset of n-gram models in which the words are divided into classes. We show that for n = 2 the maximum likelihood assignment of words to classes is equivalent to the assignment for which the average mutual information of adjacent classes is greatest. Finding an optimal assignment of words to classes is computationally hard, but we describe two algorithms for finding a suboptimal assignment. We apply mutual information to two other forms of word clustering. First, we use it to find pairs of words that function together as a single lexical entity. Then, by examining the probability that two words will appear within a reasonable distance of one another, we use it to find classes that have some loose semantic coherence. In describing our work, we draw freely on terminology and notation from the mathematical theory of communication. The reader who is unfamiliar with this field or who has allowed his or her facility with some of its concepts to fall into disrepair may profit from a brief perusal of Feller (1950) and Gallagher (1968). In the first of these, the reader should focus on conditional probabilities and on Markov chains; in the second, on entropy and mutual information.

Whether estimating probabilities of next words or of whole sequences, the N-gram model is one of the most important tools in speech and language processing. N-grams are essential in any task in which we have to identify words in noisy, ambiguous input. In speech recognition, for ample, the input speech sounds are very confusable and many words sound extremely similar.

By using n-gram algorithm we can predict the next word from previous words. The accuracy of output is directly proportional to the number of previous words used. That is accuracy of output increases when we increase the number of previous words. Here we have used previous one word i.e. Bi-gram model to get the output.

Probabilities are based on counting things. Before we do calculations for probabilities, we need to decide what we are going to count. Counting of things in natural language is based on a corpus (plural corpora), an on-line collection of text CORPUS or speech. Let's look CORPORA at two popular corpora, Brown and Switchboard. The Brown Corpus is a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.), assembled at Brown University.

2 RELATED WORK

In this section we are going to explain the related work of the Natural Language Processing model i.e. N-gram model that we have done.

2.1 OVERVIEW

Here we have introduced cleverer ways of estimating the probability of a word w given a history h, or the probability of an entire word sequence W. Let's start with a little formalizing of notation.

In order to represent the probability of a particular random variable Xi taking on the value "the", or P(Xi = "the"), we will use the simplification P(the). We'll represent a sequence of N words either as w1 . . .wn or wn1. For the joint probability of each word in a sequence having a particular value P(X = w1,Y = w2,Z = w3, ...,) we'll use P(w1,w2, ...,wn). Now we compute probabilities of entire sequences like P(w1,w2, ...,wn).

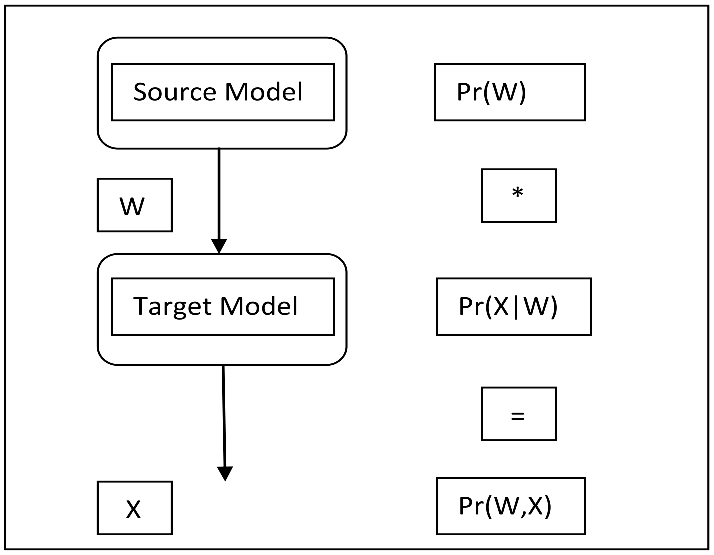


Figure 1 Source-Target Model

One thing we can do is to decompose this probability using the chain rule of probability:

$$P(X_1 \ldots X_n) = P(X_1)\, P(X_2|X_1)\, P(X_3|X^2_1) \ldots P(X_n|X^{n-1}_1)$$

$$= \Pi^n_{k=1} P(X_k|X^{k-1}_1)$$

Applying the chain rule to words, we get:

$$P(w^n_1) = P(w_1)\, P(w_2|w_1)\, P(w_3|w^2_1) \ldots P(w_n|w^{n-1}_1)$$

$$= \Pi^n_{k=1} P(w_k|w^{k-1}_1).$$

2.2 N-GRAM MODEL

The intuition of the N-gram model is that instead of computing the probability of a word given its entire history, we will approximate the history by just the last few words.

The bigram model, for example, approximates the probability BIGRAM of a word given all the previous words $P(w_n|w^{n-1}_1)$ by the conditional probability of the preceding word $P(w_n|w_{n-1})$, instead of computing the probability. This assumption that the probability of a word depends only on the previous MARKOV word is called a Markov assumption. The N-Gram model is one of the most famous models for prediction of next word. Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past. We can generalize the bigram (which looks one word into N-GRAM the past) to the trigram (which looks two words into the past) and thus to the N-gram (which looks N −1 words into the past). Thus the general equation for this N-gram approximation to the conditional probability of the next word in a sequence is:

$$P(w_n|w^{n-1}_1) \approx P(w_n|w^{n-1}_{n-N+1})$$

Probability of complete word sequence:

$$P(w^n_1) \approx \Pi^n_{k=1} P(w_k|w_{k-1}).$$

For the general case Maximum likelihood Estimation i.e. MLE N-Gram parameter estimation:

$$P(w_n|w^{n-1}_{n-N+1}) = \frac{C(w_n|w^{n-1}_{n-N+1}\, w_n)}{C(w_n|w^{n-1}_{n-N+1})}$$

Following figure shows the bigram counts from a piece of a bigram grammar from the Berkeley Restaurant Project. Note that the majority of the values are zero. In fact, we have chosen the sample words to cohere with each other; a matrix selected from a random set of seven words would be even more sparse.

	i	have	to	go	mum bai	…..
i	0.002	0.33	0	0	0.10	…..
have	0.0022	0	0.66	0.0011	0	…..
to	0.00083	0	0.0017	0.28	0.0083	…..
go	0	0	0.0027	0	0.52	…..
mum bai	0.0063	0	0	0	0	…..
…..	…..	…..	…..	…..	…..	…..

Table 1 Bigram probabilities for five words.

2.3 UNIGRAM, BIGRAM, TRIGRAM….

In case of Unigram model, we are using self-word only for prediction i.e. zero previous word. On the other hand we are using one word in case of Bigram and three words for Trigram respectively. Therefore N-gram uses N-1 previous words. The accuracy of prediction depends on number of N.

So, we have to increase the number of N to get accurate next word. By using this technique we have implemented a module for Lyrics generation by predicting next word.

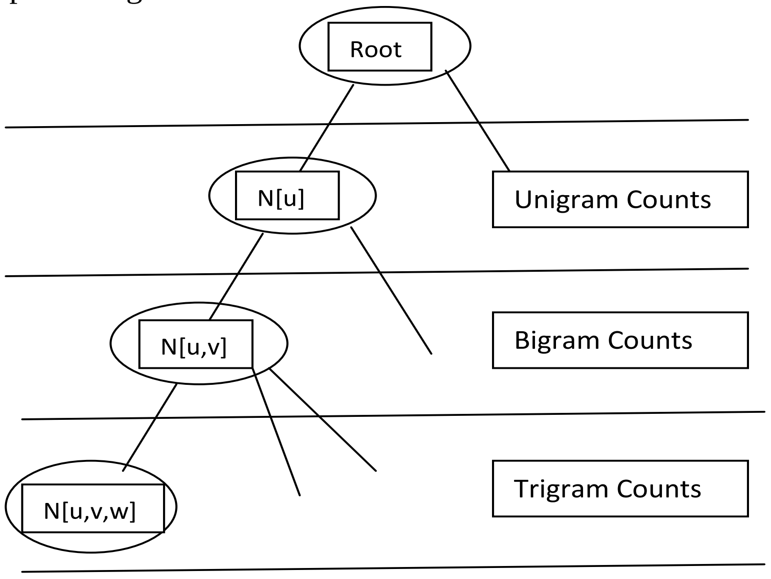


Figure 2 N-Gram Models

i	have	to	go	mumbai	…..
2533	927	2417	746	158	…..

Table 2 Number of counts in corpus for each word.

2.4 MARKOV MODEL

A bigram model is also called a first-order Markov model because it has one element of memory (one token in the past) Markov models are essentially weighted FSAs—i.e., the arcs between states have probabilities. The states in the FSA are words. Much more on Markov models when we hit POS (Part Of Speech) tagging, in which the POS as tags are given to the each word in the sentence.

3 EVALUATION OF N-GRAM

The correct way to evaluate the performance of a language model is to embed it in an application and measure the total performance of the application. Such end-to IN VIVO end evaluation, also called in vivo evaluation, is the only way to know if a particular improvement in a component is really going to help the task at hand. Thus for speech recognition, we can compare the performance of two language models by running the speech recognizer twice, once with each language model, and seeing which gives the more accurate transcription. Here below we have described some concepts related with our work.

3.1 ENTROPY

Natural language can be viewed as a sequence of random variables (a stochastic process *L*), where each random variable corresponds to a word. Based on certain theorems and assumptions (stationary) about the stochastic process, we have…..

$$H(L) = \lim_{n\to\infty} - (1|n) \log p\, (w_1\ w_2 \ldots\ldots w_n)$$

Strictly, we have an estimate q (w_1 w_2……..w_n), which is based on our language models, and we don't know what true probability of words i.e. p (w_1 w_2……..w_n) is.

3.2 CROSS ENTROPY

Cross Entropy is used when we don't know the true probability distribution p that generated the observed data (natural language). Cross Entropy $H(p,q)$ provides an upper bound on the Entropy $H(p)$. We have,

$$H(p, q) = \lim_{n \to \infty} - (1|n) \log q (w_1 w_2 \ldots\ldots w_n)$$

Here we can see that the above formula is extremely similar to above formula of entropy. Therefore, we can say entropy to mean cross entropy.

3.3 PERPLEXITY

Unfortunately, end-to-end evaluation is often very expensive; evaluating a large speech recognition test set, for example, takes hours or even days. Thus we would like a metric that can be used to quickly evaluate potential improvements in a language model. Perplexity is the most common evaluation metric for N-gram language models.

Perplexity is defined as….

$$PP(W) = \sqrt[N]{\frac{1}{P(w1\ w2\ w3 \ldots\ldots wN)}}$$

This quantity is the same as

$$PP(W) = 2^{H(W)}$$

Where H(W) is the cross entropy of the sequence W.

Therefore Perplexity: Trigram < Bigram < Unigram.

4 SMOOTHING ALGORITHMS

There are various algorithms are in the natural language processing. These are some techniques useful in N-gram model. When the problem of overflow or underflow occurs at that time these techniques are used.

4.1 ADD-ONE SMOOTHING

One way to smooth is to add a count of one to every bigram- in order to still be a probability, all probabilities need to sum to one, so, we add the number of word types to the denominator (i.e., we added one to every type of bigram, so we need to account for all our numerator additions). Therefore, this smoothing is called as add-one smoothing.

$$P^* (w_n \mid w_{n-1}) = [\, C (w_{n-1,} w_n) + 1] \mid [\, C (w_{n-1}) + V \,]$$

V = total number of word types / Vocabulary.

4.2 BACK OFF

Only use lower-order model when data for higher-order model is unavailable (i.e. count is zero). Recursively back-off to weaker models until data is available. This smoothing is called as back off.

4.3 INTERPOLATION

It linearly combines estimates of N-gram models of increasing order. It is another technique in smoothing techniques. This smoothing is called as interpolation.

$$P^{\wedge}(w_n \mid w_{n-2,} w_{n-1}) = \lambda_1 P(w_n \mid w_{n-2,} w_{n-1}) + P(w_n \mid w_{n-1}) + P^{\wedge}(w_n \mid w_n)$$

Where: $\sum_i \lambda_i = 1$

Learn proper values for λ_ι by training to (approximately) maximize the likelihood of a held-out dataset.

5 DISCOUNTTING ALGORITHMS

Another way of viewing smoothing is as discounting. Lowering non-zero counts to get the probability mass we need for the zero count items. The discounting factor can be defined as the ratio of the smoothed count to the MLE count. Jurafsky and Martin show that add-one smoothing can discount probabilities by a factor of 8!

Discount the probability mass of seen events, and redistribute the subtracted amount to unseen events. Laplace (Add-One) Smoothing is a type of "discounting" method => simple, but doesn't work well in practice. Better discounting options are

1) Good-Turing, 2) Witten-Bell, 3) Kneser-Ney.

Advanced smoothing techniques are usually a mixture of [discounting + back off] or [discounting + interpolation].

5.1 WRITTEN-BELL DISCOUNTING

Instead of simply adding one to every n-gram, compute the probability of w_{i-1}, w_i by seeing how likely w_{i-1} is at starting any bigram. Words that begin lots of bigrams lead to higher "unseen Bigram" probabilities. Non-zero bigrams are discounted in essentially the same manner as zero count bigrams!

Here we are using this algorithm for calculation of bigram probabilities.

6 IMPLEMENTATION

During the implementation of N-gram we have got some issues regarding to underflow. This is all because of more number of zeros occurred in the database.

Always handle probabilities in log space. Avoid underflow. Notice that multiplication in linear space becomes addition in log space => this is good, because addition is faster than multiplication. Therefore we are changing the expression of multiplication into addition as given below,

$$p_1 * p_2 * p_3 * p_4 = \exp(\log p_1 + \log p_2 + \log p_3 + \log p_4)$$

In the implementation part of the N-Gram we have created four tables, namely count of each word, count of each word after previous word, unsmoothed probabilities of the word and at last probabilities after smoothing the table.

The following table shows the count of each word after previous word.

	i	have	to	go	mumbai	……..
i	11	1090	0	14	0	……..
have	4	0	786	0	9	……..
to	6	0	13	866	5	……..
go	4	0	471	8	45	……..
mumbai	4	0	81	0	0	……..
……	…	……	…	…	……	……..

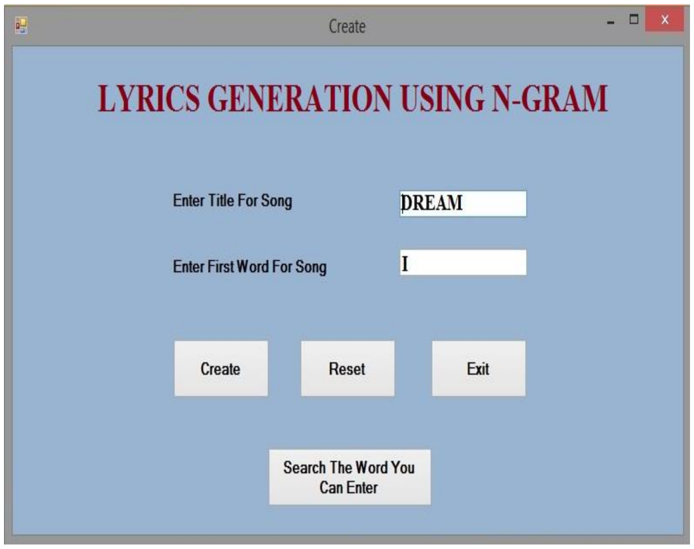
Table 3 Number of counts with previous word.

There is a problem in above table is that there are more number of zeros, therefore while calculating probabilities problem of underflow occurs. The next and last table is calculated by using various methods of smoothing as explained in previous section. Therefore we have smoothed the table 1, by using Add-one smoothing algorithm as shown in table 4.

	i	have	to	go	mum bai	….
i	0.002 1	0.331	0.002	0.00 8	0.101	….
have	0.002 4	0.000 2	0.661	0.00 14	0.007	….
to	0.000 9	0.000 3	0.001 7	0.28	0.008 32	…
go	0.005 6	0.002	0.002 7	0.00 04	0.521	….
mum bai	0.006 4	0.000 9	0.000 5	0.00 6	0.000 1	….
….	….	….	….	….	….	….

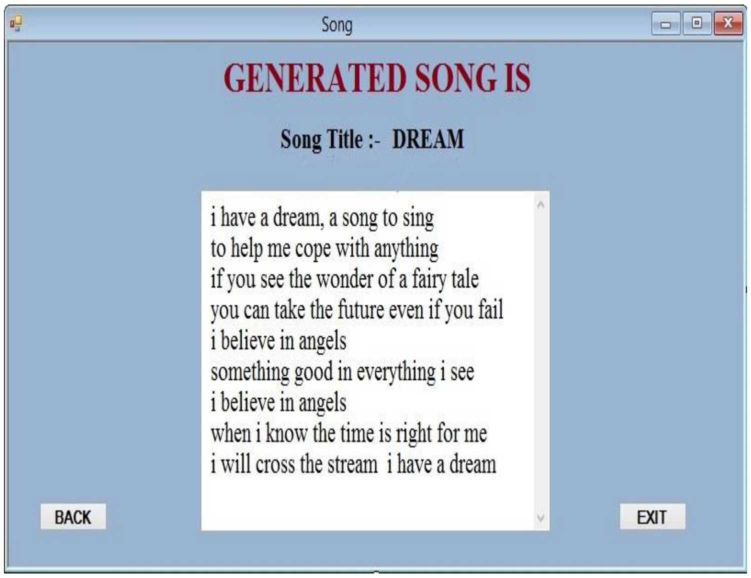
Table 4 Probabilities after smoothing.

This is the last table which shows the probability of the word with the previous words. The number of previous words is based on the order of N-Gram model. By using which we can see the next word. By using this we have implemented a model for Lyrics generation. This is fully based on above model.


Create
LYRICS GENERATION USING N-GRAM
Enter Title For Song
DREAM
Enter First Word For Song
I
Create
Reset
Exit
Search The Word You Can Enter

Output Window 1 shows to insert input.

Here, we have shown two output windows one is for getting input and another is for generating output. While process we have used above four tables and n-gram model for generation of Lyrics of song.



GENERATED SONG IS

Song Title :- DREAM

i have a dream, a song to sing
to help me cope with anything
if you see the wonder of a fairy tale
you can take the future even if you fail
i believe in angels
something good in everything i see
i believe in angels
when i know the time is right for me
i will cross the stream i have a dream

Output Window 2 shows the generated song for a word.

7 CONCLUSION

Thus, in this paper, by using n-gram technique we have created the Lyrics of the new songs. Here we focused on the applicability of n-gram co-occurrence measures for the prediction of next word. The N-Gram model is one of the most famous models for prediction of next word. In the model, the n-gram reference distribution is simply defined as the product of the probabilities of selecting lexical entries with machine learning features of word and POS N-gram. For our application you have to insert two words as input one is song title and another is the starting word of the song. Then it creates the Lyrics of song for the word that you have entered.

ACKNOWLEDGEMENT

This author would like to thank Dr. A. R. Babhulgaonkar for his guidance of N-Gram model, by using which we have implemented this. We also would like to thank Dr. A. W. Kiwelekar head of department of Computer Engineering and other staff.

REFERENCES

[1] An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition from Second Edition, by Daniel

Jurafsky belongs to Stanford University and James H. Martin belongs to University of Colorado at Boulder.

[2] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation.

[3] Marcello Federico and Mauro Cettolo. 2007. Efficient handling of n-gram language models for statistical machine translation. In Proceedings of the Second Workshop on Statistical Machine Translation.

[4] Marcello Federico and Mauro Cettolo. 2007. Efficient handling of n-gram language models for statistical machine translation. In Proceedings of the Second Workshop on Statistical Machine Translation.

[5] David Guthrie and Mark Hepple. 2010. Storing the web in memory: space efficient language models with constant time retrieval. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

[6] Boulos Harb, Ciprian Chelba, Jeffrey Dean, and Sanjay Ghemawat. 2009. Back-off language model compression. In Proceedings of Interspeech.

[7] Zhifei Li and Sanjeev Khudanpur. 2008. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation.

[8] Abby Levenberg and Miles Osborne. 2009. Streambased randomised language models for smt. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

[9] Paolo Boldi and Sebastiano Vigna 2005. Codes for the world wide web. Internet Mathematics.