3-2015

Comparing the Effect of Smoothing and N-gram Order : Finding the Best Way to Combine the Smoothing and Order of N-gram

Wenyang Zhang

Comparing the Effect of Smoothing and N-gram Order : Finding the Best Way to Combine the Smoothing and Order of N-gram

by

Wenyang Zhang

A thesis submitted to the College of Engineering at

Florida Institute of Technology

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Computer Engineering

Melbourne, Florida

March, 2015

We the undersigned committee
hereby approve the attached thesis

Comparing the Effect of Smoothing and Order of N-gram for Language Model :
Find the Best Way to Combine the Smoothing and Order of N-gram

by

Wenyang Zhang

Kepuska, Veton, Ph.D	Samuel Kozaitis, Ph.D
Associate Professor	Professor and Head
Electrical and Computer	Electrical and Computer
Engineering Department	Engineering Department
Thesis Advisor	

Otero, Carlos, Ph.D	Ribeiro, Eraldo, Ph.D
Associate Professor	Associate Professor
Electrical and Computer	Computer Sciences
Engineering Department	Department

Abstract

Comparing the Effect of Smoothing and N-gram Order : Finding the Best Way to Combine the Smoothing and Order of N-gram

By

Wenyang Zhang

Thesis Adviser: Kepuska, Veton , Ph. D.

The SRILM is a toolkit for building and applying statistical language models (LMs), designed and developed primarily for use in speech recognition, statistical tagging and segmentation, and machine translation. It has been under development in the SRI Speech Technology and Research Laboratory since 1995. The toolkit has also greatly benefited from its use and enhancements during the Johns Hopkins University/CLSP summer workshops in 1995, 1996, 1997, and 2002. In this thesis, the effect of smoothing and order of N-gram for language model we build by srilm toolkit is studied.

My primary method is to use comparison. Firstly, training corpus and testing corpus in website is downloaded. This should be checked in all of the document. Then, I use command window and training corpus to train a language model in different smoothing and order of n-gram and test another one we downloaded in website. Finally, I will get the perplexities which can weigh the language model. I will also list every perplexity and compare them in different smoothing and order of n-gram to see which language model we built has minimal perplexity. Then, we will knwhich language model we built is the best one.

Also, I will do it again by another two different corpora, one for training, another for testing, to see the effect of different corpus for language model. If the two group perplexity is the same, it means the different corpus do not affect perplexity. Otherwise, the result is opposite.

In conclusion, my measure above all is to calculate perplexity of each language model in different smoothing and order of n-gram and compare every perplexity to find the best way to match the smoothing and order of n-gram for the language model. At the same time, we will know the effect of different corpus for the language model with same smoothing and order of n-gram.

Table of Contents

Table of Contents v

List of Figures vi

List of Tables vii

Acknowledgement viii

Chapter 1 Introduction 1

1.1 Statistical Language Modeling 2

1.2 Perplexity 4

Chapter 2 Background 5

2.1 N-grams 5

2.2 N-GRAMS Application Areas 6

2.3 Word Counting & Corpora 7

2.4 Simple N-gram 7

2.5 Maximum Likelihood Estimation – Parameter Estimation 9

2.6 Training and Test Sets 11

2.7 Smoothing of N-gram Model 12

Chapter 3 Methods 26

3.1 Method1 26

3.2 Method2 29

3.3Method3 31

Chapter 4 Conclusion 33

References 34

List of Figures

Figure 1 — Comparing N-gram Order and Smoothing. 24

Figure 2 — Comparing N-gram Order and Smoothing 31

Figure 3 — Comparing N-gram Order 32

List of Tables

Table 1 — N-gram Perplexity 4

Table 2—Bigram counts for 8 of the words (out of V=1446) in Berkeley Restaurant Project corpus of 9332 sentences. 10

Table 3 — Unigram Counts 10

Table 4 — Bigram Probabilities After Normalization 11

Table 5 — Berkeley Restaurant Project Smoothed Bigram Counts (V=1446) . 14

Table 6 — Probability 18

Table 7 — Simple Example 19

Table 8 — Perplexity Chart 1 27

Table 9 — Perplexity Chart 2 29

Table 10 — Perplexity Chart 3 32

Acknowledgments

I would like to thank everybody who helped me to prepare this thesis. First, I would like to thank my research advisor Veton Kepuska. He provided me a great environment in which research was a fun activity for me. He encouraged me to tackle challenging problems and put endless efforts in teaching. Without his guidance and company. I would not finish my thesis.

I would also like to thank Andreas Stolcke and Nickolay V. Shmyrev in providing me with the training and testing corpus.

Lastly, I thank my parents Junhong Zhang and Lingchang Zhang for their boundless support in my life.

Chapter 1

Introduction

Language model is a mathematical model which describes natural language's internal law. In theory, building a language model is to conclude, find and gain natural language statistics and structural internal law. In practice, language model is widely to be used in speech recognition, handwriting-recognition system, machine translation and natural language managing area.

Language model can be divided into two types: knowledge-based language model and statistical language model. The current mature one is the statistical language model.

This thesis is mainly about the effect of smoothing and order of ngram for the language model. The Srilm is a building and using statistical language modeling toolkit. It can realize a series of training, predicting, and calculating operation. In the establishment model based on the language of the word, the word frequency estimation model accuracy depends on the corpus. When the corpus size is limited, we need to use some smoothing algorithm. For example, Linear Interpolation, Absolute-Discount, Good-Turing, Witten-Bell, which will be defined in next chapter, to solve the data sparse problem. Taking advantage of srilm, we can conveniently create and test the language model based on a variety of n-gram.

This thesis mainly studies the perplexity in different n-gram which is from 1-gram to 7-grams. We also study the perplexity in different smoothing algorithms which include simple n-gram, linear interpolation, good-turing, witten-bell and so on.

1.1 Statistical Language Modeling

The goal of Statistical Language Modeling is to build a statistical language model that can estimate the distribution of natural language as accurate as possible. A statistical language model (SLM) is a probability distribution P(s) over strings S that attempts to reflect how frequently a string S occurs as a sentence.

By expressing various language phenomena in terms of simple parameters in a statistical model, SLMs provide an easy way to deal with complex natural language in a computer.

The original (and still most important) application of SLMs is speech recognition, but SLMs also play a vital role in various other natural language applications as diverse as machine translation, part-of-speech tagging, intelligent input method and Text To Speech system.

N-gram model is the most widely used SLM today.

Without loss of generality we can express the probability of a string s: p(s) as

p(s) = p(w1)p(w2|w1)p(w3|w1w2)...p(wl|w1...wl-1) = p(wi|w1...wi-1))

In bigram models, we make the approximation that the probability of a word only depends on the identity of the immediately preceding word, hence we can approximate p(s) as:

p(s) = p(wi|wi-1)

The parameters in a traditional N-gram model can be estimated with (Maximum Likelihood Estimation) MLE technique:

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

p-probability, c-time of occurrences, w-word, i-the order of word

N-gram models have received intensive research since its invention, several enhanced N-gram models have been proposed. Here are some typical extensions to traditional N-gram model:

- Class-based N-gram model

  In order to cope with the data sparseness problem, the class-based N-gram model was proposed. Instead of dealing with separated words, class-based N-gram estimates parameters for *word classes*. By clustering words into classes, a class-based N-gram model can reduce the model size significantly with the cost of slightly higher perplexity.

- Sequence N-gram model

  Sequence N-gram is an attempt to extend N-gram models with variable length sequences. A sequence can be a sequence of words, word class, part-of-speech or whatever a sequence of *something* that the modeler believes bearing important grammar information.

1.2 Perplexity

The perplexity (sometimes called *PP* for short) of a language model on a test set is a function of the probability that the language model assigns to that test set. For a test set $W = w_1w_2 \ldots w_N$, the perplexity is the probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N)}}$$

We can use the chain rule to expand the probability of W:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 w_2 \ldots w_{i-1})}}$$

For bigram language model the perplexity of W is computed as:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}}$$

Minimizing perplexity is equivalent to maximizing the test set probability according to the language model.

Perplexity can also be interpreted as the weighted average branching factor of a language. The branching factor of a language is the number of possible next words that can follow any word.

Example of Perplexity Use：

Perplexity is used in following example to compare three N-gram models.

Table 1- Ngram Perplexity

N-gram order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unigram, Bigram, and Trigram grammars are trained on 38 million words (including start-of-sentence tokens) using WSJ corpora with 19,979 word vocabulary.

The perplexity is related inversely to the likelihood of the test sequence according to the model.

Chapter 2

Background

2.1 N-grams

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus.

An n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". Larger sizes are sometimes referred to by the value of n, e.g., "four-gram", "five-gram", and so on.

The n-gram is related to problem of word prediction. For example: very likely words: "call", "international call", or "phone call", and NOT "the".

The idea of word prediction is formalized with probabilistic models called N-grams. N-gram predict the next word from previous N-1 words. Statistical models of word sequence are also called language models or LMs. Computing probability of the next word will turn out to be closely related to computing the probability of a sequence of words. Estimators like the N-gram that assign a conditional probability to possible next words can be used to assign a joint probability to an entire sentence. N-grams models are one of the most important tools in speech and language processing. N-gram are essential in any tasks in which the words must be identified from ambiguous and noisy inputs.

2.2 N-grams Application Areas

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a (n-1)–order Markov model. N-gram models are now widely used in probability, communication theory, computational linguistics (for instance, statistical natural language processing), computational biology (for instance, biological sequence analysis), and data compression. The two core advantages of n-gram models (and algorithms that use them) are relative simplicity and the ability to scale up – by simply increasing n a model can be used to store more context with a well-understood space–time tradeoff, enabling small experiments to scale up very efficiently.

Firstly, n-gram can be used in Statistical Machine Translation. Statistical Machine Translation (SMT) is a machine translation paradigm where translations are generated on the basis of statistical models whose parameters are derived from the analysis of bilingual text corpora. The statistical approach contrasts with the rule-based approaches to machine translation as well as with example-based machine translation.

Secondly, n-gram can be used in Augmentative Communication. Augmentative Communication is helping people who are unable to use speech or sign language to communicate (e.g., Steven Hawking). Using simple body movements to select words from a menu that are spoken by the system.

Thirdly, word prediction can be used to suggest likely words for the menu.

Other areas:

- Part of-speech tagging
- Natural Language Generation
- Word Similarity
- Authorship identification
- Sentiment Extraction
- Predictive Text Input (Cell phones)

2.3 Word Counting & Corpora

The word count is the number of words in a document or passage of text. Word counting may be needed when a text is required to stay within certain numbers
9 KB (1,027 words).

Counting things in natural language is based on a corpus (plural corpora) – an on-line collection of text or speech. Popular corpora "Brown" and "Switchboard". 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 1963-1964. Switchboard Corpus – collection of 2430 telephone conversations averaging 6 minutes each – total of 240 hours of speech with about 3 million words. An example sentence from Brown corpus: He stepped out into the hall, was delighted to encounter a water brother.

2.4 Simple N-gram

Our goal is to compute the probability of a word w given some history h: P(w|h)

Example:

h = "*its water is so transparent that"*

w ="*the*"

P(*the* | *its water is so transparent that*)

How can we compute this probability?

One way is to estimate it from relative frequency counts. From a very large corpus count number of times we see "*its water is so transparent that"* and count the number of times. This is followed by *"the"*: Out of the times we saw the history ***h***, how many times was it followed by the word ***w***:

$$P(the \mid its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

Joint Probabilities is probability of an entire sequence of words like "its water is so transparent":

Out of all possible sequences of 5 words, how many of them are "*its water is* *so transparent"*. You must count of all occurrences of "*its water is so transparent"* and *divide* by the sum of counts of all possible 5 word sequences. It seems a lot of work for a simple computation of estimates.

Hence, we must figure out more clever ways of estimating the probability of a word ***w*** given some history ***h***, or an entire word sequence ***W***. If a sequence of N words: $w_1, w_2, \ldots, w_n \; or \; w_1^n$

Chain rule of Probability:

$$P(X_1,\ldots,X_n) = P(X_1)P(X_2 \mid X_1)P(X_3 \mid X_1^2)\ldots P(X_n \mid X_1^{n-1})$$
$$= \prod_{k=1}^{n} P(X_k \mid X_1^{k-1})$$

Applying the chain rule to words we get:

$$P(w_1^n) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1^2)\ldots P(w_n \mid w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k \mid w_1^{k-1})$$

The chain rule provides the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

However, we still do not know any way of computing the exact probability of a word given a long sequence of preceding words: $P(w_n \mid w_1^{n-1})$.

Idea of the N-gram model is to approximate the history by just the last few words instead of computing the probability of a word given its entire history.

The bigram model approximates the probability of a word given all the previous words $P(w_n \mid w_1^{n-1})$ by the conditional probability of the preceding $P(w_n \mid w_{n-1})$ word.

Example: Instead of computing the probability:

$$p(the \mid \; its \quad water \quad is \quad so \quad transparent \quad that)$$

It is approximated with the probability:

$$P(the \mid that)$$

The following approximation is used when the bi-gram probability is applied:

$$P(w_n \mid w_1^{n-1}) \approx P(w_n \mid w_{n-1})$$

The assumption that the conditional probability of the word depends only on the previous word is called a Markov assumption. 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.

Tri-gram: looks two words into the past. N-gram: looks N-1 words into the past.

General equation for N-gram approximation to the conditional probability of the next word in a sequence is:

$$P\left(w_n \mid w_1^{n-1}\right) \approx P\left(w_n \mid w_{n-N+1}^{n-1}\right)$$

The simplest and most intuitive way to estimate probabilities is the method called Maximum Likelihood Estimation or MLE for short.

2.5 Maximum Likelihood Estimation – Parameter Estimation

Mini-corpus containing three sentences marked with beginning sentence marker <s> and ending sentence marker </s>:
<s> I am Sam </s>
<s> Sam I am</s>
<s> I do not like green eggs and ham</s>
Some of the bi-gram calculations from this corpus:
P(I|<s>) = 2/3 = 0.66
P(Sam|<s>) = 1/3=0.33
P(am|I) = 2/3 = 0.66
P(Sam|am) = ½=0.5
P(</s>|Sam) = 1/3 = 0.33
P(</s>|am) = 1/3=0.33
P(do|I) = 1/3 = 0.33
In general case, MLE is calculated for N-gram model using the following:

$$P\left(w_n \mid w_{n-N+1}^{n-1}\right) = \frac{C\left(w_{n-N+1}^{n-1} w_n\right)}{C\left(w_{n-N+1}^{n-1}\right)} = \frac{C\left(w_{n-N+1} \ldots w_{n-2} w_{n-1} w_n\right)}{C\left(w_{n-N+1} \ldots w_{n-2} w_{n-1}\right)}$$

Table 2- Bigram counts for 8 of the words (out of V=1446) in Berkeley Restaurant Project corpus of 9332 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	628	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Table 3-Unigram Counts

i	want	to	eat	Chinese	food	lunch	spend
2533	927	214	746	158	1098	341	278

Table 4 - Bigram Probabilities After Normalization

	i	want	to	eat	Chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
Chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Clearly now we can compute probability of sentence like:

"I want English food", or "I want Chinese food" by multiplying appropriate bigram probabilities together as follows:

P(<s> i want english food </s>) =

P(i|<s>)	(0.25)
P(want|i)	(0.33)
P(english|want)	(0.0011)
P(food|english)	(0.5)
P(</s>|food)	(0.68)

= 0.25 x 0.33 x 0.0011 x 0.5 x 0.68 = 0.000031

Although we will generally show bigram models in this chapter , note that when there is sufficient training data we are more likely to use trigram models, which condition on the previous two words rather than the previous word.

To compute trigram probabilities at the very beginning of sentence, we can use two pseudo-words for the first trigram (i.e., *P*(I|<s><s>).

2.6 Training and Test Sets

N-gram models are obtained from a corpus that is trained on. Those models are used on some new data in some task (e.g. speech recognition). New data or tasks will not be exactly the same as data that was used for training. Formally, data that is used to build the N-gram (or any model) are called Training Set or Training Corpus. Data that are used to test the models comprise Test Set or Test Corpus.

Training-and-testing paradigm can also be used to evaluate different N-gram architectures: Comparing N-grams of different order N, or Using the different smoothing algorithms (to be introduced later)

Train various models using training corpus. Evaluate each model on the test corpus.

How do we measure the performance of each model on the test corpus? The answer is Perplexity. Computing probability of each sentence in the test set: the model that assigns a higher probability to the test set (hence more accurately predicts the test set) is assumed to be a better model.

For training we need as much data as possible. However, for testing we need sufficient data in order for the resulting measurements to be statistically significant. In practice often the data is often divided into 80% training 10% development and 10% evaluation.

N-gram modeling, like many statistical models, is very dependent on the training corpus.

Often the model encodes very specific facts about a given training corpus.

N-grams do a much better and better job of modeling the training corpus as we increase the value of N. This is another aspect of model being tuned to specifically to training data at the expense of generality.

2.7 Smoothing of N-gram Model

In statistics and image processing, to smooth a data set is to create an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena. In

smoothing, the data points of a signal are modified so individual points (presumably because of noise) are reduced, and points that are lower than the adjacent points are increased leading to a smoother signal. Smoothing may be used in two important ways that can aid in data analysis (1) by being able to extract more information from the data as long as the assumption of smoothing is reasonable and (2) by being able to provide analyses that are both flexible and robust. Many different algorithms are used in smoothing. Data smoothing is typically done through the simplest of all density estimators, the histogram.
There is a major problem with the maximum likelihood estimation process we have seen for training the parameters of an *N*-gram model. This is the problem of sparse data caused by the fact that our maximum likelihood estimate was based on a particular set of training data. For any *N*-gram that occurred a sufficient number of times, we might have a good estimate of its probability. But because any corpus is limited, some perfectly acceptable English word sequences are bound to be missing from it.
This missing data means that the *N*-gram matrix for any given training corpus is bound to have a very large number of cases of putative "zero probability *N*-grams" that should really have some non-zero probability.
Furthermore, the MLE method also produces poor estimates when the counts are non-zero but still small. We need a method which can help get better estimates for these zero or low frequency counts.
Zero counts turn out to cause another huge problem.
The perplexity metric defined above requires that we compute the probability of each test sentence. But if a test sentence has an *N*-gram that never appeared in the training set, the Maximum Likelihood estimate of the probability for this *N*-gram, and hence for the whole test sentence, will be zero!
This means that in order to evaluate our language models, we need to modify the MLE method to assign some non-zero probability to any *N*-gram, even one that was never observed in training.
The term smoothing is used for such modifications that address the poor estimates due to variability in small data sets. The name comes from the fact that (looking ahead a bit) we will be shaving a little bit of probability mass from the higher counts, and piling it instead on the zero counts, making the distribution a little less discontinuous.
The original Berkeley Restaurant example introduced previously will be used to show how smoothing algorithms modify the bigram probabilities.
One simple way to do smoothing is to take our matrix of bigram counts, before we normalize them into probabilities, and add one to all the counts. This algorithm is called Laplace smoothing, or Laplace's Law.
Laplace smoothing does not perform well enough to be used in modern *N*-gram models, but we begin with it because it introduces many of the concepts that we

will see in other smoothing algorithms, and also gives us a useful baseline.

Recall that the unsmoothed maximum likelihood estimate of the unigram probability of the word w_i is its count c_i normalized by the total number of word tokens N: $P(w_i) = \frac{c_i}{N}$

Laplace smoothing adds one to each count. Considering that there are V words in the vocabulary, and each one got increased, we also need to adjust the denominator to take into account the extra V observations in order to have legitimate probabilities.

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V}$$

It is convenient to describe a smoothing algorithm as a corrective constant that affects the numerator by defining an adjusted count $c*$ as follows:

$$P_{Laplace}(w_i) = \frac{c_i + 1}{N + V} = \frac{c_i + 1}{N + V}\frac{N}{N} = \left(c_i + 1\frac{N}{N + V}\right)\frac{1}{N} = \frac{c_i^*}{N}$$

$$c_i^* = c_i + 1\frac{N}{N + V}$$

A related way to view smoothing is as discounting (lowering) some non-zero counts in order to get the correct probability mass that will be assigned to the zero counts.

Thus instead of referring to the discounted counts c, we might describe a smoothing algorithm in terms of a relative discount d_c, the ratio of the discounted counts to the original counts: $d_c = \frac{c^*}{c}$

Table 5- Berkeley Restaurant Project Smoothed Bigram Counts (V=1446)

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	629	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1

spend	2	1	2	1	1	1	1	1

Recall normal bigram probabilites are computed by normalizing each raw of counts by the unigram count:

$$P(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

For add-one smoothed bigram counts we need to augment the unigram count by the number of total types in the vocabulary V:

$$P^*_{Laplace}(w_n \mid w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

It is often convenient to reconstruct the count matrix so we can see how much a smoothing algorithm has changed the original counts.
These adjusted counts can be computed by the equation presented below and the table in the next slide shows the reconstructed counts.

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n)+1]\times C(w_{n-1})}{C(w_{n-1})+V}$$

Note that add-one smoothing has made a very big change to the counts. *C*(***want to***) changed from 609 to 238! We can see this in probability space as well: *P*(***to***|***want***) decreases from .66 in the unsmoothed case to .26 in the smoothed case.

Looking at the discount *d* (the ratio between new and old counts) shows us how strikingly the counts for each prefix-word have been reduced; the discount for the bigram *want to* is .39, while the discount for Chinese food is .10, a factor of 10! The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros.

Good-Turing Estimation

Good–Turing frequency estimation was developed by Alan Turing and his assistant I. J. Good as part of their efforts at Bletchley Park to crack German ciphers for the Enigma machine during World War II. Turing at first modeled the frequencies as a multinomial distribution, but found it inaccurate. Good developed smoothing algorithms to improve the estimator's accuracy.

The discovery was recognized as significant when published by Good in 1953, but the calculations were difficult so it was not used as widely as it might have been. The method even gained some literary fame due to the Robert Harris novel *Enigma*.

In the 1990s, Geoffrey Sampson worked with William A. Gale of AT&T, to create and implement a simplified and easier-to-use variant of the Good–Turing method .

Good–Turing frequency estimation is a statistical technique for estimating the probability of encountering an object of a hitherto unseen species, given a set of past observations of objects from different species. (In drawing balls from an urn, the 'objects' would be balls and the 'species' would be the distinct colors of the balls (finite but unknown in number). After drawing R_{red} red balls, R_{black} black balls and R_{green} green balls, we would ask what is the probability of drawing a red ball, a black ball, a green ball or one of a previously unseen color.)

A number of much better algorithms have been developed that are only slightly more complex than add-one smoothing: Good-Turing.

The idea behind a number of those algorithms is to use the count of things you've seen once to help estimate the count of things you have never seen.

Good described the algorithm in 1953 in which he credits Turing for the original idea.

The basic idea in this algorithm is to re-estimate the amount of probability mass to assign to N-grams with zero counts by looking at the number of N-grams that occurred only one time. A word or N-gram that occurs once is called a singleton. Good-Turing algorithm uses the frequency of singletons as a re-estimate of the frequency of zero-count bigrams.

Algorithm Definition:

- Nc – the number of N-grams that occur c times: frequency of frequency c.
- N0 – the number of bigrams b with count 0.
- N1 – the number of bigram with count 1 (singletons), etc.
- Each Nc is a bin that stores the number of different N-grams that occur in the training set with frequency c:

$$N_c = \sum_{x:count(x)=c} 1$$

■ The MLE count for N_c is c. The Good-Turing estimate replaces this with a smoothed count $c*$, as a function of N_{c+1}:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

The previous equation presented in previous slide can be used to replace the MLE counts for all the bins N_1, N_2, and so on. Instead of using this equation directly to re-estimate the smoothed count $c*$ for N_0, the following equation is used that defines probability P^*_{GT} of the missing mass:

$$P^*_{GT}(\textit{things with frequency zero in training}) = \frac{N_1}{N}$$

N_1 – is the count of items in bin 1 (that were seen once in training), and N is total number of items we have seen in training.

Example:

- A lake with 8 species of fish (bass, carp, catfish, eel, perch, salmon, trout, whitefish)
- When fishing we have caught 6 species with the following count:10 carp 3 perch, 2 whitefish, 1 trout, 1 salmon, and 1 eel (no catfish and no bass).
- What is the probability that the next fish we catch will be a new species, i.e., one that had a zero frequency in our training set (catfish or bass)?

The MLE count ***c*** of unseen species (bass or catfish) is 0. From the equation in the previous slide the probability of a new fish being one of these unseen species is 3/18, since $\boldsymbol{N_1}$ is 3 and $\boldsymbol{N}$ is 18:

$$P^*_{GT}(\textit{things with frequency zero in training}) = \frac{N_1}{N} = \frac{3}{18}$$

Let's now estimate the probability that the next fish will be another trout? MLE count for trout is 1, so the MLE estimated probability is 1/18.

However, the Good-Turing estimate must be lower, since we took 3/18 of our probability mass to use on unseen events! Must discount MLE probabilities for observed counts (perch, whitefish, trout, salmon, and eel)

The revised count ***c****** and Good-Turing smoothed probabilities P^*_{GT} for species with counts 0 (like bass or catfish in previous example) or counts 1 (like trout, salmon, or eel) are as follows:

Table 6 - Probability

	Unseen(bass or catfish)	trout
c	0	1
MLE p		$\frac{1}{18}$
C*		$c^*(trout) = (c+1) \times \frac{N_2}{N_1} = 2 \times \frac{1}{3} = 0.67$
GT p^*_{GT}	$p^*_{GT}(unseen) = \frac{N_1}{N} = \frac{3}{18} = 0.17$	$p^*_{GT}(trout) = \frac{c^*}{N} = \frac{0.67}{18} = 0.037$

Note that the revised count c* for eel as well is discounted from c=1.0 to

c*=0.67 in order to account for some probability mass for unseen species (unseen) = 3/18=0.17 for catfish and bass.

Since we know that there are 2 unknown species, the probability of the next

fish being specifically a catfish is (catfish) = (1/2)x(3/18) = 0.085

Table 7 - Simple Example

c(MLE)	Nc	C*(GT)
0	2	1.5
1	3	1.3
2	1	1
3	1	?
4	0	?
5	0	?
6	0	?
7	0	?
8	0	?
9	0	?
10	1	?

In practice, Add-one smoothing works horribly. Good-Turing smoothing works better than Add-one, but works still horribly. So Good-Turing is often not used by itself; it is used in combination with the backoff and interpotion algorithms.

Interpolation

Discounting algorithms can help solve the problem of zero frequency N-grams. Additional knowledge that is not used:

- If trying to compute $P(w_n|w_{n-1}w_{n-2})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$,
- Estimate trigram probability based on the bigram probability $P(w_n|w_{n-1})$.
- If there are no counts for computation of bigram probability $P(w_n|w_{n-1})$, use unigram probability $P(w_n)$.

There are two ways to rely on this N-gram "hiearchy": Backoff and

Interpolation.

Backoff Relies solely on the trigram counts. When there is a zero count evidence of a trigram then the backoff to lower N-gram.

To Interpolation, probability estimates are always mixed from all N-gram estimators: weighted interpolation of trigram, bigram and unigram counts.

Simple Interpolation – Linear Interpolation.

$$\begin{aligned}\hat{P}(w_n \mid w_{n-1}w_{n-2}) = {} & \lambda_1 P(w_n \mid w_{n-1}w_{n-2}) \\ & + \lambda_2 P(w_n \mid w_{n-1}) \\ & + \lambda_3 P(w_n) \\ & \sum_i \lambda_i = 1\end{aligned}$$

Slightly more sophisticated version of linear interpolation with context dependent weights.

$$\begin{aligned}\hat{P}(w_n \mid w_{n-1}w_{n-2}) = {} & \lambda_1 (w_{n-2}^{n-1}) P(w_n \mid w_{n-1}w_{n-2}) \\ & + \lambda_2 (w_{n-2}^{n-1}) P(w_n \mid w_{n-1}) \\ & + \lambda_3 (w_{n-2}^{n-1}) P(w_n) \\ & \sum_i \lambda_i (w_{n-2}^{n-1}) = 1\end{aligned}$$

Weights are set from held-out corpus. Held-out corpus is additional training corpus that is not used to set the N-gram counts but to set other parameters like in this case interpolation weights.

Backoff

Interpolation is simple to understand and implement. There are better algorithms like backoff N-gram modeling.

Uses Good-Turing discounting based on Katz and also known as Katz backoff.

$$P_{katz}(w_n \mid w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n \mid w_{n-N+1}^{n-1}) & \text{if } C(w_{n-N+1}^{n}) > 0 \\ \alpha(w_{n-N+1}^{n-1}) P_{katz}(w_n \mid w_{n-N+2}^{n-1}) & \text{otherwise} \end{cases}$$

Equation above describes a recursive procedure. Computation of P*, the normalizing factor *a*, and other details are discussed in next section.

- β – total amount of left-over probability mass function of the

(N-1)-gram context.

For a given (N-1) gram context, the total left-over probability mass can be computed by subtracting from 1 the total discounted probability mass for all N-grams starting with that context.

$$\beta\left(w_{n-N+1}^{n-1}\right)=1-\sum_{w_n:c\left(w_{n-N+1}^{n-1}\right)>0} P*\left(w_n \mid w_{n-N+1}^{n-1}\right)$$

- *This gives us the total probability mass that we are ready to distribute to all (N-1)-gram (e.g., bigrams if our original model was trigram)*

Each individual (*N*-1)-gram (bigram) will only get a fraction of this mass, so we need to normalize *b* by the total probability of all the (*N*-1)-grams (bigrams) that begin some *N*-gram (trigram) that has zero count. The final equation for computing how much probability mass to distribute from an *N*-gram to an (*N*-1)-gram is represented by the function *a*:

$$\alpha\left(w_{n-N+1}^{n-1}\right)=\frac{\beta\left(w_{n-N+1}^{n-1}\right)}{\sum_{w_n:c\left(w_{n-N+1}^{n-1}\right)>0} P_{katz}\left(w_n \mid w_{n-N+2}^{n-1}\right)}$$

$$=\frac{1-\sum_{w_n:c\left(w_{n-N+1}^{n-1}\right)>0} P^*\left(w_n \mid w_{n-N+1}^{n-1}\right)}{1-\sum_{w_n:c\left(w_{n-N+1}^{n-1}\right)>0} P^*\left(w_n \mid w_{n-N+2}^{n-1}\right)}$$

Note that *a* is a function of the preceding word string, that is, of w_{n-N+1}^{n-1}; thus the amount by which we discount each trigram (*d*), and the mass that gets reassigned to lower order *N*-grams (*a*) are recomputed for every (*N*-1)-gram that occurs in any *N*-gram.

We only need to specify what to do when the counts of an (*N*-1)-gram context are 0, (i.e., when $c\left(w_{n-N+1}^{n-1}\right)=0$) and our definition is complete:

$$\begin{array}{ll}
P_{katz}\left(w_n \mid w_{n-N+1}^{n-1}\right)=P_{katz}\left(w_n \mid w_{n-N+2}^{n-1}\right) & \text{if } c\left(w_{n-N+1}^{n-1}\right)=0 \\
P^*\left(w_n \mid w_{n-N+1}^{n-1}\right)=0 & \text{if } c\left(w_{n-N+1}^{n-1}\right)=0 \\
\beta\left(w_n \mid w_{n-N+1}^{n-1}\right)=1 & \text{if } c\left(w_{n-N+1}^{n-1}\right)=0
\end{array}$$

Advanced Smoothing Methods: Kneser-Ney Smoothing

Language models are an essential element of natural language processing, central to tasks ranging from spellchecking to machine translation. Given an arbitrary piece of text, a language model determines whether that text belongs to a given language.

We can give a concrete example with a probabilistic language model, a specific construction which uses probabilities to estimate how likely any given string belongs to a language. Consider a probabilistic English language model PE. We would expect the probability

PE(I went to the store)

To be quite high, since we can confirm this is valid English. On the other hand, we expect the probabilities

PE(store went to I the),PE(Ich habe eine Katz)

To be very low, since these fragments do not constitute proper English text.

I don't aim to cover the entirety of language models at the moment — that would be an ambitious task for a single blog post. If you haven't encountered language models or n-grams before, I recommend the following resources:

- "Language model" on Wikipedia
- Chapter 4 of Jurafsky and Martin's Speech and Language Processing
- Chapter 7 of Statistical Machine Translation (see summary slides online)

I'd like to jump ahead to a trickier subject within language modeling known as Kneser-Ney smoothing. This smoothing method is most commonly applied in an interpolated form,[1] and this is the form that I'll present today.

Kneser-Ney evolved from absolute-discounting interpolation, which makes use of both higher-order (i.e., higher-n) and lower-order language models, reallocating some probability mass from 4-grams or 3-grams to simpler unigram models. The formula for absolute-discounting smoothing as applied to a bigram language model is presented below:

$$P_{abs}(w_i \mid w_i) = \frac{\max(c(w_{i-1} w_i) - \delta, 0)}{\sum_{w'} c(w_{i-1} w')} + \alpha \, p_{abs}(w_i)$$

Here δ refers to a fixed discount value, and α is a normalizing constant. The details of this smoothing are covered in Chen and Goodman (1999).

The essence of Kneser-Ney is in the clever observation that we can take advantage of this interpolation as a sort of backoff model. When the first term (in this case, the discounted relative bigram count) is near zero, the second term (the lower-order model) carries more weight. Inversely, when the higher-order model matches strongly, the second lower-order term has little weight.

The Kneser-Ney design retains the first term of absolute discounting interpolation, but rewrites the second term to take advantage of this relationship. Whereas absolute discounting interpolation in a bigram model would simply default to a unigram model in the second term, Kneser-Ney depends upon the idea of a continuation probability associated with each unigram.

This probability for a given token Wi is proportional to the number of bigrams which it completes:

$$P_{continuation}(w_i) \propto \left|\left\{w_{i-1} : c(w_{i-1}, w_i) \rangle 0\right\}\right|$$

This quantity is normalized by dividing by the total number of bigram types (note that j
is a free variable):

$$P_{continuation}(w_i) = \frac{\left|\left\{w_{i-1} : c(w_{i-1}, w_i) > 0\right\}\right|}{\left|\left\{w_{j-1} : c(w_{j-1}, w_j) > 0\right\}\right|}$$

The common example used to demonstrate the efficacy of Kneser-Ney is the phrase San Francisco. Suppose this phrase is abundant in a given training corpus. Then the unigram probability of Francisco will also be high. If we unwisely use something like absolute discounting interpolation in a context where our bigram model is weak, the unigram model portion may take over and lead to some strange results.

Dan Jurafsky gives the following example context:

I can’t see without my reading _____.

A fluent English speaker reading this sentence knows that the word glasses should fill in the blank. But since San Francisco is a common term, absolute-discounting interpolation might declare that Francisco is a better fit: $P_{abs}(Francisco) > P_{abs}(glasses)$

Kneser-Ney fixes this problem by asking a slightly harder question of our lower-order model. Whereas the unigram model simply provides how likely a word wi is to appear, Kneser-Ney’s second term determines how likely a word wi is to appear in an unfamiliar bigram context.

Kneser-Ney in whole follows:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - \delta, 0)}{\sum_{w'} c(w_{i-1}w')} + \lambda \frac{|\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|}{|\{w_{j-1} : c(w_{j-1}, w_j) > 0\}|}$$

λ is a normalizing constant

$$\lambda(w_{i-1}) = \frac{\delta}{c(w_{i-1})} \left|\{w' : c(w_{i-1}, w') > 0\}\right| .$$

Note that the denominator of the first term can be simplified to a unigram count. Here is the final interpolated Kneser-Ney smoothed bigram model, in all its glory:

$$P_{KN}(w_i \mid w_{i-1}) = \frac{\max(c(w_{i-1}w_i) - \delta, 0)}{c(w_{i-1})} + \lambda \frac{|\{w_{i-1} : c(w_{i-1}, w_i) > 0\}|}{|\{w_{j-1} : c(w_{j-1}, w_j) > 0\}|}$$

A brief introduction to the most commonly used modern *N*-gram smoothing is method, the Interpolated Kneser-Ney algorithm:
An algorithm is based on absolute discounting method.

It is a more elaborate method of a computing revised count *c** than the Good-Turing discount formula. Re-estimated counts *c** for greater than 1 counts could be estimated pretty well by just subtracting 0.75 from the MLE count *c*. Absolute discounting method formalizes this intuition by subtracting a fixed (absolute) discount d from each count.
The rational is that we have good estimates already for the high counts, and a small discount d won't affect them much.

The affected are only the smaller counts for which we do not necessarily trust the estimate anyhow.
In practice, distinct discount values *D* for the 0 and 1 counts are computed.
Kneser-Ney discounting augments absolute discounting with a more sophisticated way to handle the backoff distribution. Consider the job of predicting the next word in the sentence, assuming we are backing off to a unigram model:
I can't see without my reading XXXXXX. The word "*glasses*" seem much more likely to follow than the word "*Francisco*".But "*Francisco*" is in fact more common, and thus a unigram model will prefer it to "*glasses*".
Thus we would like to capture that although "*Francisco*" is frequent, it is only frequent after the word "*San*". The word "*glasses*" has a much wider distribution.
Thus the idea is instead of backing off to the unigram MLE count (the number of times the word *w* has been seen), we want to use a completely different backoff distribution!
We want a heuristic that more accurately estimates the number of times we might expect to see word *w* in a new unseen context.
The Kneser-Ney intuition is to base our estimate on the number of different contexts word w has appeared in.
Words that have appeared in more contexts are more likely to appear in some new context as well. New backoff probability can be expressed as the "*continuation probability*" presented in following expression:
Continuation Probability:

$$P_{continuation}(w_i) = \frac{\left|\left\{w_{i-1} : c(w_{i-1}w_i) > 0\right\}\right|}{\sum_{w_i}\left|\left\{w_{i-1} : c(w_{i-1}w_i) > 0\right\}\right|}$$

Kneser-Ney backoff is formalized as follows assuming proper coefficient a on the backoff to make everything sum to one:

$$p_{KN}(w_i \mid w_{i-1}) = \begin{cases} \frac{c(w_{i-1}w_i) - D}{c(w_{i-1})}, & \textit{if } c(w_{i-1}w_i) > 0 \\ \alpha(w_i)\frac{\left|\left\{w_{i-1} : c(w_{i-1}w_i) > 0\right\}\right|}{\sum_{w_i}\left|\left\{w_{i-1} : c(w_{i-1}w_i) > 0\right\}\right|} & \textit{otherwise} \end{cases}$$

Chapter 3

Methods

3.1 Method 1

Here we describe my method which includes the steps of the experiment, generating counting file from corpus, and calculating testing data perplexity.
I downloaded the data in a website and used command window to generate the n-gram counting file. The sentence we used in command window is “ngram-count -text news-commentary-v6.en -order 1 -write-order news-commentary-v6.en.count”. And then I used the counting file to train language model. The sentence I used was “ngram-count -read news-commentary-v6.en.count -order 3 -lm news-commentary-v6.en.lm”. The last thing I did was used the language model we created above to calculate testing data perplexity. The sentence we used in cmd(command window) was “ngram -ppl europarl-v6.en -order 1 -lm news-commentary-v6.en.lm >   news-commentary-v6.en.lm.ppl”.
My testing corpus was named the Raw momolingual language model training data. The website in which I downloaded it was https://code.google.com/p/1-billion-word-language-modeling-benchmark/. it contained 2.0154e+006 sentences, 4.93701e+007 words, and 1.24923e+006 OOVs, 0 zeroprobs, logprob= -1.56782e+008, ppl=?, ppl=10^{-{logP(T)}/{Sen+Word}}. Sen and Word represent the quantity of sentences and words.

Table 8 - Perplexity Chart 1

Perplexity	Order1	Order2	Order3	Order4	Order5	Order6	Order7
Simple ngram	1340.01	352.833	313.079	312.092	312.837	313.025	313.077
interpolate	1340.01	352.833	313.079	312.092	312.837	313.025	313.077
Good turing discounting	1340.01	352.833	313.079	312.092	312.837	313.025	313.077
Natural discounting+interpolate	1340.01	376.867	315.429	312.005	312.339	312.834	313.006
Witten-Bell discounting+interpolate	1364.44	368.307	311.097	311.248	312.321	312.837	313.004
Modified Knes	1343.05	347.347	313.393	311.248	312.247	312.814	313.016

er-Ney discounting+interpolate							
Original Kneser-Ney discounting+interpolate	1340.01	349.052	316.915	313.483	312.703	312.858	313.007

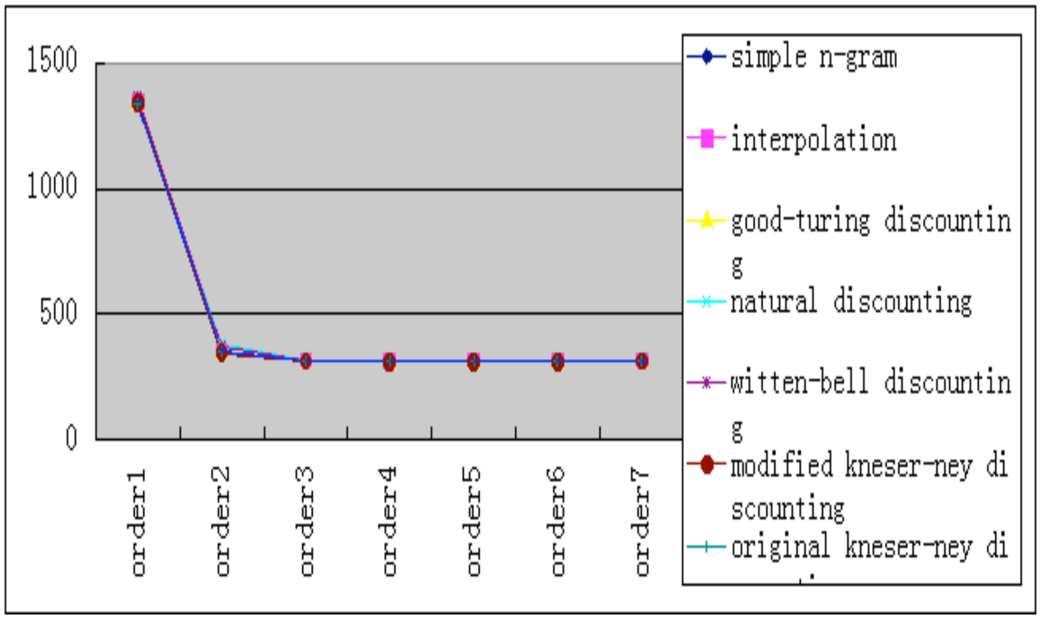

Figure 1 - Comparing N-gram Order and Smoothing

3.2 Method 2

My testing corpus was named European Parliament Proceedings Parallel Corpus 1996-2011. The website in which I downloaded it was http://www.statmt.org/europarl/. It contained 649697 sentences, 1.56859e+007 words, and 2131 OOVs, 0 zeroprobs, logprob= -4.9319e+007, ppl=?, ppl=10^{-{logP(T)}/{Sen+Word}}. Sen and Word represent the quantity of sentences and words.

Table 9 - Perplexity Chart 2

Perplexity	Order1	Order2	Order3	Order4	Order5	Order6	Order 7
Simple ngram	1045.92	96.4242	57.2009	49.1651	47.9456	47.8842	47.9993
Interpolate dicounting	1045.92	96.4242	57.2009	49.1651	47.9456	47.8842	47.9993
Good-turing discounting	1045.92	96.4242	57.2009	49.1651	47.9456	47.8842	47.9993
Natural discountin	1045.92	89.5422	56.771	48.0942	46.5125	46.5675	46.9316

g							
Witten-bell	1047.71	92.2205	56.4033	47.3234	45.9748	46.2979	46.8127
Modified kneser-ney	1046.21	97.0386	57.159	48.1449	46.6114	46.7785	47.2074
Original knerser-ney	1045.92	94.8735	56.4876	47.6903	46.2829	46.4957	46.9573
Kneser-ney	1045.92	96.4242	57.2009	49.1651	47.9456	47.8842	47.9993

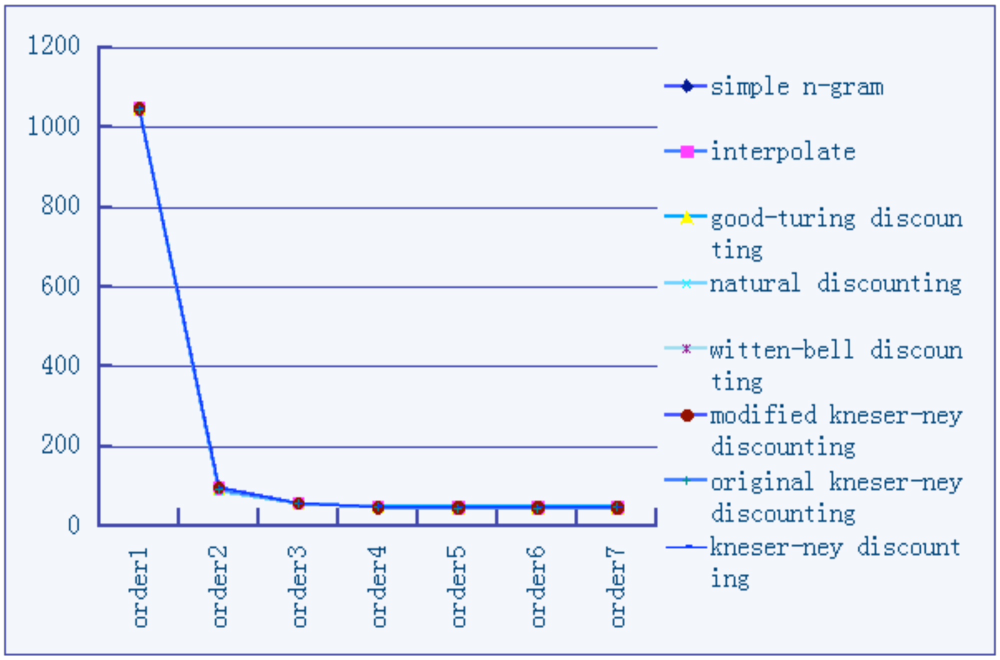

Figure 2 - Comparing N-gram Order and Smoothing

3.3 Method 3

I downloaded the corpus which name is Berkeley Restaurant Project. In Berkeley Restaurant Project file, every word in it is related to restaurant. It contains 1446 OOVs, 7500 sentences, 1500 words. I used sentence "ngram -order N training.en.lm -vocab icslp94-berp.ps-2 -limit-vocab -renorm -write-lm icslp94-berp.ps-2.lm" in command window to generate the new language model file. Then I used the sentence "ngram -ppl testing.en -order N -lm icslp94-berp.ps-2.lm> icslp94-berp.ps-2.lm.ppl" in command window to generate perplexity file, and to compare the perplexity from 1-gram to 7-grams.

Testing corpus contains 649697 sentences, 1.56859e+007 words, and 1.52601e+007 OOVs, 0 zeroprobs, logprob= -4.9319e+007, ppl=?, ppl=10^{-{logP(T)}/{Sen+Word}}. Sen and Word represent the quantity of sentences and words.

Table 10 - Perplexity Chart 3

Order1	Order2	Order3	Order4	Order5	Order6	Order7
53.5936	52.5935	52.5939	52.5939	52.5939	52.5939	52.5939

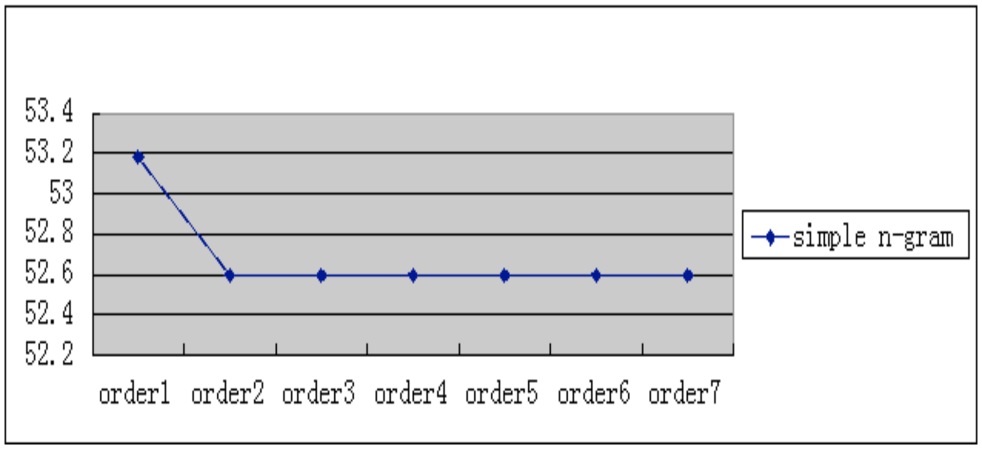

Figure 3 – Comparing N-gram Order

Chapter 4

Conclusion

From the result I got, I think in the first experiment, the best matching is witten-bell discounting and 3-gram in which the perplexity is 311.097. The second experiment best matching is witten-bell and 5-gram discounting in which the perplexity is 45.9748.

It means that the corpus size will effect the perplexity and the matching result. So different corpus will have different best matching between smoothing and order of n-gram.

In third experiment, I can conclude that we can build the language model by general corpus in specific domain. There is no need for us to find specific corpus. I can find some words and sentence in website that is related to some specific domain to generate new language model in some specific domain like Berkeley Restaurant Project that is about restaurant words.

From all 3 figures, I can prove that the higher order gram has better perplexity. Because the trigram mode has better perplexity than unigram mode. In specific domain, the perplexity is generally lower.

In the future, we can use the building language model technology to apply for some specific domain like law domain, management domain and so on to build the language model by general corpus instead of specific corpus. Also, we can study more different corpus to find the best matching between smoothing and n-gram order to apply for different speech recognition applications.

References

Levin, B. and Rappaport Hovav, M. (2005). Argument Realization. Cambridge University Press.

Li, X. and Roth, D. (2002). Learning question classifiers. In COLING-02, PP. 556-562.

Hobbs, J. R. (1979a). Metaphor, metaphor schemata, and selective inferencing. Tech. rep. 204, SRI.

Galley, M., Hopkins, M., Knight, K., and Marcu, D. (2004). What's in a translation rule?. In HLT-NAACL-04.

Edmunson, H. (1969). New methods in automatic extracting. Journal of the ACM, 16(2), 264-285

Cohen, P. R. (1995). Empirical Methods for Artificial Intelligence. MIT Press.

Branco, A., McEnery, T., and Mitkov, R. (2002). Anaphora Processing. John Henjamins.

Balentine, B. and D. Morgan, *How to build a Speech Recognition Application*, 1999, Enterprise Integration Group.

Galitz, W.O., *User-Interface Screen Design*, 1993, Wellesley, MA, Q. E. D. Information Sciences Inc.

Furui, S., "Resent Advances in Speaker Recognition," *Pattern Recognition Letters*, 1997, **18**, pp. 859-872.

Galitz, W.O., *Handbook of Screen Format Design*, 1985, Wellesley, MA, Q.E.D. Information Sciences Inc.

Wood, L., *User Interface Design*: Bridging the Gap from User Requirements to Design, 1997, CRC Press.

Wixon, D. and J. Ramey, *Field Methods Casebook for Software Design,* 1996, New York, John Wiley.

Weinschenk, S. and D. Barker, *Designing Effective Speech Interfaces*, 2000, New York, John Wiley.