NUS National University of Singapore | School of Computing

N-Gram Language Models

CS4248 Natural Language Processing

Week 03

Anab Maulana BARIK and Min-Yen KAN

3

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Prof. Hwee Tou Ng (NUS), and Dan Jurafsky (Stanford)

Recap of Week 02

Regular Expressions

Corpus Preprocessing: Getting to Words

- *Detour*: Morphology / Byte Pair Encoding

Normalization

Spelling Errors

Noisy Channel

Edit Distance

Week 03 Agenda

Language Models
n-grams
The Markov Assumption
Estimating n-gram Probabilities
Evaluating Language Models

Unknown Words, Redux
Smoothing
Backoff and Interpolation
Kneser-Ney Smoothing

Language Models

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

Motivation

Which one makes more sense?

on guys all I of notice sidewalk three a sudden standing the

Or ...

all of a sudden I notice three guys standing on the sidewalk

But why?

The probability of the latter sentence is ***higher*** *!*

P("on guys … the") > *P("all of a sudden I … the sidewalk")*

What are Language Models?

Language models are models that assign probabilities to a sentence.

i, j = set notation

Probability of sequence of words

$$P(W) = P(w_1, w_2, w_3, \dots, w_n)$$

$$P(\text{“please turn your homework”})$$

Probability of an upcoming word

$$P(w_n | w_1, \dots, w_{n-1})$$

$$P(\text{“homework”} | \text{“please turn your”})$$

Where to apply?

Many real-world applications for assigning probabilities to sentences

- Spelling Correction

$$P(\text{“… has no } \underline{mistake}\text{”}) > P(\text{“… has no } \underline{mistaek}\text{”})$$

- Speech Recognition

$$P(\text{“I will be } \underline{back\ soonish}\text{”}) > P(\text{“I will be } \underline{bassoon\ dish}\text{”})$$

Application, Cont.

- Grammatical Error Correction

$$P(\text{“… has } \underline{\text{improved}}\text{”}) > P(\text{“… has } \underline{\text{improve}}\text{”})$$

- Machine Translation

他 向 记者 介绍了 主要 内容

He to reporters introduced main content

$$P(\text{“he briefed reporters } \cancel{\text{on}} \text{ the main contents of the statement”})$$

$$>$$

$$P(\text{“he briefed } \cancel{\text{to}} \text{ reporters the main contents of the statement”})$$

n-Grams

School of Computing

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

Soaking in the hot mineral water with a picturesque natural background is a rewarding treat for visitors to Tram Tau District. … Its water is so transparent that …

Source http://english.vietnamnet.vn/fms/travel/211310/take-a-hot-bath-surrounded-by-nature.html

Probabilities of sentences

Given a sentence *"its water is so transparent that"*

… we'll need a method to compute the relevant probability:

$P(\mathit{its}, \mathit{water}, \mathit{is}, \mathit{so}, \mathit{transparent}, \mathit{that})$ or

$P(\mathit{the} \mid \mathit{its}, \mathit{water}, \mathit{is}, \mathit{so}, \mathit{transparent}, \mathit{that})$

Review: Chain Rule Probability

The chain rule for 2 random events (variables) is:

$$P(A_1, A_2) = P(A_2|A_1)\times P(A_1) = P(A_1|A_2) \times P(A_2)$$

The chain rule for 3 random events is:

$$P(A_1, A_2, A_3) = P(A_3|A_1, A_2)\times P(A_1, A_2)$$
$$= P(A_3|A_1, A_2)\times P(A_2|A_1)\times P(A_1)$$

Chain Rule Probability Cont.

Let's generalize. The chain rule for N random events is:

$i:j$ = sequence notation

start any one

$$P(A_1,\ldots,A_N) = P(A_1)\times P(A_2|A_1)\times P(A_3|A_{1:2})\times\cdots\times P(A_N|A_{1:N-1})$$

$$= \prod_{i=1}^{N} P(A_i|A_{1:i-1})$$

Estimate the probability with MLE:

$$P(A_i|A_{1:i-1}) = \frac{Count(A_{1:i})}{Count(A_{1:i-1})}$$

more common

Chain Rule, applied to Words

We can then apply the chain rule to sequences of words:

$$P(its\ water\ is) = P(its) \times P(water|its) \times P(is|its\ water)$$

Estimate $P(water|its) = \frac{Count(its\ water)}{Count(its)}$

Estimate $P(is|its, water) = \frac{Count(its\ water\ is)}{Count(its\ water)}$

All's good … or is it?

Chain Rule, applied to Words Cont.

How about for long sentences. How about this (not very long) one: $P(the|its\ water\ is\ so\ transparent\ that) =$

$$\frac{Count(its\ water\ is\ so\ transparent\ that\ the)}{Count(its\ water\ is\ so\ transparent\ that)}$$

See any problem? sparcity.

Chain Rule, applied to Words Cont.

How about for long sentences? Take this (not very long) one:
$P(the|its\ water\ is\ so\ transparent\ that) =$

$$\frac{Count(its\ water\ is\ so\ transparent\ that\ the)}{Count(its\ water\ is\ so\ transparent\ that)}$$

What's the problem?

- Joint probability table for many entries;
- Either the sentence (or a subsequence) may not have been seen. It may have a count of **zero**.

The Markov Assumption

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

AMERICAN
Scientist

“The first application of [A. A. Markov’s chains] was to a textual analysis of Alexander Pushkin's poem *Eugene Onegin*. Here a snippet of one verse appears (in Russian and English) along with Pushkin's own sketch of his protagonist Onegin.”

Source
https://www.americanscientist.org/article/first-links-in-the-markov-chain

Markov Assumption

Approximate the probability by assuming that it is just dependent on the last n words:

$$P(the|its\ water\ is\ so\ transparent\ that) \approx P(the|\boldsymbol{that})$$

Or

$$\approx P(the|\boldsymbol{transparent\ that})$$

Markov Assumption Cont.

If the probability only depends on k preceding words, then:

go back to the first word

$$P(A_1 \dots A_N) = \prod_{i=1}^{N} P(A_i | A_{\mathbf{1}:i-1})$$

go back to last k units.

$$= \prod_{i=1}^{N} P(A_i | A_{\boldsymbol{i-k}:i-1})$$

$P(A_i | A_{1:i-1})$ = Prob. of A_i given $A_1, A_2, \ldots A_{i-1}$

n-Gram models

Intuition: approximate the probability by looking at the n preceding words

last one word

- Unigram (1-gram) : $P(A_i|A_{1:i-1})$ $= P(A_i)$

last two words

- Bigram (2-gram) : $P(A_i|A_{1:i-1})$ $= P(A_i | A_{i-1})$
- Trigram (3-gram) : $P(A_i|A_{1:i-1})$ $P(A_i | A_{i-2} A_{i-1})$

n-Gram models

Intuition: approximate the probability by looking at the n preceding words

- Unigram (1-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i)$
- Bigram (2-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i|A_{i-1})$
- Trigram (3-gram) : $P(A_i|A_{1:i-1}) \approx P(A_i|A_{i-2}A_{i-1})$

n-Gram models Cont.

It is common to use more than bigram models; e.g., 3-gram, 4-gram, and 5-gram models.

However, larger grams require more training data.

To think about: How much more?

Estimating n-gram Probabilities

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and Prof. Hwee Tou Ng (NUS)

Maximum Likelihood Estimation

Estimate the probabilities by getting **counts** from corpus and **normalizing** by the sum of all n-grams that share the preceding words.

$$P_{MLE}(A_i|A_{i-1}) = \frac{Count(A_{i-1}A_i)}{\sum_w Count(A_{i-1}A_i)} \text{ (bigram)}$$

Sum of all bigrams that starts with w is equal to unigram A_i, then

$$P_{MLE}(A_i|A_{i-1}) = \frac{Count(A_{i-1}A_i)}{\cancel{Count(A_i)}} \text{ (bigram)}$$

Count(Ai-1)

Maximum Likelihood Estimation Cont.

General MLE with n-gram:

longer sequence smaller prob.

$$P_{MLE}(A_i | A_{i-N+1:i-1}) = \frac{Count(A_{i-N+1} \dots A_i)}{Count(A_{i-N+1} \dots A_{i-1})}$$ shorter seq.

Bigram Example

starting of sentence

Sentences:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P_{MLE}(I|<s>) = \frac{Count(<s>\ I)}{Count(<s>)} = \frac{2}{3}$$

$$P_{MLE}(am|I) = \frac{Count(I\ am)}{Count(I)} = \frac{2}{3}.$$

$$P_{MLE}(Sam|am) = \frac{Count(am\ Sam)}{Count(am)} = \frac{1}{2}$$

$$P_{MLE}(</s>|Sam) = \frac{Count(Sam\ </s>)}{Count(Sam)} = \frac{1}{2}$$

Bigram Example

Sentences:

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

$$P_{MLE}(I|<s>) = \frac{Count(<s> I)}{Count(<s>)} = \frac{2}{3}$$

$$P_{MLE}(am|I) = \frac{Count(I\ am)}{Count(I)} = \frac{2}{3}$$

$$P_{MLE}(Sam|am) = \frac{Count(am\ Sam)}{Count(am)} = \frac{1}{2}$$

$$P_{MLE}(</s>|Sam) = \frac{Count(Sam\ </s>)}{Count(Sam)} = \frac{1}{2}$$

Bigram Example – 2

$p(\text{to} \mid \text{want}) = \frac{\text{count}(\text{want to})}{\text{count}(\text{want})} = \frac{608}{927}$

Larger Corpora – Unigram Counts

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Bigram probabilities

too many zeros!

	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	$\boxed{0.66}$	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

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

Bigram Example – 3

Bigram probabilities

$i+1$ (column index) ; i (row index)

	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

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

Other probabilities (not in table)

$P(I|<s>) = 0.25$

$P(</s>|food>) = 0.68$

Calculate:

$P(<s>\ I\ want\ chinese\ food\ </s>)$

$P(I|<s>) \times$ 0.25

$P(want|I) \times$ 0.33

$P(chinese|want) \times$ 0.0065

$P(food|chinese) \times$ 0.52

$P(</s>|food)$ 0.68

$=$

Bigram Example – 3

Bigram probabilities

$i+1$

i

	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

Figure 3.2 Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences.

Other probabilities (not in table)

$P(I|<s>) = 0.25$

$P(</s>|food>) = 0.68$

Calculate:

$P(<s>\ I\ want\ chinese\ food\ </s>)$

$P(I|<s>) \times$
$P(want|I) \times$
$P(chinese|want) \times$
$P(food|chinese) \times$
$P(</s>|food)$
$=$

$0.25 \times 0.33 \times 0.0065 \times 0.52 \times 0.68$
$= 0.00019$

Practical Issues

Multiplying MLE probabilities could result in underflow.

Hence we always use an equivalent logarithmic format:

$$P_1 \times P_2 \times P_3 \times P_4 \quad \propto \quad \log P_1 + \log P_2 + \log P_3 + \log P_4$$

Evaluating Language Models

Introducing Perplexity

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Prof. Hwee Tou Ng, and Dan Jurfasky (Stanford)

Does our LM perform well?

Does the model assign

- higher probabilities to **frequently occurring** sentences?
- Lower probabilities to **rarely occurring** sentences?

Two ways of evaluating a language model:

- Extrinsic evaluation
- Intrinsic evaluation

Intrinsic versus Extrinsic Evaluation

Intrinsic

internal

Requires **intrinsic metric** to evaluate the model itself (E.g., **perplexity**).

Cheaper and **quicker**.

- evaluation on specific, intermediate task.

We are more interested in intrinsic evaluation here.

Extrinsic

external

Requires a **downstream task** (E.g., running a speech recognizer twice, once with each LM, comparing the results)

Running downstream task is **expensive** and **time-consuming**.

But when would an extrinsic task be more useful?

- evaluation on a real task.

Slides adapted from Prof. Hwee Tou Ng (NUS)

Intrinsic Evaluation

Intrinsic evaluation involves three steps:

1. Train the model on a **training set.**
2. Tune parameters of the model on a **development set.**
3. Test the model on a **test set.**
   Use the **evaluation metric** (here for LMs, **perplexity**) to assess model performance.

Common breakdown = **80:10:10**

- 80% training set
- 10% development set
- 10% test set

Intuition of Perplexity

The Shannon Game: How well can we predict the next word?

I always order pizza with cheese and …

The 33rd President of the US was …

I saw a …

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

always return same result

e.g. the (with highest probability)

Unigrams are terrible at this game

But why?

need context!

Slide adapted from Dan Jurafsky (Stanford)

Perplexity

The best language model is one that best predicts an unseen test set (highest $P(\text{sentence})$)

Perplexity: the inverse probability of the test set, normalized by the number of words. Denoted as $PP(W)$.

Minimizing the perplexity is the same as maximizing the probability

Slide adapted from Dan Jurafsky (Stanford)

Perplexity Cont.

N = length of sentence / corpus

For a test set $W = w_1 w_2 \dots w_N$: $PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

Using the chain rule:

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_{i|} | w_1 \dots w_{i-1})}}$$

Perplexity on bigram model

$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_{i|} | w_{i-1})}}$$

last 2 words

Slide adapted from Dan Jurafsky (Stanford)

Another Interpretation of Perplexity

Perplexity can be thought of as the weighted average branching factor: the number of possible next words that can follow any word

Example:

- Consider task recognizing the digits (*0-9*), each have equal probability $P = \frac{1}{10}$.
- The perplexity will be *10*: $PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$

$$= \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = 10$$

equally likely.

That is, there are 10 outcomes (digits) that can come next, which the system can't decide among.

Unknown Words, Redux

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition

Closed versus Open Vocabulary

Closed Vocabulary

Vocabulary is fixed. All dataset contains words from this vocabulary

No unknown words

Open Vocabulary

Test set may contain words that is not in the vocabulary.

(OOV $\equiv$ out of vocabulary words)

Example: Proper Noun

What's the problem?
The count (or equivalently, the probability) might be **zero**

Handling Unknown Words

1. Choose a vocabulary list in advance.
2. Convert all words that are not in the vocabulary to unknown token `<UNK>` in a normalization step.
3. Estimate the probability for `<UNK>` like other regular words in the training set.

… also, use subword morphological processing (BPE; Week 02).

Or consider smoothing…

Smoothing

Handling OOV

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and Prof. Hwee Tou Ng (NUS)

Zero Counts

Let's say the words that follow the bigram ***"denied the"*** in the WSJ Treebank corpus are

"denied the allegations" = 5

"denied the speculation" = 2

"denied the reports" = 1

And let's say in our test set, we have the phrase

"denied the offer"

Any problems?

- Probability of $P(\textit{offer} \mid \textit{denied the}) = 0$
- Hmm. We can't calculate the perplexity of the test set. (Why?)

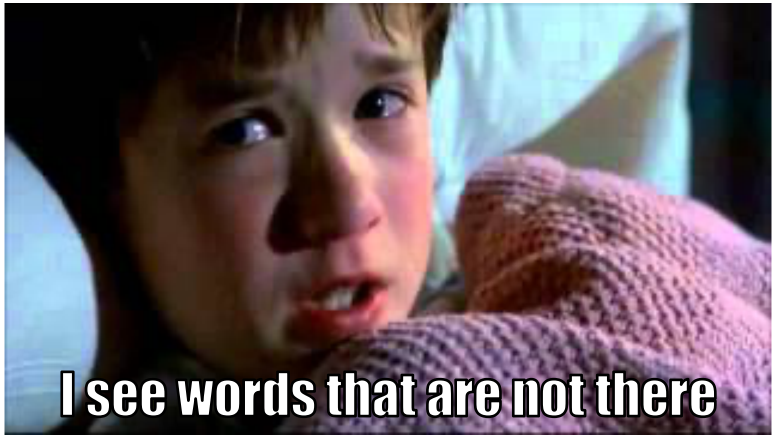
Smoothing

Another way overcome zero counts: **Smoothing**

Robin Hood

Smoothing: take off a bit of non-zero probability n-grams and give it to zero probability n-gram.

Also called **discounting**: lowering non-zero n-gram counts in order to assign some probability mass to the zero n-grams.



Slide adapted from Prof. Hwee Tou Ng (NUS). Photo capture from The Sixth Sense, distributed by Beuna Vista Pictures.

Laplace (Add-1) Smoothing for Bigrams

Add 1 to all counts.

$$C_{laplace}(w) = C(w) + 1$$

"Ingenious." – anonymous

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	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

Figure 3.1 Bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	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

Figure 3.5 Add-one smoothed bigram counts for eight of the words (out of $V = 1446$) in the Berkeley Restaurant Project corpus of 9332 sentences.

Laplace Smoothing for Bigram Cont.

Hence, the probability will be:

$$P_{laplace}(w_n|w_{n-1}) = \frac{C_{laplace}(w_{n-1}w)}{\sum_w C_{laplace}(w_{n-1}w)}$$

table is V by V size

→ original count +1

$$= \frac{C(w_{n-1}w)+1}{\sum_w (C(w_{n-1}w)+1)}$$

$$= \frac{C(w_{n-1}w)+1}{C(w_{n-1})+V}$$

bigram count

Laplace Discounted Count

Probability decreases because 分母 increases!

Discounted count: Adjusted bigram count $C^*(w_{n-1}w_n)$

$$P_{laplace}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} = \frac{C^*(w_{n-1}w_n)}{C(w_{n-1})}$$

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

$$\frac{\frac{C(w_{n-1}w_n)}{C(w_{n-1})}}{\frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}}$$

$\therefore C(w_{n-1}w_n) = [C(w_{n-1}w_n)+1] \cdot \frac{C(w_{n-1})}{C(w_{n-1})+V}.$

Slide adapted from Prof. Hwee Tou Ng (NUS)

Laplace Discount

Discount: ratio of the discounted counts to the original counts d_c.

$$d_c = \frac{C^*(w_{n-1}w_n)}{C(w_{n-1}w_n)}$$

$$d_c = \frac{\{C(w_{n-1}w_n) + 1\}}{C(w_{n-1}w_n)} \times \frac{C(w_{n-1})}{C(w_{n-1}) + V}$$

Slide adapted from Prof. Hwee Tou Ng (NUS)

Add-k smoothing

Generalize Add-1 to add k instead of 1:

But why would we want to do this instead?

$$P_{add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{Count(w_{n-1}) + kV}$$

Slide adapted from Prof. Hwee Tou Ng (NUS)

Backoff and Interpolation

Handling OOV

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and Prof. Hwee Tou Ng (NUS)

Backoff and Interpolation

Intuition: sometimes using **less context** is a good thing.

Take a trigram model for example.

- If the context for trigram exists, we use the trigram. Calculate $P(w_2|w_0 w_1)$ as usual.

  But if what if it doesn't?

- Estimate the probability using bigram $P(w_2|w_1)$

- Otherwise, estimate using unigram $P(w_2)$

Backoff and Interpolation Cont.

Interpolation Intuition: mix the probability estimates from all the n-grams estimators; e.g. weighing and combining trigram, bigram, and unigram counts.

$$P(w_2|w_0w_1) = \underline{\lambda_1 P(w_2|w_0w_1)} + \underline{\lambda_2 P(w_2|w_1)} + \underline{\lambda_3 P(w_2)}$$

Where the sum of $\sum \lambda_i = 1$

λ will be estimated through parameter tuning

randomly try numbers

Kneser Ney Smoothing

Discount + Backoff

School of Computing

Slides adapted from An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and Prof. Hwee Tou Ng (NUS)

Kneser-Ney Smoothing

Intuition: **Absolute discounting.** Discount the seen n-grams count and distribute it to the unseen n-grams.

Bigrams with counts 2–9 in a held-out set was estimated by subtracting **0.75** from the training set.

Bigram count in training set	Bigram count in heldout set
0	0.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

∃ bigram in test set ∉ training set

Reason of decreasing by 0.75

From Church and Gale (1991)

Kneser-Ney Smoothing Cont.

Hence, we the probability of the seen n-grams will be

$$P_{kneser}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) - \delta}{C(w_{n-1})}, C(w_{n-1}w_n) > 0$$

In case of bigrams, we can set $\delta = 0.75$

(alternatively, we can set 0.5 for count = 1 and 0.75 for others)

Knesser-Ney Smoothing Cont.

How about the probability of the **unseen n-grams**?

Use backoff interpolation, based on the number of **different context word w_n has appeared**, normalized by **total bigram types.**

$$P_{kneser}(w_n|w_{n-1}) = \overline{\lambda_1(w_{n-1})} \times \frac{|\{w: C(ww_n) > 0\}|}{\sum_{w'}|\{v: C(vw') > 0\}|}, C(w_{n-1}w_n) = 0$$

eg: I lost my reading ____ ?

If use unigram, "Kong" will be returned. A lot of "Hong Kong" in corpus.

glasses - pair of glasses
- sun glasses
- two glasses

∴ glasses will get returned!

Kneser-Ney Summary

Summing up:

seen in corpus

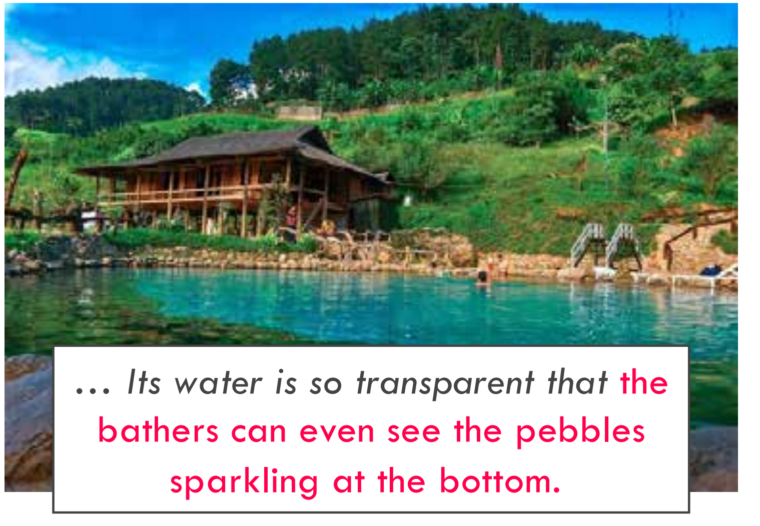
$$P_{kneser}(w_n|w_{n-1}) = \begin{cases} \dfrac{C(w_{n-1}w_n) - \delta}{C(w_{n-1})}, & \boxed{C(w_{n-1}w_n) > 0} \\[2ex] \lambda_1(w_{n-1}) \times \dfrac{|\{w: C(ww_n) > 0\}|}{\sum_{w'}|\{v: C(vw') > 0\}|}, & \boxed{C(w_{n-1}w_n) = 0} \end{cases}$$

unseen bigram.

Summary

Language models: Means to predict the likelihood of a sequence (or next word)

Introduced more ways to handle pesky zero counts: **smoothing** and **backoff**.



Source http://english.vietnamnet.vn/fms/travel/211310/take-a-hot-bath-surrounded-by-nature.html