

# **ERROR CORRECTION PREDICTION N-GRAMS**

---

Ing. R. Tedesco. PhD, AA 20-21

(mostly from: Speech and Language Processing - Jurafsky and Martin)

# Today

- Spelling Errors - Correction
- Word prediction task
- Language modeling (N-grams)
  - N-gram intro
  - The chain rule
  - Model evaluation
  - Smoothing

# ERRORS CORRECTION

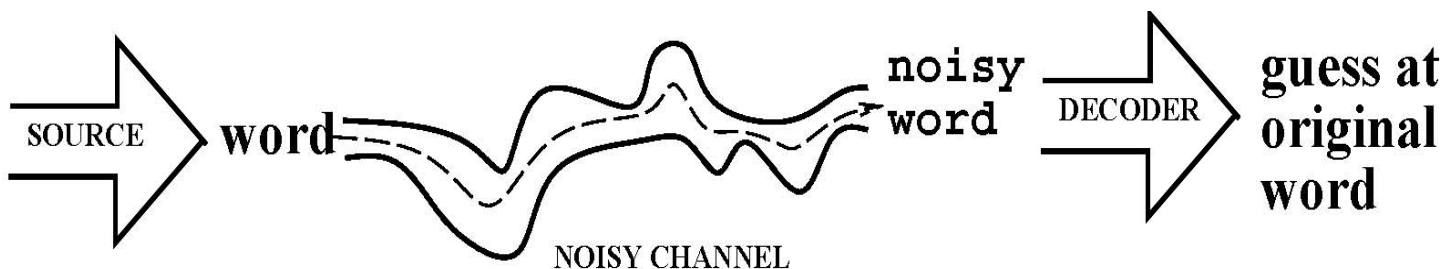
- Optical character recognition (OCR)
- Handwriting recognition
- Spelling mistakes (spelling errors)
  - Detection of non-words
  - Correction of errors (isolated words)
  - Detection and correction of errors depending on the context (even when produce a word -- real-word errors)
  - Typos (linked to the keyboard, replacement, insertion, deletion, transposition)
  - OCR errors (substitution, deletion, addition of spaces, ...)
  - Cognitive errors (phonetic errors - homonyms)
- The probability of a sequence of words
  - Predicting the next word
  - Detection and correction of spelling errors

# Probabilistic Models

- Bayes' rule and the model of the channel noise
- N-grams - Markov chains - Hidden Markov Models (HMM), ...
- Dynamic programming
  - Minimum Edit Distance (spelling errors)
  - .....
- Smoothing

# The Bayesian Model

- Probabilistic models are used for the analysis of errors
- Noise channel model
- The problem of correcting spelling errors (a result of beatings or acquisition by a scanner) can be treated as a mapping of a string of symbols into another
- We model the channel to be able to go back to the original word without the "noise" that makes it difficult recognition (speech recognition - IBM labs - Jelinek, 1976)



# Spelling Correction

- How do I understand that  $O$  is actually an error?

- $O$  not found in a vocabulary

- Several algorithms for the spelling
- Bayesian algorithm

$$\hat{w} = \arg \max_{w \in V} P(w | O)$$

Using:  $P(x | y) = \frac{P(y | x)P(x)}{P(y)}$

$$\hat{w} = \arg \max_{w \in V} \frac{P(O | w)P(w)}{P(O)}$$

$$\hat{w} = \arg \max_{w \in V} \frac{P(O | w)P(w)}{P(O)} = \arg \max_{w \in V} \underbrace{P(O | w)}_{\text{likelihood of } w} \underbrace{P(w)}_{\text{prior of } w}$$

# The Bayesian Methodology

- Kernighan et al. (99)
  - It proposes potential corrections
  - It gives a score to each potential correction
  - Simplifying initial assumption: **that there is only one error per word**
- I step: find possible corrections (1 error max)

$t$ Error	$c$ Possible Correction	Transformation			
		Correct Letter	Error Letter	Position (Letter #)	Error Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	2	5	insertion
acress	acres	—	2	4	insertion

# Bayesian Methodology

- II step; for each  $c$  found at step I, calculate:

$$P(O|w) = P(t \mid c)$$

$$P(w) = P(c)$$

- $P(c)$ : Normalization and “smoothing”:  $P(c) = \frac{C(c) + 0.5}{N + 0.5}$
- K. Et al. take corpus of  $N = 44$  millions of words:

<b>c</b>	<b>freq(c)</b>	<b>p(c)</b>
actress	1343	.0000315
cress	0	.000000014
caress	4	.0000001
access	2280	.000058
across	8436	.00019
acres	2879	.000065



# Bayesian Methodology

- $P(t \mid c)$  requires a huge annotated corpus and other info
  - Who the typist was, whether they were left-handed or right-handed, and many other factors
- K. et al estimates  $P(t=\text{acress} \mid c=\text{across})$  counting in a corpus of errors (how many times *e* has been observed instead of *o*)
  - confusion matrix (del, ins, sub, trans) – **need a training corpus annotated with corrections**
  - $P(t \mid c)$  using del, ins, sub, or trans, according to the difference between *O* and *w*
  - Remember hp: just one error

# Bayesian Methodology

- $\text{del}[x, y]$ : the number of times characters  $xy$  in the correct word were typed as  $x$ , in an annotated corpus
- $\text{ins}[x, y]$ : the number of times character  $x$  in the correct word was typed as  $xy$ .
- $\text{sub}[x, y]$ : the number of times that  $x$  was typed as  $y$ .
- $\text{trans}[x, y]$ : the number of times that  $xy$  was typed as  $yx$ .

$$P(t|c) = \begin{cases} \frac{\text{del}[c_{p-1}, c_p]}{\text{count}[c_{p-1}c_p]}, & \text{if deletion} \\ \frac{\text{ins}[c_{p-1}, t_p]}{\text{count}[c_{p-1}]}, & \text{if insertion} \\ \frac{\text{sub}[t_p, c_p]}{\text{count}[c_p]}, & \text{if substitution} \\ \frac{\text{trans}[c_p, c_{p+1}]}{\text{count}[c_p c_{p+1}]}, & \text{if transposition} \end{cases}$$

Where  $c_p$  is the  $p$ -th character of the word  $c$

# Confusion matrices

- Four confusion matrices, for del, ins, sub, trans)
- E.g.: sub:

X	Y (correct)																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	0	2	21	3	0	0	0	0	3	0

# Bayesian Methodology

- Previous approach for estimating  $P(t | c)$ :
  - Annotated corpus is needed to create confusion matrices
- Another approach (Kernighan et al., 1990)
  - Compute the matrices by iteratively using this very spelling error correction algorithm itself
    1. initializes the matrices with equal values
    2. the spelling error correction algorithm is run on a set of spelling errors
    3. the set of typos is paired with their corrections, the confusion matrices can now be recomputed
    4. goto 2, unless some termination condition is true
  - It is an instance of the EM algorithm (Dempster et al., 1977)

# Minimum Edit Distance

- Still Bayesian methodology
- Removing the simplifying assumption of one error per word
- Consider the distance between the strings as a parameter related to the similarity between strings
- To find the distance between strings:
  - Minimum edit distance (Levenshtein distance) between two strings is the minimum number of editing operations needed to transform one string into another (Wagner and Fischer, 74)
  - Equal weights for each operation
  - Different weights

# Minimum Edit Distance

- $d(O, w)$  = *the min. number of edit operations needed to transform the misspelled word  $O$  into the candidate word  $w$*

representations

Trace

```

i n t e n t i o n
 / / / / | | | |
e x e c u t i o n
    
```

E.g.:  $d(\text{desert}, \text{desertic})=3$

Alignment

```

i n t e n t i o n
ε e x e c u t i o n
    
```

- $P(O|w) = f(d(O, w))$

example:

Operation

List

$$P(O|w) = 1 - \frac{d(O, w)}{\sum_{w'} d(O, w')}$$

$w' \in \{\text{candidate words: words within a predefined max distance}\}$

```

                                i n t e n t i o n
delete i →                      n t e n t i o n
substitute n by e →             e t e n t i o n
substitute t by x →             e x e n t i o n
insert u →                      e x e n u t i o n
substitute n by c →             e x e c u t i o n
                                e x e c u t i o n
    
```

- $P(w)$  = frequency of  $w$  in a corpus

# Context is needed

c	freq(c)	p(c)	p(t c)	p(t c)p(c)	%
actress	1343	.0000315	.000117	$3.69 \times 10^{-9}$	37%
cress	0	.000000014	.00000144	$2.02 \times 10^{-14}$	0%
caress	4	.0000001	.00000164	$1.64 \times 10^{-13}$	0%
access	2280	.000058	.000000209	$1.21 \times 10^{-11}$	0%
across	8436	.00019	.0000093	$1.77 \times 10^{-9}$	18%
acres	2879	.000065	.0000321	$2.09 \times 10^{-9}$	21%
acres	2879	.000065	.0000342	$2.22 \times 10^{-9}$	23%

} 44%

- The Bayesian algorithm predicts **acres** (44%) as a correction to be made and **actress** (37%) as a second chance.
- Prediction ... Incorrect! ... This much is clear from the **context**  
*... Was Called a "stellar and versatile **acress** whose combination of sass and glamor has defined her ..."*

# Word Prediction

- Guess the next word...
  - *... I notice three guys standing on the ???*
- There are many sources of knowledge that can be used to inform this task, including arbitrary world knowledge.
- But it turns out that you can do pretty well by simply looking at the preceding words and keeping track of some fairly simple counts.



# Word Prediction

- We can formalize this task using what are called  $N$ -gram models.
- $N$ -grams are token sequences of length  $N$ .
- Our earlier example contains the following 2-grams (aka bigrams)
  - (I notice), (notice three), (three guys), (guys standing), (standing on), (on the)
- Given knowledge of counts of  $N$ -grams such as these, we can guess likely next words in a sequence.

# ***N*-Gram Models**

- More formally, we can use knowledge of the counts of *N*-grams to assess the conditional probability of candidate words as the next word in a sequence.
- Or, we can use them to assess the probability of an entire sequence of words.
  - Pretty much the same thing as we'll see...

# Applications

- It turns out that being able to predict the next word (or any linguistic unit) in a sequence is an extremely useful thing to be able to do.
- It lies at the core of many applications:
  - Automatic speech recognition
  - Handwriting and character recognition
  - Spelling correction
  - Machine translation
  - And many more.

# Context-based error correction: *N*-grams

- Predict next word
- Useful for context-based error correction

$$\hat{w}_1^n = \operatorname{argmax}_{w_1^n \in V^n} P(O_1^n | w_1^n) P(w_1^n)$$

$$P(w_1^n) = \prod_i^n P(w_i | w_{i-1}) \quad \text{Markov approximation (more on that later...)}$$

# Counting

- Simple counting lies at the core of any probabilistic approach.
- Let's first take a look at what we're counting.
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - 16 tokens, 18 if we include “,” and “.” as separate tokens.
    - Assuming we include the comma and period, how many bigrams are there?

# Counting

- Not always that simple
  - *I do uh main- mainly business data processing*
- Spoken language poses various challenges.
  - Should we count “uh” and other fillers as tokens?
  - What about the repetition of “mainly”? Should such do-overs count twice or just once?
  - The answers depend on the application.
    - If we’re focusing on something like ASR to support indexing for search, then “uh” isn’t helpful (it’s not likely to occur as a query).
    - But filled pauses are very useful in dialog management, so we might want them there.

# Counting: Types and Tokens

- How about
  - *They picnicked by the pool, then lay back on the grass and looked at the stars.*
    - 18 tokens (again counting punctuation)
- But we might also note that “*the*” is used 3 times, so there are only 16 *unique word* (or **types**), as opposed to 18 **tokens**.
- In going forward, we’ll have occasion to focus on counting both types and tokens of both words and *N*-grams.

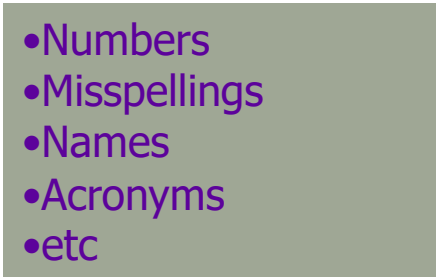
# Counting: Wordforms

- Should “cats” and “cat” count as the same when we’re counting?
- How about “geese” and “goose”?
- Some terminology:
  - **Lemma**: the orthographic base form of a lexeme
  - **Wordform**: an inflected surface form
- Again, we’ll have occasion to count both lemmas and wordforms



# Counting: Corpora

- So what happens when we look at large bodies of text instead of single utterances?
- Brown et al (1992) large corpus of English text
  - 583 million wordform tokens
  - 293 181 wordform types
- Google
  - Crawl of 1 024 908 267 229 English tokens
  - 13 588 391 wordform types
    - That seems like a lot of types... After all, even large dictionaries of English have only around 500k types. Why so many here?



- Numbers
- Misspellings
- Names
- Acronyms
- etc

# Language Modeling

- Back to word prediction
- We can model the word prediction task as the ability to assess the conditional probability of a word given the previous words in the sequence:
  - $P(w_n | w_1, w_2 \dots w_{n-1})$
- We'll call a statistical model that can assess this a *Language Model*

# Language Modeling

- How might we go about calculating such a conditional probability?

- One way is to use the definition of conditional probabilities and look for counts. So to get:

- "its water is so transparent that ..."

- $P(\text{the} \mid \text{its water is so transparent that})$



the ?

- By definition that's:

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

Warning for mathematicians: in these probabilities, the order of words matters!

# Very Easy Estimate

- How to estimate?
- We can get each of those from counts in a large corpus:

$$P(\text{the | its water is so transparent that}) = \\ = \frac{\textit{Count}(\text{its water is so transparent that the})}{\textit{Count}(\text{its water is so transparent that})}$$

# Very Easy Estimate

- According to Google those counts are 5/9.
  - Unfortunately... 2 of those were to these slides... So maybe it's really
  - 3/7
  - In any case, *that's not terribly convincing due to the small numbers involved*
- Finding a sample that is composed of several tokens is usually quite difficult
  - You will find a few samples...

# Language Modeling

- Unfortunately, for most sequences and for most text collections we won't get good estimates from this method.
  - What we're likely to get is 0. Or worse 0/0.
- Clearly, we'll have to be a little more clever.
  - Let's use the chain rule of probability
  - And a particularly useful independence assumption.

# The Chain Rule

- Recall the definition of conditional probabilities:

$$P(A | B) = \frac{P(A, B)}{P(B)}$$

- Rewriting:  $P(A, B) = P(A | B)P(B)$

- For sequences...

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

- And in general...

$$P(x_1, x_2, x_3, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1 \dots x_{n-1})$$

# The Chain Rule

$$P(w_1, w_2, \dots, w_n) = P(w_1^n) \quad \text{A shorthand...}$$

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned} \quad \begin{array}{l} \text{NB: defining} \\ P(w_1 | w_1^0) \equiv P(w_1) \end{array}$$

$$P(\text{its water was so transparent}) =$$

$$= P(\text{its}) \cdot$$

$$P(\text{water} | \text{its}) \cdot$$

$$P(\text{was} | \text{its water}) \cdot$$

$$P(\text{so} | \text{its water was}) \cdot$$

$$P(\text{transparent} | \text{its water was so})$$



# Unfortunately

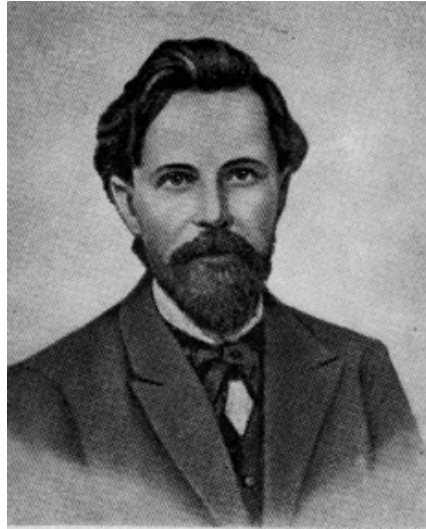
- There are still a lot of possible sentences
- In general, we'll never be able to get enough data to compute the statistics for those longer prefixes
  - Same problem we had for the strings themselves

# Independence Assumption

- Make the simplifying assumption
  - “the other day I was walking along and saw a lizard”
  - $P(\text{lizard} \mid \text{the,other,day,I,was,walking,along,and,saw,a}) = P(\text{lizard} \mid \text{a})$
- Or maybe
  - $P(\text{lizard} \mid \text{the,other,day,I,was,walking,along,and,saw,a}) = P(\text{lizard} \mid \text{saw a})$
- That is, the probability in question is independent of its earlier history.
  - Keep in mind that this assumption does not actually hold, in this case. It is just an approximation!

# Independence Assumption

- This particular kind of independence assumption is called a *Markov assumption* after the Russian mathematician Andrei Markov.



# Markov Assumption

So, for each component in the product replace with the approximation (assuming a prefix of  $N$ ; i.e., an  $N$ -gram)

$$P(w_k | w_1^{k-1}) \approx P(w_k | w_{k-(N-1)}^{k-1}) = P(w_k | w_{k-N+1}^{k-1})$$

Bigram version ( $N=2$ ):

$$P(w_k | w_1^{k-1}) \approx P(w_k | w_{k-1})$$

# Chain rule with Markov Assumption

Thus, the chain rule becomes (assuming a prefix of  $N$ ):

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k \mid w_{k-N+1}^{k-1})$$

Initialization (e.g.,  $N=3$ ):  $\langle s \rangle, \langle s \rangle, w_1, w_2, \dots$

$$P(w_2 \mid w_0^1) \equiv P(w_2 \mid \langle s \rangle, w_1)$$

$$P(w_1 \mid w_{-1}^0) \equiv P(w_1 \mid \langle s \rangle, \langle s \rangle)$$

Bigram version ( $N=2$ ):

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k \mid w_{k-1}) \quad \begin{array}{l} \text{where } P(w_1 \mid w_0) \equiv P(w_1) \\ \text{or } P(w_1 \mid w_0) \equiv P(w_1 \mid \langle s \rangle) \end{array}$$

# Estimating Bigram Probabilities

- The Maximum Likelihood Estimate (MLE)

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

- Bigram version:

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

# An Example

- Corpus:

<s> I am Sam </s>

<s> Sam I am </s>

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

- Notice that <s> and </s> permit to state that a word is the first/last one

$$P(I | <s>) = \frac{2}{3} = .67 \quad P(\text{Sam} | <s>) = \frac{1}{3} = .33 \quad P(\text{am} | I) = \frac{2}{3} = .67$$

$$P(</s> | \text{Sam}) = \frac{1}{2} = 0.5 \quad P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 \quad P(\text{do} | I) = \frac{1}{3} = .33$$

$$P(<s>) = 3/20$$

$$P(<s> I \text{ am Sam } </s>)$$

$$\cong P(<s>) P(I | <s>) P(\text{am} | I) P(\text{Sam} | \text{am}) P(</s> | \text{Sam})$$

$$\cong P(I | <s>) P(\text{am} | I) P(\text{Sam} | \text{am}) P(</s> | \text{Sam})$$

# Maximum Likelihood Estimates

- The maximum likelihood estimate of some parameter of a model  $M$  from a training set  $T$ 
  - Is the estimate that maximizes the likelihood of the training set  $T$  given the model  $M$
- Suppose the word “Chinese” occurs 400 times in a corpus of a million words (Brown corpus)
- What is the probability that a random word from some other text from the same distribution will be “Chinese”
- MLE estimate is  $400/1000000 = .004$ 
  - This may be a bad estimate for some other corpus
- But it is the **estimate** that makes it **most likely** that “Chinese” will occur 400 times in a million word corpus.



# Berkeley Restaurant Project

## Sentences

- *can you tell me about any good cantonese restaurants close by.*
- *mid priced thai food is what I'm looking for.*
- *tell me about chez panisse.*
- *can you give me a listing of the kinds of food that are available.*
- *I'm looking for a good place to eat breakfast.*
- *when is caffe venezia open during the day.*
- ...

# Bigram Counts

- Out of 9222 sentences
  - Eg. “I want” occurred 827 times

	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

# Bigram Probabilities

- Divide bigram counts by prefix unigram counts to get conditional probabilities.

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

	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

E.g.:  $P(\text{want} | i) = C(i \text{ want}) / C(i) = 827 / 2533 = 0.33$

# Bigram Estimates of Sentence Probabilities

$$\begin{aligned} P(< s > \text{ I want English food } < / s >) &\cong \\ P(\text{I} \mid < s >) &\cdot \\ P(\text{want} \mid \text{I}) &\cdot \\ P(\text{english} \mid \text{want}) &\cdot \\ P(\text{food} \mid \text{english}) &\cdot \\ P(< / s > \mid \text{food}) &= \\ &= 0.000031 \end{aligned}$$

# Kinds of Knowledge

- As crude as they are,  $N$ -gram probabilities capture a range of interesting facts about language.

- $P(\text{english}|\text{want}) = .0011$

- $P(\text{chinese}|\text{want}) = .0065$

World knowledge

- $P(\text{to}|\text{want}) = .66$

- $P(\text{eat} | \text{to}) = .28$

- $P(\text{food} | \text{to}) = 0$

- $P(\text{want} | \text{spend}) = 0$

Syntax

- $P(i | \langle s \rangle) = .25$

Discourse

# Shannon's Method

- Assigning probabilities to sentences is all well and good, but it's not terribly illuminating. A more interesting task is to turn the model around and use it to **generate** random sentences that are *like* the sentences from which the model was derived.
- Generally attributed to Claude Shannon.



# Shannon's Method

- Sample a random bigram ( $\langle s \rangle, w$ ) according to its probability  $P(w \mid \langle s \rangle)$
- Now sample a random bigram  $(w, x)$  according to its probability  $P(x \mid w)$ 
  - Where the prefix  $w$  matches the suffix of the first.
- And so on until we randomly choose a  $(y, \langle /s \rangle)$
- Then, string the words together
- $\langle s \rangle$  I

I want

want to

to eat

eat Chinese

Chinese food

food  $\langle /s \rangle$

# Shakespeare

Unigram	<ul style="list-style-type: none"> <li>• To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have</li> <li>• Every enter now severally so, let</li> <li>• Hill he late speaks; or! a more to leg less first you enter</li> <li>• Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like</li> </ul>
Bigram	<ul style="list-style-type: none"> <li>• What means, sir. I confess she? then all sorts, he is trim, captain.</li> <li>• Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.</li> <li>• What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?</li> <li>• Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt</li> </ul>
Trigram	<ul style="list-style-type: none"> <li>• Sweet prince, Falstaff shall die. Harry of Monmouth's grave.</li> <li>• This shall forbid it should be branded, if renown made it empty.</li> <li>• Indeed the duke; and had a very good friend.</li> <li>• Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.</li> </ul>
Quadrigram	<ul style="list-style-type: none"> <li>• King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;</li> <li>• Will you not tell me who I am?</li> <li>• It cannot be but so.</li> <li>• Indeed the short and the long. Marry, 'tis a noble Lepidus.</li> </ul>



# Shakespeare as a Corpus

- $N=884\,647$  tokens,  $V=29\,066$  types
- Shakespeare actually produced 300 000 bigram types out of  $V^2= 844$  million possible bigram types...
  - So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - This is the biggest problem in language modeling; we'll come back to it.
- Higher  $N$ -grams look better
  - but... text generated by quadrigrams looks like Shakespeare because it *is* Shakespeare

# The Wall Street Journal is Not Shakespeare

*unigram:* Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

*bigram:* Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

*trigram:* They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

- $N$ -gram trained on Shakespeare is not a good Language Model for representing the Wall Street Journal!
- A typical issue for any Language Model...

# Evaluation

- How do we know if our models are any good?
  - And in particular, how do we know if one model is better than another.
- Well, Shannon's game gives us an intuition.
  - The generated texts from the higher order models sure look better. That is, they sound more like the text the model was obtained from.
  - But what does that mean? Can we make that notion operational?

# Extrinsic Evaluation

- Best evaluation for a language model
  - Put model A into an application
    - For example, a speech recognizer
    - Metric: precision of the recognition
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models
- Time-consuming: can take days to run an experiment

# Intrinsic Evaluation

- Train parameters of our model on a **training set**.
- Look at the model performance on some new data
  - We want to know how our model performs on data we haven't seen
- So use a **test set**. A dataset which is different than our training set, but is drawn from the same source
- Then we need an **evaluation metric** to tell us how well our model is doing on the test set.
  - One such metric is **perplexity**

# Perplexity

- Perplexity is the probability of the test set (assigned by the language model), normalized by the number of words:
$$\text{PP}(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$
$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$
- Chain rule: 
$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$
- N-grams: 
$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-N+1}^{i-1})}}$$
- For bigrams: 
$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$
- Minimizing perplexity is the same as maximizing probability
- **The best language model is one that best predicts an unseen test set**

# Lower perplexity means a better model

- Training 38 million words, test 1.5 million words, WSJ

<i>N</i> -gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

- As the Shannon's game showed, the higher order model is the best one

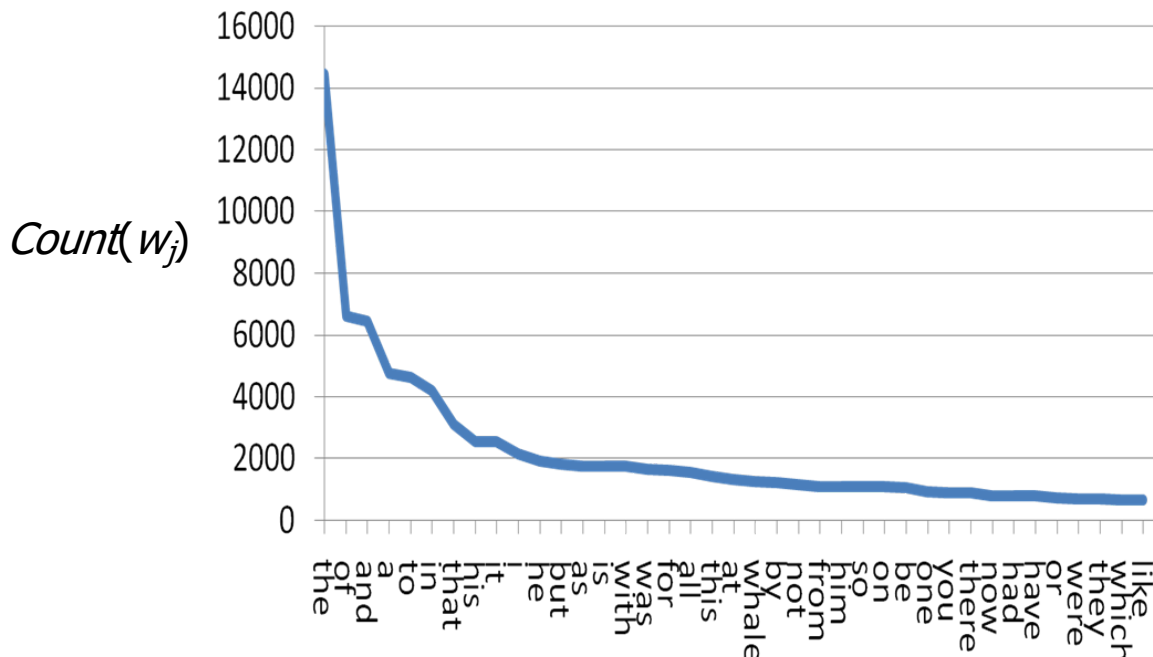
# Zero Counts

- Back to Shakespeare
  - Recall that Shakespeare produced 300 000 bigram types out of  $V^2 = 844$  million possible bigrams...
  - So, 99.96% of the possible bigrams were never seen (have zero entries in the table)
  - Does that mean that any sentence that contains one of those bigrams should have a probability of 0?
- It depends...



# Zero Counts

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just **rare** events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).
- Zipf's Law (long tail phenomenon):
  - "given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table."

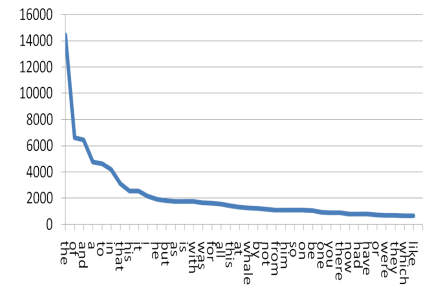


$$Count(w_j) = \frac{K}{j^{1-\theta}}$$
$$0 < \theta < 1$$

word rank  $j$

# Zero Counts

- Some of those zeros are really zeros...
  - Things that really can't or shouldn't happen.
- On the other hand, some of them are just rare events.
  - If the training corpus had been a little bigger they would have had a count (probably a count of 1!).
- Zipf's Law (long tail phenomenon):
  - "given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table."
  - A small number of words occur with high frequency
  - A large number of words occur with low frequency
  - You can quickly collect statistics on the high frequency events
  - You might have to wait an arbitrarily long time to get valid statistics on low frequency events
- Result:
  - Our estimates are sparse! We have no counts at all for the vast bulk of things we want to estimate!
- Answer:
  - Estimate the likelihood of unseen (zero count)  $N$ -grams!



# Laplace Smoothing

- Also called add-one smoothing
- Just add one to all the counts!
- Very simple



- MLE estimate for a unique word:  $P(w_i) = \frac{c_i}{N}$
- Laplace estimate for a unique word:  $P^*(w_i) = \frac{c_i + 1}{N + V}$
- Smoothed counts:
  - I.e.: using MLE on them, I obtain again the  $P^*(w_i)$ :
$$c_i^* = (c_i + 1) \frac{N}{N + V}$$
$$P^*(w_i) = \frac{c_i^*}{N}$$

# Laplace-Smoothed Bigram Counts

	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

- Add 1 to bigram counts

# Laplace-Smoothed Bigram Probabilities

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

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

- New, smoothed bigram probabilities

# Smoothed Counts

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

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

- Using MLE on that, I obtain again the Laplace estimation

# Big Change to the Counts!

- $C(\text{want to})$  went from 608 to 238!
- $P(\text{to} \mid \text{want})$  from .66 to .26!
- Define the *discount*  $d = c^*/c$ 
  - $d$  for “chinese food” =  $8.2/82 = 0.1$ 
    - A 10x reduction!!!
  - So in general, Laplace is a blunt instrument
  - Could use more fine-grained method (add-k)
- But Laplace smoothing not used for  $N$ -grams, as we have much better methods
- Despite its flaws Laplace (add-k) is however still used to smooth other probabilistic models in NLP, especially
  - For pilot studies
  - in domains where the number of zeros isn't so huge.

# Good-Turing Intuition

- Laplace smoothing is not the best one
  - Counts change too much
- An intuition used by many smoothing algorithms...
  - Good-Turing, Kneser-Ney, Witten-Bell
- ... is to use the count of things we've seen *once* to help estimate the count of things we've *never* seen
- In general, Good-Turing estimates probability of word types occurring  $c$  times with *adjusted* MLE estimation of word types occurring  $c+1$  times



# Good-Turing: example

- **Notation:**
  - $N_c$  is the frequency-of-frequency- $c$   
(how many word types occur  $c$  times)
  - $N$  is the total number of tokens
- The probability of *the set* of word types occurring  $c$  times is:

$$P_c = \frac{C(\text{word types occurring } c \text{ times})}{N} = \frac{N_c \cdot c}{N}$$

- According to the Good-Turing intuition:

$$P_c = \frac{N_{c+1} \cdot (c + 1)}{N}$$

- Thus:  $\frac{N_c \cdot c^*}{N} = \frac{N_{c+1} \cdot (c+1)}{N} \rightarrow c^* = (c + 1) \cdot \frac{N_{c+1}}{N_c}$
- The probability of *any* word type occurring  $c$  times is:  $P_c^* = \frac{c^*}{N}$

# Good-Turing: example

- Imagine you are fishing
  - There are 8 species: carp, perch, whitefish, trout, salmon, eel, catfish, bass
- You have caught
  - 10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel  
= 18 fish
- Thus:
  - $N_0=2$ ; Number of fish species never seen is 2: catfish, bass
  - $N_1=3$ ; Number of fish species seen once is 3: trout, salmon, eel
  - $N_2=1$ ; Number of fish seen twice is 1: whitefish
  - $N_3=1$ ; Number of fish seen three times is 1: perch
  - $N_4, \dots, N_9=0$ ; no fish found 4...9 times
  - $N_{10}=1$ ; Number of fish seen ten times is 1: carp

# Good-Turing: example

- How likely is it that the next fish caught is from a new species: one not seen in our previous catch (catfish, bass)?
- To estimate total number of unseen species ( $c = 0$ ); for example for bass:

$$c^*(\text{bass}) = (0 + 1) \cdot \frac{N_{0+1}}{N_0} = \frac{3}{2}; \quad P_c^*(\text{bass}) = \frac{3/2}{18} = \frac{1}{12}$$

- Counts and probabilities of seen species must be discounted; for example trout ( $c=1$ ):

$$c^*(\text{trout}) = (1 + 1) \cdot \frac{N_{1+1}}{N_1} = \frac{2}{3}; \quad P_c^*(\text{trout}) = \frac{2/3}{18} = \frac{1}{27}$$

- Notice that, without discounting,  $P_c(\text{trout}) = \frac{1}{18}$

# Bigram Frequencies of Frequencies and GT Re-estimates

AP Newswire			Berkeley Restaurant—		
$c$ (MLE)	$N_c$	$c^*$ (GT)	$c$ (MLE)	$N_c$	$c^*$ (GT)
0	74,671,100,000	0.0000270	0	2,081,496	0.002553
1	2,018,046	0.446	1	5315	0.533960
2	449,721	1.26	2	1419	1.357294
3	188,933	2.24	3	642	2.373832
4	105,668	3.24	4	381	4.081365
5	68,379	4.22	5	311	3.781350
6	48,190	5.19	6	196	4.500000

$c$  and  $c^*$  are not that different...

# Complications

- If  $N_{c+1}=0$ ? To avoid this, replace all the zeros using a linear regression curve:

$$\log(N_c) = a + b \cdot \log(c) \quad \text{Holds for words, not fish!}$$

- In practice, discounted  $c^*$  are not used for large counts ( $c > k$  for some  $k$ ):

$$c^* = c \text{ for } c > k$$

- That complicates how  $c^*$  are recalculated, making it (Katz, '87):

$$c^* = \frac{(c+1) \frac{N_{c+1}}{N_c} - c \frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}}, \text{ for } 1 \leq c \leq k.$$

- Also: we assume singleton counts  $c=1$  are unreliable, so treat  $N$ -grams with count of 1 as if they were count=0
- Good-Turing is usually used in combination with *backoff* and *interpolation*

# Backoff

- Another really useful source of knowledge
  - Do not “invent” a value for an unseen  $N$ -gram
  - Backoff to  $(N-1)$ gram
  - E.g., if we are estimating:
    - Trigram  $P(z \mid x, y)$
    - But  $count(xyz)=0$
  - Try to use info from:
    - Bigram  $P(z \mid y)$
    - If  $count(yz)$  is zero
  - Try to use info from:
    - Unigram  $P(z)$
- How to combine trigram, bigram, unigram?

# Backoff Vs. Interpolation

- **Backoff:** use trigram if you have it, otherwise bigram, otherwise unigram
- **Interpolation:** mix all three

# Interpolation

- Simple interpolation:

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

$$\sum_i \lambda_i(w_{n-2}^{n-1}) = 1$$



# How to Set the Lambdas?

- Use a **held-out, or development**, corpus
- Choose lambdas which maximize the probability of some held-out data:
  - Set the initial  $N$ -gram probabilities, using the training corpus
  - Then search for lambda values that, when plugged into previous equation, give largest probability for held-out set
    - Compute prob. of the held-out set (or Perplexity)
  - Can use Expectation Maximization (EM) iterative algorithm to do this search

# Katz Backoff

- Integrates Good-Turing:

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

- E.g. for trigrams  $x, y, z$ :

$$P_{\text{katz}}(z | y, x) = \begin{cases} P^*(z | y, x); & \text{if } C(x, y, z) > 0 \\ \alpha(x, y) P_{\text{katz}}(z | y); & \text{otherwise} \end{cases}$$

$$P_{\text{katz}}(z | y) = \begin{cases} P^*(z | y); & \text{if } C(y, z) > 0 \\ \alpha(y) P^*(z); & \text{otherwise} \end{cases}$$

# Katz Backoff

- Calculating  $\alpha$ :

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{\{w_n: C(w_{n-N+1}^n) > 0\}} P^*(w_n | w_{n-N+1}^{n-1})$$

$$\alpha(w_{n-N+1}^{n-1}) = \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{\{w_n: C(w_{n-N+1}^n) = 0\}} P_{katz}(w_n | w_{n-N+1}^{n-1})}$$

- E.g. for trigrams  $x, y, z$ :

$$\beta(x, y) = 1 - \sum_{\{z: C(x, y, z) > 0\}} P^*(z | y, x)$$

$$\alpha(x, y) = \frac{\beta(x, y)}{\sum_{\{z: C(x, y, z) = 0\}} P_{katz}(z | y)}$$

Show notes...

# Why discounts $P^*$ and $\alpha$ ?

- We need to add  $P^*$  and  $\alpha$  constants, so that the resulting model is still a probability distribution
  - Adding probability mass to unseen items, we need discount
  - The  $\alpha$  is used to ensure that the probability mass from all the lower order *N-grams* sums up to exactly the amount that we saved by discounting the higher-order *N-grams*.

# Backoff Smoothed Bigram Probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.0014	0.326	0.00248	0.00355	0.000205	0.0017	0.00073	0.000489
want	0.00134	0.00152	0.656	0.000483	0.00455	0.00455	0.00384	0.000483
to	0.000512	0.00152	0.00165	0.284	0.000512	0.0017	0.00175	0.0873
eat	0.00101	0.00152	0.00166	0.00189	0.0214	0.00166	0.0563	0.000585
chinese	0.00283	0.00152	0.00248	0.00189	0.000205	0.519	0.00283	0.000585
food	0.0137	0.00152	0.0137	0.00189	0.000409	0.00366	0.00073	0.000585
lunch	0.00363	0.00152	0.00248	0.00189	0.000205	0.00131	0.00073	0.000585
spend	0.00161	0.00152	0.00161	0.00189	0.000205	0.0017	0.00073	0.000585

# OOV words: <UNK> word

- Usually, a language model is learned starting from:
  - A collection of documents
  - A vocabulary (i.e., a *lexicon*) containing all the words we want to recognize (generated from the whole document collection)
- Then, the document collection is split into two parts:
  - A training set, used for learning the language model
  - A test set, used for testing the language model (e.g., perplexity)
- At training-time, the words not found into the training set are managed using smoothing/interpolation/back-off
  - Because these words are still into the vocabulary
- At test time, using the test set, no new words can appear
- At run-time, two approaches...

# OOV words: <UNK> word

## Closed vocabulary: no new words

- Assumption: at run-time (i.e., applying the language model on new documents), new words cannot appear

## Open vocabulary: we model unknown words by adding a pseudo-word called <UNK>

- We assume that, at run-time, new words may appear
- **Out Of Vocabulary** = OOV words
- We don't use smoothing/interpolation/back-off for these
- Instead: we create an unknown word token <UNK>
  - Training of <UNK> probabilities:
    - Define a fixed lexicon L (usually, contains the most common words)
    - Any training word not in L (i.e., rare word), is changed to <UNK>
    - Now we calculate the probability of <UNK> like a normal word
  - At run-time:
    - In text input: Use <UNK> probabilities for any word not in training

# A note on vocabulary

- **First approach: the vocabulary is pre-defined**
  - Hypothesis: the corpus should not contain words not belonging to the vocabulary
  - But you can find words into the vocabulary that are not into the corpus
  - So, smoothing gives probability to these words
- **Second approach: the vocabulary is created starting from the corpus**
  - So, no words with a count=0 can be found into the vocabulary
  - Smoothing is still useful to give better probability to rare words
- **For both approaches, at run-time you can assume open vocabulary or closed vocabulary**



# Practical Issues

- We do everything in log space

- Avoid underflow

- (also adding is faster than multiplying)

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

- How to store N-grams: the ARPA format

- File format for back-off models

- Es.: trigram

unigram:	$\log P^*(w_i)$	$w_i$	$\log \alpha(w_i)$
----------	-----------------	-------	--------------------

bigram:	$\log P^*(w_i   w_{i-1})$	$w_{i-1} w_i$	$\log \alpha(w_i, w_{i-1})$
---------	---------------------------	---------------	-----------------------------

trigram:	$\log P^*(w_i   w_{i-2}, w_{i-1})$	$w_{i-2}, w_{i-1}, w_i$	
----------	------------------------------------	-------------------------	--

# ARPA Format

```
\data\  
ngram 1=1447  
ngram 2=9420  
ngram 3=5201
```

```
\1-grams:\br/>-0.8679678      </s>  
-99             <s>                -1.068532  
-4.743076      chow-fun          -0.1943932  
-4.266155      fries             -0.5432462  
-3.175167      thursday          -0.7510199  
-1.776296      want              -1.04292  
...
```

```
\2-grams:\br/>-0.6077676      <s>      i                -0.6257131  
-0.4861297      i        want          0.0425899  
-2.832415       to      drink          -0.06423882  
-0.5469525      to      eat            -0.008193135  
-0.09403705     today   </s>  
...
```

```
\3-grams:\br/>-2.579416       <s>      i                prefer  
-1.148009       <s>      about          fifteen  
-0.4120701      to      go             to  
-0.3735807      me      a              list  
-0.260361       at      jupiter         </s>  
-0.260361       a      malaysian      restaurant  
...  
\end\
```