

Basic Text Processing

Regular Expressions

Regular Expressions: Negation in Disjunction

- Negations `[^Ss]`
 - Carat means negation only when first in []

Pattern	Matches	
<code>[^A-Z]</code>	Not an upper case letter	O <u>y</u> fn pripetchik
<code>[^Ss]</code>	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
<code>[^e^]</code>	Neither e nor ^	Look h <u>e</u> re
<code>a^b</code>	The pattern a carat b	Look up <u>a^b</u> now

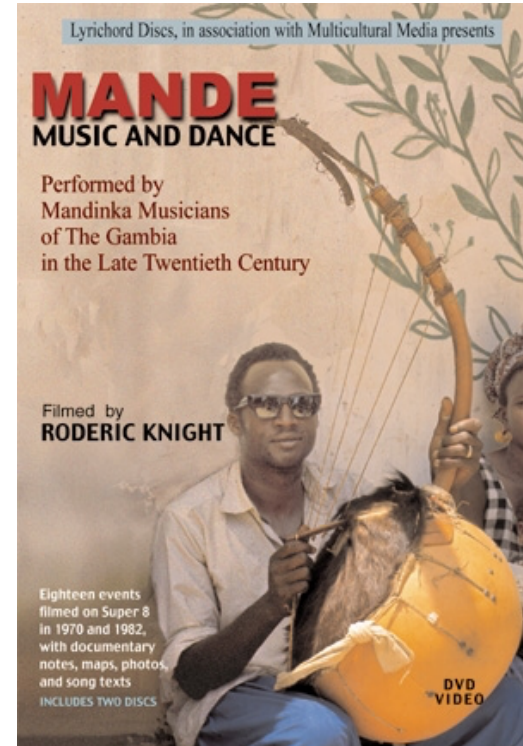
Role of Regular Expressions

- Regular expressions play a surprisingly large role
 - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
 - But regular expressions are used as features in the classifiers
 - Can be very useful in capturing generalizations

Hearst Patterns

- In her seminal 1992 paper, entitled Automatic Acquisition of Hyponyms from Large Text Corpora, Marti Hearst defined a set of patterns for identifying *hypernym/hyponym* relations (also known as *is-a*)

The bow lute, **such as** the Bambara ndang, is plucked and has an individual curved neck for each string



Hearst's Patterns for extracting IS-A relations

Hearst pattern	Example occurrences
X and other Y	...temples, treasuries, and other important civic buildings.
X or other Y	Bruises, wounds, broken bones or other injuries...
Y such as X	The bow lute, such as the Bambara ndang...
Such Y as X	... such authors as Herrick, Goldsmith, and Shakespeare.
Y including X	...common-law countries, including Canada and England...
Y , especially X	European countries, especially France, England, and Spain...

Unix utility: grep

- `zcat * | grep "such as" | more`

management consultants **such as** McKinsey and CSC Index.
social evils **such as** prostitution, drug addiction and HIV
new set of potentially lucrative services, **such as** movies on demand
the students use canned chicken broth such as Swanson's
in treating medical conditions such as psoriasis, seasonal

- `zcat * | grep "and other" | more`

sanitation problems, the endless red tape **and other** difficulties
Court records **and other** documents show that Angela Tene
providing dominoes, card games **and other** recreation to help asylum
malls, swap meets, colleges, barber shops and other popular haunts

Basic Text Processing

Word tokenization

Text Normalization

- Every NLP task needs to do text normalization:
 1. Segmenting/tokenizing words in running text
 2. Normalizing word formats
 3. Segmenting sentences in running text

How many words?

- I do uh main- mainly business data processing
 - Fragments, filled pauses
- Seuss's **cat** in the hat is different from other **cats**!
 - **Lemma**: same stem, part of speech, rough word sense
 - **cat** and **cats** = same lemma
 - **Wordform**: the full inflected surface form
 - **cat** and **cats** = different wordforms

How many words?

they lay back on the San Francisco grass and looked at the stars and their

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
 - 15 tokens (or 14)
 - 13 types (or 12) (or 11?)

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{1/2})$

	Tokens = N	Types = $ V $
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

1945 A	25 Aaron
72 AARON	6 Abate
19 ABBESS	1 Abates
5 ABBOT	5 Abbess
	6 Abbey
...	3 Abbot
...
	...

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

```
THE  
SONNETS  
by  
William  
Shakespeare  
From  
fairest  
creatures  
We  
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

A

A

A

A

A

A

A

A

A

...

More counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954 d
```

What happened here?

Issues in Tokenization

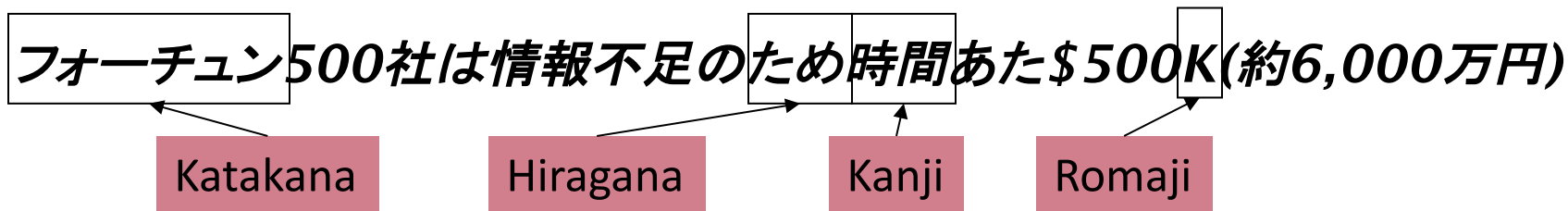
- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → one token or two?
- m.p.h., PhD. → ??

Tokenization: language issues

- French
 - *L'ensemble* → one token or two?
 - *L ? L' ? Le ?*
 - Want *l'ensemble* to match with *un ensemble*
- German noun compounds are not segmented
 - *Lebensversicherungsgesellschaftsangestellter*
 - 'life insurance company employee'
 - German information retrieval needs **compound splitter**

Tokenization: language issues

- Chinese and Japanese no spaces between words:
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
 - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
 - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
 - Characters are generally 1 syllable and 1 morpheme.
 - Average word is 2.4 characters long.
- Standard baseline segmentation algorithm:
 - Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation Algorithm

- Given a wordlist of Chinese, and a string.
 - 1) Start a pointer at the beginning of the string
 - 2) Find the longest word in dictionary that matches the string starting at pointer
 - 3) Move the pointer over the word in string
 - 4) Go to 2

Max-match segmentation illustration

- Thecatinthehat the cat in the hat
- Thetabledownthere the table down there
 theta bled own there
- Doesn't generally work in English!
- But works astonishingly well in Chinese
 - 莎拉波娃现在居住在美国东南部的佛罗里达。
 - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic segmentation algorithms even better

Basic Text Processing

Word tokenization

Basic Text Processing

Word Normalization and
Stemming

Normalization

- Need to “normalize” terms
 - Information Retrieval: indexed text & query terms must have same form.
 - We want to match ***U.S.A.*** and ***USA***
- We implicitly define equivalence classes of terms
 - e.g., deleting periods in a term
- Alternative: asymmetric expansion:
 - Enter: ***window*** Search: ***window, windows***
 - Enter: ***windows*** Search: ***Windows, windows, window***
 - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Case folding

- Applications like IR: reduce all letters to lower case
 - Since users tend to use lower case
 - Possible exception: upper case in mid-sentence?
 - e.g., *General Motors*
 - *Fed* vs. *fed*
 - *SAIL* vs. *sail*
- For sentiment analysis, MT, Information extraction
 - Case is helpful (**US** versus *us* is important)

Lemmatization

- Reduce inflections or variant forms to base form
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization: have to find correct dictionary headword form
- Machine translation
 - Spanish **quiero** ('I want'), **quieres** ('you want') same lemma as **querer** 'want'

Morphology

- **Morphemes:**
 - The small meaningful units that make up words
 - **Stems:** The core meaning-bearing units
 - **Affixes:** Bits and pieces that adhere to stems
 - Often with grammatical functions

Stemming

- Reduce terms to their stems in information retrieval
- *Stemming* is crude chopping of affixes
 - language dependent
 - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

*for example compressed
and compression are both
accepted as equivalent to
compress.*



for exampl compress and
compress ar both accept
as equival to compress

Porter's algorithm

The most common English stemmer

Step 1a

sses → ss	caresses → caress
ies → i	ponies → poni
ss → ss	caress → caress
s → ∅	cats → cat

Step 2 (for long stems)

ational → ate	relational → relate
izer → ize	digitizer → digitize
ator → ate	operator → operate
...	

Step 1b

(*v*)ing → ∅	walking → walk
	sing → sing
(*v*)ed → ∅	plastered → plaster
...	

Step 3 (for longer stems)

al → ∅	revival → reviv
able → ∅	adjustable → adjust
ate → ∅	activate → activ
...	

Viewing morphology in a corpus

Why only strip -ing if there is a vowel?

(*v*)ing → ∅ walking → walk
 sing → sing

Viewing morphology in a corpus

Why only strip –ing if there is a vowel?

`(*v*)ing` → `∅` `walking` → `walk`
 `sing` → `sing`

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort -nr
```

1312 King	548 being
548 being	541 nothing
541 nothing	152 something
388 king	145 coming
375 bring	130 morning
358 thing	122 having
307 ring	120 living
152 something	117 loving
145 coming	116 Being
130 morning	102 going

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort -nr
```

Dealing with complex morphology is sometimes necessary

- Some languages requires complex morpheme segmentation
 - Turkish
 - **Uygarlastiramadiklarimizdanmissinizcasina**
 - `(behaving) as if you are among those whom we could not civilize’
 - **Uygar** `civilized’ + **las** `become’
 - + **tir** `cause’ + **ama** `not able’
 - + **dik** `past’ + **lar** `plural’
 - + **imiz** `p1pl’ + **dan** `abl’
 - + **mis** `past’ + **siniz** `2pl’ + **casina** `as if’

Basic Text Processing

Word Normalization and
Stemming

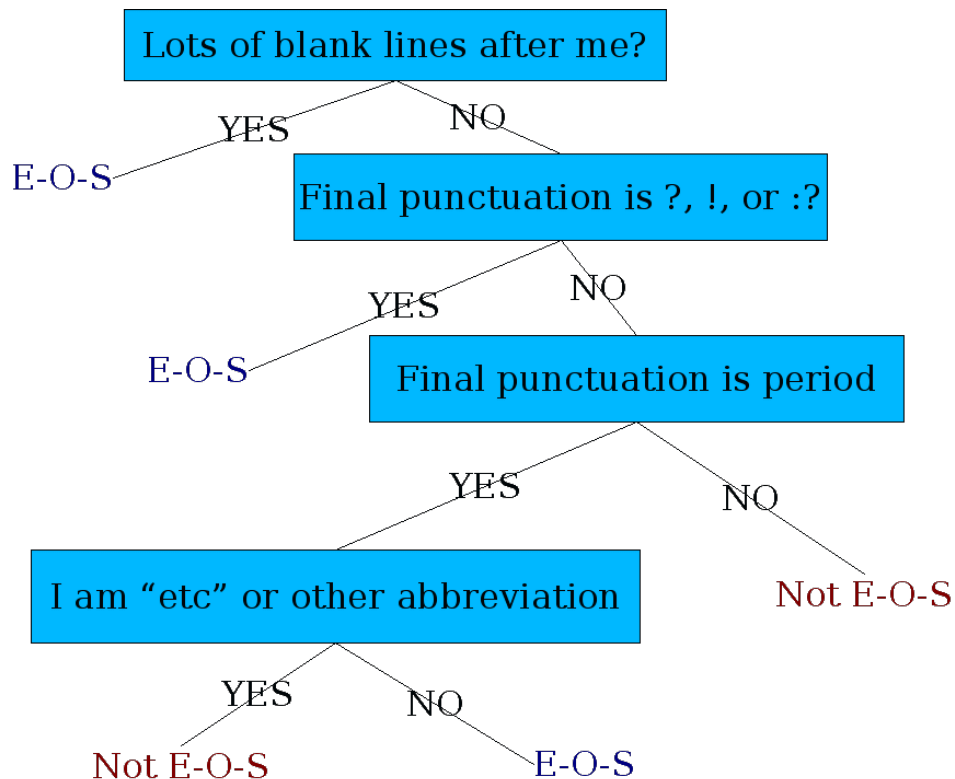
Basic Text Processing

Sentence Segmentation
and Decision Trees

Sentence Segmentation

- !, ? are relatively unambiguous
- Period “.” is quite ambiguous
 - Sentence boundary
 - Abbreviations like Inc. or Dr.
 - Numbers like .02% or 4.3
- Build a binary classifier
 - Looks at a “.”
 - Decides EndOfSentence/NotEndOfSentence
 - Classifiers: hand-written rules, regular expressions, or machine-learning

Determining if a word is end-of-sentence: a Decision Tree



More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number
- Numeric features
 - Length of word with “.”
 - Probability(word with “.” occurs at end-of-s)
 - Probability(word after “.” occurs at beginning-of-s)

Implementing Decision Trees

- A decision tree is just an if-then-else statement
- The interesting research is choosing the features
- Setting up the structure is often too hard to do by hand
 - Hand-building only possible for very simple features, domains
 - For numeric features, it's too hard to pick each threshold
 - Instead, structure usually learned by machine learning from a training corpus

Decision Trees and other classifiers

- We can think of the questions in a decision tree
- As features that could be exploited by any kind of classifier
 - Logistic regression
 - SVM
 - Neural Nets
 - etc.

Language Modeling

Introduction to N-grams

Probabilistic Language Models

- Today's goal: assign a probability to a sentence

- Machine Translation:

- $P(\text{high winds tonite}) > P(\text{large winds tonite})$

- Spelling Correction

- The office is about fifteen **minuets** from my house

- $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$

- Speech Recognition

- $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$

- + Summarization, question-answering, etc., etc.!!

Why?

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \dots w_{n-1})$ is called a **language model**.

- Better: **the grammar** But **language model** or **LM** is standard

How to compute $P(W)$

- How to compute this joint probability:
 - $P(\text{its, water, is, so, transparent, that})$
- Intuition: let's rely on the Chain Rule of Probability

Reminder: The Chain Rule

- Recall the definition of conditional probabilities

$$p(\mathbf{B} | \mathbf{A}) = \mathbf{P}(\mathbf{A}, \mathbf{B}) / \mathbf{P}(\mathbf{A}) \quad \text{Rewriting: } \mathbf{P}(\mathbf{A}, \mathbf{B}) = \mathbf{P}(\mathbf{A})\mathbf{P}(\mathbf{B} | \mathbf{A})$$

- More variables:

$$\mathbf{P}(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}) = \mathbf{P}(\mathbf{A})\mathbf{P}(\mathbf{B} | \mathbf{A})\mathbf{P}(\mathbf{C} | \mathbf{A}, \mathbf{B})\mathbf{P}(\mathbf{D} | \mathbf{A}, \mathbf{B}, \mathbf{C})$$

- The Chain Rule in General

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

The Chain Rule applied to compute joint probability of words in sentence

$$P(w_1 w_2 \dots w_n) = \prod_i P(w_i \mid w_1 w_2 \dots w_{i-1})$$

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

$P(\text{its}) \times P(\text{water} \mid \text{its}) \times P(\text{is} \mid \text{its water})$

$\times P(\text{so} \mid \text{its water is}) \times P(\text{transparent} \mid \text{its water is so})$

How to estimate these probabilities

- Could we just count and divide?

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

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption



Andrei Markov

- Simplifying assumption:

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

- Or maybe

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

Markov Assumption

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i \mid w_{i-k} \dots w_{i-1})$$

- In other words, we approximate each component in the product

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-k} \dots w_{i-1})$$

Simplest case: Unigram model

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

Some automatically generated sentences from a unigram model

fifth, an, of, futures, the, an, incorporated, a,
a, the, inflation, most, dollars, quarter, in, is,
mass

thrift, did, eighty, said, hard, 'm, july, bullish

that, or, limited, the

Bigram model

- Condition on the previous word:

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$

texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached
this, would, be, a, record, november

N-gram models

- We can extend to trigrams, 4-grams, 5-grams
- In general this is an insufficient model of language
 - because language has **long-distance dependencies**:
“The computer(s) which I had just put into the machine room on the fifth floor is (are) crashing.”
- But we can often get away with N-gram models

Language Modeling

Introduction to N-grams

Language Modeling

Estimating N-gram
Probabilities

Estimating bigram probabilities

- The Maximum Likelihood Estimate

$$P(w_i | w_{i-1}) = \frac{\textit{count}(w_{i-1}, w_i)}{\textit{count}(w_{i-1})}$$

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

An example

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>

<s> Sam I am </s>

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

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

$$P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\text{</s>} | \text{Sam}) = \frac{1}{2} = 0.5$$

$$P(\text{Sam} | \text{am}) = \frac{1}{2} = .5$$

$$P(\text{do} | \text{I}) = \frac{1}{3} = .33$$

More examples:

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

Raw bigram counts

- Out of 9222 sentences

	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

Raw bigram probabilities

- Normalize by unigrams:

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

- Result:

	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

Bigram estimates of sentence probabilities

$P(< s> \text{ I want english food } < /s>) =$

$P(\text{I} | < s>)$

$\times P(\text{want} | \text{I})$

$\times P(\text{english} | \text{want})$

$\times P(\text{food} | \text{english})$

$\times P(< /s> | \text{food})$

$= .000031$

What kinds of knowledge?

- $P(\text{english} | \text{want}) = .0011$
- $P(\text{chinese} | \text{want}) = .0065$
- $P(\text{to} | \text{want}) = .66$
- $P(\text{eat} | \text{to}) = .28$
- $P(\text{food} | \text{to}) = 0$
- $P(\text{want} | \text{spend}) = 0$
- $P(i | \langle s \rangle) = .25$

Practical Issues

- We do everything in log space
 - Avoid underflow
 - (also adding is faster than multiplying)

$$\log(p_1 \times p_2 \times p_3 \times p_4) = \log p_1 + \log p_2 + \log p_3 + \log p_4$$

Language Modeling Toolkits

- SRILM
 - <http://www.speech.sri.com/projects/srilm/>
- KenLM
 - <https://kheafield.com/code/kenlm/>

Google N-Gram Release, August 2006

AUG

3

All Our N-gram are Belong to You

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects,

...

That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

Google N-Gram Release

- serve as the incoming 92
- serve as the incubator 99
- serve as the independent 794
- serve as the index 223
- serve as the indication 72
- serve as the indicator 120
- serve as the indicators 45
- serve as the indispensable 111
- serve as the indispensable 40
- serve as the individual 234

<http://googleresearch.blogspot.com/2006/08/all-our-n-gram-are-belong-to-you.html>

Google Book N-grams

- <http://ngrams.googlelabs.com/>

Language Modeling

Estimating N-gram
Probabilities

Language Modeling

Evaluation and
Perplexity

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
 - Assign higher probability to “real” or “frequently observed” sentences
 - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
 - A **test set** is an unseen dataset that is different from our training set, totally unused.
 - An **evaluation metric** tells us how well our model does on the test set.

Training on the test set

- We can't allow test sentences into the training set
- We will assign it an artificially high probability when we set it in the test set
- “Training on the test set”
- Bad science!
- And violates the honor code

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
 - Put each model in a task
 - spelling corrector, speech recognizer, MT system
 - Run the task, get an accuracy for A and for B
 - How many misspelled words corrected properly
 - How many words translated correctly
 - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
 - Time-consuming; can take days or weeks
- So
 - Sometimes use **intrinsic** evaluation: **perplexity**
 - Bad approximation
 - unless the test data looks **just** like the training data
 - So **generally only useful in pilot experiments**
 - But is helpful to think about.

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 _____

- Unigrams are terrible at this game. (Why?)

- A better model of a text

- is one which assigns a higher probability to the word that actually occurs



mushrooms 0.1
pepperoni 0.1
anchovies 0.01
....
fried rice 0.0001
....
and 1e-100

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}}$$

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

Chain rule:

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

For bigrams:

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

Minimizing perplexity is the same as maximizing probability

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$\begin{aligned}\text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{N}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10\end{aligned}$$

Lower perplexity = better model

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

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Language Modeling

Evaluation and
Perplexity

Language Modeling

Generalization and
zeros

The Shannon Visualization Method

- Choose a random bigram
($\langle s \rangle$, w) according to its probability
- Now choose a random bigram
(w , x) according to its probability
- And so on until we choose $\langle /s \rangle$
- Then string the words together

```
<s> I
    I want
      want to
        to eat
          eat Chinese
            Chinese food
              food </s>

I want to eat Chinese food
```

Approximating Shakespeare

1

gram

–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have

–Hill he late speaks; or! a more to leg less first you enter

2

gram

–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

–What means, sir. I confess she? then all sorts, he is trim, captain.

3

gram

–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

–This shall forbid it should be branded, if renown made it empty.

4

gram

–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;

–It cannot be but so.

Shakespeare as corpus

- $N=884,647$ tokens, $V=29,066$
- 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)
- Quadrigrams worse: What's coming out looks like Shakespeare because it *is* Shakespeare

The wall street journal is not shakespeare (no offense)

1
gram

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

2
gram

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

3
gram

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

Can you guess the author of these random 3-gram sentences?

- 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 gram Brazil on market conditions
- This shall forbid it should be branded, if renown made it empty.
- “You are uniformly charming!” cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

The perils of overfitting

- N-grams only work well for word prediction if the test corpus looks like the training corpus
 - In real life, it often doesn't
 - We need to train robust models that generalize!
 - One kind of generalization: Zeros!
 - Things that don't ever occur in the training set
 - But occur in the test set

Zeros

- Training set:
 - ... denied the allegations
 - ... denied the reports
 - ... denied the claims
 - ... denied the request
- Test set
 - ... denied the offer
 - ... denied the loan

$$P(\text{"offer"} \mid \text{denied the}) = 0$$

Zero probability bigrams

- Bigrams with zero probability
 - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

Language Modeling

Generalization and
zeros

Language Modeling

Smoothing: Add-one
(Laplace) smoothing

The intuition of smoothing (from Dan Klein)

- When we have sparse statistics:

$P(w \mid \text{denied the})$

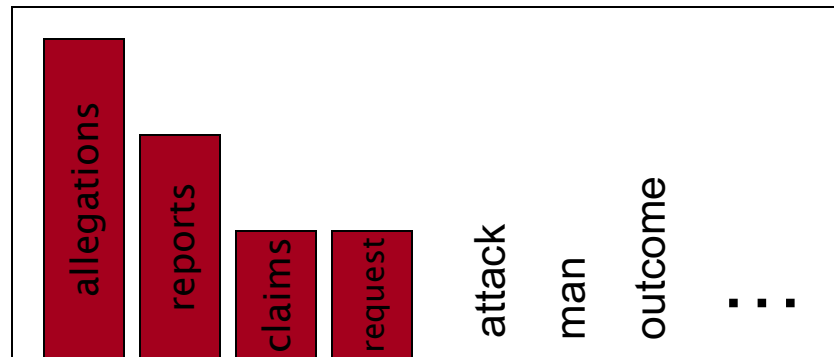
3 allegations

2 reports

1 claims

1 request

7 total



- Steal probability mass to generalize better

$P(w \mid \text{denied the})$

2.5 allegations

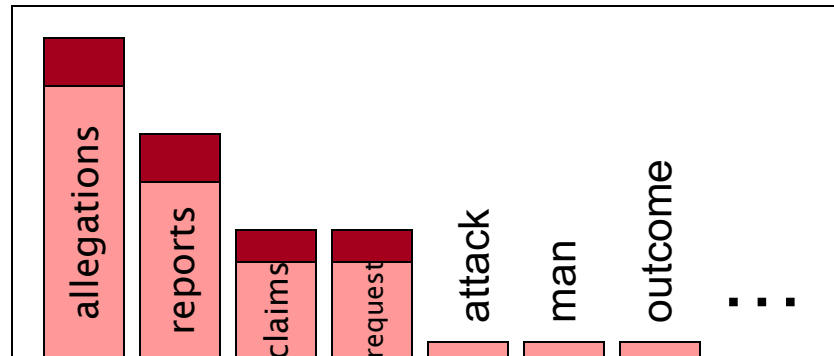
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Add-one estimation

- Also called Laplace smoothing
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

- MLE estimate:
$$P_{MLE}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

- Add-1 estimate:
$$P_{Add-1}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

Maximum Likelihood Estimates

- The maximum likelihood estimate
 - of some parameter of a model M from a training set T
 - maximizes the likelihood of the training set T given the model M
- Suppose the word “bagel” occurs 400 times in a corpus of a million words
- What is the probability that a random word from some other text will be “bagel”?
- MLE estimate is $400/1,000,000 = .0004$
- This may be a bad estimate for some other corpus
 - But it is the **estimate** that makes it **most likely** that “bagel” will occur 400 times in a million word corpus.

Berkeley Restaurant Corpus: 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

Laplace-smoothed bigrams

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

	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

Reconstituted counts

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

	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

Compare with raw 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

	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

Add-1 estimation is a blunt instrument

- So add-1 isn't used for N-grams:
 - We'll see better methods
- But add-1 is used to smooth other NLP models
 - For text classification
 - In domains where the number of zeros isn't so huge.

Language Modeling

Smoothing: Add-one
(Laplace) smoothing

Language Modeling

Interpolation, Backoff,
and Web-Scale LMs

Backoff and Interpolation

- Sometimes it helps to use **less** context
 - Condition on less context for contexts you haven't learned much about
- **Backoff:**
 - use trigram if you have good evidence,
 - otherwise bigram, otherwise unigram
- **Interpolation:**
 - mix unigram, bigram, trigram
- Interpolation works better

Linear Interpolation

- Simple interpolation

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

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

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

How to set the lambdas?

- Use a **held-out** corpus

Training Data

Held-Out
Data

Test
Data

- Choose λ s to maximize the probability of held-out data:
 - Fix the N-gram probabilities (on the training data)
 - Then search for λ s that give largest probability to held-out set:

$$\log P(w_1 \dots w_n \mid M(\lambda_1 \dots \lambda_k)) = \sum_i \log P_{M(\lambda_1 \dots \lambda_k)}(w_i \mid w_{i-1})$$

Unknown words: Open versus closed vocabulary tasks

- If we know all the words in advanced
 - Vocabulary V is fixed
 - Closed vocabulary task
- Often we don't know this
 - **Out Of Vocabulary** = OOV words
 - Open vocabulary task
- Instead: create an unknown word token <UNK>
 - Training of <UNK> probabilities
 - Create a fixed lexicon L of size V
 - At text normalization phase, any training word not in L changed to <UNK>
 - Now we train its probabilities like a normal word
 - At decoding time
 - If text input: Use UNK probabilities for any word not in training

Huge web-scale n-grams

- How to deal with, e.g., Google N-gram corpus
- Pruning
 - Only store N-grams with count $>$ threshold.
 - Remove singletons of higher-order n-grams
 - Entropy-based pruning
- Efficiency
 - Efficient data structures like tries
 - Bloom filters: approximate language models
 - Store words as indexes, not strings
 - Use Huffman coding to fit large numbers of words into two bytes
 - Quantize probabilities (4-8 bits instead of 8-byte float)

Smoothing for Web-scale N-grams

- “Stupid backoff” (Brants *et al.* 2007)
- No discounting, just use relative frequencies

$$S(w_i | w_{i-k+1}^{i-1}) = \begin{cases} \frac{\text{count}(w_{i-k+1}^i)}{\text{count}(w_{i-k+1}^{i-1})} & \text{if } \text{count}(w_{i-k+1}^i) > 0 \\ 0.4S(w_i | w_{i-k+2}^{i-1}) & \text{otherwise} \end{cases}$$

$$S(w_i) = \frac{\text{count}(w_i)}{N}$$

N-gram Smoothing Summary

- Add-1 smoothing:
 - OK for text categorization, not for language modeling
- The most commonly used method:
 - Extended Interpolated Kneser-Ney
- For very large N-grams like the Web:
 - Stupid backoff

Advanced Language Modeling

- Discriminative models:
 - choose n-gram weights to improve a task, not to fit the training set
- Parsing-based models
- Caching Models
 - Recently used words are more likely to appear

$$P_{CACHE}(w | history) = \lambda P(w_i | w_{i-2} w_{i-1}) + (1 - \lambda) \frac{c(w \in history)}{|history|}$$

- These perform very poorly for speech recognition (why?)

Language Modeling

Interpolation, Backoff,
and Web-Scale LMs

Language Modeling

Advanced:
Kneser-Ney Smoothing

Absolute discounting: just subtract a little from each count

- Suppose we wanted to subtract a little from a count of 4 to save probability mass for the zeros
- How much to subtract ?
- Church and Gale (1991)'s clever idea
- Divide up 22 million words of AP Newswire
 - Training and held-out set
 - for each bigram in the training set
 - see the actual count in the held-out set!

Bigram count in training	Bigram count in heldout set
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

Absolute Discounting Interpolation

- Save ourselves some time and just subtract 0.75 (or some d)!

discounted bigram

Interpolation weight

$$P_{\text{AbsoluteDiscounting}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1}) P(w)$$

unigram

- (Maybe keeping a couple extra values of d for counts 1 and 2)
- But should we really just use the regular unigram $P(w)$?

Kneser-Ney Smoothing I

- Better estimate for probabilities of lower-order unigrams!
 - Shannon game: *I can't see without my reading* Francisco ?
 - “Francisco” is more common than “glasses”
 - ... but “Francisco” always follows “San”
- The unigram is useful exactly when we haven't seen this bigram!
- Instead of $P(w)$: “How likely is w ”
- $P_{\text{continuation}}(w)$: “How likely is w to appear as a novel continuation?”
 - For each word, count the number of bigram types it completes
 - Every bigram type was a novel continuation the first time it was seen

$$P_{\text{CONTINUATION}}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

Kneser-Ney Smoothing II

- How many times does w appear as a novel continuation:

$$P_{CONTINUATION}(w) \propto |\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- Normalized by the total number of word bigram types

$$|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|$$

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

Kneser-Ney Smoothing III

- Alternative metaphor: The number of # of word types seen to precede w

$$|\{w_{i-1} : c(w_{i-1}, w) > 0\}|$$

- normalized by the # of words preceding all words:

$$P_{CONTINUATION}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{\sum_{w'} |\{w'_{i-1} : c(w'_{i-1}, w') > 0\}|}$$

- A frequent word (Francisco) occurring in only one context (San) will have a low continuation probability

Kneser-Ney Smoothing IV

$$P_{KN}(w_i | w_{i-1}) = \frac{\max(c(w_{i-1}, w_i) - d, 0)}{c(w_{i-1})} + \lambda(w_{i-1})P_{CONTINUATION}(w_i)$$

λ is a normalizing constant; the probability mass we've discounted

$$\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} |\{w : c(w_{i-1}, w) > 0\}|$$

the normalized discount

The number of word types that can follow w_{i-1}
= # of word types we discounted
= # of times we applied normalized discount

Kneser-Ney Smoothing: Recursive formulation

$$P_{KN}(w_i | w_{i-n+1}^{i-1}) = \frac{\max(c_{KN}(w_{i-n+1}^i) - d, 0)}{c_{KN}(w_{i-n+1}^{i-1})} + \lambda(w_{i-n+1}^{i-1})P_{KN}(w_i | w_{i-n+2}^{i-1})$$

$$c_{KN}(\bullet) = \begin{cases} \textit{count}(\bullet) & \text{for the highest order} \\ \textit{continuationcount}(\bullet) & \text{for lower order} \end{cases}$$

Continuation count = Number of unique single word contexts for •

Language Modeling

Advanced:
Kneser-Ney Smoothing