

Natural Language Processing

By
Prof Grishma S

Text is everywhere!

Social media



Text is everywhere!

Research papers, news, etc.

The screenshot shows a search result for a patent. The title is "Efficiency of quantum vs. classical annealing in nonconvex learning problems". It includes a thumbnail of the patent document, the abstract, and a detailed view of the patent text.

The image is a horizontal collage of logos for various science and technology websites and platforms. From left to right, it includes: a 'SUBSCRIBE' button for Scientific American; the 'SCIENTIFIC AMERICAN' logo; a navigation bar for 'HOME', 'ABOUT', 'CONTACT', 'LOGOUT'; the 'ScienceBlogs' logo with a blue square icon; a 'LOG IN' button; and a 'LOG OUT' button. Below these are several news snippets: 'Trading Our Future for Fossil Fuels' by Matt Friedman, 'Why fossil fuel corporations killed us' by Michael Mann, 'How to build a better battery' by Mark Wainberg, 'Fake News: The 2016 US Presidential Election' by Michael E. Mann, 'Climate Change: The Case for Action' by Michael E. Mann, 'Purposed Metal Casting Would Identify As Power Outage' by Michael E. Mann, 'James Webb Can We Detect Earthlike Planets Around Sunlike Stars?' by Michael E. Mann, 'Is the Universe Finite or Infinite?' by Michael E. Mann, and 'Channel Surfing' by Michael E. Mann. To the right of these snippets are three large logos: 'GitHub' with its cat icon, 'stack overflow' with its orange flame icon, and 'ORCID' with its green 'iD' icon. Below the GitHub logo is the tagline 'Connecting Research and Researchers'. At the bottom right is the text 'Crowd-sourced platforms'.

Diversity of Languages (Worldwide)

How many Languages are spoken today?

¹Source: <https://www.ethnologue.com/guides/how-many-languages>

Diversity of Languages (Worldwide)

How many Languages are spoken today?

7,111¹

¹Source: <https://www.ethnologue.com/guides/how-many-languages>

Diversity of Languages (Worldwide)

How many Languages are spoken today?

7,111¹

Can we understand the majority of the

¹Source: <https://www.ethnologue.com/guides/how-many-languages>

Diversity of Languages (Worldwide)

How many Languages are spoken today?

7,111¹

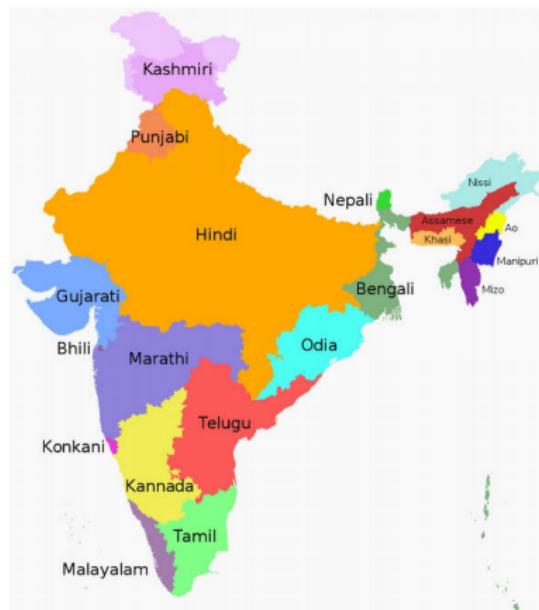
Can we understand the majority of the

Rank ↗	Language	Native speakers in millions 2007 (2010) ↗	Percentage of world population (2007) ↗
1	Mandarin (entire branch)	935 (955)	14.1%
2	Spanish	390 (405)	5.85%
3	English	365 (360)	5.52%
4	Hindi ^[b]	295 (310)	4.46%
5	Arabic	280 (295)	4.23%
6	Portuguese	205 (215)	3.08%
7	Bengali (Bangla)	200 (205)	3.05%
8	Russian	160 (155)	2.42%
9	Japanese	125 (125)	1.92%
10	Punjabi	95 (100)	1.44%

¹Source: <https://www.ethnologue.com/guides/how-many-languages>

Diversity of Languages (India)

Can we understand the majority of the India's data?



Total Languages: > 1650 (2011 consensus)

Hindi: 57.1%

English: 10.6%

Bengali: 8.9%

Marathi: 8.2%

Telugu: 7.8 %

...

...

The goals of NLP

French Sentence: Tu Bois un Coca-cola

The goals of NLP

French Sentence: Tu Bois un Coca-cola

English Translation: You drink a Coca-cola

The goals of NLP

French Sentence: Tu Bois un Coca-cola

English Translation: You drink a Coca-cola

We try to understand a foreign language using some known keywords

The goals of NLP

French Sentence: Tu Bois un Coca-cola

English Translation: You drink a Coca-cola

We try to understand a foreign language using some known keywords

Goals of NLP

- Deep understanding of broad language constructs.
- Achieve human-like comprehension of texts/languages.
- Make computer systems to understand, draw inferences from, summarize, translate and generate accurate and natural human text and language.

Some Applications: Language Translation

The screenshot shows the Google Translate interface. At the top, there's a navigation bar with the Google logo, followed by "Translate" and a "Turn off instant translation" link. Below the navigation bar, there are language selection buttons for English, Spanish, Hindi, and Detect language. The main area shows a translation from English to Hindi. The English input is "Let's go out for a date". The Hindi output is "चलो एक तारीख के लिए बाहर जाओ". Below the translation, there are edit controls (undo, redo, etc.) and a "Suggest an edit" link. The bottom of the interface shows the original text in Hindi: "chalo ek taareekh ke lie baahar jao".

Language Translation

Google Translate

Turn off instant translation

English Spanish Hindi Detect language English Spanish Hindi Translate

मेरे हाथ में तेरा हाथ हो सारी जन्नतें मेरे साथ हो। All hands should be with me in my hand.

49/5000 Suggest an edit

mere haath mein tera haath ho saaree jannaten mere saath ho

Language Translation is not easy even for humans

Pepsi Chinese blunder

“Come alive with the Pepsi Generation”, when translated into Chinese meant,
“Pepsi brings your relatives back from the dead.”

Language Translation is not easy even for humans

Pepsi Chinese blunder

“Come alive with the Pepsi Generation”, when translated into Chinese meant, “Pepsi brings your relatives back from the dead.”

KFC's Chinese blunder

KFC's slogan, “Finger lickin' good”, when translated into Chinese meant “We'll eat your fingers off.”

Some more examples...



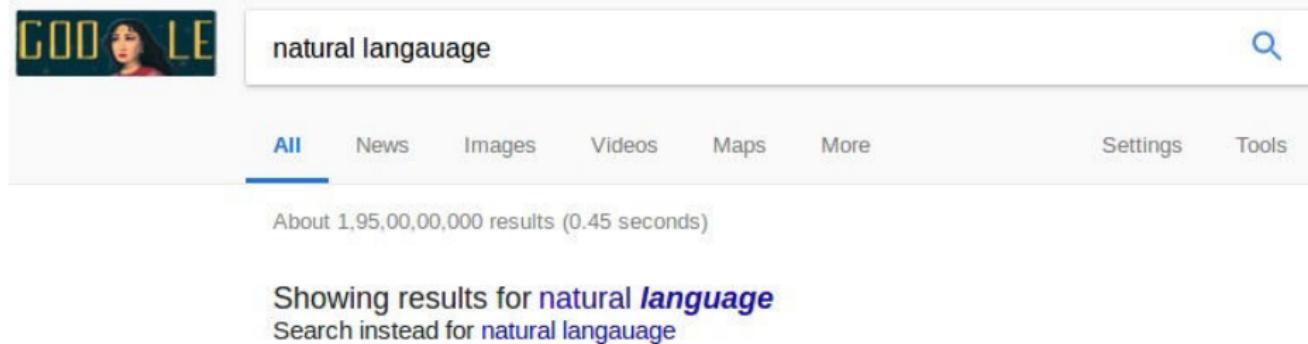
Query Recommendation in Search Engines



Dhoni is

- dhoni is from which state**
- dhoni is back**
- dhoni is in which team**
- dhoni is**
- dhoni is retiring**
- dhoni is best**
- dhoni is photo**
- dhoni is a legend**
- dhoni is injured**
- dhoni is age**

Spelling Correction



A screenshot of a Google search results page. The search bar at the top contains the query "natural langauge". Below the search bar, the "All" tab is selected, followed by "News", "Images", "Videos", "Maps", and "More". To the right of these tabs are "Settings" and "Tools". A blue horizontal bar indicates the search term "natural langauge". Below the tabs, the text "About 1,95,00,00,000 results (0.45 seconds)" is displayed. The main content area shows a snippet of search results starting with "Showing results for natural *language*".

natural langauge

All News Images Videos Maps More Settings Tools

About 1,95,00,00,000 results (0.45 seconds)

Showing results for natural *language*
Search instead for natural langauge

Information Extraction

New York Times Co. named Russell T. Lewis, 45, president and general manager of its flagship New York Times newspaper, responsible for all business-side activities. He was executive vice president and deputy general manager. He succeeds Lance R. Primis, who in September was named president and chief operating officer of the parent.

Person	Company	Post	State
Russell T. Lewis	New York Times newspaper	president and general manager	start
Russell T. Lewis	New York Times newspaper	executive vice president	end
Lance R. Primis	New York Times Co.	president and CEO	start

Sentiment Analysis



Saravana Kumar



Worst software update Needed

4 May 2019

Style: 3GB RAM | Size: 32GB | Colour: Elegant Blue | **Verified Purchase**

Bad software

Phone hangs a lot

Many issues and bugs in phone

Only Battery and front camera is good

Worst from MI

208 people found this helpful

Helpful

▼ 1 Comment

| Report abuse

Recent Trends: Fake news detection

Lies

IndiaToday.in
New Delhi, April 13, 2017 | UPDATED 18:31 IST

Uttar Pradesh government ends reservation for SC, ST, OBC candidates in private medical colleges

The Uttar Pradesh government made an announcement to end caste-based reservations in private medical and dental colleges of up. An order has been passed to do away with the quota for candidates belonging to SC, ST and OBC categories.



Uttar Pradesh CM Yogi Adityanath

RELATED STORIES

- ❑ Union Cabinet approves IIPV Visakhapatnam's status as institute of national importance
- ❑ Delhi HC suggests outsiders should be banned from entering JNU
- ❑ 25,000 Indian students to be trained in cyber security by Cisco
- ❑ Haryana boy makes it to IIM-A with 89.67 percentile

Truth

'Caste-based reservation never existed in private medical colleges in Uttar Pradesh'

Shalivaa Sharda THN | Apr 13, 2017, 07:27 PM IST

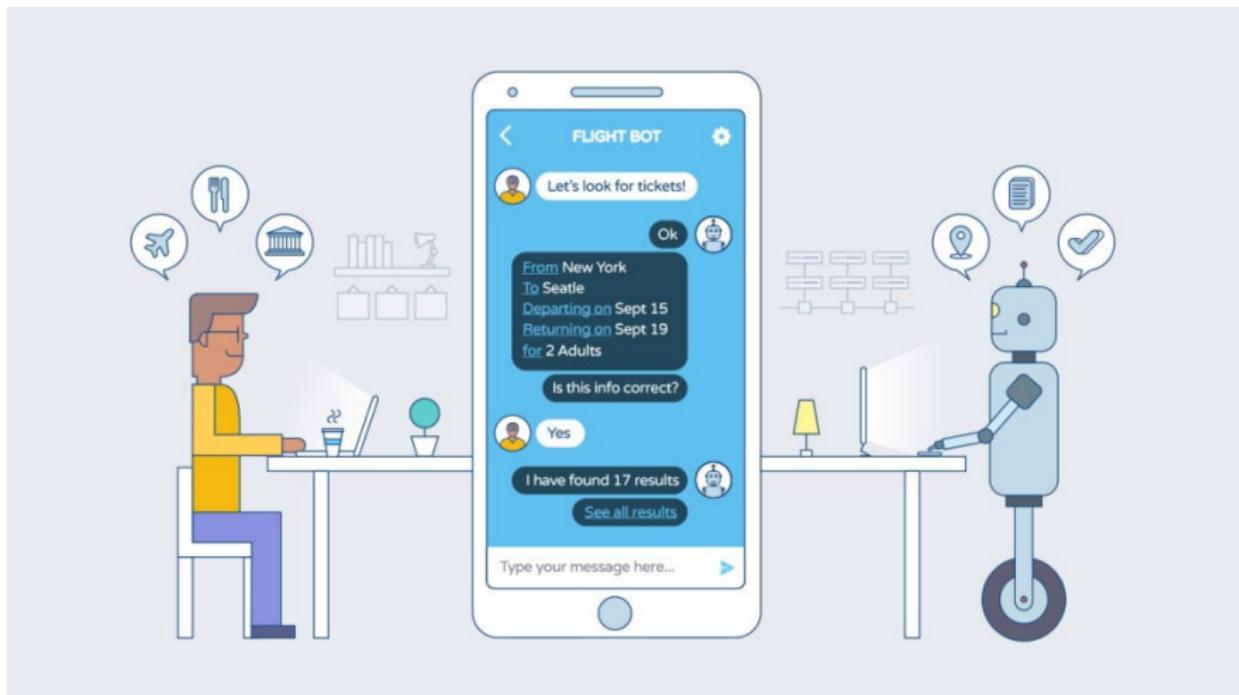


Pile photograph used for representational purpose

HIGHLIGHTS

- ❑ UP medical education department refuted reports of Yogi government abolishing reservations in private medical colleges
- ❑ It said caste-based reservation was never a part of the admission process in private colleges in UP

Recent Trends: Chatbots



Other Goals

- Text Summarization
- Opinion dynamics
- Spam detection
- . . .

Other Goals

- Text Summarization
- Opinion dynamics
- Spam detection
- . . .

Natural Language
Technology not yet
perfect

But still good enough for
several useful
applications

Why is NLP hard?

Compounding

निरन्तरान्धकारित-दिगन्तर-कन्दलदमन्द-सुधारस-विन्दु-सान्दरतर-घनाघन-वृन्द-
सन्देहकर-स्यन्दमान-मकरन्द-विन्दु-बन्धुरतर-माकन्द-तरु-कुल-तल्प-कल्प-
मृदुल-सिकता-जाल-जटिल-मूल-तल-मरुवक-मिलदलधु-लघु-लय-कलित-
रमणीय-पानीय-शालिका-बालिका-करार-विन्द-गलन्तिका-गलदेला-लवड़ग-
पाटल-घनसार-कस्तूरिकातिसौरभ-मेदुर-लघुतर-मधुर-शीतलतर-सलिलधारा-
निराकरिष्णु-तदीय-विमल-विलोचन-मयूख-रेखापसारित-पिपासायास-पथिक-
लोकान्

195 characters (with 428 characters when transliterated into the roman writing system).

Why is NLP hard?

Ambiguity

Lexical Ambiguity

The presence of two or more possible meanings within a single word.



"I saw her duck."

Syntactic Ambiguity

The presence of two or more possible meanings within a single sentence or sequence of words.



"The chicken is ready to eat."

Why is NLP hard?

Ambiguity



@DOGMODOG

Why else is NLP hard?

Shorthand text



Why else is NLP hard?

Non-standard English

Why else is NLP hard?

Segmentation Issues

the New York New Haven Railroad

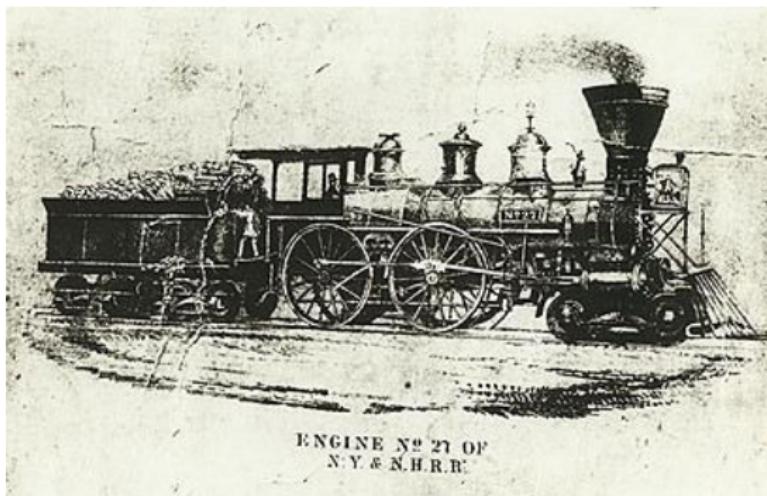
Why else is NLP hard?

Segmentation Issues

the New York New Haven Railroad

the [New] [York New] [Haven] [Railroad]

the [New York] [New Haven] [Railroad]



Why else is NLP hard?

Idioms

- Dark horse
- Ball in your court
- Burn the midnight oil

Why else is NLP hard?

Idioms : An expression whose meaning is different from the meanings of the individual words in it.

Idioms Example

- Dark horse
- Ball in your court Burn the midnight oil
-

Neologisms: A new word, phrase or expression, or a new meaning of a familiar word

- Unfriend
- Retweet
- Google/Skype/photoshop

Why is NLP hard?

New Senses of a word

- That's *sick* dude!
- Giants

Why is NLP hard?

New Senses of a word

- That's *sick* dude!

Giants ...

*multinationals,
conglomerates,
manufacturers*

Why is NLP hard?

New Senses of a word

- That's *sick* dude!

Giants ...

*multinationals,
conglomerates,
manufacturers*



Tricky Entity Names

Where is *A Bug's Life*
playing ...

Let It Be was recorded

...

Why is NLP hard?

Code Mixing/switching



Romanization

BAHUT TENSION HAI 0
PUKKA IDIOT!
EK CHANCE MILEGA?
CUTTING CHAI
ADJUST KIJIYE
CHUTNEFYING
ENGLISH

What we do in NLP?

Create annotated corpora

Brown Corpus, Google Books Ngram Corpus, Reuters Newswire Topic Classification, IMDB Movie Review Sentiment Classification, Project Gutenberg, etc.

Create Models/Algorithms

LDA, BERT, CKY, Edit Distance, CRF++, etc.

Create Tools

CoreNLP, NLTK, Gensim, SpaCy, etc.

Stages in NLP (*traditional view*)

- *Phonetics and phonology*
- *Morphology*
- *Lexical Analysis*
- *Syntactic Analysis*
- *Semantic Analysis*
- *Pragmatics*
- *Discourse*

Source: IITB NLP Course by Pushpak Bhattacharyya

Phonetics

- Processing of speech
- Challenges
 - Homophones: *bank (finance)* vs. *bank (river bank)*
 - Near Homophones: *maatraqa* vs. *maatra (hin)*
 - Word Boundary
 - *aajaayenge (aa jaayenge (will come) or aaj aayenge (will come today)*
 - *I got [ua]plate*
 - Phrase boundary
 - *mtech1 students are especially exhorted to attend as such seminars are integral to one's post-graduate education*
 - Disfluency: *ah, um, ahem etc.*

Morphology

- Word formation rules from *root words*
- Nouns: Plural (*boy-boys*); Gender marking (*czar-czarina*)
- Verbs: Tense (*stretch-stretched*); Aspect (e.g. perfective *sit-had sat*); Modality (e.g. request *khaanaa* \sqcup *khaaiie*)
- First crucial first step in NLP
- Languages rich in morphology: e.g., Dravidian, Hungarian, Turkish
- Languages poor in morphology: Chinese, English
- Languages with rich morphology have the advantage of easier processing at higher stages of processing
- A task of interest to computer science: *Finite State Machines for Word Morphology*

Lexical Analysis

- Essentially refers to dictionary access and obtaining the properties of the word
 - e.g. *dog*
 - noun (lexical property)*
 - take-'s'-in-plural (morph property)*
 - animate (semantic property)*
 - 4-legged (-do-)*
 - carnivore (-do)*
- Challenge: *Lexical or word sense disambiguation*

Lexical Disambiguation

First step: *part of Speech Disambiguation*

Dog as a noun (animal)

Dog as a verb (*to pursue*)

Sense Disambiguation

Dog (as *animal*)

Dog (as a very detestable person)

Needs word relationships in a context

The chair emphasized the need for adult education

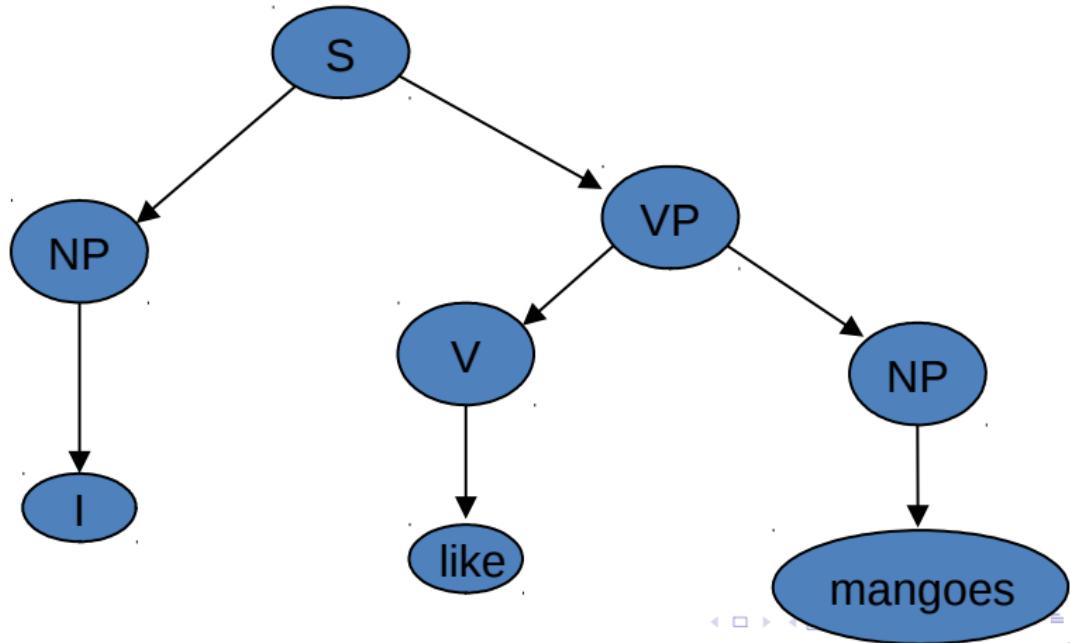
Very common in day to day communications

Satellite Channel Ad: *Watch what you want, when you want* (two senses of watch)

e.g., Ground breaking ceremony/research

Syntax Processing Stage

Structure Detection



Parsing Strategy

Driven by grammar

$S \rightarrow NP\ VP$

$NP \rightarrow N \mid PRON$

$VP \rightarrow V\ NP \mid V\ PP$

$N \rightarrow \text{Mangoes}$

$PRON \rightarrow I$

$V \rightarrow \text{like}$

Challenges in Syntactic Processing: Structural Ambiguity

- *Scope Ambiguity*

1. *The old men and women were taken to safe locations*
(old men and women) vs. ((old men) and women)
2. *No smoking areas will allow Hookas inside*

- *Preposition Phrase Attachment*

- *I saw the boy with a telescope*
(who has the telescope?)
- *I saw the mountain with a telescope*
*(world knowledge: *mountain* cannot be an instrument of seeing)*
- *I saw the boy with the pony-tail*
*(world knowledge: *pony-tail* cannot be an instrument of seeing)*
- *Very ubiquitous: newspaper headline “20 years later, BMC pays father 20 lakhs for causing son’s death”*

Structural Ambiguity...

Overheard

I did not know my PDA had a phone for 3 months

An actual sentence in the newspaper

The camera man shot the man with the gun when he was near Tendulkar

Semantic Analysis

- *Representation in terms of*
Predicate calculus/Semantic
Nets/Frames/Conceptual Dependencies
and Scripts
- *John gave a book to Mary*
Give action: Agent: John, Object: Book,
Recipient: Mary
- *Challenge: ambiguity in semantic role labeling*
(Eng) Visiting aunts can be a nuisance
(Hin) aapko mujhe mithaai khilaanii padegii
(ambiguous in Marathi and Bengali too; not
in Dravidian languages)

Pragmatics

- *Very hard problem*
- *Model user intention*
 - *Tourist (in a hurry, checking out of the hotel, motioning to the service boy): Boy, go upstairs and see if my sandals are under the divan. Do not be late. I just have 15 minutes to catch the train.*
 - *Boy (running upstairs and coming back panting): yes sir, they are there.*
- *World knowledge*
 - *WHY INDIA NEEDS A SECOND OCTOBER*

Discourse

- Processing of sequence of sentences
 - *Mother to John:*
 - *John go to school. It is open today. Should you bunk? Father will be very angry.*
- Ambiguity of *open*
- *bunk what?*
- *Why will the father be angry?*
 - *Complex chain of reasoning and application of world knowledge*
 - *Ambiguity of father*
father as parent
or
father as headmaster

Reference Books

- Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. 2nd edition. Prentice-Hall. 2009.
- Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press. 1999.
- Steven Bird, Ewan Klein and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media. 2009.
- Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press. 2016.

Regular Expressions and Finite State Automata

Mausam

(Based on slides by Jurafsky & Martin,
Julia Hirschberg)

Regular Expressions and Text Searching

- Everybody does it
 - ◆ Emacs, vi, perl, grep, etc..
- Regular expressions are a compact textual representation of a set of strings representing a language.

RE	Example Patterns Matched
/woodchucks/	“interesting links to <u>woodchucks</u> and lemurs”
/a/	“ <u>Mary</u> Ann stopped by <u>Mona</u> ’s”
/Claire_says,/	“ “Dagmar, my gift please,” <u>Claire</u> says,”
/DOROTHY/	“SURRENDER <u>DOROTHY</u> ”
/ ! /	“You’ve left the burglar behind again!” said Nori

Regular Expressions

RE	Match	Example Patterns
/ [wW]oodchuck /	Woodchuck or woodchuck	“ <u>Woodchuck</u> ”
/ [abc] /	‘a’, ‘b’, <i>or</i> ‘c’	“In uomini, in soldati”
/ [1234567890] /	any digit	“plenty of <u>7</u> to <u>5</u> ”

Regular Expressions

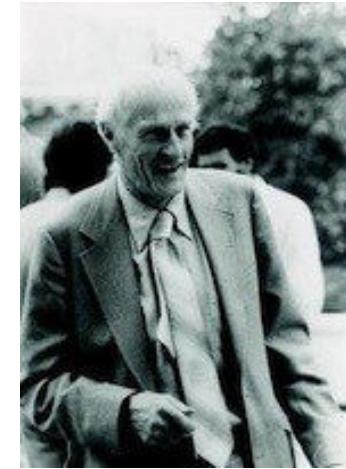
RE	Match	Example Patterns Matched
/ [A-Z] /	an upper case letter	“we should call it ‘ <u>Drenched Blossoms</u> ’ ”
/ [a-z] /	a lower case letter	“ <u>my</u> beans were impatient to be hoed!”
/ [0-9] /	a single digit	“Chapter <u>1</u> : Down the Rabbit Hole”

Regular Expressions

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an upper case letter	“Oyfn pripetchik”
[^Ss]	neither ‘S’ nor ‘s’	“I have no exquisite reason for’t”
[^\.\.]	not a period	“our resident Djinn”
[e^]	either ‘e’ or ‘^’	“look up ^ now”
a^b	the pattern ‘a^b’	“look up a^ b now”

Regular Expressions: ? * + .

Pattern	Matches	
colou?r	Optional previous char	<u>color</u> <u>colour</u>
oo*h!	0 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
o+h!	1 or more of previous char	<u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>
baa+		<u>baa</u> <u>baaa</u> <u>baaaa</u> <u>baaaaa</u>
beg.n		<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene

Kleene *, Kleene -

Regular Expressions: Anchors

^ \$

Pattern	Matches
^ [A-Z]	Palo Alto
^ [^A-Za-z]	<u>1</u> "Hello"
\. \$	The end <u>.</u>
. \$	The end <u>?</u> The end <u>!</u>

Regular Expressions

RE	Expansion	Match	Examples
\d	[0-9]	any digit	Party_of_5
\D	[^0-9]	any non-digit	Blue_moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!
\s	[\r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in_Concord

Regular Expressions

RE	Match	Example Patterns Matched
*	an asterisk “*”	“K*A*P*L*A*N”
\.	a period “.”	“Dr. <u>_</u> Livingston, I presume”
\?	a question mark	“Why don’t they come and lend a hand <u>_</u> ?”
\n	a newline	
\t	a tab	

Example

- Find all the instances of the word “the” in a text.
 - ◆ `/the/`
 - ◆ `/ [tT] he/`
 - ◆ `/\b [tT] he \b/`
 - ◆ `[^a-zA-Z] [tT] he [^a-zA-Z]`
 - ◆ `(^ | [^a-zA-Z]) [tT] he ($ | [^a-zA-Z])`

Errors

- The process we just went through was based on two fixing kinds of errors
 - ◆ Matching strings that we should not have matched (**there**, **then**, **other**)
 - **False positives** (Type I)
 - ◆ Not matching things that we should have matched (**The**)
 - **False negatives** (Type II)

Errors

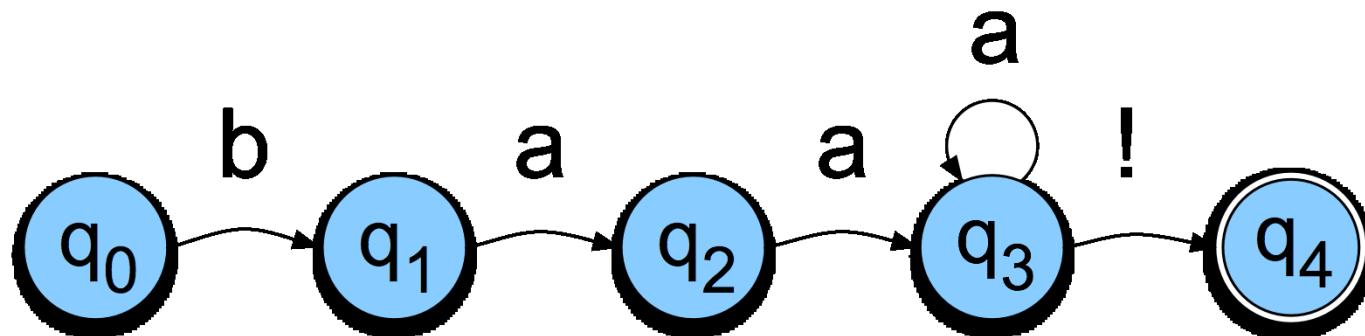
- We'll be telling the same story for many tasks, all semester. Reducing the error rate for an application often involves two **antagonistic** efforts:
 - ◆ Increasing accuracy, or precision, (minimizing false positives)
 - ◆ Increasing coverage, or recall, (minimizing false negatives).

Finite State Automata

- Regular expressions can be viewed as a textual way of specifying the structure of finite-state automata.
- FSAs capture significant aspects of what linguists say we need for **morphology** and parts of **syntax**.

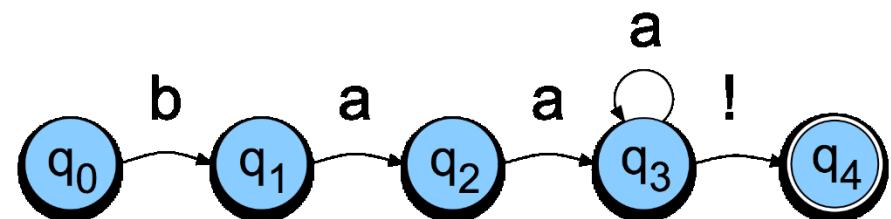
FSAs as Graphs

- Let's start with the sheep language from Chapter 2
 - /baa+ ! /



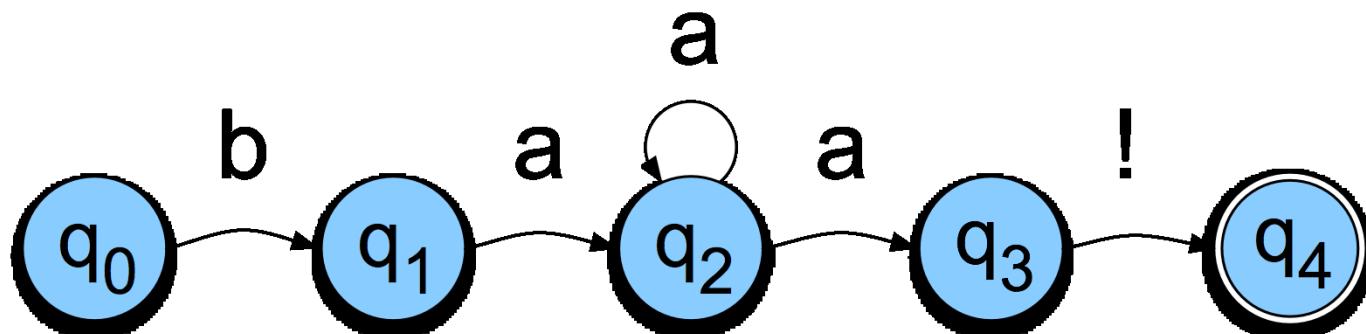
Sheep FSA

- We can say the following things about this machine
 - ◆ It has 5 states
 - ◆ **b**, **a**, and **!** are in its alphabet
 - ◆ q_0 is the start state
 - ◆ q_4 is an accept state
 - ◆ It has 5 transitions



But Note

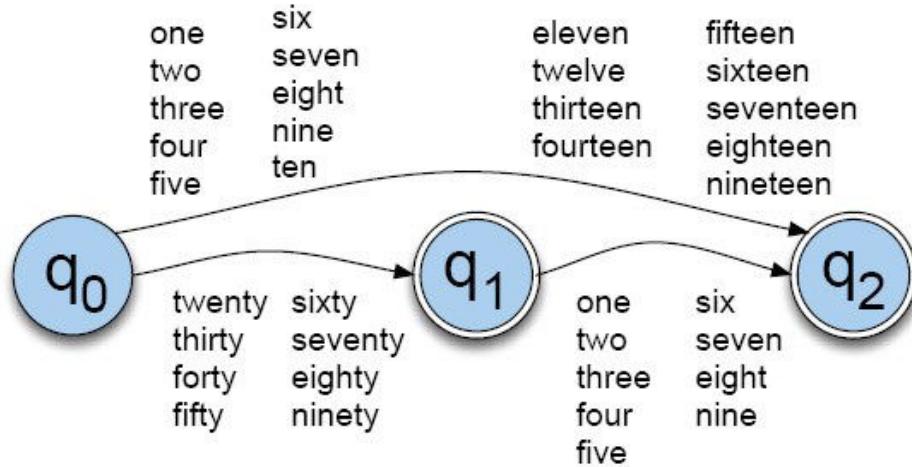
- There are other machines that correspond to this same language



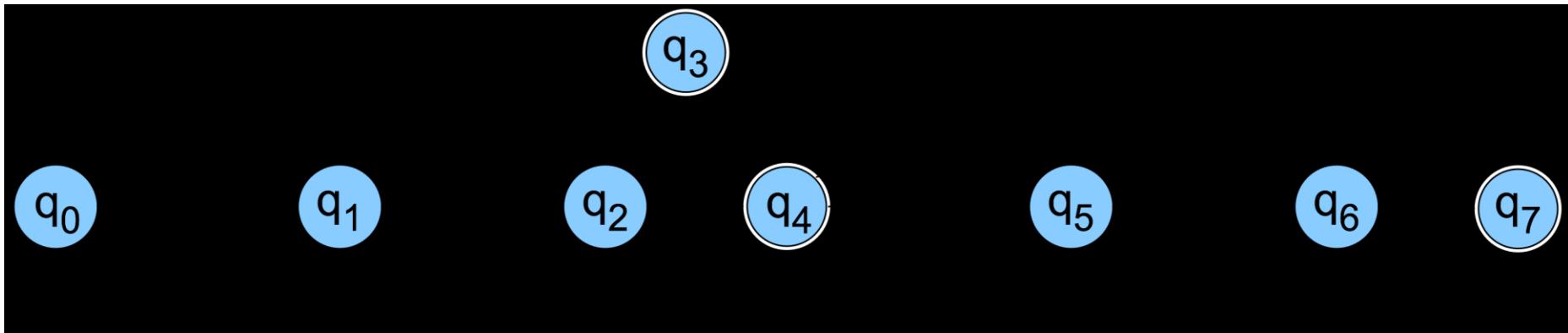
More Formally

- You can specify an FSA by enumerating the following things.
 - ◆ The set of states: Q
 - ◆ A finite alphabet: Σ
 - ◆ A start state
 - ◆ A set of accept/final states
 - ◆ A transition function that maps $Q \times \Sigma$ to Q

Dollars and Cents



Dollars and Cents

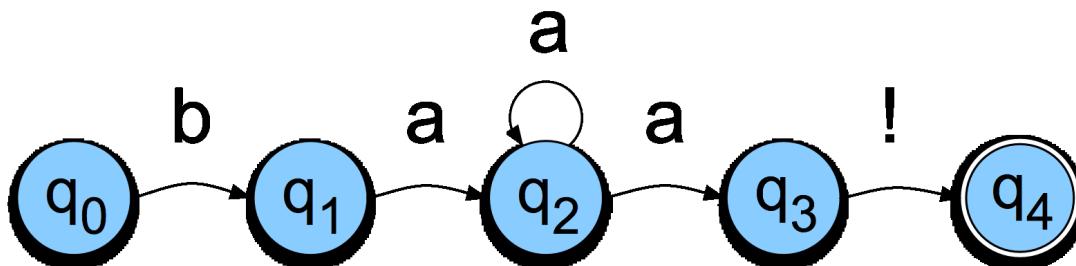


Yet Another View

- The guts of FSAs can ultimately be represented as tables

If you're in state 1 and you're looking at an a, go to state 2

	b	a	!	e
0		1		
1			2	
2			2,3	
3				4
4				

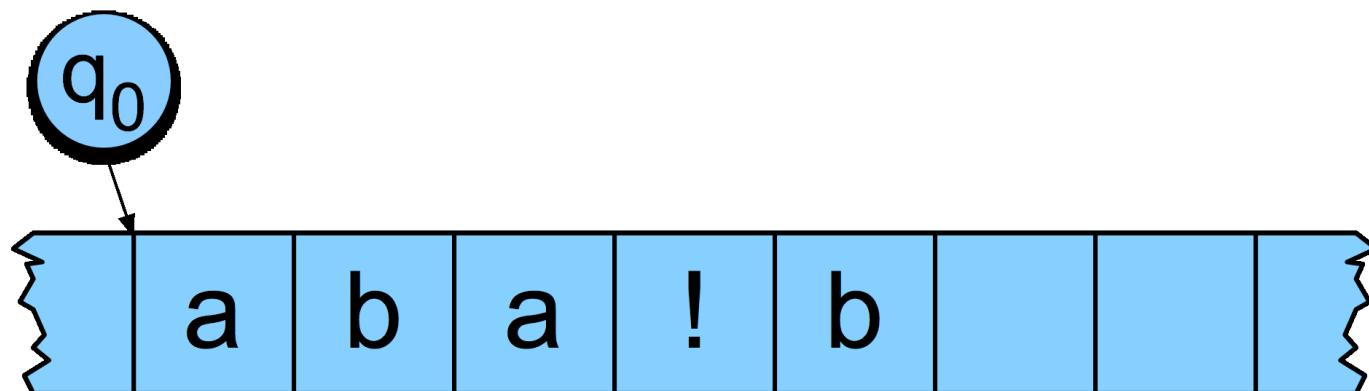


Recognition

- Recognition is the process of determining if a string should be accepted by a machine
- Or... it's the process of determining if a string is in the language we're defining with the machine
- Or... it's the process of determining if a regular expression matches a string
- Those all amount the same thing in the end

Recognition

- Traditionally, (Turing's notion) this process is depicted with a tape.



Recognition

- Simply a process of starting in the start state
- Examining the current input
- Consulting the table
- Going to a new state and updating the tape pointer.
- Until you run out of tape.

D-Recognize

function D-RECOGNIZE(*tape, machine*) **returns** accept or reject

index \leftarrow Beginning of tape

current-state \leftarrow Initial state of machine

loop

if End of input has been reached **then**

if current-state is an accept state **then**

return accept

else

return reject

elsif *transition-table[current-state,tape[index]]* is empty **then**

return reject

else

current-state \leftarrow *transition-table[current-state,tape[index]]*

index \leftarrow *index* + 1

end

Key Points

- Deterministic means that at each point in processing there is always one unique thing to do (no choices).
- D-recognize is a simple table-driven interpreter
- The algorithm is universal for all unambiguous regular languages.
 - ◆ To change the machine, you simply change the table.

Key Points

- Crudely therefore... matching strings with regular expressions (ala Perl, grep, etc.) is a matter of
 - ◆ translating the regular expression into a machine (a table) and
 - ◆ passing the table and the string to an interpreter

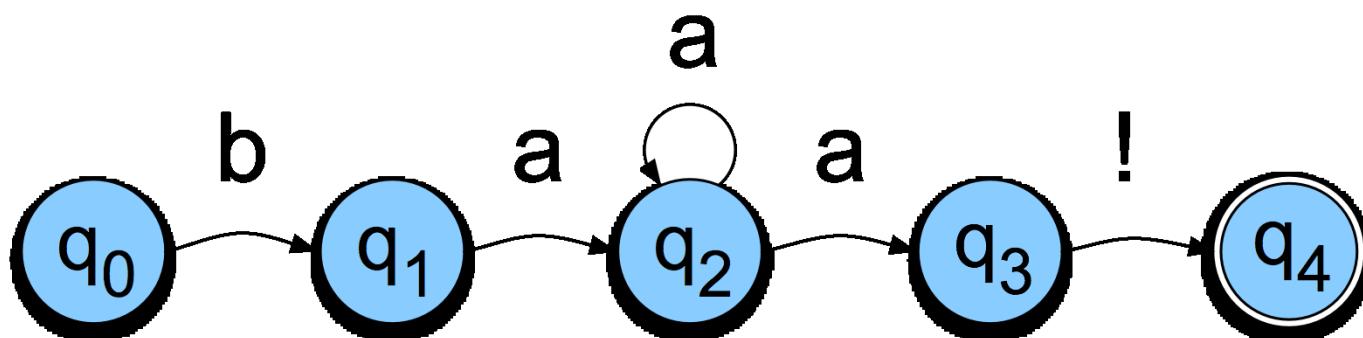
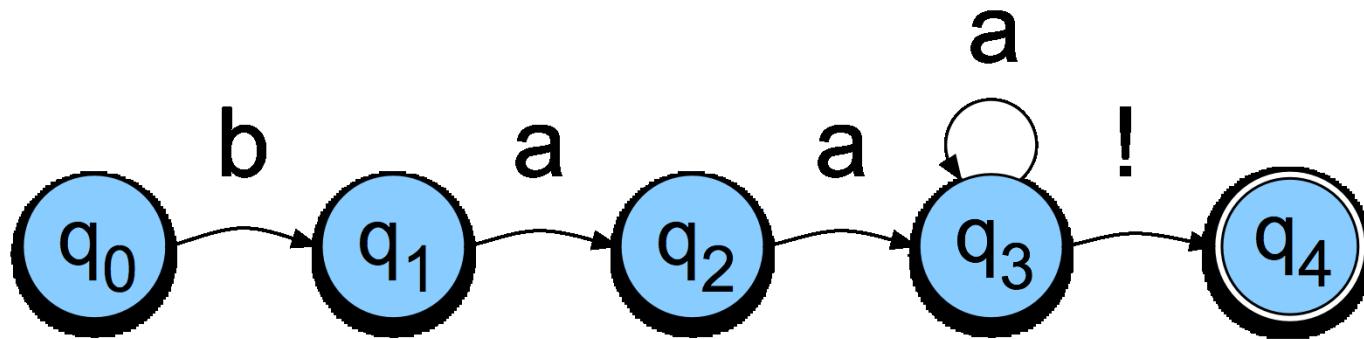
Generative Formalisms

- *Formal Languages* are sets of strings composed of symbols from a finite set of symbols.
- Finite-state automata define formal languages (without having to enumerate all the strings in the language)
- The term *Generative* is based on the view that you can run the machine as a generator to get strings from the language.

Generative Formalisms

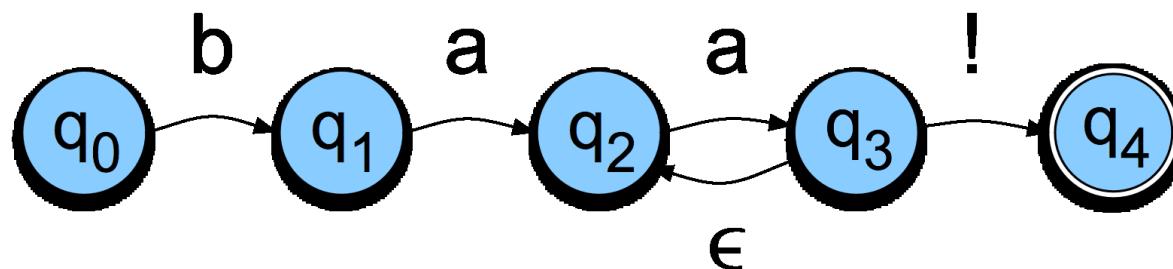
- FSAs can be viewed from two perspectives:
 - ◆ Acceptors that can tell you if a string is in the language
 - ◆ Generators to produce *all and only* the strings in the language

Non-Determinism



Non-Determinism cont.

- Yet another technique
 - ◆ Epsilon transitions
 - ◆ Key point: these transitions do not examine or advance the tape during recognition



Equivalence

- Non-deterministic machines can be converted to deterministic ones with a fairly simple construction
- That means that they have the same power; non-deterministic machines are not more powerful than deterministic ones in terms of the languages they can accept

ND Recognition

- Two basic approaches (used in all major implementations of regular expressions, see Friedl 2006)
 1. Either take a ND machine and convert it to a D machine and then do recognition with that.
 2. Or explicitly manage the process of recognition as a state-space search (leaving the machine as is).

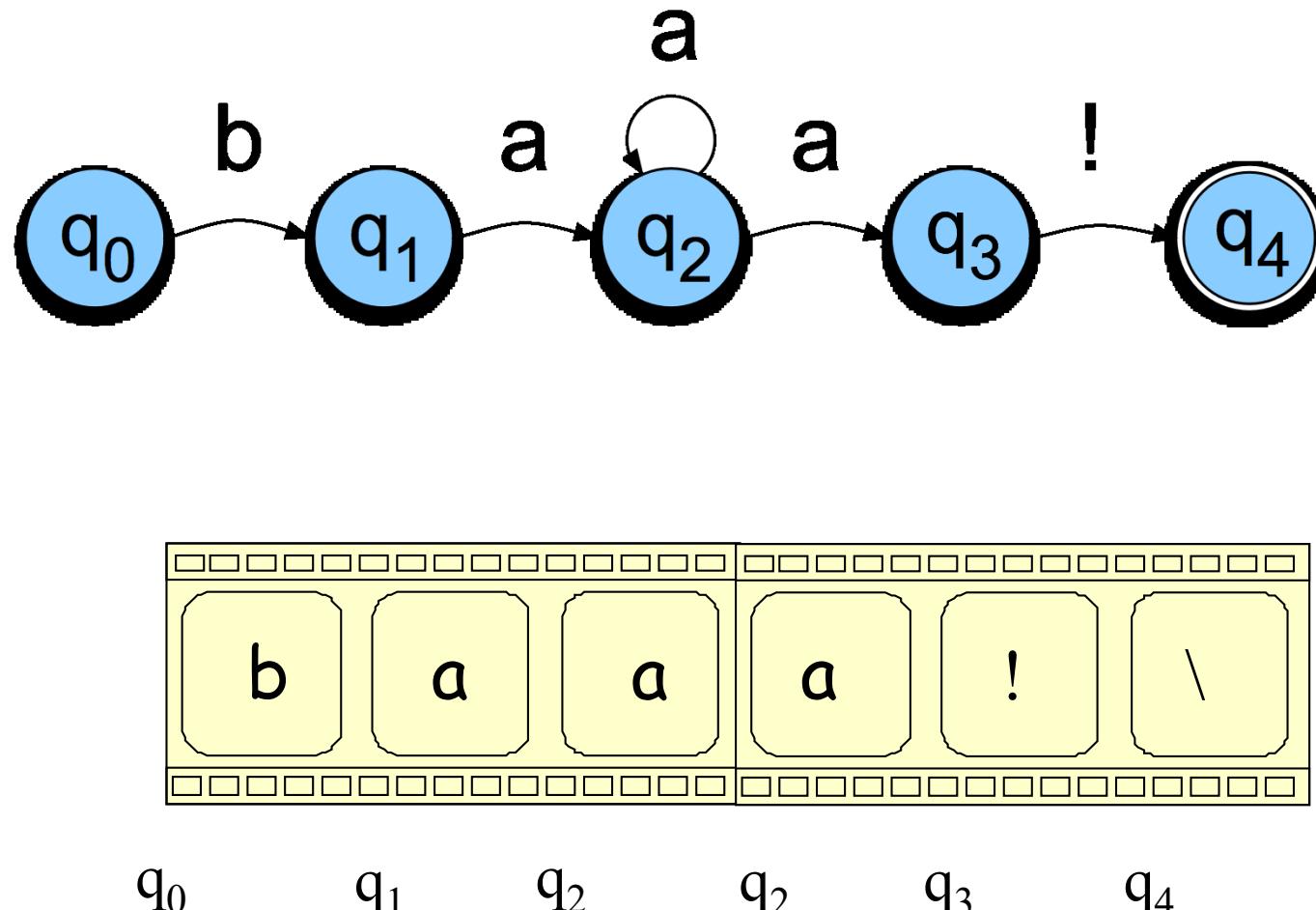
Non-Deterministic Recognition: Search

- In a ND FSA **there exists** at least one path through the machine for a string that is in the language defined by the machine.
- **But not all paths** directed through the machine for an accept string lead to an accept state.
- **No paths** through the machine lead to an accept state for a string not in the language.

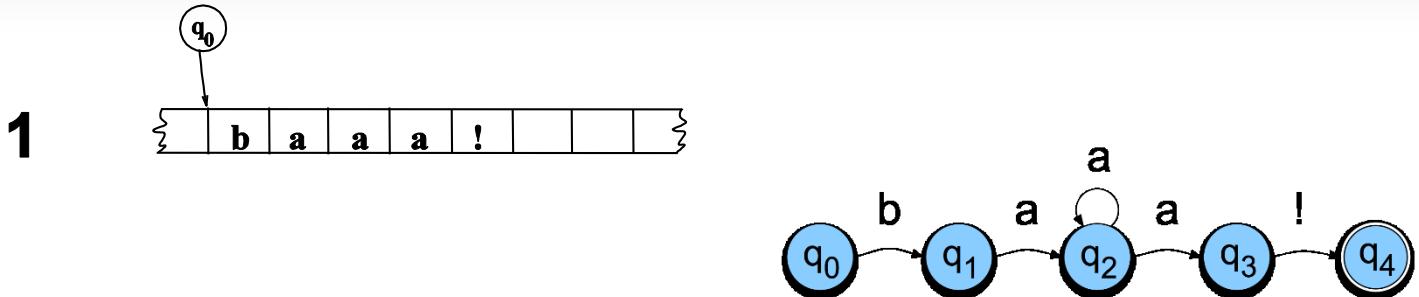
Non-Deterministic Recognition

- So **success** in non-deterministic recognition occurs when a path is found through the machine that ends in an accept.
- **Failure** occurs when **all** of the possible paths for a given string lead to failure.

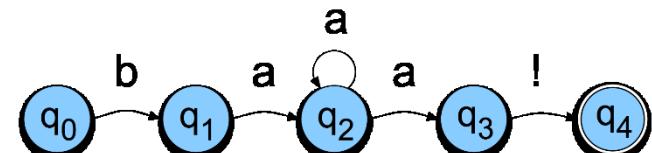
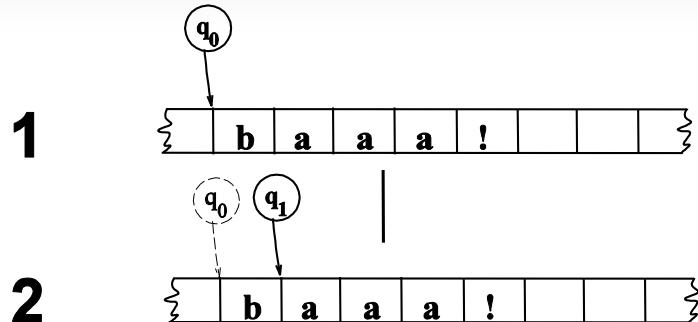
Example



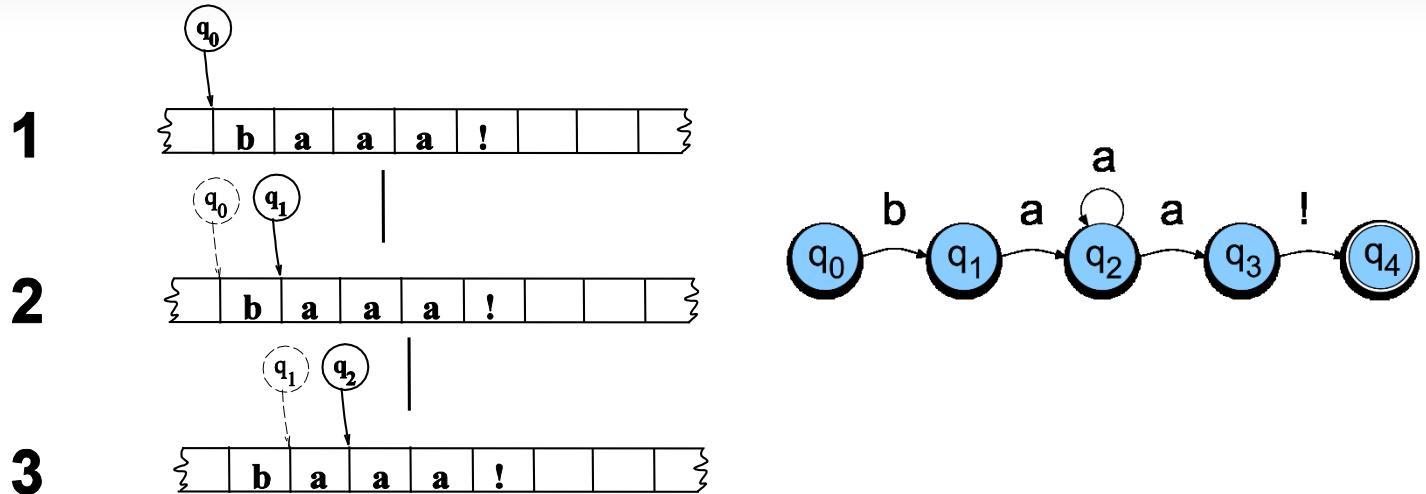
Example



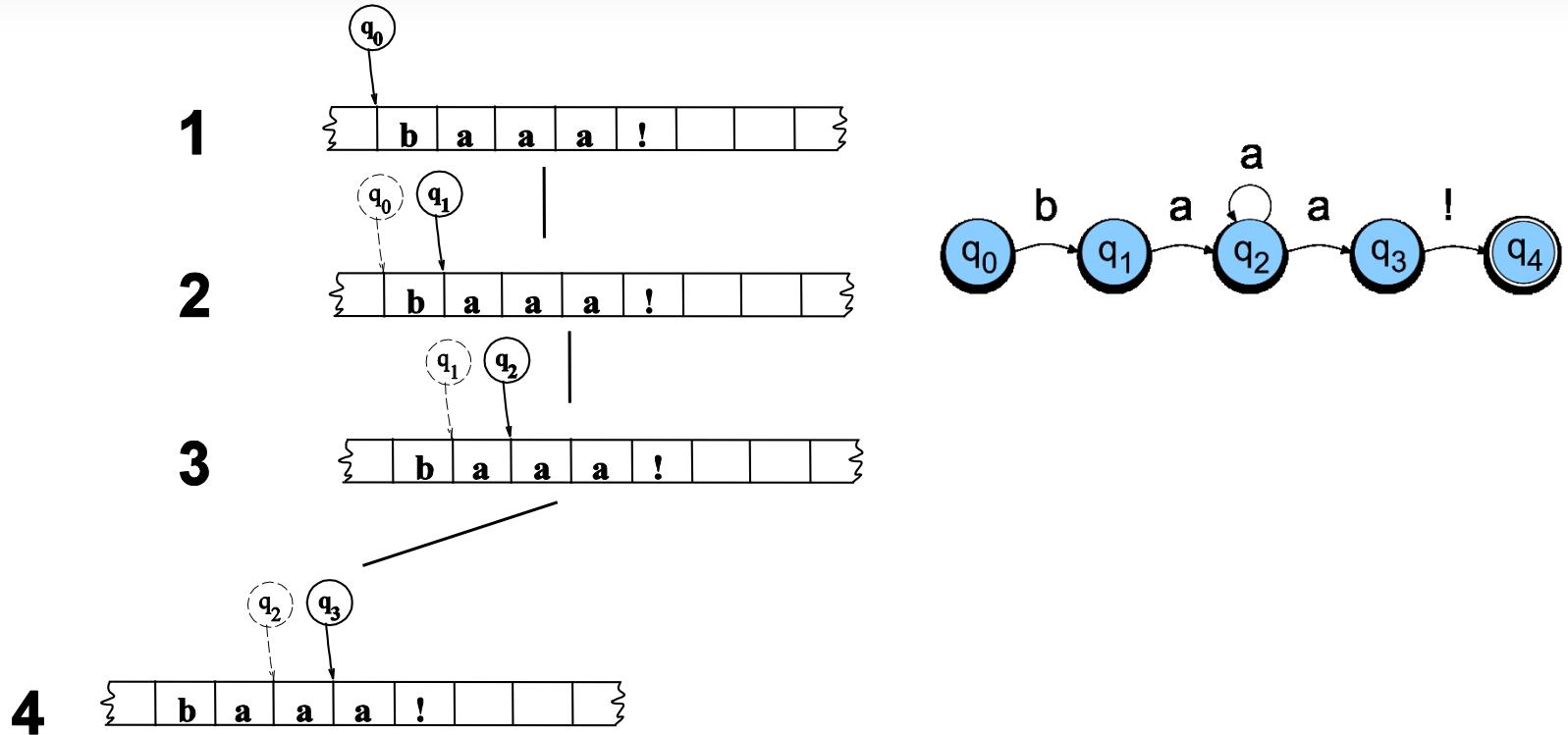
Example



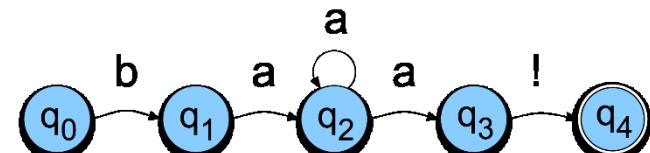
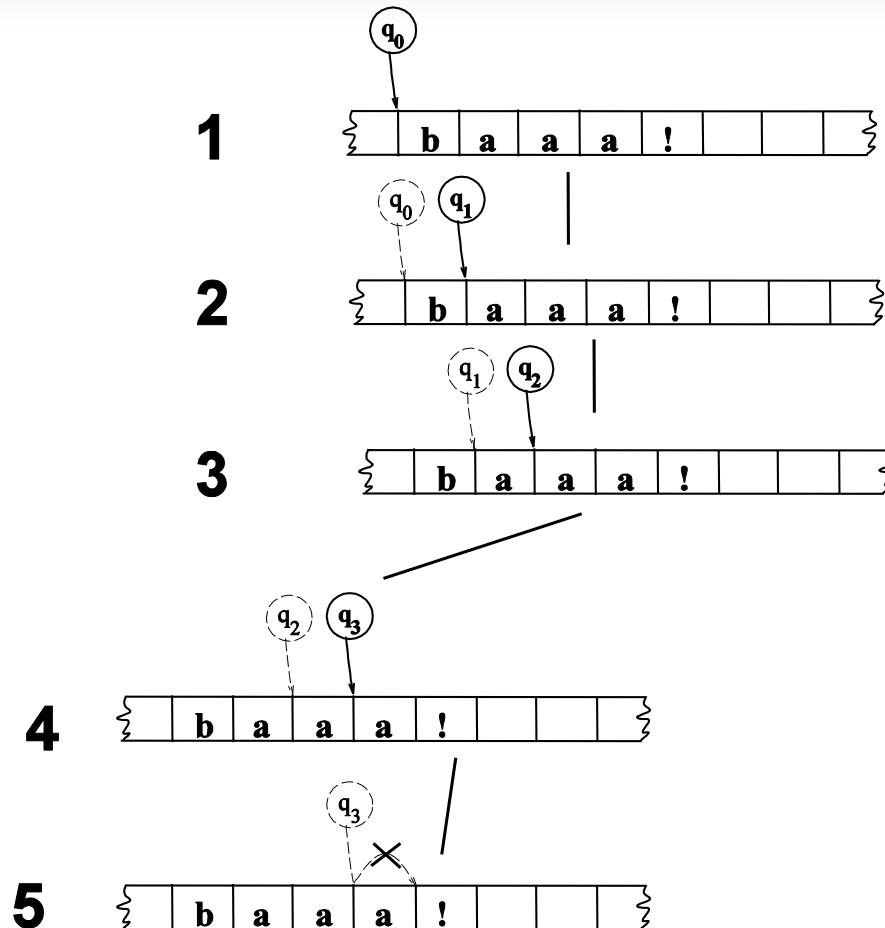
Example



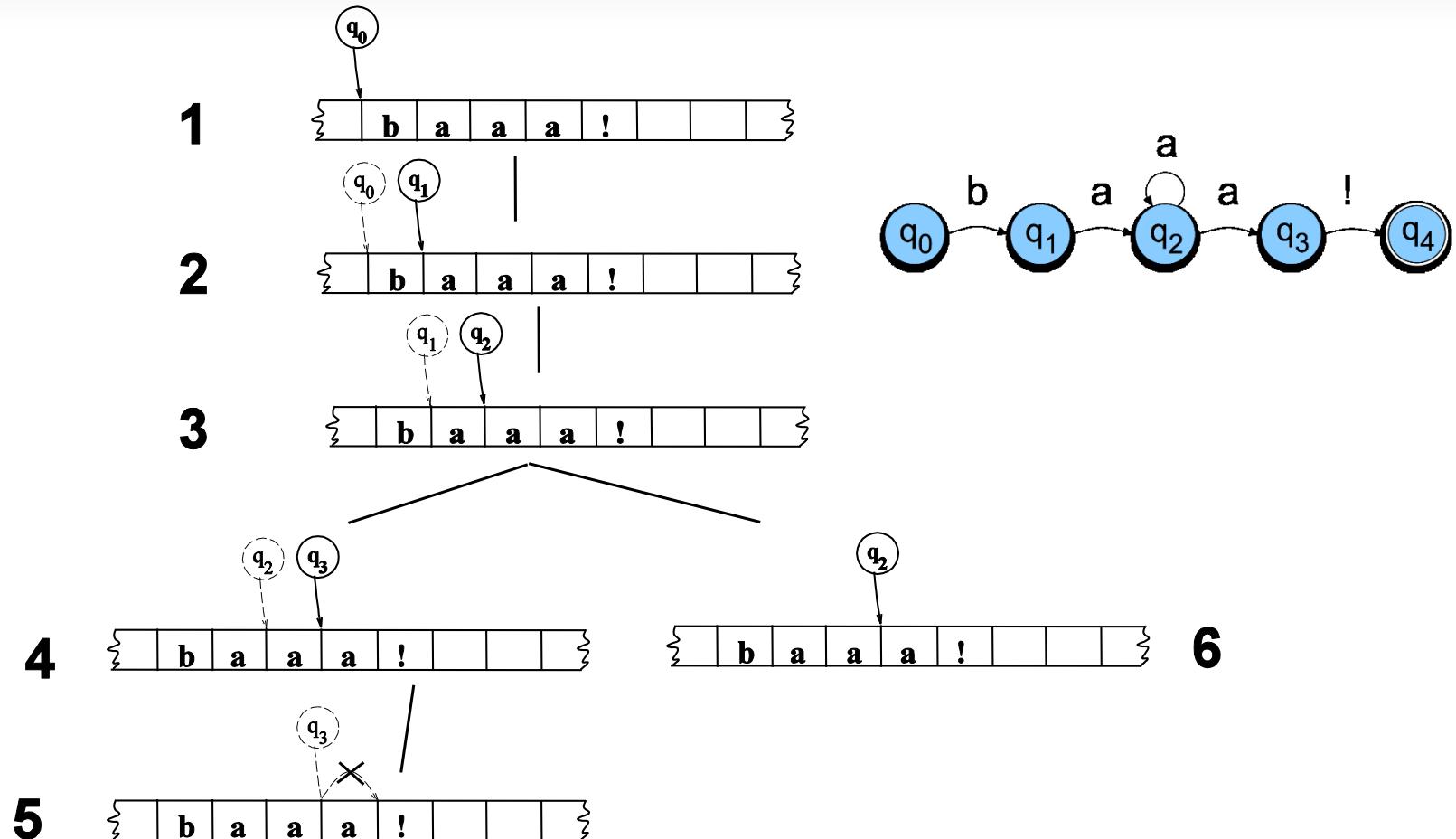
Example



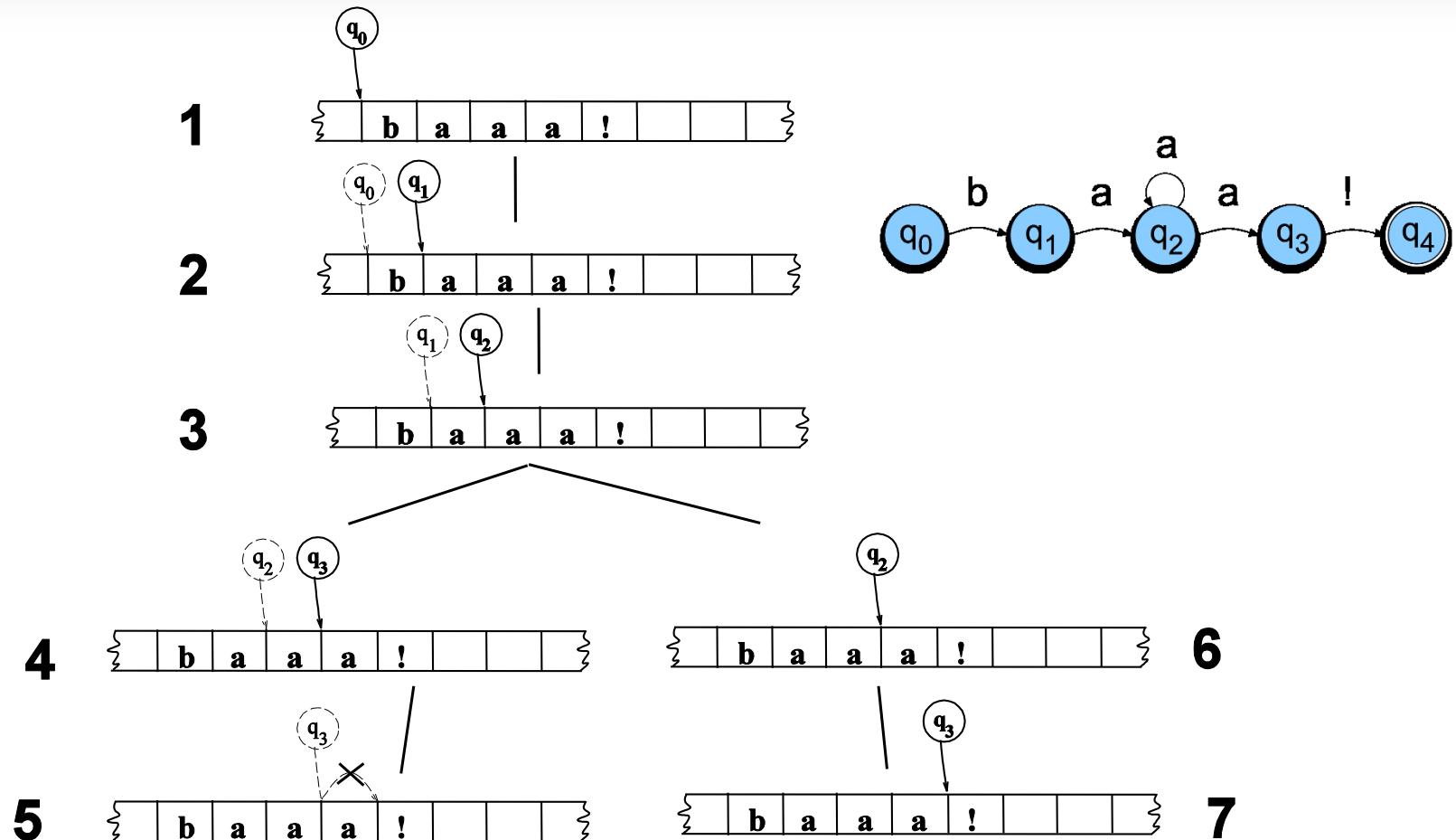
Example



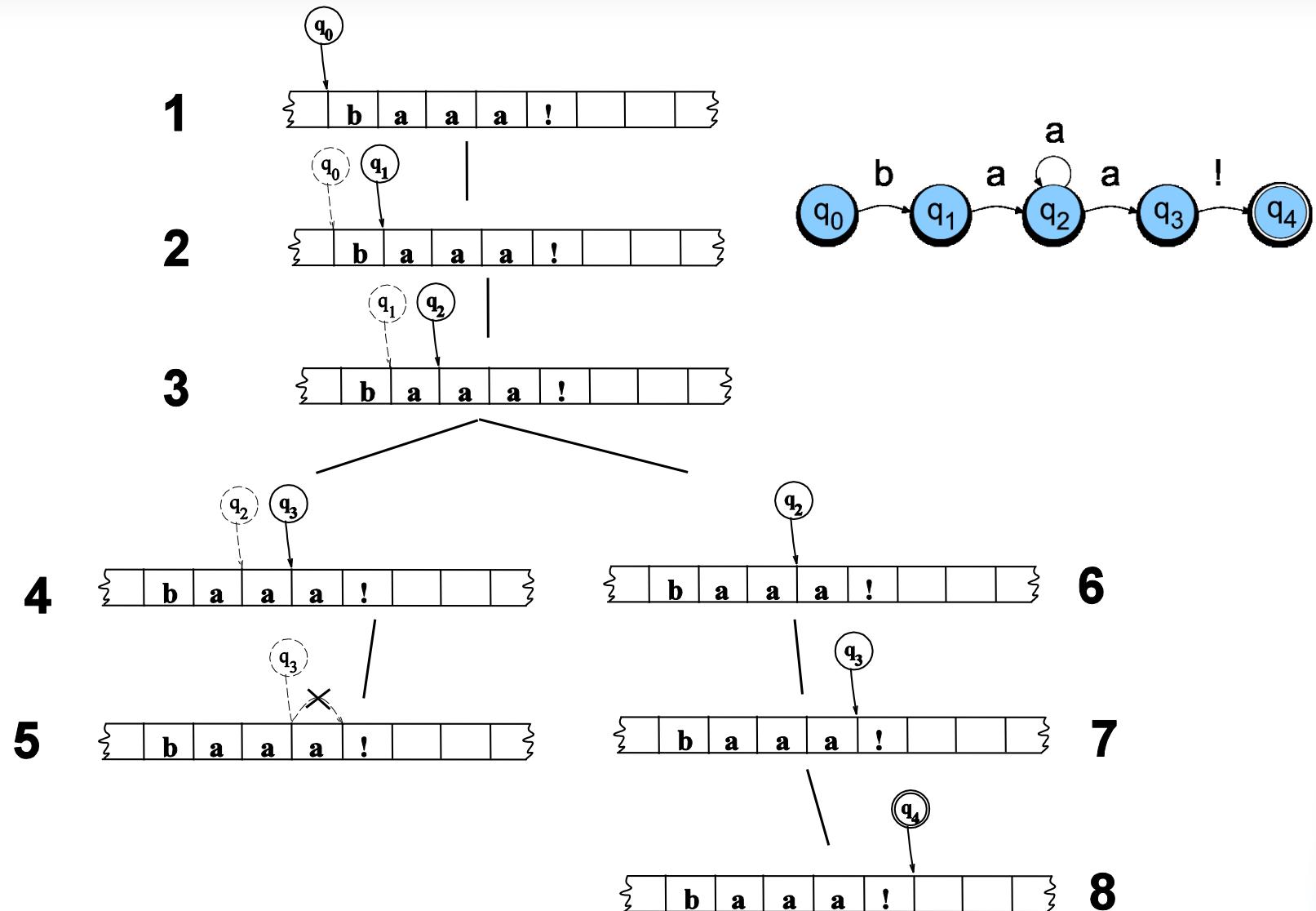
Example



Example



Example



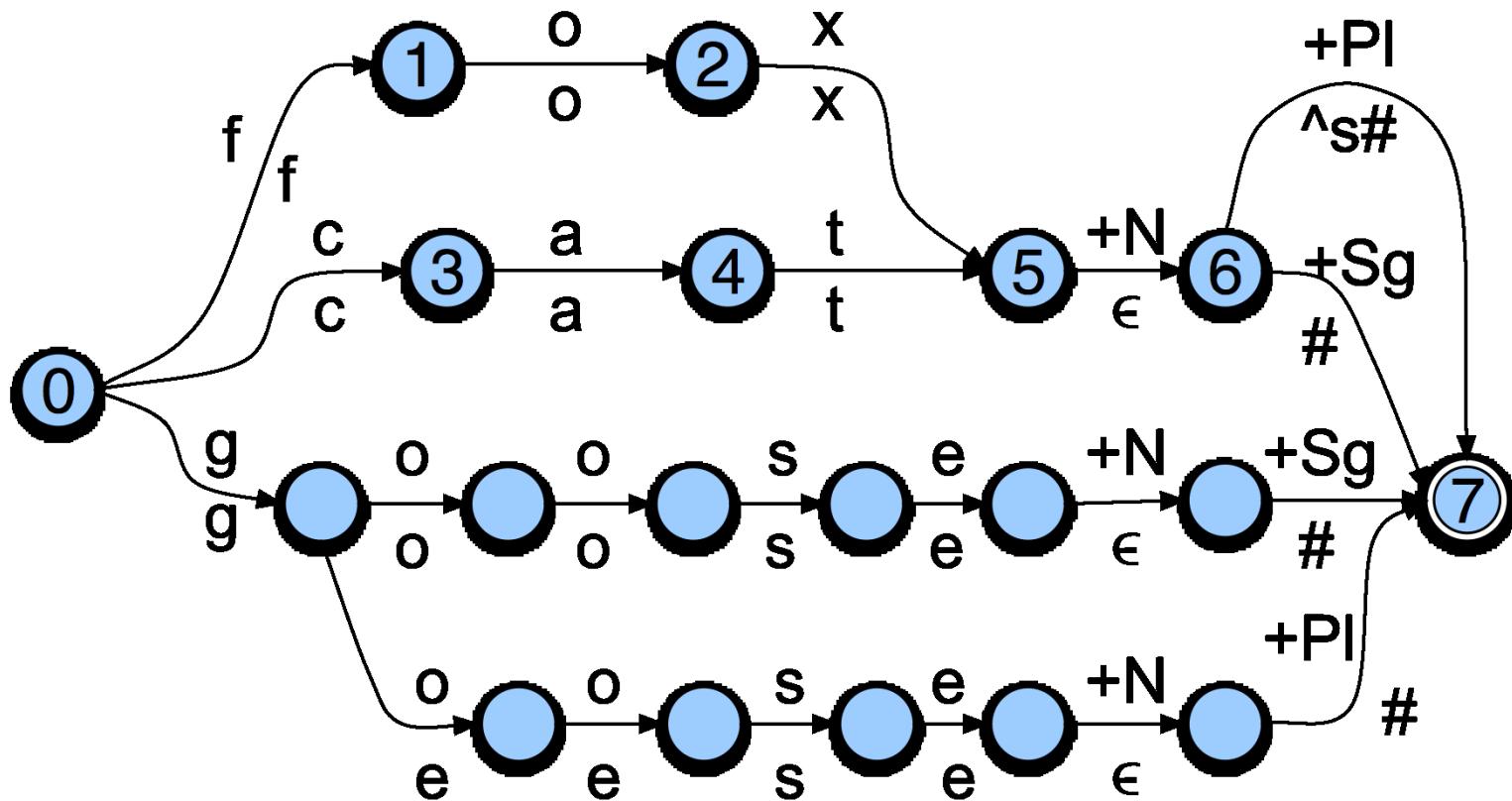
Key Points

- States in the search space are pairings of tape positions and states in the machine.
- By keeping track of as yet unexplored states, a recognizer can systematically explore all the paths through the machine given an input.

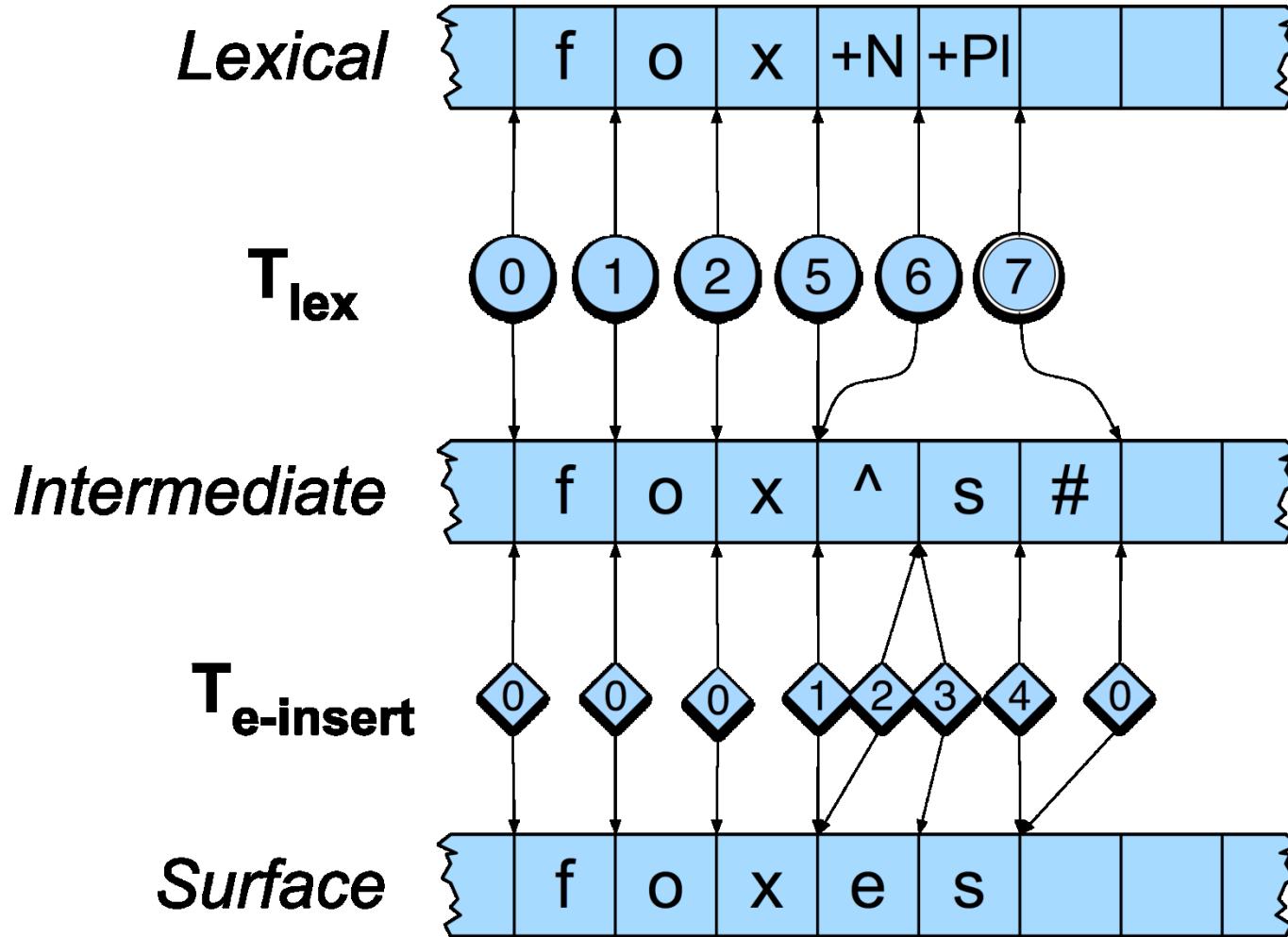
FSTs (Contd)

FST Fragment: Lexical to Intermediate

- ^ is morpheme boundary; # is word boundary



Putting Them Together



Practical Uses

- This kind of parsing is normally called morphological analysis
- Can be
 - An important stand-alone component of an application (spelling correction, information retrieval, part-of-speech tagging,...)
 - Or simply a link in a chain of processing (machine translation, parsing,...)

FST-based Tokenization

```
#!/usr/bin/perl

$letternumber = "[A-Za-z0-9]";
$notletter = "[^A-Za-z0-9]";
$alwayssep = "[\\?!()\";/\\[\\]]";
$clitic = "('':|-|'S'|'D'|'M'|'LL'|'RE|'VE|'N'T|'s|'d|'m|'ll|'re|'ve|'n't)";

$abbr{"Co."} = 1; $abbr{"Dr."} = 1; $abbr{"Jan."} = 1; $abbr{"Feb."} = 1;

while ($line = <>){ # read the next line from standard input

    # put whitespace around unambiguous separators
    $line =~ s/$alwayssep/ $& /g;

    # put whitespace around commas that aren't inside numbers
    $line =~ s/([^\0-9]),/$1 , /g;
    $line =~ s/,([^\0-9])/ , $1/g;

    # distinguish single quotes from apostrophes by
    # segmenting off single quotes not preceded by letter
    $line =~ s/'/$& /g;
    $line =~ s/($notletter)'/ $1 '/g;

    # segment off unambiguous word-final clitics and punctuation
    $line =~ s/$clitic$/ $& /g;
    $line =~ s/$clitic($notletter)/ $1 $2/g;

    # now deal with periods. For each possible word
    @possiblewords=split(/\s+/, $line);
    foreach $word (@possiblewords) {
        # if it ends in a period,
        if (($word =~ $letternumber\.) )
            && !($abbr{$word}) # and isn't on the abbreviation list
            # and isn't a sequence of letters and periods (U.S.)
            # and doesn't resemble an abbreviation (no vowels: Inc.)
            && !($word =~
                  ^([A-Za-z]\.([A-Za-z]\.)*|[A-Z][bcdfghj-nptvxz]+\.)$/) {
                # then segment off the period
                $word =~ s/\.$/ \./;
            }
        # expand clitics
        $word =~ s/'ve/have/;
        $word =~ s/'m/am/;
        print $word, " ";
    }
    print "\n";
}
```

Porter Stemmer (1980)

- Common algorithm for stemming English
- Conventions + 5 phases of reductions
 - ◆ phases applied sequentially
 - ◆ each phase consists of a set of commands
 - ◆ sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Porter Stemmer (1980)

- Standard, very popular and usable **stemmer** (IR, IE) – identify a word's **stem**
- Sequence of cascaded rewrite rules, e.g.
 - ◆ IZE → ε (e.g. unionize → union)
 - ◆ CY → T (e.g. frequency → frequent)
 - ◆ ING → ε , if stem contains vowel (motoring → motor)
- Can be implemented as a lexicon-free FST (many implementations available on the web)
- <http://text-processing.com/demo/stem/>

Eliza

User₁: Men are all alike.

ELIZA₁: IN WHAT WAY

User₂: They're always bugging us about something or other.

ELIZA₂: CAN YOU THINK OF A SPECIFIC EXAMPLE

User₃: Well, my boyfriend made me come here.

ELIZA₃: YOUR BOYFRIEND MADE YOU COME HERE

User₄: He says I'm depressed much of the time.

ELIZA₄: I AM SORRY TO HEAR YOU ARE DEPRESSED

Eliza FST

```
s/.* I'M (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/  
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/  
s/.* all .*/IN WHAT WAY/  
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
```

RelNoun: Nominal Open IE

Constructions	Phrase	Extraction
Verb1	Francis Collins is the director of NIH	(Francis Collins; is the director of; NIH)
Verb2	the director of NIH is Francis Collins	(Francis Collins; is the director of; NIH)
Appositive1	Francis Collins, the director of NIH	(Francis Collins; [is] the director of; NIH)
Appositive2	the director of NIH, Francis Collins,	(Francis Collins; [is] the director of; NIH)
Appositive3	Francis Collins, the NIH director	(Francis Collins; [is] the director [of]; NIH)
AppositiveTitle	Francis Collins, the director,	(Francis Collins; [is]; the director)
CompoundNoun	<i>NIH director Francis Collins</i>	(Francis Collins; [is] director [of]; NIH)
Possessive	NIH's director Francis Collins	(Francis Collins; [is] director [of]; NIH)
PossessiveAppositive	NIH's director, Francis Collins	(Francis Collins; [is] director [of]; NIH)
AppositivePossessive	Francis Collins, NIH's director	(Francis Collins; [is] director [of]; NIH)
PossessiveVerb	NIH's director is Francis Collins	(Francis Collins; is director [of]; NIH)
VerbPossessive	Francis Collins is NIH's director	(Francis Collins; is director [of]; NIH)

Compound Noun Extraction

Baseline

- NIH Director Francis Collins

(Francis Collins, is the Director of, NIH)

- Challenges

- ◆ New York Banker Association

ORG NAMES

- ◆ German Chancellor Angela Merkel

DEMONYMS

- ◆ Prime Minister Modi

COMPOUND
RELATIONAL NOUNS

- ◆ GM Vice Chairman Bob Lutz

Rule-Based System

- Classifies and filters orgs
- List of demonyms
 - ◆ appropriate location conversion
- Bootstrap a list of relational noun *prefixes*
 - ◆ vice, ex, health, ...

Summing Up

- Regular expressions and FSAs can represent subsets of natural language as well as regular languages
 - ◆ Both representations may be difficult for humans to use for any real subset of a language
 - ◆ But quick, powerful and easy to use for small problems
- Finite state transducers and rules are common ways to incorporate linguistic ideas in NLP for small applications
- Particularly useful for no data setting

Finite-state methods for morphology

Julia Hockenmaier

Today's lecture

What are words? How many words are there?

What is the **structure of words**?

(in English, Chinese, Arabic,...)

Morphology: the area of linguistics that deals with this.

How can we identify the structure of words?

We need to build a **morphological analyzer** (parser).

We will use **finite-state transducers** for this task.

Finite-State Automata and Regular Languages

(Review)

Morphology: What is a word?

A Turkish word

uygarlaştıramadıklarımızdanmışsınızcasına
uygar_laş_tır_ama_dık_lar_ımız_dan_mış_sınız_casına

“as if you are among those whom we were not able to civilize (=cause to become civilized)”

uygar: *civilized*

laş: *become*

tır: *cause somebody to do something*

ama: *not able*

dık: *past participle*

lar: *plural*

ımız: *1st person plural possessive (our)*

dan: *among (ablative case)*

mış: *past*

sınız: *2nd person plural (you)*

casına: *as if (forms an adverb from a verb)*

K. Oflazer pc to J&M

Basic word classes (parts of speech)

Content words (open-class):

- Nouns: student, university, knowledge,...
- Verbs: write, learn, teach,...
- Adjectives: difficult, boring, hard,
- Adverbs: easily, repeatedly,...

Function words (closed-class):

- Prepositions: in, with, under,...
- Conjunctions: and, or,...
- Determiners: a, the, every,...

How many words are there?

The Unix command “`wc -w`” counts the words in a file.

```
> cat example.txt
```

This company isn't New York-based anymore

We moved to Chicago

```
> wc -w example.txt
```

10 example.txt

“`wc -w`” uses blanks to identify words:

This₁ company₂ isn't₃ New₄ York-based₅ anymore₆

We₇ moved₈ to₉ Chicago₁₀

Words aren't just defined by blanks

Problem 1: Compounding

“ice cream”, “website”, “web site”, “New York-based”

Problem 2: Other writing systems have no blanks

Chinese: 我开始写小说 = 我 开始 写 小说
 I start(ed) writing novel(s)

Problem 3: Clitics

English: “doesn’t”, “I’m”,

Italian: “dirglielo” = dir + gli(e) + lo
 tell + him + it

How many words are there?

Of course he wants to take the advanced course too.
He already took two beginners' courses.

This is a bad question. Did I mean:

How many **word tokens** are there?
(16 to 19, depending on how we count punctuation)

How many **word types** are there?
(i.e. How many different words are there?
Again, this depends on how you count, but it's
usually much less than the number of tokens)

How many words are there?

Of course he wants to take the advanced course too.
He already took two beginners' courses.

The same (underlying) word can take different forms:
course/courses, take/took

We distinguish concrete word forms (take, taking)
from abstract lemmas or dictionary forms (take)

Different words may be spelled/pronounced the same:
of course vs. advanced course
two vs. too

How many different words are there?

Inflection creates different forms of the same word:

Verbs: to be, being, I am, you are, he is, I was,

Nouns: one book, two books

Derivation creates different words from the same lemma:

grace ⇒ disgrace ⇒ disgraceful ⇒ disgracefully

Compounding combines two words into a new word:

cream ⇒ ice cream ⇒ ice cream cone ⇒ ice cream cone bakery

Word formation is productive:

New words are subject to all of these processes:

Google ⇒ Googler, to google, to ungoogle, to misgoogle, googlification, ungooglification, googlified, Google Maps, Google Maps service,...

Inflectional morphology in English

Verbs:

- Infinitive/present tense: walk, go
- 3rd person singular present tense (s-form): walks, goes
- Simple past: walked, went
- Past participle (ed-form): walked, gone
- Present participle (ing-form): walking, going

Nouns:

- Number: singular (book) vs. plural (books)
- Plural: books
- Possessive (~ genitive case): book's, books
- Personal pronouns inflect for person, number, gender, case:
I saw him; he saw me; you saw her; we saw them; they saw us.

Derivational morphology

Nominalization:

V + -ation: computerization

V+ -er: killer

Adj + -ness: fuzziness

Negation:

un-: undo, unseen, ...

mis-: mistake,...

Adjectivization:

V+ -able: doable

N + -al: national

Morphemes: stems, affixes

dis-grace-ful-ly
prefix-stem-suffix-suffix

Many word forms consist of a **stem** plus a number of **affixes** (*prefixes or suffixes*)

Infixes are inserted inside the stem.

Circumfixes (German gesehen) surround the stem

Morphemes: the smallest (meaningful/grammatical) parts of words.

Stems (grace) are often **free morphemes**.

Free morphemes can occur by themselves as words.

Affixes (dis-, -ful, -ly) are usually **bound morphemes**.

Bound morphemes have to combine with others to form words.

Morphemes and morphs

There are many *irregular* word forms:

- Plural nouns add -s to singular: book-books,
but: box-boxes, fly-flies, child-children
- Past tense verbs add -ed to infinitive: walk-walked,
but: like-liked, leap-leapt

Morphemes are abstract categories

Examples: plural morpheme, past tense morpheme

The same morpheme (e.g. for plural nouns) can be realized as different surface forms (morphs):

-s/-es/-ren

Allomorphs: two different realizations (-s/-es/-ren)
of the same underlying morpheme (plural)

Morphological parsing and generation

Morphological parsing

disgracefully
dis grace ful ly
prefix stem suffix suffix
NEG grace+N +ADJ +ADV

Morphological generation

Generate possible English words:

grace, graceful, gracefully
disgrace, disgraceful, disgracefully,
ungraceful, ungracefully,
undisgraceful, undisgracefully,...

Don't generate impossible English words:

*gracelyful, *gracefuly, *disungracefully,...

Review: Finite-State Automata and Regular Languages

Formal languages

An alphabet Σ is a set of symbols:

e.g. $\Sigma = \{a, b, c\}$

A string ω is a sequence of symbols, e.g $\omega = abcb$.

The empty string ε consists of zero symbols.

The Kleene closure Σ^* ('sigma star') is the (infinite) set of all strings that can be formed from Σ :

$$\Sigma^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, \dots\}$$

A language $L \subseteq \Sigma^*$ over Σ is also a set of strings.

Typically we only care about proper subsets of Σ^* ($L \subset \Sigma^*$).

Automata and languages

An **automaton** is an abstract model of a computer which reads an **input string**, and **changes its internal state** depending on the current input symbol. It can either **accept or reject** the input string.

Every automaton defines a language
(the set of strings it accepts).

Different automata define different language classes:

- **Finite-state** automata define **regular** languages
- **Pushdown** automata define **context-free** languages
- **Turing machines** define **recursively enumerable** languages

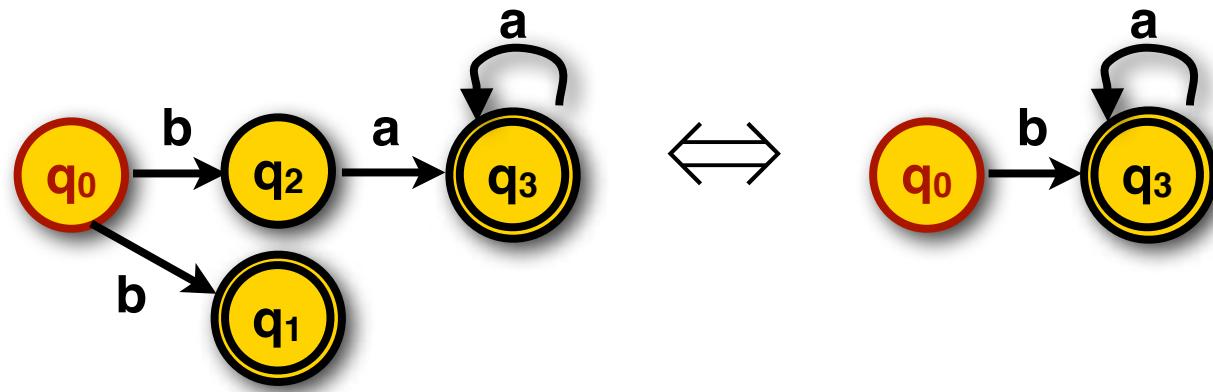
Finite State Automata (FSAs)

A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \dots, q_n\}$
- A finite alphabet Σ of input symbols (e.g. $\Sigma = \{a, b, c, \dots\}$)
- A designated start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ :
 - The transition function for a deterministic (D)FSA: $Q \times \Sigma \rightarrow Q$
$$\delta(q, w) = q' \quad \text{for } q, q' \in Q, w \in \Sigma$$
If the current state is q and the current input is w , go to q'
 - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$
$$\delta(q, w) = Q' \quad \text{for } q \in Q, Q' \subseteq Q, w \in \Sigma$$
If the current state is q and the current input is w , go to any $q' \in Q'$

Finite State Automata (FSAs)

Every NFA can be transformed into an equivalent DFA:



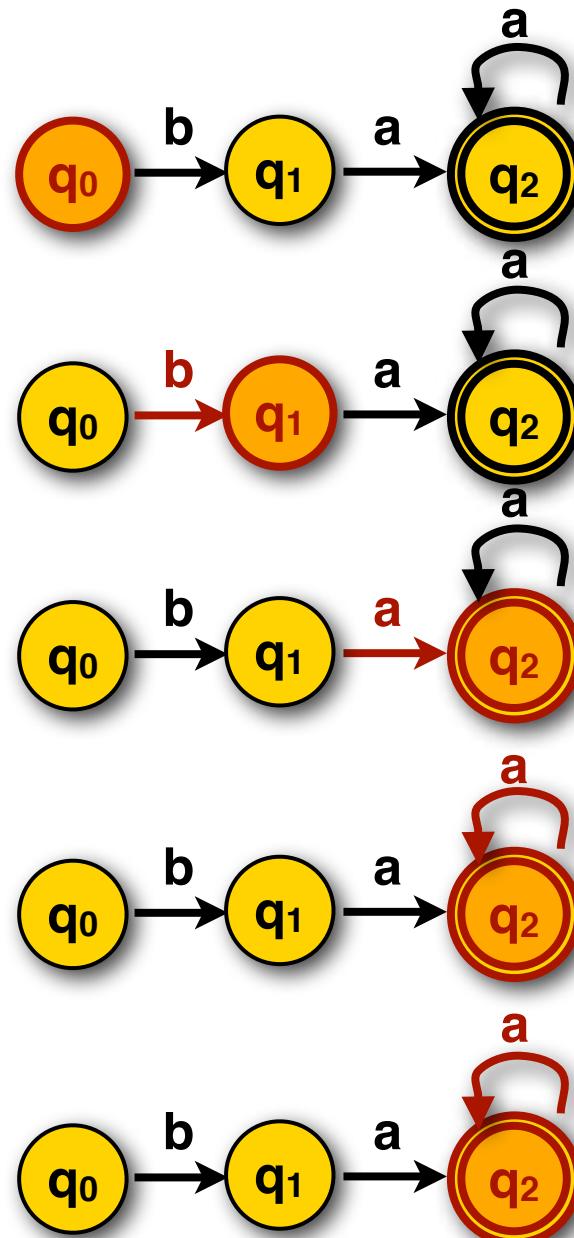
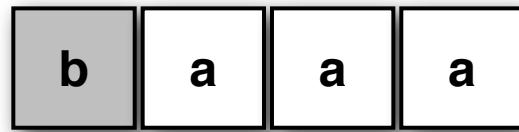
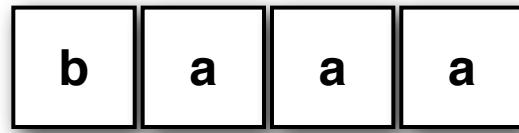
Recognition of a string w with a DFA is linear in the length of w

Finite-state automata define the class of **regular languages**

$L_1 = \{ a^n b^m \} = \{ ab, aab, abb, aaab, abb, \dots \}$ is a regular language,

$L_2 = \{ a^n b^n \} = \{ ab, aabb, aaabbb, \dots \}$ is not (it's context-free).

You cannot construct an FSA that accepts all the strings in L_2 and nothing else.



Regular Expressions

Simple patterns:

- **Standard characters** match themselves: ‘*a*’, ‘*I*’
- **Character classes**: ‘[*abc*]’, ‘[0-9]’, **negation**: ‘[^aeiou]’
(Predefined: \s (whitespace), \w (alphanumeric), etc.)
- **Any character** (except newline) is matched by ‘.’

Complex patterns: (e.g. ^[A-Z][a-z]+\\s)

- **Group**: ‘(...)'
- **Repetition**: 0 or more times: ‘*’, 1 or more times: ‘+’
- **Disjunction**: ‘...|...’
- **Beginning of line** ‘^’ and **end of line** ‘\$’

Python: Regular Expressions

```
>>> import re                                %% Import re package
>>> ex = re.compile('a.c')                   %% '...': reg.expression
>>> m = ex.search('ab')                     %% Does 'ab' contain ex?
>>> print m                                 %% No.

None

>>> m = ex.search('abc')                   %% Does 'abc' contain ex?
>>> print m                                 %% Yes.

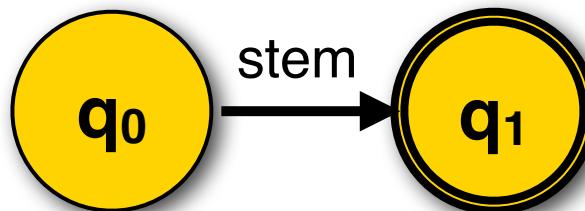
<_sre.SRE_Match object at 0x70640>
```

<http://docs.python.org/dev/howto/regex.html>
<http://docs.python.org/lib/module-re.html>

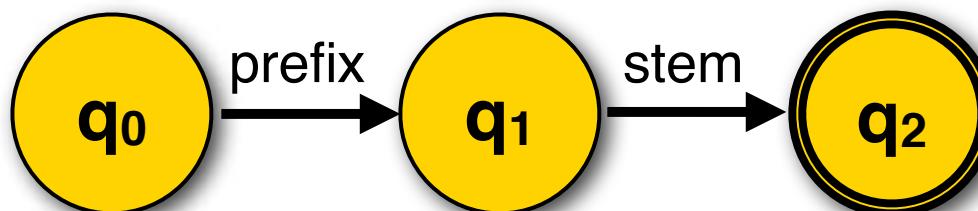
Finite-state methods for morphology

Finite state automata for morphology

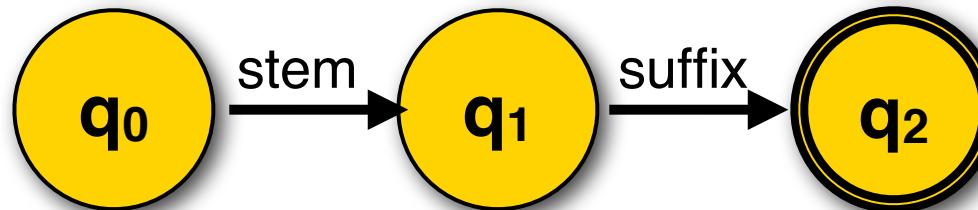
grace:



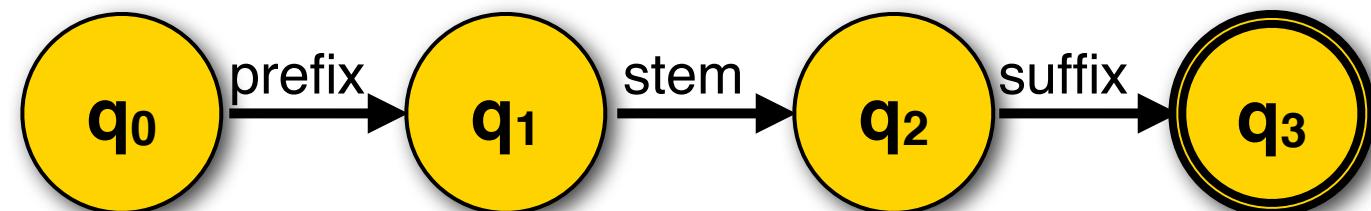
dis-grace:



grace-ful:

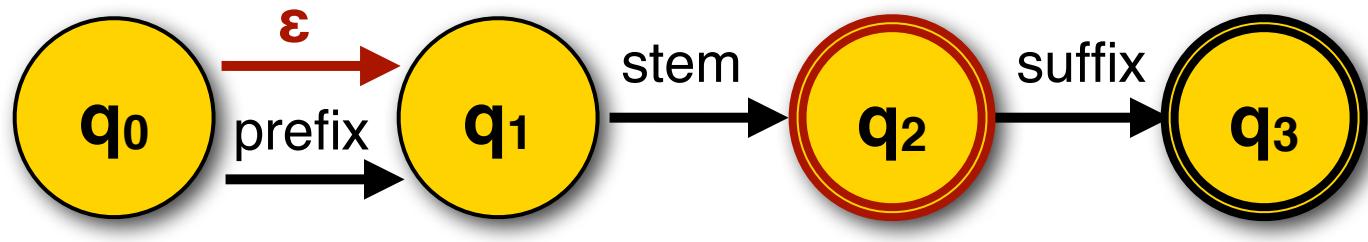


dis-grace-ful:



Union: merging automata

grace,
dis-grace,
grace-ful,
dis-grace-ful



Stem changes

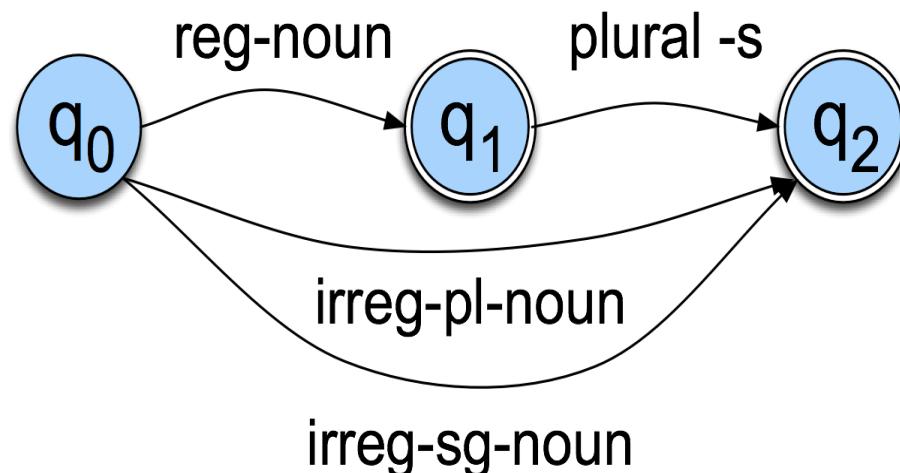
Some irregular words require stem changes:

Past tense verbs:

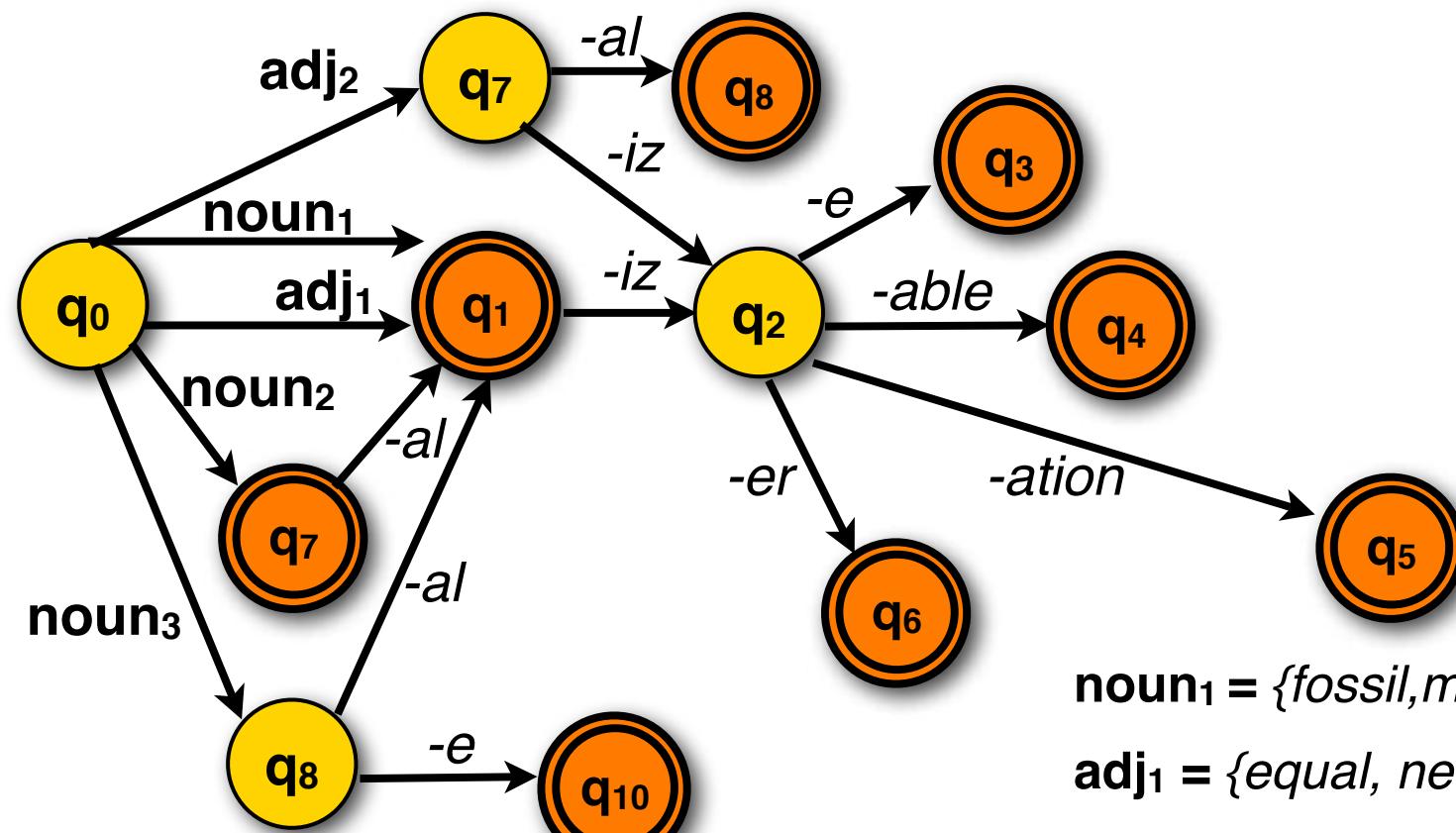
teach-taught, go-went, write-wrote

Plural nouns:

mouse-mice, foot-feet, wife-wives



FSAs for derivational morphology



noun₁ = {fossil, mineral, ...}

adj₁ = {equal, neutral}

adj₂ = {minim, maxim}

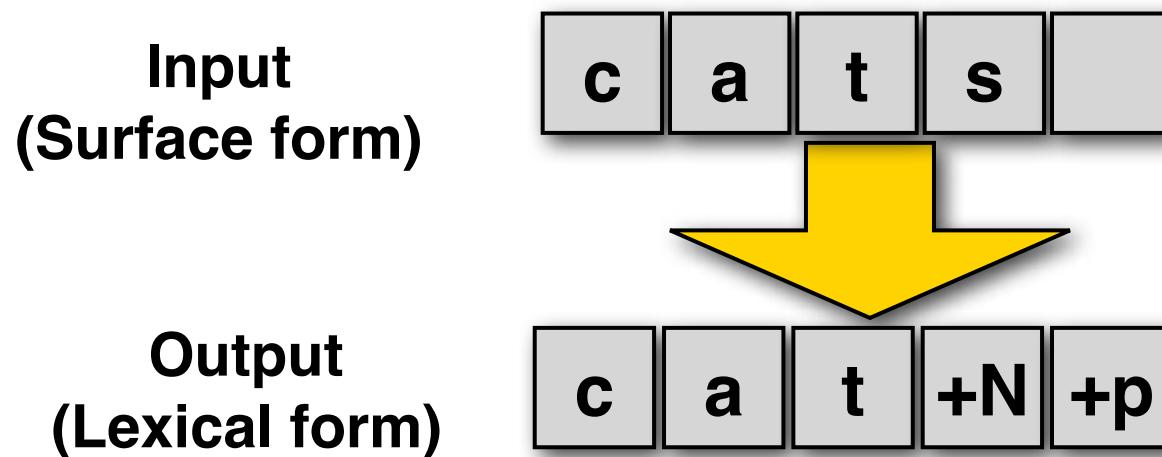
noun₂ = {nation, form, ...}

noun₃ = {natur, structur, ...}

Recognition vs. Analysis

FSAs can recognize (**accept**) a string, but they don't tell us its internal structure.

We need is a machine that maps (**transduces**) the input string into an output string that encodes its structure:



Finite-state transducers

A **finite-state transducer** $T = \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$ consists of:

- A finite **set of states** $Q = \{q_0, q_1, \dots, q_n\}$
 - A finite alphabet Σ of **input symbols** (e.g. $\Sigma = \{a, b, c, \dots\}$)
 - A finite alphabet Δ of **output symbols** (e.g. $\Delta = \{+N, +pl, \dots\}$)
 - A designated **start state** $q_0 \in Q$
 - A set of **final states** $F \subseteq Q$
 - A **transition function** $\delta: Q \times \Sigma \rightarrow 2^Q$
 $\delta(q, w) = Q'$ for $q \in Q, Q' \subseteq Q, w \in \Sigma$
 - **An output function** $\sigma: Q \times \Sigma \rightarrow \Delta^*$
 $\sigma(q, w) = \omega$ for $q \in Q, w \in \Sigma, \omega \in \Delta^*$
- If the current state is q and the current input is w , write ω .

Finite-state transducers

An FST $T = L_{in} \times L_{out}$ defines a **relation between two regular languages L_{in} and L_{out} :**

$$L_{in} = \{\mathbf{cat}, \mathbf{cats}, \mathbf{fox}, \mathbf{foxes}, \dots\}$$
$$L_{out} = \{cat+N+sg, cat+N+pl, fox+N+sg, fox+N+PL \dots\}$$


$$T = \{ <\mathbf{cat}, cat+N+sg>, \\ <\mathbf{cats}, cat+N+pl>, \\ <\mathbf{fox}, fox+N+sg>, \\ <\mathbf{foxes}, fox+N+pl> \}$$

Some FST operations

Inversion T^{-1} :

The inversion (T^{-1}) of a transducer switches input and output labels.

*This can be used to switch from **parsing** words to **generating** words.*

Composition ($T \circ T'$): (*Cascade*)

Two transducers $T = L_1 \times L_2$ and $T' = L_2 \times L_3$ can be composed into a third transducer $T'' = L_1 \times L_3$.

*Sometimes **intermediate representations** are useful*

English spelling rules

English spelling (orthography) is funny:

The underlying morphemes (*plural-s*, etc.) can have different orthographic surface realizations (-s, -es)

Spelling changes at morpheme boundaries:

- E-insertion: fox +s = fox**e**s
- E-deletion: make**e** +ing = making

Intermediate representations

English plural -s: cat \Rightarrow cats dog \Rightarrow dogs
but: fox \Rightarrow foxes, bus \Rightarrow buses buzz \Rightarrow buzzes

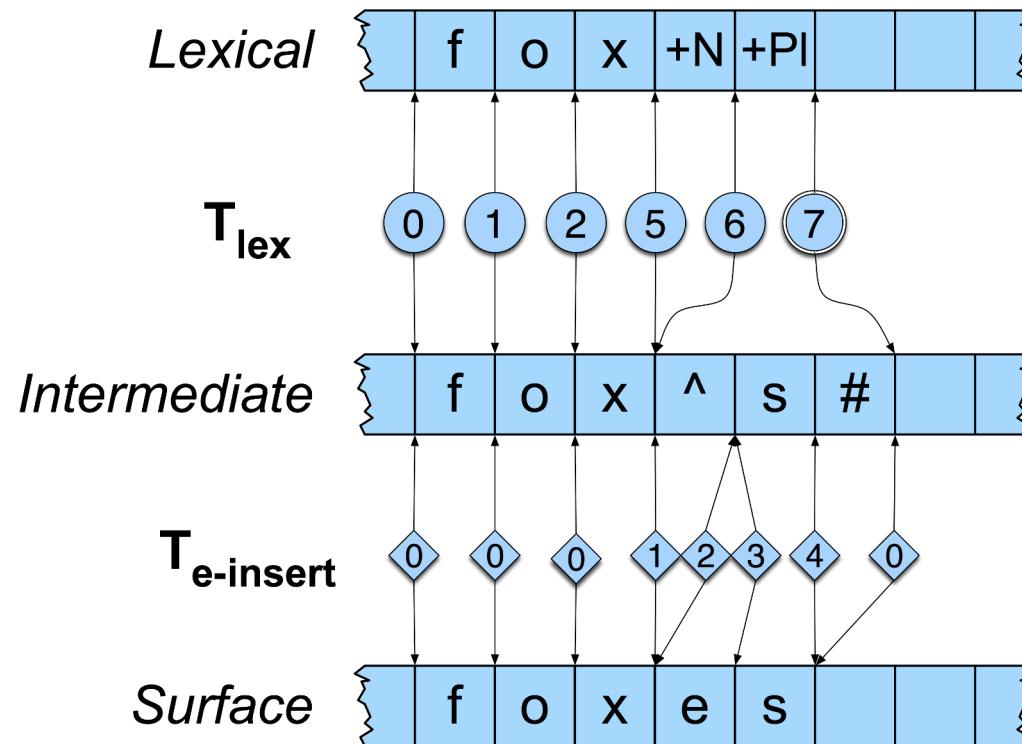
We define an **intermediate representation** which captures morpheme boundaries (^) and word boundaries (#):

<i>Lexicon:</i>	<i>cat+N+PL</i>	<i>fox+N+PL</i>
\Rightarrow <i>Intermediate representation:</i>	<i>cat^s#</i>	<i>fox^s#</i>
\Rightarrow <i>Surface string:</i>	<i>cats</i>	<i>foxes</i>

Intermediate-to-Surface Spelling Rule:

If plural '**s**' follows a morpheme ending in '**x**', '**z**' or '**s**', insert '**e**'.

FST composition/cascade:

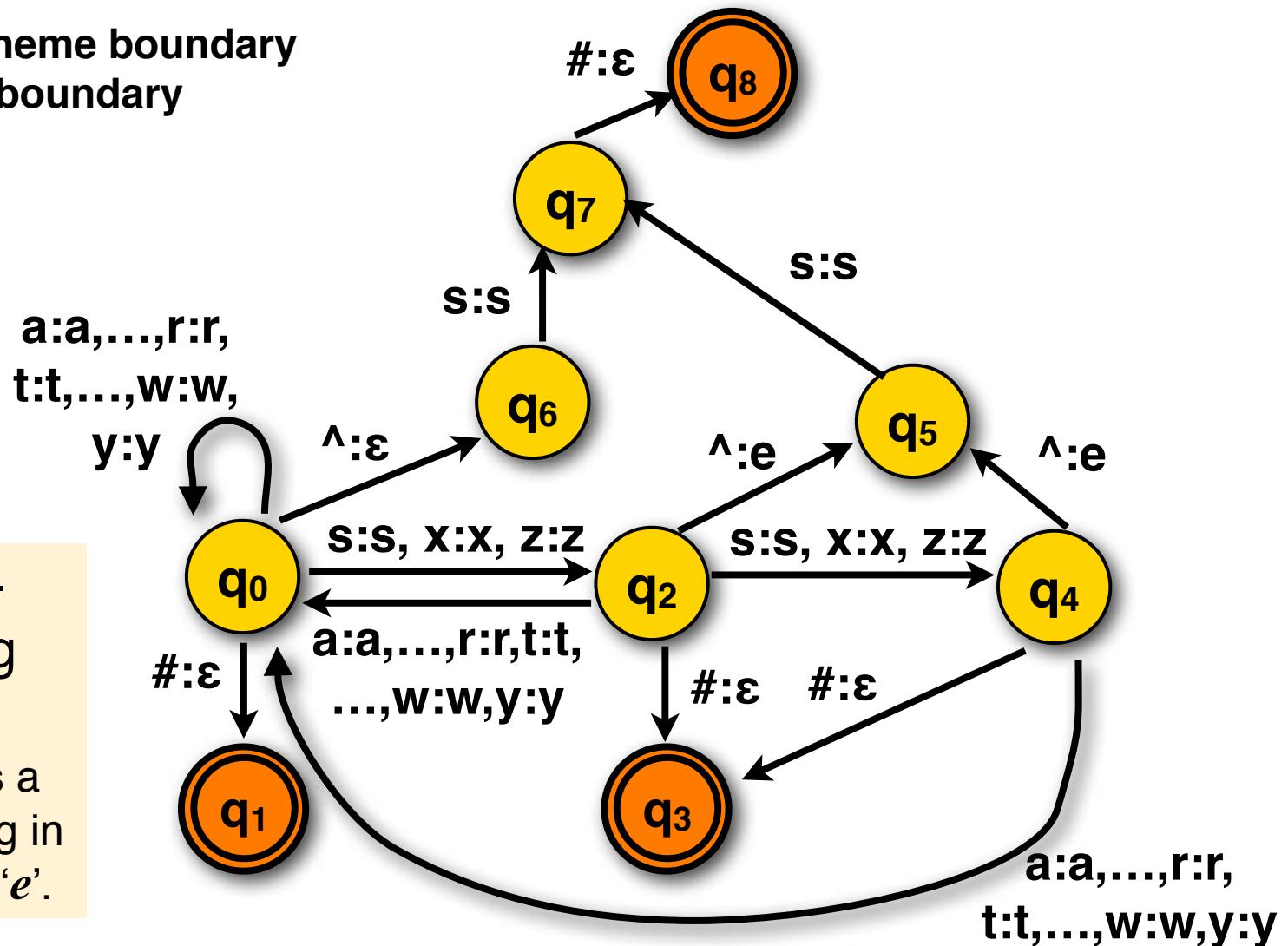


FST: intermediate to surface level

\wedge = morpheme boundary
 $\#$ = word boundary

Intermediate-to-Surface Spelling Rule:

If plural 's' follows a morpheme ending in 'x', 'z' or 's', insert 'e'.



Dealing with ambiguity

book: $book + N + sg$ or $book + V?$

Generating words is generally unambiguous, but
analyzing words often requires disambiguation.

Efficiency problem:

Not every nondeterministic FST can be translated into
a deterministic one!

A computer therapist?

Computer: TELL ME MORE ABOUT YOUR FAMILY

Human: My mother takes care of me.

Computer: WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

Human: My father.

Computer: YOUR FATHER

Human: You are like my father in some ways.

Computer: WHAT RESEMBLANCE DO YOU SEE

Human: You are not very aggressive but I think you don't want me to notice that.

Computer: WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

Human: You don't argue with me.

Computer: WHY DO YOU THINK I DON'T ARGUE WITH YOU

Human: You are afraid of me.

Weizenbaum (1966), ELIZA.

ELIZA as a FST cascade

Human: You don't argue with me.

Computer: WHY DO YOU THINK I DON'T ARGUE WITH YOU

1. Replace **you** with **I** and **me** with **you**:

I don't argue with you.

2. Replace <...> with **Why do you think <...>**:

Why do you think I don't argue with you.

What about compounds?

Compounds have hierarchical structure:

((ice cream) cone) bakery

not (ice ((cream cone) bakery))

((computer science) (graduate student))

not (computer ((science graduate) student))

We will need context-free grammars to capture this underlying structure.

CHAPTER

6

Vector Semantics and Embeddings

荃者所以在鱼，得鱼而忘荃 Nets are for fish;

Once you get the fish, you can forget the net.

言者所以在意，得意而忘言 Words are for meaning;

Once you get the meaning, you can forget the words

庄子(Zhuangzi), Chapter 26

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or saber-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. *Thylacosmilus* was a marsupial whereas *Smilodon* was a placental mammal, but *Thylacosmilus* had the same long upper canines and, like *Smilodon*, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many examples of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).



The role of context is also important in the similarity of a less biological kind of organism: the word. Words that occur in *similar contexts* tend to have *similar meanings*. This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**. The hypothesis was first formulated in the 1950s by linguists like Joos (1950), Harris (1954), and Firth (1957), who noticed that words which are synonyms (like *oculist* and *eye-doctor*) tended to occur in the same environment (e.g., near words like *eye* or *examined*) with the amount of meaning difference between two words “corresponding roughly to the amount of difference in their environments” (Harris, 1954, 157).

distributional hypothesis

vector semantics embeddings

representation learning

In this chapter we introduce **vector semantics**, which instantiates this linguistic hypothesis by learning representations of the meaning of words, called **embeddings**, directly from their distributions in texts. These representations are used in every natural language processing application that makes use of meaning, and the **static embeddings** we introduce here underlie the more powerful dynamic or **contextualized embeddings** like **BERT** that we will see in Chapter 11.

These word representations are also the first example in this book of **representation learning**, automatically learning useful representations of the input text. Finding such **self-supervised** ways to learn representations of the input, instead of creating representations by hand via **feature engineering**, is an important focus of NLP research (Bengio et al., 2013).

6.1 Lexical Semantics

Let's begin by introducing some basic principles of word meaning. How should we represent the meaning of a word? In the n-gram models of Chapter 3, and in classical NLP applications, our only representation of a word is as a string of letters, or an index in a vocabulary list. This representation is not that different from a tradition in philosophy, perhaps you've seen it in introductory logic classes, in which the meaning of words is represented by just spelling the word with small capital letters; representing the meaning of "dog" as DOG, and "cat" as CAT, or by using an apostrophe (DOG').

Representing the meaning of a word by capitalizing it is a pretty unsatisfactory model. You might have seen a version of a joke due originally to semanticist Barbara Partee ([Carlson, 1977](#)):

Q: What's the meaning of life?

A: LIFE'

Surely we can do better than this! After all, we'll want a model of word meaning to do all sorts of things for us. It should tell us that some words have similar meanings (*cat* is similar to *dog*), others are antonyms (*cold* is the opposite of *hot*), some have positive connotations (*happy*) while others have negative connotations (*sad*). It should represent the fact that the meanings of *buy*, *sell*, and *pay* offer differing perspectives on the same underlying purchasing event (If I buy something from you, you've probably sold it to me, and I likely paid you). More generally, a model of word meaning should allow us to draw inferences to address meaning-related tasks like question-answering or dialogue.

lexical semantics

In this section we summarize some of these desiderata, drawing on results in the linguistic study of word meaning, which is called **lexical semantics**; we'll return to and expand on this list in Chapter 23 and Chapter 24.

Lemmas and Senses Let's start by looking at how one word (we'll choose *mouse*) might be defined in a dictionary (simplified from the online dictionary WordNet):

- mouse (N)
 - 1. any of numerous small rodents...
 - 2. a hand-operated device that controls a cursor...

lemma
citation form

wordform

Here the form *mouse* is the **lemma**, also called the **citation form**. The form *mouse* would also be the lemma for the word *mice*; dictionaries don't have separate definitions for inflected forms like *mice*. Similarly *sing* is the lemma for *sing*, *sang*, *sung*. In many languages the infinitive form is used as the lemma for the verb, so Spanish *dormir* "to sleep" is the lemma for *duermes* "you sleep". The specific forms *sung* or *carpets* or *sing* or *duermes* are called **wordforms**.

As the example above shows, each lemma can have multiple meanings; the lemma *mouse* can refer to the rodent or the cursor control device. We call each of these aspects of the meaning of *mouse* a **word sense**. The fact that lemmas can be **polysemous** (have multiple senses) can make interpretation difficult (is someone who types "mouse info" into a search engine looking for a pet or a tool?). Chapter 23 will discuss the problem of polysemy, and introduce **word sense disambiguation**, the task of determining which sense of a word is being used in a particular context.

Synonymy One important component of word meaning is the relationship between word senses. For example when one word has a sense whose meaning is

synonym identical to a sense of another word, or nearly identical, we say the two senses of those two words are **synonyms**. Synonyms include such pairs as

couch/sofa vomit/throw up filbert/hazelnut car/automobile

A more formal definition of synonymy (between words rather than senses) is that two words are synonymous if they are substitutable for one another in any sentence without changing the *truth conditions* of the sentence, the situations in which the sentence would be true. We often say in this case that the two words have the same **propositional meaning**.

While substitutions between some pairs of words like *car / automobile* or *water / H₂O* are truth preserving, the words are still not identical in meaning. Indeed, probably no two words are absolutely identical in meaning. One of the fundamental tenets of semantics, called the **principle of contrast** (Girard 1718, Bréal 1897, Clark 1987), states that a difference in linguistic form is always associated with some difference in meaning. For example, the word *H₂O* is used in scientific contexts and would be inappropriate in a hiking guide—*water* would be more appropriate—and this genre difference is part of the meaning of the word. In practice, the word *synonym* is therefore used to describe a relationship of approximate or rough synonymy.

Word Similarity While words don't have many synonyms, most words do have lots of *similar* words. *Cat* is not a synonym of *dog*, but *cats* and *dogs* are certainly similar words. In moving from synonymy to similarity, it will be useful to shift from talking about relations between word senses (like synonymy) to relations between words (like similarity). Dealing with words avoids having to commit to a particular representation of word senses, which will turn out to simplify our task.

The notion of word **similarity** is very useful in larger semantic tasks. Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of tasks like question answering, paraphrasing, and summarization. One way of getting values for word similarity is to ask humans to judge how similar one word is to another. A number of datasets have resulted from such experiments. For example the SimLex-999 dataset (Hill et al., 2015) gives values on a scale from 0 to 10, like the examples below, which range from near-synonyms (*vanish, disappear*) to pairs that scarcely seem to have anything in common (*hole, agreement*):

vanish	disappear	9.8
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

relatedness association The meaning of two words can be related in ways other than similarity. One such class of connections is called word **relatedness** (Budanitsky and Hirst, 2006), also traditionally called word **association** in psychology.

Consider the meanings of the words *coffee* and *cup*. Coffee is not similar to cup; they share practically no features (coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape). But coffee and cup are clearly related; they are associated by co-participating in an everyday event (the event of drinking coffee out of a cup). Similarly *scalpel* and *surgeon* are not similar but are related eventively (a surgeon tends to make use of a scalpel).

One common kind of relatedness between words is if they belong to the same **semantic field**. A semantic field is a set of words which cover a particular semantic

topic models

domain and bear structured relations with each other. For example, words might be related by being in the semantic field of hospitals (*surgeon, scalpel, nurse, anesthetic, hospital*), restaurants (*waiter, menu, plate, food, chef*), or houses (*door, roof, kitchen, family, bed*). Semantic fields are also related to **topic models**, like **Latent Dirichlet Allocation, LDA**, which apply unsupervised learning on large sets of texts to induce sets of associated words from text. Semantic fields and topic models are very useful tools for discovering topical structure in documents.

In Chapter 23 we'll introduce more relations between senses like **hypernymy** or **IS-A, antonymy** (opposites) and **meronymy** (part-whole relations).

semantic frame

Semantic Frames and Roles Closely related to semantic fields is the idea of a **semantic frame**. A semantic frame is a set of words that denote perspectives or participants in a particular type of event. A commercial transaction, for example, is a kind of event in which one entity trades money to another entity in return for some good or service, after which the good changes hands or perhaps the service is performed. This event can be encoded lexically by using verbs like *buy* (the event from the perspective of the buyer), *sell* (from the perspective of the seller), *pay* (focusing on the monetary aspect), or nouns like *buyer*. Frames have semantic roles (like *buyer, seller, goods, money*), and words in a sentence can take on these roles.

Knowing that *buy* and *sell* have this relation makes it possible for a system to know that a sentence like *Sam bought the book from Ling* could be paraphrased as *Ling sold the book to Sam*, and that Sam has the role of the *buyer* in the frame and Ling the *seller*. Being able to recognize such paraphrases is important for question answering, and can help in shifting perspective for machine translation.

connotations

Connotation Finally, words have *affective meanings* or **connotations**. The word *connotation* has different meanings in different fields, but here we use it to mean the aspects of a word's meaning that are related to a writer or reader's emotions, sentiment, opinions, or evaluations. For example some words have positive connotations (*happy*) while others have negative connotations (*sad*). Even words whose meanings are similar in other ways can vary in connotation; consider the difference in connotations between *fake, knockoff, forgery*, on the one hand, and *copy, replica, reproduction* on the other, or *innocent* (positive connotation) and *naive* (negative connotation). Some words describe positive evaluation (*great, love*) and others negative evaluation (*terrible, hate*). Positive or negative evaluation language is called **sentiment**, as we saw in Chapter 4, and word sentiment plays a role in important tasks like sentiment analysis, stance detection, and applications of NLP to the language of politics and consumer reviews.

sentiment

Early work on affective meaning (Osgood et al., 1957) found that words varied along three important dimensions of affective meaning:

valence: the pleasantness of the stimulus

arousal: the intensity of emotion provoked by the stimulus

dominance: the degree of control exerted by the stimulus

Thus words like *happy* or *satisfied* are high on valence, while *unhappy* or *annoyed* are low on valence. *Excited* is high on arousal, while *calm* is low on arousal. *Controlling* is high on dominance, while *awed* or *influenced* are low on dominance. Each word is thus represented by three numbers, corresponding to its value on each of the three dimensions:

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24

Osgood et al. (1957) noticed that in using these 3 numbers to represent the meaning of a word, the model was representing each word as a point in a three-dimensional space, a vector whose three dimensions corresponded to the word's rating on the three scales. This revolutionary idea that word meaning could be represented as a point in space (e.g., that part of the meaning of *heartbreak* can be represented as the point [2.45, 5.65, 3.58]) was the first expression of the vector semantics models that we introduce next.

6.2 Vector Semantics

vector semantics

Vector semantics is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning we saw in the previous section. The roots of the model lie in the 1950s when two big ideas converged: Osgood's 1957 idea mentioned above to use a point in three-dimensional space to represent the connotation of a word, and the proposal by linguists like Joos (1950), Harris (1954), and Firth (1957) to define the meaning of a word by its **distribution** in language use, meaning its neighboring words or grammatical environments. Their idea was that two words that occur in very similar distributions (whose neighboring words are similar) have similar meanings.

For example, suppose you didn't know the meaning of the word *ongchoi* (a recent borrowing from Cantonese) but you see it in the following contexts:

- (6.1) Ongchoi is delicious sauteed with garlic.
- (6.2) Ongchoi is superb over rice.
- (6.3) ...ongchoi leaves with salty sauces...

And suppose that you had seen many of these context words in other contexts:

- (6.4) ...spinach sauteed with garlic over rice...
- (6.5) ...chard stems and leaves are delicious...
- (6.6) ...collard greens and other salty leafy greens

The fact that *ongchoi* occurs with words like *rice* and *garlic* and *delicious* and *salty*, as do words like *spinach*, *chard*, and *collard greens* might suggest that *ongchoi* is a leafy green similar to these other leafy greens.¹ We can do the same thing computationally by just counting words in the context of *ongchoi*.

embeddings

The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived (in ways we'll see) from the distributions of word neighbors. Vectors for representing words are called **embeddings** (although the term is sometimes more strictly applied only to dense vectors like word2vec (Section 6.8), rather than sparse tf-idf or PPMI vectors (Section 6.3–Section 6.6)). The word “embedding” derives from its mathematical sense as a mapping from one space or structure to another, although the meaning has shifted; see the end of the chapter.

¹ It's in fact *Ipomoea aquatica*, a relative of morning glory sometimes called *water spinach* in English.



Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015) with colors added for explanation.

Fig. 6.1 shows a visualization of embeddings learned for sentiment analysis, showing the location of selected words projected down from 60-dimensional space into a two dimensional space. Notice the distinct regions containing positive words, negative words, and neutral function words.

The fine-grained model of word similarity of vector semantics offers enormous power to NLP applications. NLP applications like the sentiment classifiers of Chapter 4 or Chapter 5 depend on the same words appearing in the training and test sets. But by representing words as embeddings, classifiers can assign sentiment as long as it sees some words with *similar meanings*. And as we'll see, vector semantic models can be learned automatically from text without supervision.

In this chapter we'll introduce the two most commonly used models. In the **tf-idf** model, an important baseline, the meaning of a word is defined by a simple function of the counts of nearby words. We will see that this method results in very long vectors that are **sparse**, i.e. mostly zeros (since most words simply never occur in the context of others). We'll introduce the **word2vec** model family for constructing short, **dense** vectors that have useful semantic properties. We'll also introduce the **cosine**, the standard way to use embeddings to compute *semantic similarity*, between two words, two sentences, or two documents, an important tool in practical applications like question answering, summarization, or automatic essay grading.

6.3 Words and Vectors

“The most important attributes of a vector in 3-space are {Location, Location, Location}”
Randall Munroe, <https://xkcd.com/2358/>

Vector or distributional models of meaning are generally based on a **co-occurrence matrix**, a way of representing how often words co-occur. We'll look at two popular matrices: the term-document matrix and the term-term matrix.

6.3.1 Vectors and documents

term-document matrix

In a **term-document matrix**, each row represents a word in the vocabulary and each column represents a document from some collection of documents. Fig. 6.2 shows a small selection from a term-document matrix showing the occurrence of four words in four plays by Shakespeare. Each cell in this matrix represents the number of times

a particular word (defined by the row) occurs in a particular document (defined by the column). Thus *fool* appeared 58 times in *Twelfth Night*.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

vector space
model

The term-document matrix of Fig. 6.2 was first defined as part of the **vector space model** of information retrieval (Salton, 1971). In this model, a document is represented as a count vector, a column in Fig. 6.3.

vector

vector space
dimension

To review some basic linear algebra, a **vector** is, at heart, just a list or array of numbers. So *As You Like It* is represented as the list [1,114,36,20] (the first **column vector** in Fig. 6.3) and *Julius Caesar* is represented as the list [7,62,1,2] (the third column vector). A **vector space** is a collection of vectors, characterized by their **dimension**. In the example in Fig. 6.3, the document vectors are of dimension 4, just so they fit on the page; in real term-document matrices, the vectors representing each document would have dimensionality $|V|$, the vocabulary size.

The ordering of the numbers in a vector space indicates different meaningful dimensions on which documents vary. Thus the first dimension for both these vectors corresponds to the number of times the word *battle* occurs, and we can compare each dimension, noting for example that the vectors for *As You Like It* and *Twelfth Night* have similar values (1 and 0, respectively) for the first dimension.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.3 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each document is represented as a column vector of length four.

We can think of the vector for a document as a point in $|V|$ -dimensional space; thus the documents in Fig. 6.3 are points in 4-dimensional space. Since 4-dimensional spaces are hard to visualize, Fig. 6.4 shows a visualization in two dimensions; we've arbitrarily chosen the dimensions corresponding to the words *battle* and *fool*.

Term-document matrices were originally defined as a means of finding similar documents for the task of document **information retrieval**. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar. The vectors for the comedies *As You Like It* [1,114,36,20] and *Twelfth Night* [0,80,58,15] look a lot more like each other (more fools and wit than battles) than they look like *Julius Caesar* [7,62,1,2] or *Henry V* [13,89,4,3]. This is clear with the raw numbers; in the first dimension (battle) the comedies have low numbers and the others have high numbers, and we can see it visually in Fig. 6.4; we'll see very shortly how to quantify this intuition more formally.

A real term-document matrix, of course, wouldn't just have 4 rows and columns, let alone 2. More generally, the term-document matrix has $|V|$ rows (one for each word type in the vocabulary) and D columns (one for each document in the collec-

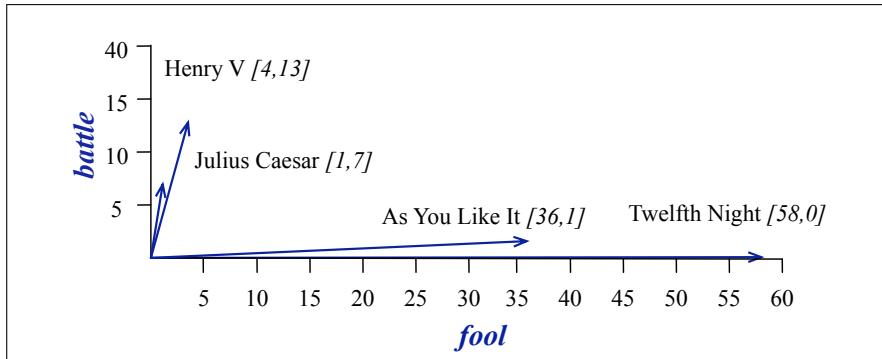


Figure 6.4 A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words *battle* and *fool*. The comedies have high values for the *fool* dimension and low values for the *battle* dimension.

information retrieval

tion); as we'll see, vocabulary sizes are generally in the tens of thousands, and the number of documents can be enormous (think about all the pages on the web).

Information retrieval (IR) is the task of finding the document d from the D documents in some collection that best matches a query q . For IR we'll therefore also represent a query by a vector, also of length $|V|$, and we'll need a way to compare two vectors to find how similar they are. (Doing IR will also require efficient ways to store and manipulate these vectors by making use of the convenient fact that these vectors are sparse, i.e., mostly zeros).

Later in the chapter we'll introduce some of the components of this vector comparison process: the tf-idf term weighting, and the cosine similarity metric.

row vector

6.3.2 Words as vectors: document dimensions

We've seen that documents can be represented as vectors in a vector space. But vector semantics can also be used to represent the meaning of *words*. We do this by associating each word with a word vector—a **row vector** rather than a column vector, hence with different dimensions, as shown in Fig. 6.5. The four dimensions of the vector for *fool*, [36,58,1,4], correspond to the four Shakespeare plays. Word counts in the same four dimensions are used to form the vectors for the other 3 words: *wit*, [20,15,2,3]; *battle*, [1,0,7,13]; and *good* [114,80,62,89].

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.5 The term-document matrix for four words in four Shakespeare plays. The red boxes show that each word is represented as a row vector of length four.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in.

6.3.3 Words as vectors: word dimensions

word-word matrix

An alternative to using the term-document matrix to represent words as vectors of document counts, is to use the **term-term matrix**, also called the **word-word matrix** or the **term-context matrix**, in which the columns are labeled by words rather than documents. This matrix is thus of dimensionality $|V| \times |V|$ and each cell records the number of times the row (target) word and the column (context) word co-occur in some context in some training corpus. The context could be the document, in which case the cell represents the number of times the two words appear in the same document. It is most common, however, to use smaller contexts, generally a window around the word, for example of 4 words to the left and 4 words to the right, in which case the cell represents the number of times (in some training corpus) the column word occurs in such a ± 4 word window around the row word. Here are four examples of words in their windows:

is traditionally followed by **cherry** pie, a traditional dessert
 often mixed, such as **strawberry** rhubarb pie. Apple pie
 computer peripherals and personal **digital** assistants. These devices usually
 a computer. This includes **information** available on the internet

If we then take every occurrence of each word (say **strawberry**) and count the context words around it, we get a word-word co-occurrence matrix. Fig. 6.6 shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus (Davies, 2015).

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Figure 6.6 Co-occurrence vectors for four words in the Wikipedia corpus, showing six of the dimensions (hand-picked for pedagogical purposes). The vector for *digital* is outlined in red. Note that a real vector would have vastly more dimensions and thus be much sparser.

Note in Fig. 6.6 that the two words *cherry* and *strawberry* are more similar to each other (both *pie* and *sugar* tend to occur in their window) than they are to other words like *digital*; conversely, *digital* and *information* are more similar to each other than, say, to *strawberry*. Fig. 6.7 shows a spatial visualization.

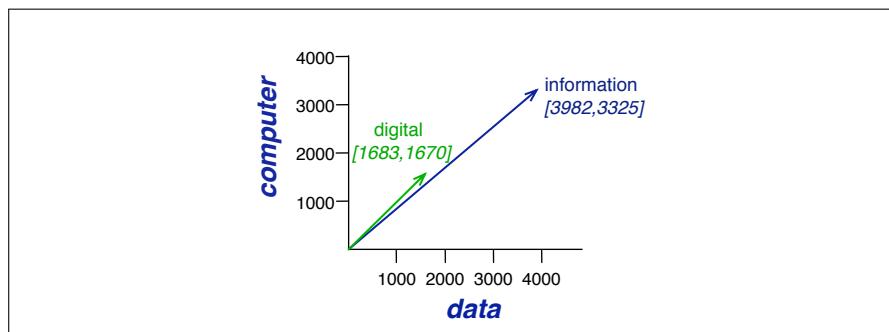


Figure 6.7 A spatial visualization of word vectors for *digital* and *information*, showing just two of the dimensions, corresponding to the words *data* and *computer*.

Note that $|V|$, the dimensionality of the vector, is generally the size of the vocabulary, often between 10,000 and 50,000 words (using the most frequent words

in the training corpus; keeping words after about the most frequent 50,000 or so is generally not helpful). Since most of these numbers are zero these are **sparse** vector representations; there are efficient algorithms for storing and computing with sparse matrices.

Now that we have some intuitions, let's move on to examine the details of computing word similarity. Afterwards we'll discuss methods for weighting cells.

6.4 Cosine for measuring similarity

To measure similarity between two target words v and w , we need a metric that takes two vectors (of the same dimensionality, either both with words as dimensions, hence of length $|V|$, or both with documents as dimensions, of length $|D|$) and gives a measure of their similarity. By far the most common similarity metric is the **cosine** of the angle between the vectors.

The cosine—like most measures for vector similarity used in NLP—is based on the **dot product** operator from linear algebra, also called the **inner product**:

dot product
inner product

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N \quad (6.7)$$

The dot product acts as a similarity metric because it will tend to be high just when the two vectors have large values in the same dimensions. Alternatively, vectors that have zeros in different dimensions—orthogonal vectors—will have a dot product of 0, representing their strong dissimilarity.

This raw dot product, however, has a problem as a similarity metric: it favors **long** vectors. The **vector length** is defined as

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2} \quad (6.8)$$

The dot product is higher if a vector is longer, with higher values in each dimension. More frequent words have longer vectors, since they tend to co-occur with more words and have higher co-occurrence values with each of them. The raw dot product thus will be higher for frequent words. But this is a problem; we'd like a similarity metric that tells us how similar two words are regardless of their frequency.

We modify the dot product to normalize for the vector length by dividing the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors, following from the definition of the dot product between two vectors \mathbf{a} and \mathbf{b} :

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= |\mathbf{a}| |\mathbf{b}| \cos \theta \\ \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} &= \cos \theta \end{aligned} \quad (6.9)$$

cosine The **cosine** similarity metric between two vectors \mathbf{v} and \mathbf{w} thus can be computed as:

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}| |\mathbf{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}} \quad (6.10)$$

For some applications we pre-normalize each vector, by dividing it by its length, creating a **unit vector** of length 1. Thus we could compute a unit vector from \mathbf{a} by dividing it by $|\mathbf{a}|$. For unit vectors, the dot product is the same as the cosine.

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for orthogonal vectors, to -1 for vectors pointing in opposite directions. But since raw frequency values are non-negative, the cosine for these vectors ranges from 0–1.

Let's see how the cosine computes which of the words *cherry* or *digital* is closer in meaning to *information*, just using raw counts from the following shortened table:

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry}, \text{information}) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .018$$

$$\cos(\text{digital}, \text{information}) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2} \sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

The model decides that *information* is way closer to *digital* than it is to *cherry*, a result that seems sensible. Fig. 6.8 shows a visualization.

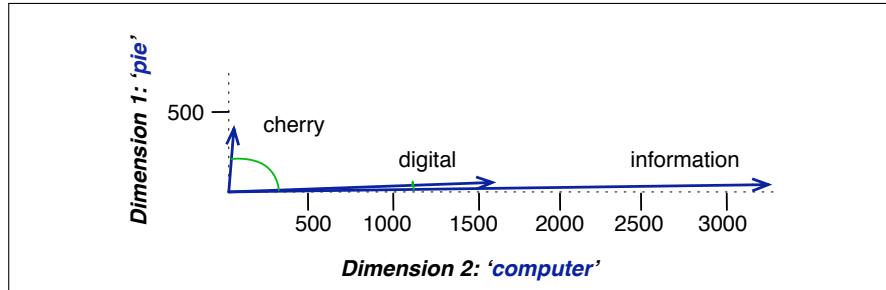


Figure 6.8 A (rough) graphical demonstration of cosine similarity, showing vectors for three words (*cherry*, *digital*, and *information*) in the two dimensional space defined by counts of the words *computer* and *pie* nearby. The figure doesn't show the cosine, but it highlights the angles; note that the angle between *digital* and *information* is smaller than the angle between *cherry* and *information*. When two vectors are more similar, the cosine is larger but the angle is smaller; the cosine has its maximum (1) when the angle between two vectors is smallest (0°); the cosine of all other angles is less than 1.

6.5 TF-IDF: Weighing terms in the vector

The co-occurrence matrices above represent each cell by frequencies, either of words with documents (Fig. 6.5), or words with other words (Fig. 6.6). But raw frequency

is not the best measure of association between words. Raw frequency is very skewed and not very discriminative. If we want to know what kinds of contexts are shared by *cherry* and *strawberry* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words and aren't informative about any particular word. We saw this also in Fig. 6.3 for the Shakespeare corpus; the dimension for the word *good* is not very discriminative between plays; *good* is simply a frequent word and has roughly equivalent high frequencies in each of the plays.

It's a bit of a paradox. Words that occur nearby frequently (maybe *pie* nearby *cherry*) are more important than words that only appear once or twice. Yet words that are too frequent—ubiquitous, like *the* or *good*—are unimportant. How can we balance these two conflicting constraints?

There are two common solutions to this problem: in this section we'll describe the **tf-idf** weighting, usually used when the dimensions are documents. In the next we introduce the **PPMI** algorithm (usually used when the dimensions are words).

The **tf-idf weighting** (the ‘-’ here is a hyphen, not a minus sign) is the product of two terms, each term capturing one of these two intuitions:

The first is the **term frequency** (Luhn, 1957): the frequency of the word t in the document d . We can just use the raw count as the term frequency:

$$\text{tf}_{t,d} = \text{count}(t,d) \quad (6.11)$$

More commonly we squash the raw frequency a bit, by using the \log_{10} of the frequency instead. The intuition is that a word appearing 100 times in a document doesn't make that word 100 times more likely to be relevant to the meaning of the document. Because we can't take the log of 0, we normally add 1 to the count:²

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d) + 1) \quad (6.12)$$

If we use log weighting, terms which occur 0 times in a document would have $\text{tf} = \log_{10}(1) = 0$, 10 times in a document $\text{tf} = \log_{10}(11) = 1.04$, 100 times $\text{tf} = \log_{10}(101) = 2.004$, 1000 times $\text{tf} = 3.00044$, and so on.

The second factor in tf-idf is used to give a higher weight to words that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection aren't as helpful. The **document frequency** df_t of a term t is the number of documents it occurs in. Document frequency is not the same as the **collection frequency** of a term, which is the total number of times the word appears in the whole collection in any document. Consider in the collection of Shakespeare's 37 plays the two words *Romeo* and *action*. The words have identical collection frequencies (they both occur 113 times in all the plays) but very different document frequencies, since *Romeo* only occurs in a single play. If our goal is to find documents about the romantic tribulations of Romeo, the word *Romeo* should be highly weighted, but not *action*:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

We emphasize discriminative words like *Romeo* via the **inverse document frequency** or **idf** term weight (Sparck Jones, 1972). The idf is defined using the frac-

² Or we can use this alternative: $\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$

tion N/df_t , where N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. It's usually clear what counts as a document: in Shakespeare we would use a play; when processing a collection of encyclopedia articles like Wikipedia, the document is a Wikipedia page; in processing newspaper articles, the document is a single article. Occasionally your corpus might not have appropriate document divisions and you might need to break up the corpus into documents yourself for the purposes of computing idf.

Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

$$\text{idf}_t = \log_{10} \left(\frac{N}{\text{df}_t} \right) \quad (6.13)$$

Here are some idf values for some words in the Shakespeare corpus, ranging from extremely informative words which occur in only one play like *Romeo*, to those that occur in a few like *salad* or *Falstaff*, to those which are very common like *fool* or so common as to be completely non-discriminative since they occur in all 37 plays like *good* or *sweet*.³

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

tf-idf

The **tf-idf** weighted value $w_{t,d}$ for word t in document d thus combines term frequency $\text{tf}_{t,d}$ (defined either by Eq. 6.11 or by Eq. 6.12) with idf from Eq. 6.13:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \quad (6.14)$$

Fig. 6.9 applies tf-idf weighting to the Shakespeare term-document matrix in Fig. 6.2, using the tf equation Eq. 6.12. Note that the tf-idf values for the dimension corresponding to the word *good* have now all become 0; since this word appears in every document, the tf-idf weighting leads it to be ignored. Similarly, the word *fool*, which appears in 36 out of the 37 plays, has a much lower weight.

The tf-idf weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline, the simple thing to try first. We'll look at other weightings like PPMI (Positive Pointwise Mutual Information) in Section 6.6.

³ *Sweet* was one of Shakespeare's favorite adjectives, a fact probably related to the increased use of sugar in European recipes around the turn of the 16th century (Jurafsky, 2014, p. 175).

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.9 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of $\text{tf} = \log_{10}(20 + 1) = 1.322$ and $\text{idf} = .037$. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

6.6 Pointwise Mutual Information (PMI)

pointwise mutual information

An alternative weighting function to tf-idf, PPMI (positive pointwise mutual information), is used for term-term-matrices, when the vector dimensions correspond to words rather than documents. PPMI draws on the intuition that the best way to weigh the association between two words is to ask how much **more** the two words co-occur in our corpus than we would have a priori expected them to appear by chance.

Pointwise mutual information (Fano, 1961)⁴ is one of the most important concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \quad (6.16)$$

The pointwise mutual information between a target word w and a context word c (Church and Hanks 1989, Church and Hanks 1990) is then defined as:

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)} \quad (6.17)$$

The numerator tells us how often we observed the two words together (assuming we compute probability by using the MLE). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently; recall that the probability of two independent events both occurring is just the product of the probabilities of the two events. Thus, the ratio gives us an estimate of how much more the two words co-occur than we expect by chance. PMI is a useful tool whenever we need to find words that are strongly associated.

PMI values range from negative to positive infinity. But negative PMI values (which imply things are co-occurring *less often* than we would expect by chance) tend to be unreliable unless our corpora are enormous. To distinguish whether two words whose individual probability is each 10^{-6} occur together less often than chance, we would need to be certain that the probability of the two occurring together is significantly less than 10^{-12} , and this kind of granularity would require an enormous corpus. Furthermore it's not clear whether it's even possible to evaluate such scores of 'unrelatedness' with human judgments. For this reason it is more

⁴ PMI is based on the **mutual information** between two random variables X and Y , defined as:

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \quad (6.15)$$

In a confusion of terminology, Fano used the phrase *mutual information* to refer to what we now call *pointwise mutual information* and the phrase *expectation of the mutual information* for what we now call *mutual information*

PPMI common to use Positive PMI (called **PPMI**) which replaces all negative PMI values with zero (Church and Hanks 1989, Dagan et al. 1993, Niwa and Nitta 1994)⁵:

$$\text{PPMI}(w, c) = \max(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0) \quad (6.18)$$

More formally, let's assume we have a co-occurrence matrix F with W rows (words) and C columns (contexts), where f_{ij} gives the number of times word w_i occurs with context c_j . This can be turned into a PPMI matrix where PPMI_{ij} gives the PPMI value of word w_i with context c_j (which we can also express as $\text{PPMI}(w_i, c_j)$ or $\text{PPMI}(w = i, c = j)$) as follows:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}, \quad p_{i*} = \frac{\sum_{j=1}^C f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}}, \quad p_{*j} = \frac{\sum_{i=1}^W f_{ij}}{\sum_{i=1}^W \sum_{j=1}^C f_{ij}} \quad (6.19)$$

$$\text{PPMI}_{ij} = \max(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0) \quad (6.20)$$

Let's see some PPMI calculations. We'll use Fig. 6.10, which repeats Fig. 6.6 plus all the count marginals, and let's pretend for ease of calculation that these are the only words/context that matter.

	computer	data	result	pie	sugar	count(w)
cherry	2	8	9	442	25	486
strawberry	0	0	1	60	19	80
digital	1670	1683	85	5	4	3447
information	3325	3982	378	5	13	7703
count(context)	4997	5673	473	512	61	11716

Figure 6.10 Co-occurrence counts for four words in 5 contexts in the Wikipedia corpus, together with the marginals, pretending for the purpose of this calculation that no other words/context matter.

Thus for example we could compute $\text{PPMI}(\text{information}, \text{data})$, assuming we pretended that Fig. 6.6 encompassed all the relevant word contexts/dimensions, as follows:

$$\begin{aligned} P(w=\text{information}, c=\text{data}) &= \frac{3982}{11716} = .3399 \\ P(w=\text{information}) &= \frac{7703}{11716} = .6575 \\ P(c=\text{data}) &= \frac{5673}{11716} = .4842 \\ \text{PPMI}(\text{information}, \text{data}) &= \log_2(.3399 / (.6575 * .4842)) = .0944 \end{aligned}$$

Fig. 6.11 shows the joint probabilities computed from the counts in Fig. 6.10, and Fig. 6.12 shows the PPMI values. Not surprisingly, *cherry* and *strawberry* are highly associated with both *pie* and *sugar*, and *data* is mildly associated with *information*.

PMI has the problem of being biased toward infrequent events; very rare words tend to have very high PMI values. One way to reduce this bias toward low frequency

⁵ Positive PMI also cleanly solves the problem of what to do with zero counts, using 0 to replace the $-\infty$ from $\log(0)$.

	$p(w, \text{context})$					$p(w)$
	computer	data	result	pie	sugar	$p(w)$
cherry	0.0002	0.0007	0.0008	0.0377	0.0021	0.0415
strawberry	0.0000	0.0000	0.0001	0.0051	0.0016	0.0068
digital	0.1425	0.1436	0.0073	0.0004	0.0003	0.2942
information	0.2838	0.3399	0.0323	0.0004	0.0011	0.6575
$p(\text{context})$	0.4265	0.4842	0.0404	0.0437	0.0052	

Figure 6.11 Replacing the counts in Fig. 6.6 with joint probabilities, showing the marginals in the right column and the bottom row.

	computer	data	result	pie	sugar
cherry	0	0	0	4.38	3.30
strawberry	0	0	0	4.10	5.51
digital	0.18	0.01	0	0	0
information	0.02	0.09	0.28	0	0

Figure 6.12 The PPMI matrix showing the association between words and context words, computed from the counts in Fig. 6.11. Note that most of the 0 PPMI values are ones that had a negative PMI; for example $\text{PMI}(\text{cherry}, \text{computer}) = -6.7$, meaning that *cherry* and *computer* co-occur on Wikipedia less often than we would expect by chance, and with PPMI we replace negative values by zero.

events is to slightly change the computation for $P(c)$, using a different function $P_\alpha(c)$ that raises the probability of the context word to the power of α :

$$\text{PPMI}_\alpha(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P_\alpha(c)}, 0\right) \quad (6.21)$$

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}(c)^\alpha} \quad (6.22)$$

Levy et al. (2015) found that a setting of $\alpha = 0.75$ improved performance of embeddings on a wide range of tasks (drawing on a similar weighting used for skip-grams described below in Eq. 6.32). This works because raising the count to $\alpha = 0.75$ increases the probability assigned to rare contexts, and hence lowers their PMI ($P_\alpha(c) > P(c)$ when c is rare).

Another possible solution is Laplace smoothing: Before computing PMI, a small constant k (values of 0.1-3 are common) is added to each of the counts, shrinking (discounting) all the non-zero values. The larger the k , the more the non-zero counts are discounted.

6.7 Applications of the tf-idf or PPMI vector models

In summary, the vector semantics model we've described so far represents a target word as a vector with dimensions corresponding either to the documents in a large collection (the term-document matrix) or to the counts of words in some neighboring window (the term-term matrix). The values in each dimension are counts, weighted by tf-idf (for term-document matrices) or PPMI (for term-term matrices), and the vectors are sparse (since most values are zero).

The model computes the similarity between two words x and y by taking the cosine of their tf-idf or PPMI vectors; high cosine, high similarity. This entire model

is sometimes referred to as the **tf-idf** model or the **PPMI** model, after the weighting function.

The tf-idf model of meaning is often used for document functions like deciding if two documents are similar. We represent a document by taking the vectors of all the words in the document, and computing the **centroid** of all those vectors. The centroid is the multidimensional version of the mean; the centroid of a set of vectors is a single vector that has the minimum sum of squared distances to each of the vectors in the set. Given k word vectors w_1, w_2, \dots, w_k , the centroid **document vector** d is:

$$d = \frac{w_1 + w_2 + \dots + w_k}{k} \quad (6.23)$$

Given two documents, we can then compute their document vectors d_1 and d_2 , and estimate the similarity between the two documents by $\cos(d_1, d_2)$. Document similarity is also useful for all sorts of applications; information retrieval, plagiarism detection, news recommender systems, and even for digital humanities tasks like comparing different versions of a text to see which are similar to each other.

Either the PPMI model or the tf-idf model can be used to compute word similarity, for tasks like finding word paraphrases, tracking changes in word meaning, or automatically discovering meanings of words in different corpora. For example, we can find the 10 most similar words to any target word w by computing the cosines between w and each of the $V - 1$ other words, sorting, and looking at the top 10.

6.8 Word2vec

In the previous sections we saw how to represent a word as a sparse, long vector with dimensions corresponding to words in the vocabulary or documents in a collection. We now introduce a more powerful word representation: **embeddings**, short dense vectors. Unlike the vectors we've seen so far, embeddings are **short**, with number of dimensions d ranging from 50-1000, rather than the much larger vocabulary size $|V|$ or number of documents D we've seen. These d dimensions don't have a clear interpretation. And the vectors are **dense**: instead of vector entries being sparse, mostly-zero counts or functions of counts, the values will be real-valued numbers that can be negative.

It turns out that dense vectors work better in every NLP task than sparse vectors. While we don't completely understand all the reasons for this, we have some intuitions. Representing words as 300-dimensional dense vectors requires our classifiers to learn far fewer weights than if we represented words as 50,000-dimensional vectors, and the smaller parameter space possibly helps with generalization and avoiding overfitting. Dense vectors may also do a better job of capturing synonymy. For example, in a sparse vector representation, dimensions for synonyms like *car* and *automobile* dimension are distinct and unrelated; sparse vectors may thus fail to capture the similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor.

In this section we introduce one method for computing embeddings: **skip-gram** with **negative sampling**, sometimes called **SGNS**. The skip-gram algorithm is one of two algorithms in a software package called **word2vec**, and so sometimes the algorithm is loosely referred to as word2vec (Mikolov et al. 2013a, Mikolov et al. 2013b). The word2vec methods are fast, efficient to train, and easily available on-

**static
embeddings**

line with code and pretrained embeddings. Word2vec embeddings are **static embeddings**, meaning that the method learns one fixed embedding for each word in the vocabulary. In Chapter 11 we'll introduce methods for learning dynamic **contextual embeddings** like the popular family of **BERT** representations, in which the vector for each word is different in different contexts.

The intuition of word2vec is that instead of counting how often each word w occurs near, say, *apricot*, we'll instead train a classifier on a binary prediction task: “Is word w likely to show up near *apricot*?”. We don't actually care about this prediction task; instead we'll take the learned classifier *weights* as the word embeddings.

self-supervision

The revolutionary intuition here is that we can just use running text as implicitly supervised training data for such a classifier; a word c that occurs near the target word *apricot* acts as gold ‘correct answer’ to the question “Is word c likely to show up near *apricot*?”. This method, often called **self-supervision**, avoids the need for any sort of hand-labeled supervision signal. This idea was first proposed in the task of neural language modeling, when [Bengio et al. \(2003\)](#) and [Collobert et al. \(2011\)](#) showed that a neural language model (a neural network that learned to predict the next word from prior words) could just use the next word in running text as its supervision signal, and could be used to learn an embedding representation for each word as part of doing this prediction task.

We'll see how to do neural networks in the next chapter, but word2vec is a much simpler model than the neural network language model, in two ways. First, word2vec simplifies the task (making it binary classification instead of word prediction). Second, word2vec simplifies the architecture (training a logistic regression classifier instead of a multi-layer neural network with hidden layers that demand more sophisticated training algorithms). The intuition of skip-gram is:

1. Treat the target word and a neighboring context word as positive examples.
2. Randomly sample other words in the lexicon to get negative samples.
3. Use logistic regression to train a classifier to distinguish those two cases.
4. Use the learned weights as the embeddings.

6.8.1 The classifier

Let's start by thinking about the classification task, and then turn to how to train. Imagine a sentence like the following, with a target word *apricot*, and assume we're using a window of ± 2 context words:

... lemon,	a [tablespoon of	apricot jam,	a]	pinch ...
c1	c2	w	c3	c4

Our goal is to train a classifier such that, given a tuple (w, c) of a target word w paired with a candidate context word c (for example $(\text{apricot}, \text{jam})$, or perhaps $(\text{apricot}, \text{aardvark})$) it will return the probability that c is a real context word (true for *jam*, false for *aardvark*):

$$P(+|w, c) \tag{6.24}$$

The probability that word c is not a real context word for w is just 1 minus Eq. 6.24:

$$P(-|w, c) = 1 - P(+|w, c) \tag{6.25}$$

How does the classifier compute the probability P ? The intuition of the skip-gram model is to base this probability on embedding similarity: a word is likely to

occur near the target if its embedding vector is similar to the target embedding. To compute similarity between these dense embeddings, we rely on the intuition that two vectors are similar if they have a high **dot product** (after all, cosine is just a normalized dot product). In other words:

$$\text{Similarity}(w, c) \approx \mathbf{c} \cdot \mathbf{w} \quad (6.26)$$

The dot product $\mathbf{c} \cdot \mathbf{w}$ is not a probability, it's just a number ranging from $-\infty$ to ∞ (since the elements in word2vec embeddings can be negative, the dot product can be negative). To turn the dot product into a probability, we'll use the **logistic** or **sigmoid** function $\sigma(x)$, the fundamental core of logistic regression:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (6.27)$$

We model the probability that word c is a real context word for target word w as:

$$P(+|w, c) = \sigma(\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{c} \cdot \mathbf{w})} \quad (6.28)$$

The sigmoid function returns a number between 0 and 1, but to make it a probability we'll also need the total probability of the two possible events (c is a context word, and c isn't a context word) to sum to 1. We thus estimate the probability that word c is not a real context word for w as:

$$\begin{aligned} P(-|w, c) &= 1 - P(+|w, c) \\ &= \sigma(-\mathbf{c} \cdot \mathbf{w}) = \frac{1}{1 + \exp(\mathbf{c} \cdot \mathbf{w})} \end{aligned} \quad (6.29)$$

Equation 6.28 gives us the probability for one word, but there are many context words in the window. Skip-gram makes the simplifying assumption that all context words are independent, allowing us to just multiply their probabilities:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(\mathbf{c}_i \cdot \mathbf{w}) \quad (6.30)$$

$$\log P(+|w, c_{1:L}) = \sum_{i=1}^L \log \sigma(\mathbf{c}_i \cdot \mathbf{w}) \quad (6.31)$$

In summary, skip-gram trains a probabilistic classifier that, given a test target word w and its context window of L words $c_{1:L}$, assigns a probability based on how similar this context window is to the target word. The probability is based on applying the logistic (sigmoid) function to the dot product of the embeddings of the target word with each context word. To compute this probability, we just need embeddings for each target word and context word in the vocabulary.

Fig. 6.13 shows the intuition of the parameters we'll need. Skip-gram actually stores two embeddings for each word, one for the word as a target, and one for the word considered as context. Thus the parameters we need to learn are two matrices \mathbf{W} and \mathbf{C} , each containing an embedding for every one of the $|V|$ words in the vocabulary V .⁶ Let's now turn to learning these embeddings (which is the real goal of training this classifier in the first place).

⁶ In principle the target matrix and the context matrix could use different vocabularies, but we'll simplify by assuming one shared vocabulary V .

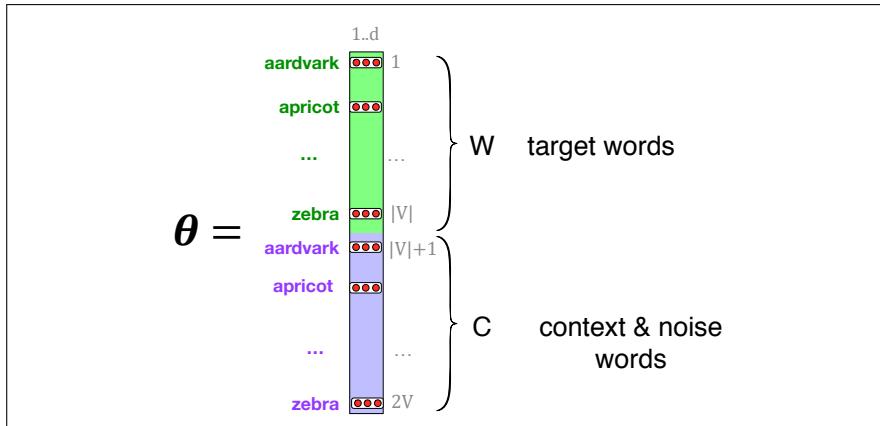


Figure 6.13 The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d , formed by concatenating two matrices, the target embeddings \mathbf{W} and the context+noise embeddings \mathbf{C} .

6.8.2 Learning skip-gram embeddings

The learning algorithm for skip-gram embeddings takes as input a corpus of text, and a chosen vocabulary size N . It begins by assigning a random embedding vector for each of the N vocabulary words, and then proceeds to iteratively shift the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don't occur nearby. Let's start by considering a single piece of training data:

```
... lemon, a [tablespoon of apricot jam,      a] pinch ...
          c1       c2     w   c3      c4
```

This example has a target word w (apricot), and 4 context words in the $L = \pm 2$ window, resulting in 4 positive training instances (on the left below):

positive examples +		negative examples -	
w	c_{pos}	w	c_{neg}
apricot	tablespoon	apricot	aardvark
apricot	of	apricot	my
apricot	jam	apricot	where
apricot	a	apricot	coaxial
		apricot	seven
		apricot	forever
		apricot	dear
		apricot	if

For training a binary classifier we also need negative examples. In fact skip-gram with negative sampling (SGNS) uses more negative examples than positive examples (with the ratio between them set by a parameter k). So for each of these (w, c_{pos}) training instances we'll create k negative samples, each consisting of the target w plus a ‘noise word’ c_{neg} . A noise word is a random word from the lexicon, constrained not to be the target word w . The right above shows the setting where $k = 2$, so we'll have 2 negative examples in the negative training set – for each positive example w, c_{pos} .

The noise words are chosen according to their weighted unigram frequency $p_\alpha(w)$, where α is a weight. If we were sampling according to unweighted frequency $p(w)$, it would mean that with unigram probability $p(\text{"the"})$ we would choose the word *the* as a noise word, with unigram probability $p(\text{"aardvark"})$ we would choose *aardvark*, and so on. But in practice it is common to set $\alpha = .75$, i.e. use the

weighting $p^{\frac{3}{4}}(w)$:

$$P_\alpha(w) = \frac{\text{count}(w)^\alpha}{\sum_{w'} \text{count}(w')^\alpha} \quad (6.32)$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$. To illustrate this intuition, it might help to work out the probabilities for an example with two events, $P(a) = .99$ and $P(b) = .01$:

$$\begin{aligned} P_\alpha(a) &= \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97 \\ P_\alpha(b) &= \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03 \end{aligned} \quad (6.33)$$

Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings to

- Maximize the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive examples
- Minimize the similarity of the (w, c_{neg}) pairs from the negative examples.

If we consider one word/context pair (w, c_{pos}) with its k noise words $c_{neg_1} \dots c_{neg_k}$, we can express these two goals as the following loss function L to be minimized (hence the $-$); here the first term expresses that we want the classifier to assign the real context word c_{pos} a high probability of being a neighbor, and the second term expresses that we want to assign each of the noise words c_{neg_i} a high probability of being a non-neighbor, all multiplied because we assume independence:

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \prod_{i=1}^k P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log P(-|w, c_{neg_i}) \right] \\ &= - \left[\log P(+|w, c_{pos}) + \sum_{i=1}^k \log (1 - P(+|w, c_{neg_i})) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg_i} \cdot w) \right] \end{aligned} \quad (6.34)$$

That is, we want to maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the k negative sampled non-neighbor words.

We minimize this loss function using stochastic gradient descent. Fig. 6.14 shows the intuition of one step of learning.

To get the gradient, we need to take the derivative of Eq. 6.34 with respect to the different embeddings. It turns out the derivatives are the following (we leave the

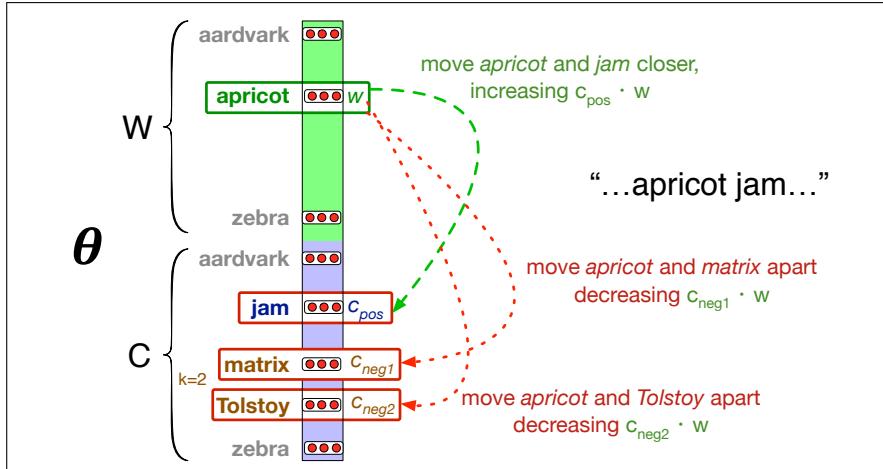


Figure 6.14 Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

proof as an exercise at the end of the chapter):

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w} \quad (6.35)$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(\mathbf{c}_{neg} \cdot \mathbf{w})]\mathbf{w} \quad (6.36)$$

$$\frac{\partial L_{CE}}{\partial \mathbf{w}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^k [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i} \quad (6.37)$$

The update equations going from time step t to $t+1$ in stochastic gradient descent are thus:

$$\mathbf{c}_{pos}^{t+1} = \mathbf{c}_{pos}^t - \eta[\sigma(\mathbf{c}_{pos}^t \cdot \mathbf{w}^t) - 1]\mathbf{w}^t \quad (6.38)$$

$$\mathbf{c}_{neg}^{t+1} = \mathbf{c}_{neg}^t - \eta[\sigma(\mathbf{c}_{neg}^t \cdot \mathbf{w}^t)]\mathbf{w}^t \quad (6.39)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^t) - 1]\mathbf{c}_{pos} + \sum_{i=1}^k [\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}^t)]\mathbf{c}_{neg_i} \right] \quad (6.40)$$

Just as in logistic regression, then, the learning algorithm starts with randomly initialized \mathbf{W} and \mathbf{C} matrices, and then walks through the training corpus using gradient descent to move \mathbf{W} and \mathbf{C} so as to minimize the loss in Eq. 6.34 by making the updates in (Eq. 6.38)-(Eq. 6.40).

target
embedding
context
embedding

Recall that the skip-gram model learns **two** separate embeddings for each word i : the **target embedding** \mathbf{w}_i and the **context embedding** \mathbf{c}_i , stored in two matrices, the **target matrix** \mathbf{W} and the **context matrix** \mathbf{C} . It's common to just add them together, representing word i with the vector $\mathbf{w}_i + \mathbf{c}_i$. Alternatively we can throw away the \mathbf{C} matrix and just represent each word i by the vector \mathbf{w}_i .

As with the simple count-based methods like tf-idf, the context window size L affects the performance of skip-gram embeddings, and experiments often tune the parameter L on a devset.

6.8.3 Other kinds of static embeddings

fasttext

There are many kinds of static embeddings. An extension of word2vec, **fasttext** (Bojanowski et al., 2017), addresses a problem with word2vec as we have presented it so far: it has no good way to deal with **unknown words**—words that appear in a test corpus but were unseen in the training corpus. A related problem is word sparsity, such as in languages with rich morphology, where some of the many forms for each noun and verb may only occur rarely. Fasttext deals with these problems by using subword models, representing each word as itself plus a bag of constituent n-grams, with special boundary symbols < and > added to each word. For example, with $n = 3$ the word *where* would be represented by the sequence <where> plus the character n-grams:

<wh, whe, her, ere, re>

Then a skipgram embedding is learned for each constituent n-gram, and the word *where* is represented by the sum of all of the embeddings of its constituent n-grams. Unknown words can then be presented only by the sum of the constituent n-grams. A fasttext open-source library, including pretrained embeddings for 157 languages, is available at <https://fasttext.cc>.

Another very widely used static embedding model is GloVe (Pennington et al., 2014), short for Global Vectors, because the model is based on capturing global corpus statistics. GloVe is based on ratios of probabilities from the word-word co-occurrence matrix, combining the intuitions of count-based models like PPMI while also capturing the linear structures used by methods like word2vec.

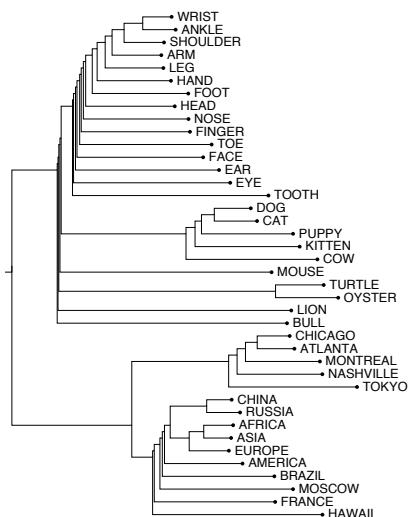
It turns out that dense embeddings like word2vec actually have an elegant mathematical relationship with sparse embeddings like PPMI, in which word2vec can be seen as implicitly optimizing a shifted version of a PPMI matrix (Levy and Goldberg, 2014c).

6.9 Visualizing Embeddings

“I see well in many dimensions as long as the dimensions are around two.”

The late economist Martin Shubik

Visualizing embeddings is an important goal in helping understand, apply, and improve these models of word meaning. But how can we visualize a (for example) 100-dimensional vector?



The simplest way to visualize the meaning of a word w embedded in a space is to list the most similar words to w by sorting the vectors for all words in the vocabulary by their cosine with the vector for w . For example the 7 closest words to *frog* using the GloVe embeddings are: *frogs*, *toad*, *litoria*, *leptodactylidae*, *rana*, *lizard*, and *eleutherodactylus* (Pennington et al., 2014).

Yet another visualization method is to use a clustering algorithm to show a hierarchical representation of which words are similar to others in the embedding space. The uncaptioned figure on the left uses hierarchical clustering of some embedding vectors for nouns as a visualization method (Rohde et al., 2006).

Probably the most common visualization method, however, is to project the 100 dimensions of a word down into 2 dimensions. Fig. 6.1 showed one such visualization, as does Fig. 6.16, using a projection method called t-SNE (van der Maaten and Hinton, 2008).

6.10 Semantic properties of embeddings

In this section we briefly summarize some of the semantic properties of embeddings that have been studied.

Different types of similarity or association: One parameter of vector semantic models that is relevant to both sparse tf-idf vectors and dense word2vec vectors is the size of the context window used to collect counts. This is generally between 1 and 10 words on each side of the target word (for a total context of 2-20 words).

The choice depends on the goals of the representation. Shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words. When the vectors are computed from short context windows, the most similar words to a target word w tend to be semantically similar words with the same parts of speech. When vectors are computed from long context windows, the highest cosine words to a target word w tend to be words that are topically related but not similar.

For example Levy and Goldberg (2014a) showed that using skip-gram with a window of ± 2 , the most similar words to the word *Hogwarts* (from the *Harry Potter* series) were names of other fictional schools: *Sunnydale* (from *Buffy the Vampire Slayer*) or *Evernight* (from a vampire series). With a window of ± 5 , the most similar words to *Hogwarts* were other words topically related to the *Harry Potter* series: *Dumbledore*, *Malfoy*, and *half-blood*.

first-order
co-occurrence

It's also often useful to distinguish two kinds of similarity or association between words (Schütze and Pedersen, 1993). Two words have **first-order co-occurrence** (sometimes called **syntagmatic association**) if they are typically nearby each other. Thus *wrote* is a first-order associate of *book* or *poem*. Two words have **second-order co-occurrence** (sometimes called **paradigmatic association**) if they have similar neighbors. Thus *wrote* is a second-order associate of words like *said* or *remarked*.

second-order
co-occurrence

parallelogram
model

Analogy/Relational Similarity: Another semantic property of embeddings is their ability to capture relational meanings. In an important early vector space model of cognition, Rumelhart and Abrahamson (1973) proposed the **parallelogram model** for solving simple analogy problems of the form a is to b as a^* is to what?. In such problems, a system is given a problem like $apple:tree::grape:?$, i.e., *apple* is to *tree* as *grape* is to ____, and must fill in the word *vine*. In the parallelogram model, illustrated in Fig. 6.15, the vector from the word *apple* to the word *tree* (= $\overrightarrow{tree} - \overrightarrow{apple}$) is added to the vector for *grape* (\overrightarrow{grape}); the nearest word to that point is returned.

In early work with sparse embeddings, scholars showed that sparse vector models of meaning could solve such analogy problems (Turney and Littman, 2005), but the parallelogram method received more modern attention because of its success with word2vec or GloVe vectors (Mikolov et al. 2013c, Levy and Goldberg 2014b, Pennington et al. 2014). For example, the result of the expression $\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is a vector close to $\overrightarrow{\text{queen}}$. Similarly, $\overrightarrow{\text{Paris}} - \overrightarrow{\text{France}} + \overrightarrow{\text{Italy}}$ results

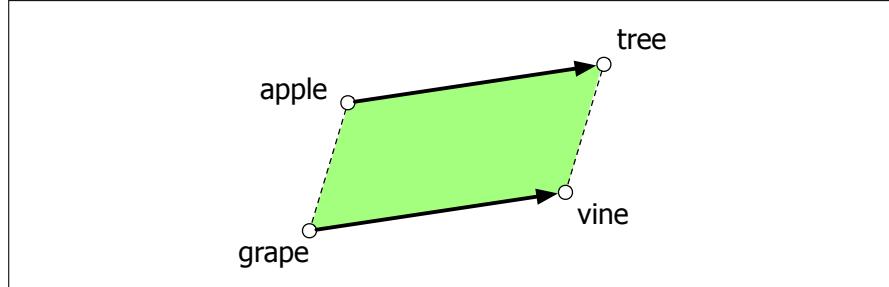


Figure 6.15 The parallelogram model for analogy problems (Rumelhart and Abrahamson, 1973): the location of vine can be found by subtracting apple from tree and adding grape.

in a vector that is close to $\overrightarrow{\text{Rome}}$. The embedding model thus seems to be extracting representations of relations like MALE-FEMALE, or CAPITAL-CITY-OF, or even COMPARATIVE/SUPERLATIVE, as shown in Fig. 6.16 from GloVe.

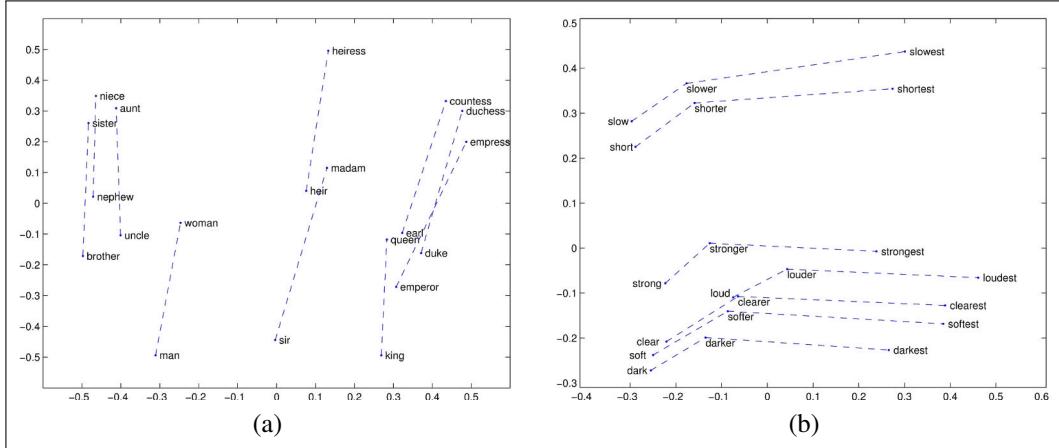


Figure 6.16 Relational properties of the GloVe vector space, shown by projecting vectors onto two dimensions. (a) $\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}}$ is close to $\overrightarrow{\text{queen}}$. (b) offsets seem to capture comparative and superlative morphology (Pennington et al., 2014).

For a $\mathbf{a} : \mathbf{b} :: \mathbf{a}^* : \mathbf{b}^*$ problem, meaning the algorithm is given vectors \mathbf{a} , \mathbf{b} , and \mathbf{a}^* and must find \mathbf{b}^* , the parallelogram method is thus:

$$\hat{\mathbf{b}}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \operatorname{distance}(\mathbf{x}, \mathbf{b} - \mathbf{a} + \mathbf{a}^*) \quad (6.41)$$

with some distance function, such as Euclidean distance.

There are some caveats. For example, the closest value returned by the parallelogram algorithm in word2vec or GloVe embedding spaces is usually not in fact \mathbf{b}^* but one of the 3 input words or their morphological variants (i.e., *cherry:red :: potato:x* returns *potato* or *potatoes* instead of *brown*), so these must be explicitly excluded. Furthermore while embedding spaces perform well if the task involves frequent words, small distances, and certain relations (like relating countries with their capitals or verbs/nouns with their inflected forms), the parallelogram method with embeddings doesn't work as well for other relations (Linzen 2016, Gladkova et al. 2016, Schluter 2018, Ethayarajh et al. 2019a), and indeed Peterson et al. (2020) argue that the parallelogram method is in general too simple to model the human cognitive process of forming analogies of this kind.

6.10.1 Embeddings and Historical Semantics

Embeddings can also be a useful tool for studying how meaning changes over time, by computing multiple embedding spaces, each from texts written in a particular time period. For example Fig. 6.17 shows a visualization of changes in meaning in English words over the last two centuries, computed by building separate embedding spaces for each decade from historical corpora like Google n-grams (Lin et al., 2012) and the Corpus of Historical American English (Davies, 2012).

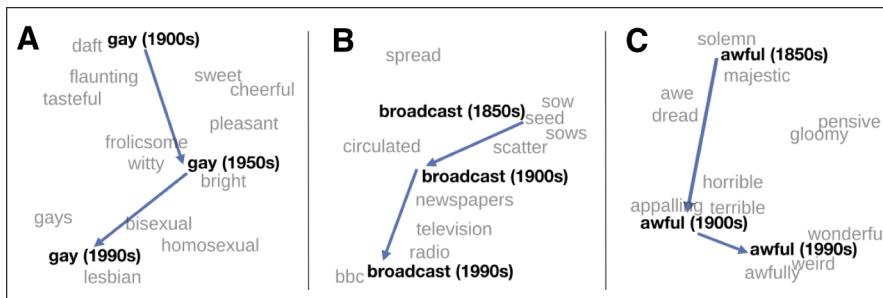


Figure 6.17 A t-SNE visualization of the semantic change of 3 words in English using word2vec vectors. The modern sense of each word, and the grey context words, are computed from the most recent (modern) time-point embedding space. Earlier points are computed from earlier historical embedding spaces. The visualizations show the changes in the word *gay* from meanings related to “cheerful” or “frolicsome” to referring to homosexuality, the development of the modern “transmission” sense of *broadcast* from its original sense of sowing seeds, and the pejoration of the word *awful* as it shifted from meaning “full of awe” to meaning “terrible or appalling” (Hamilton et al., 2016).

6.11 Bias and Embeddings

In addition to their ability to learn word meaning from text, embeddings, alas, also reproduce the implicit biases and stereotypes that were latent in the text. As the prior section just showed, embeddings can roughly model relational similarity: ‘queen’ as the closest word to ‘king’ - ‘man’ + ‘woman’ implies the analogy *man:woman::king:queen*. But these same embedding analogies also exhibit gender stereotypes. For example Bolukbasi et al. (2016) find that the closest occupation to ‘computer programmer’ - ‘man’ + ‘woman’ in word2vec embeddings trained on news text is ‘homemaker’, and that the embeddings similarly suggest the analogy ‘father’ is to ‘doctor’ as ‘mother’ is to ‘nurse’. This could result in what Crawford (2017) and Blodgett et al. (2020) call an **allocational harm**, when a system allocates resources (jobs or credit) unfairly to different groups. For example algorithms that use embeddings as part of a search for hiring potential programmers or doctors might thus incorrectly downweight documents with women’s names.

allocational
harm

bias
amplification

It turns out that embeddings don’t just reflect the statistics of their input, but also **amplify bias**; gendered terms become **more** gendered in embedding space than they were in the input text statistics (Zhao et al. 2017, Ethayarajh et al. 2019b, Jia et al. 2020), and biases are more exaggerated than in actual labor employment statistics (Garg et al., 2018).

Embeddings also encode the implicit associations that are a property of human reasoning. The Implicit Association Test (Greenwald et al., 1998) measures peo-

representational harm

ple's associations between concepts (like 'flowers' or 'insects') and attributes (like 'pleasantness' and 'unpleasantness') by measuring differences in the latency with which they label words in the various categories.⁷ Using such methods, people in the United States have been shown to associate African-American names with unpleasant words (more than European-American names), male names more with mathematics and female names with the arts, and old people's names with unpleasant words (Greenwald et al. 1998, Nosek et al. 2002a, Nosek et al. 2002b). Caliskan et al. (2017) replicated all these findings of implicit associations using GloVe vectors and cosine similarity instead of human latencies. For example African-American names like 'Leroy' and 'Shaniqua' had a higher GloVe cosine with unpleasant words while European-American names ('Brad', 'Greg', 'Courtney') had a higher cosine with pleasant words. These problems with embeddings are an example of a **representational harm** (Crawford 2017, Blodgett et al. 2020), which is a harm caused by a system demeaning or even ignoring some social groups. Any embedding-aware algorithm that made use of word sentiment could thus exacerbate bias against African Americans.

debiasing

Recent research focuses on ways to try to remove these kinds of biases, for example by developing a transformation of the embedding space that removes gender stereotypes but preserves definitional gender (Bolukbasi et al. 2016, Zhao et al. 2017) or changing the training procedure (Zhao et al., 2018). However, although these sorts of **debiasing** may reduce bias in embeddings, they do not eliminate it (Gonen and Goldberg, 2019), and this remains an open problem.

Historical embeddings are also being used to measure biases in the past. Garg et al. (2018) used embeddings from historical texts to measure the association between embeddings for occupations and embeddings for names of various ethnicities or genders (for example the relative cosine similarity of women's names versus men's to occupation words like 'librarian' or 'carpenter') across the 20th century. They found that the cosines correlate with the empirical historical percentages of women or ethnic groups in those occupations. Historical embeddings also replicated old surveys of ethnic stereotypes; the tendency of experimental participants in 1933 to associate adjectives like 'industrious' or 'superstitious' with, e.g., Chinese ethnicity, correlates with the cosine between Chinese last names and those adjectives using embeddings trained on 1930s text. They also were able to document historical gender biases, such as the fact that embeddings for adjectives related to competence ('smart', 'wise', 'thoughtful', 'resourceful') had a higher cosine with male than female words, and showed that this bias has been slowly decreasing since 1960. We return in later chapters to this question about the role of bias in natural language processing.

6.12 Evaluating Vector Models

The most important evaluation metric for vector models is extrinsic evaluation on tasks, i.e., using vectors in an NLP task and seeing whether this improves performance over some other model.

⁷ Roughly speaking, if humans associate 'flowers' with 'pleasantness' and 'insects' with 'unpleasantness', when they are instructed to push a green button for 'flowers' (daisy, iris, lilac) and 'pleasant words' (love, laughter, pleasure) and a red button for 'insects' (flea, spider, mosquito) and 'unpleasant words' (abuse, hatred, ugly) they are faster than in an incongruous condition where they push a red button for 'flowers' and 'unpleasant words' and a green button for 'insects' and 'pleasant words'.

Nonetheless it is useful to have intrinsic evaluations. The most common metric is to test their performance on **similarity**, computing the correlation between an algorithm’s word similarity scores and word similarity ratings assigned by humans. **WordSim-353** (Finkelstein et al., 2002) is a commonly used set of ratings from 0 to 10 for 353 noun pairs; for example (*plane*, *car*) had an average score of 5.77. **SimLex-999** (Hill et al., 2015) is a more difficult dataset that quantifies similarity (*cup*, *mug*) rather than relatedness (*cup*, *coffee*), and including both concrete and abstract adjective, noun and verb pairs. The **TOEFL dataset** is a set of 80 questions, each consisting of a target word with 4 additional word choices; the task is to choose which is the correct synonym, as in the example: *Levied is closest in meaning to: imposed, believed, requested, correlated* (Landauer and Dumais, 1997). All of these datasets present words without context.

Slightly more realistic are intrinsic similarity tasks that include context. The Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012) and the Word-in-Context (WiC) dataset (Pilehvar and Camacho-Collados, 2019) offer richer evaluation scenarios. SCWS gives human judgments on 2,003 pairs of words in their sentential context, while WiC gives target words in two sentential contexts that are either in the same or different senses; see Section ???. The *semantic textual similarity* task (Agirre et al. 2012, Agirre et al. 2015) evaluates the performance of sentence-level similarity algorithms, consisting of a set of pairs of sentences, each pair with human-labeled similarity scores.

Another task used for evaluation is the analogy task, discussed on page 24, where the system has to solve problems of the form a is to b as a^* is to b^* , given a , b , and a^* and having to find b^* (Turney and Littman, 2005). A number of sets of tuples have been created for this task, (Mikolov et al. 2013a, Mikolov et al. 2013c, Gladkova et al. 2016), covering morphology (*city:cities::child:children*), lexicographic relations (*leg:table::spout:teapot*) and encyclopedia relations (*Beijing:China::Dublin:Ireland*), some drawing from the SemEval-2012 Task 2 dataset of 79 different relations (Jurgens et al., 2012).

All embedding algorithms suffer from inherent variability. For example because of randomness in the initialization and the random negative sampling, algorithms like word2vec may produce different results even from the same dataset, and individual documents in a collection may strongly impact the resulting embeddings (Tian et al. 2016, Hellrich and Hahn 2016, Antoniak and Mimno 2018). When embeddings are used to study word associations in particular corpora, therefore, it is best practice to train multiple embeddings with bootstrap sampling over documents and average the results (Antoniak and Mimno, 2018).

6.13 Summary

- In vector semantics, a word is modeled as a vector—a point in high-dimensional space, also called an **embedding**. In this chapter we focus on **static embeddings**, where each word is mapped to a fixed embedding.
- Vector semantic models fall into two classes: **sparse** and **dense**. In sparse models each dimension corresponds to a word in the vocabulary V and cells are functions of **co-occurrence counts**. The **term-document** matrix has a row for each word (**term**) in the vocabulary and a column for each document. The **word-context** or **term-term** matrix has a row for each (target) word in

the vocabulary and a column for each context term in the vocabulary. Two sparse weightings are common: the **tf-idf** weighting which weights each cell by its **term frequency** and **inverse document frequency**, and **PPMI** (pointwise positive mutual information), which is most common for word-context matrices.

- Dense vector models have dimensionality 50–1000. **Word2vec** algorithms like **skip-gram** are a popular way to compute dense embeddings. Skip-gram trains a logistic regression classifier to compute the probability that two words are ‘likely to occur nearby in text’. This probability is computed from the dot product between the embeddings for the two words.
- Skip-gram uses stochastic gradient descent to train the classifier, by learning embeddings that have a high dot product with embeddings of words that occur nearby and a low dot product with noise words.
- Other important embedding algorithms include **GloVe**, a method based on ratios of word co-occurrence probabilities.
- Whether using sparse or dense vectors, word and document similarities are computed by some function of the **dot product** between vectors. The cosine of two vectors—a normalized dot product—is the most popular such metric.

Bibliographical and Historical Notes

The idea of vector semantics arose out of research in the 1950s in three distinct fields: linguistics, psychology, and computer science, each of which contributed a fundamental aspect of the model.

The idea that meaning is related to the distribution of words in context was widespread in linguistic theory of the 1950s, among distributionalists like Zellig Harris, Martin Joos, and J. R. Firth, and semioticians like Thomas Sebeok. As [Joos \(1950\)](#) put it,

the linguist’s “meaning” of a morpheme...is by definition the set of conditional probabilities of its occurrence in context with all other morphemes.

The idea that the meaning of a word might be modeled as a point in a multi-dimensional semantic space came from psychologists like Charles E. Osgood, who had been studying how people responded to the meaning of words by assigning values along scales like *happy/sad* or *hard/soft*. [Osgood et al. \(1957\)](#) proposed that the meaning of a word in general could be modeled as a point in a multidimensional Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points in the space.

mechanical indexing

A final intellectual source in the 1950s and early 1960s was the field then called **mechanical indexing**, now known as **information retrieval**. In what became known as the **vector space model** for information retrieval ([Salton 1971](#), [Sparck Jones 1986](#)), researchers demonstrated new ways to define the meaning of words in terms of vectors ([Switzer, 1965](#)), and refined methods for word similarity based on measures of statistical association between words like mutual information ([Giuliano, 1965](#)) and idf ([Sparck Jones, 1972](#)), and showed that the meaning of documents could be represented in the same vector spaces used for words.

Some of the philosophical underpinning of the distributional way of thinking came from the late writings of the philosopher Wittgenstein, who was skeptical of

the possibility of building a completely formal theory of meaning definitions for each word, suggesting instead that “the meaning of a word is its use in the language” (Wittgenstein, 1953, PI 43). That is, instead of using some logical language to define each word, or drawing on denotations or truth values, Wittgenstein’s idea is that we should define a word by how it is used by people in speaking and understanding in their day-to-day interactions, thus prefiguring the movement toward embodied and experiential models in linguistics and NLP (Glenberg and Robertson 2000, Lake and Murphy 2021, Bisk et al. 2020, Bender and Koller 2020).

More distantly related is the idea of defining words by a vector of discrete features, which has roots at least as far back as Descartes and Leibniz (Wierzbicka 1992, Wierzbicka 1996). By the middle of the 20th century, beginning with the work of Hjelmslev (Hjelmslev, 1969) (originally 1943) and fleshed out in early models of generative grammar (Katz and Fodor, 1963), the idea arose of representing meaning with **semantic features**, symbols that represent some sort of primitive meaning. For example words like *hen*, *rooster*, or *chick*, have something in common (they all describe chickens) and something different (their age and sex), representable as:

hen +female, +chicken, +adult
rooster -female, +chicken, +adult
chick +chicken, -adult

The dimensions used by vector models of meaning to define words, however, are only abstractly related to this idea of a small fixed number of hand-built dimensions. Nonetheless, there has been some attempt to show that certain dimensions of embedding models do contribute some specific compositional aspect of meaning like these early semantic features.

The use of dense vectors to model word meaning, and indeed the term **embedding**, grew out of the **latent semantic indexing** (LSI) model (Deerwester et al., 1988) recast as **LSA** (**latent semantic analysis**) (Deerwester et al., 1990). In LSA **singular value decomposition—SVD**—is applied to a term-document matrix (each cell weighted by log frequency and normalized by entropy), and then the first 300 dimensions are used as the LSA embedding. Singular Value Decomposition (SVD) is a method for finding the most important dimensions of a data set, those dimensions along which the data varies the most. LSA was then quickly widely applied: as a cognitive model Landauer and Dumais (1997), and for tasks like spell checking (Jones and Martin, 1997), language modeling (Bellegarda 1997, Coccaro and Jurafsky 1998, Bellegarda 2000) morphology induction (Schone and Jurafsky 2000, Schone and Jurafsky 2001b), multiword expressions (MWEs) (Schone and Jurafsky, 2001a), and essay grading (Rehder et al., 1998). Related models were simultaneously developed and applied to word sense disambiguation by Schütze (1992). LSA also led to the earliest use of embeddings to represent words in a probabilistic classifier, in the logistic regression document router of Schütze et al. (1995). The idea of SVD on the term-term matrix (rather than the term-document matrix) as a model of meaning for NLP was proposed soon after LSA by Schütze (1992). Schütze applied the low-rank (97-dimensional) embeddings produced by SVD to the task of word sense disambiguation, analyzed the resulting semantic space, and also suggested possible techniques like dropping high-order dimensions. See Schütze (1997).

A number of alternative matrix models followed on from the early SVD work, including Probabilistic Latent Semantic Indexing (PLSI) (Hofmann, 1999), Latent Dirichlet Allocation (LDA) (Blei et al., 2003), and Non-negative Matrix Factorization (NMF) (Lee and Seung, 1999).

The LSA community seems to have first used the word “embedding” in Landauer

semantic feature

SVD

[et al.](#) (1997), in a variant of its mathematical meaning as a mapping from one space or mathematical structure to another. In LSA, the word embedding seems to have described the mapping from the space of sparse count vectors to the latent space of SVD dense vectors. Although the word thus originally meant the mapping from one space to another, it has metonymically shifted to mean the resulting dense vector in the latent space, and it is in this sense that we currently use the word.

By the next decade, [Bengio et al.](#) (2003) and [Bengio et al.](#) (2006) showed that neural language models could also be used to develop embeddings as part of the task of word prediction. [Collobert and Weston](#) (2007), [Collobert and Weston](#) (2008), and [Collobert et al.](#) (2011) then demonstrated that embeddings could be used to represent word meanings for a number of NLP tasks. [Turian et al.](#) (2010) compared the value of different kinds of embeddings for different NLP tasks. [Mikolov et al.](#) (2011) showed that recurrent neural nets could be used as language models. The idea of simplifying the hidden layer of these neural net language models to create the skip-gram (and also CBOW) algorithms was proposed by [Mikolov et al.](#) (2013a). The negative sampling training algorithm was proposed in [Mikolov et al.](#) (2013b). There are numerous surveys of static embeddings and their parameterizations ([Bullinaria and Levy 2007](#), [Bullinaria and Levy 2012](#), [Lapesa and Evert 2014](#), [Kiela and Clark 2014](#), [Levy et al. 2015](#)).

See [Manning et al.](#) (2008) for a deeper understanding of the role of vectors in information retrieval, including how to compare queries with documents, more details on tf-idf, and issues of scaling to very large datasets. See [Kim \(2019\)](#) for a clear and comprehensive tutorial on word2vec. [Cruse \(2004\)](#) is a useful introductory linguistic text on lexical semantics.

Exercises

- Agirre, E., C. Banea, C. Cardie, D. Cer, M. Diab, A. Gonzalez-Agirre, W. Guo, I. Lopez-Gazpio, M. Martíxalar, R. Mihalcea, G. Rigau, L. Uriar, and J. Wiebe. 2015. [SemEval-2015 task 2: Semantic textual similarity, English, Spanish and pilot on interpretability](#). *SemEval-15*.
- Agirre, E., M. Diab, D. Cer, and A. Gonzalez-Agirre. 2012. [SemEval-2012 task 6: A pilot on semantic textual similarity](#). *SemEval-12*.
- Antoniak, M. and D. Mimno. 2018. [Evaluating the stability of embedding-based word similarities](#). *TACL*, 6:107–119.
- Bellegarda, J. R. 1997. [A latent semantic analysis framework for large-span language modeling](#). *EUROSPEECH*.
- Bellegarda, J. R. 2000. [Exploiting latent semantic information in statistical language modeling](#). *Proceedings of the IEEE*, 89(8):1279–1296.
- Bender, E. M. and A. Koller. 2020. [Climbing towards NLU: On meaning, form, and understanding in the age of data](#). *ACL*.
- Bengio, Y., A. Courville, and P. Vincent. 2013. [Representation learning: A review and new perspectives](#). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.
- Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin. 2003. [A neural probabilistic language model](#). *JMLR*, 3:1137–1155.
- Bengio, Y., H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. 2006. [Neural probabilistic language models](#). In *Innovations in Machine Learning*, pages 137–186. Springer.
- Bisk, Y., A. Holtzman, J. Thomason, J. Andreas, Y. Bengio, J. Chai, M. Lapata, A. Lazaridou, J. May, A. Nisnevich, N. Pinto, and J. Turian. 2020. [Experience grounds language](#). *EMNLP*.
- Blei, D. M., A. Y. Ng, and M. I. Jordan. 2003. [Latent Dirichlet allocation](#). *JMLR*, 3(5):993–1022.
- Blodgett, S. L., S. Barocas, H. Daumé III, and H. Wallach. 2020. [Language \(technology\) is power: A critical survey of “bias” in NLP](#). *ACL*.
- Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov. 2017. [Enriching word vectors with subword information](#). *TACL*, 5:135–146.
- Bolukbasi, T., K.-W. Chang, J. Zou, V. Saligrama, and A. T. Kalai. 2016. [Man is to computer programmer as woman is to homemaker? Debiasing word embeddings](#). *NeurIPS*.
- Bréal, M. 1897. *Essai de Sémantique: Science des significations*. Hachette.
- Budanitsky, A. and G. Hirst. 2006. [Evaluating WordNet-based measures of lexical semantic relatedness](#). *Computational Linguistics*, 32(1):13–47.
- Bullinaria, J. A. and J. P. Levy. 2007. [Extracting semantic representations from word co-occurrence statistics: A computational study](#). *Behavior research methods*, 39(3):510–526.
- Bullinaria, J. A. and J. P. Levy. 2012. [Extracting semantic representations from word co-occurrence statistics: stoplists, stemming, and SVD](#). *Behavior research methods*, 44(3):890–907.
- Caliskan, A., J. J. Bryson, and A. Narayanan. 2017. [Semantics derived automatically from language corpora contain human-like biases](#). *Science*, 356(6334):183–186.
- Carlson, G. N. 1977. [Reference to kinds in English](#). Ph.D. thesis, University of Massachusetts, Amherst. Forward.
- Church, K. W. and P. Hanks. 1989. [Word association norms, mutual information, and lexicography](#). *ACL*.
- Church, K. W. and P. Hanks. 1990. [Word association norms, mutual information, and lexicography](#). *Computational Linguistics*, 16(1):22–29.
- Clark, E. 1987. [The principle of contrast: A constraint on language acquisition](#). In B. MacWhinney, editor, *Mechanisms of language acquisition*, pages 1–33. LEA.
- Coccato, N. and D. Jurafsky. 1998. [Towards better integration of semantic predictors in statistical language modeling](#). *ICSLP*.
- Collobert, R. and J. Weston. 2007. [Fast semantic extraction using a novel neural network architecture](#). *ACL*.
- Collobert, R. and J. Weston. 2008. [A unified architecture for natural language processing: Deep neural networks with multitask learning](#). *ICML*.
- Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. 2011. [Natural language processing \(almost\) from scratch](#). *JMLR*, 12:2493–2537.
- Crawford, K. 2017. [The trouble with bias](#). Keynote at NeurIPS.
- Cruse, D. A. 2004. *Meaning in Language: an Introduction to Semantics and Pragmatics*. Oxford University Press. Second edition.
- Dagan, I., S. Marcus, and S. Markovitch. 1993. [Contextual word similarity and estimation from sparse data](#). *ACL*.
- Davies, M. 2012. [Expanding horizons in historical linguistics with the 400-million word Corpus of Historical American English](#). *Corpora*, 7(2):121–157.
- Davies, M. 2015. [The Wikipedia Corpus: 4.6 million articles, 1.9 billion words](#). Adapted from Wikipedia. <https://www.english-corpora.org/wiki/>.
- Deerwester, S. C., S. T. Dumais, G. W. Furnas, R. A. Harshman, T. K. Landauer, K. E. Lochbaum, and L. Streeter. 1988. [Computer information retrieval using latent semantic structure](#): US Patent 4,839,853.
- Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. 1990. [Indexing by latent semantics analysis](#). *JASIS*, 41(6):391–407.
- Ethayarajh, K., D. Duvenaud, and G. Hirst. 2019a. [Towards understanding linear word analogies](#). *ACL*.
- Ethayarajh, K., D. Duvenaud, and G. Hirst. 2019b. [Understanding undesirable word embedding associations](#). *ACL*.
- Fano, R. M. 1961. *Transmission of Information: A Statistical Theory of Communications*. MIT Press.
- Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. 2002. [Placing search in context: The concept revisited](#). *ACM Transactions on Information Systems*, 20(1):116—131.
- Firth, J. R. 1957. [A synopsis of linguistic theory 1930–1955](#). In *Studies in Linguistic Analysis*. Philological Society. Reprinted in Palmer, F. (ed.) 1968. *Selected Papers of J. R. Firth*. Longman, Harlow.

- Garg, N., L. Schiebinger, D. Jurafsky, and J. Zou. 2018. Word embeddings quantify 100 years of gender and ethnic stereotypes. *Proceedings of the National Academy of Sciences*, 115(16):E3635–E3644.
- Girard, G. 1718. *La justesse de la langue françoise: ou les différentes significations des mots qui passent pour synonymes*. Laurent d'Houry, Paris.
- Giuliano, V. E. 1965. The interpretation of word associations. *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings. Washington, D.C., USA, March 17, 1964*. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf>.
- Gladkova, A., A. Drozd, and S. Matsuoka. 2016. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. *NAACL Student Research Workshop*. Association for Computational Linguistics.
- Glenberg, A. M. and D. A. Robertson. 2000. Symbol grounding and meaning: A comparison of high-dimensional and embodied theories of meaning. *Journal of memory and language*, 43(3):379–401.
- Gonen, H. and Y. Goldberg. 2019. Lipstick on a pig: Debiasing methods cover up systematic gender biases in word embeddings but do not remove them. *NAACL HLT*.
- Gould, S. J. 1980. *The Panda's Thumb*. Penguin Group.
- Greenwald, A. G., D. E. McGhee, and J. L. K. Schwartz. 1998. Measuring individual differences in implicit cognition: the implicit association test. *Journal of personality and social psychology*, 74(6):1464–1480.
- Hamilton, W. L., J. Leskovec, and D. Jurafsky. 2016. Diachronic word embeddings reveal statistical laws of semantic change. *ACL*.
- Harris, Z. S. 1954. Distributional structure. *Word*, 10:146–162. Reprinted in J. Fodor and J. Katz, *The Structure of Language*, Prentice Hall, 1964 and in Z. S. Harris, *Papers in Structural and Transformational Linguistics*, Reidel, 1970, 775–794.
- Hellrich, J. and U. Hahn. 2016. Bad company—Neighborhoods in neural embedding spaces considered harmful. *COLING*.
- Hill, F., R. Reichart, and A. Korhonen. 2015. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.
- Hjelmslev, L. 1969. *Prologomena to a Theory of Language*. University of Wisconsin Press. Translated by Francis J. Whitfield; original Danish edition 1943.
- Hofmann, T. 1999. Probabilistic latent semantic indexing. *SIGIR-99*.
- Huang, E. H., R. Socher, C. D. Manning, and A. Y. Ng. 2012. Improving word representations via global context and multiple word prototypes. *ACL*.
- Jia, S., T. Meng, J. Zhao, and K.-W. Chang. 2020. Mitigating gender bias amplification in distribution by posterior regularization. *ACL*.
- Jones, M. P. and J. H. Martin. 1997. Contextual spelling correction using latent semantic analysis. *ANLP*.
- Joos, M. 1950. Description of language design. *JASA*, 22:701–708.
- Jurafsky, D. 2014. *The Language of Food*. W. W. Norton, New York.
- Jurgens, D., S. M. Mohammad, P. Turney, and K. Holyoak. 2012. SemEval-2012 task 2: Measuring degrees of relational similarity. *SEM 2012.
- Katz, J. J. and J. A. Fodor. 1963. The structure of a semantic theory. *Language*, 39:170–210.
- Kiela, D. and S. Clark. 2014. A systematic study of semantic vector space model parameters. *EACL 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)*.
- Kim, E. 2019. Optimize computational efficiency of skip-gram with negative sampling. https://aegis4048.github.io/optimize_computational_efficiency_of_skip-gram_with_negative_sampling.
- Lake, B. M. and G. L. Murphy. 2021. Word meaning in minds and machines. *Psychological Review*. In press.
- Landauer, T. K. and S. T. Dumais. 1997. A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240.
- Landauer, T. K., D. Laham, B. Rehder, and M. E. Schreiner. 1997. How well can passage meaning be derived without using word order? A comparison of Latent Semantic Analysis and humans. *COGSCI*.
- Lapesa, G. and S. Evert. 2014. A large scale evaluation of distributional semantic models: Parameters, interactions and model selection. *TACL*, 2:531–545.
- Lee, D. D. and H. S. Seung. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.
- Levy, O. and Y. Goldberg. 2014a. Dependency-based word embeddings. *ACL*.
- Levy, O. and Y. Goldberg. 2014b. Linguistic regularities in sparse and explicit word representations. *CoNLL*.
- Levy, O. and Y. Goldberg. 2014c. Neural word embedding as implicit matrix factorization. *NeurIPS*.
- Levy, O., Y. Goldberg, and I. Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *TACL*, 3:211–225.
- Li, J., X. Chen, E. H. Hovy, and D. Jurafsky. 2015. Visualizing and understanding neural models in NLP. *NAACL HLT*.
- Lin, Y., J.-B. Michel, E. Lieberman Aiden, J. Orwant, W. Brockman, and S. Petrov. 2012. Syntactic annotations for the Google Books NGram corpus. *ACL*.
- Linzen, T. 2016. Issues in evaluating semantic spaces using word analogies. *1st Workshop on Evaluating Vector-Space Representations for NLP*.
- Luhn, H. P. 1957. A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317.
- Manning, C. D., P. Raghavan, and H. Schütze. 2008. *Introduction to Information Retrieval*. Cambridge.
- Mikolov, T., K. Chen, G. S. Corrado, and J. Dean. 2013a. Efficient estimation of word representations in vector space. *ICLR 2013*.

- Mikolov, T., S. Kombrink, L. Burget, J. H. Černocký, and S. Khudanpur. 2011. Extensions of recurrent neural network language model. *ICASSP*.
- Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013b. Distributed representations of words and phrases and their compositionality. *NeurIPS*.
- Mikolov, T., W.-t. Yih, and G. Zweig. 2013c. [Linguistic regularities in continuous space word representations](#). *NAACL HLT*.
- Niwa, Y. and Y. Nitta. 1994. [Co-occurrence vectors from corpora vs. distance vectors from dictionaries](#). *COLING*.
- Nosek, B. A., M. R. Banaji, and A. G. Greenwald. 2002a. Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice*, 6(1):101.
- Nosek, B. A., M. R. Banaji, and A. G. Greenwald. 2002b. Math=male, me=female, therefore math \neq me. *Journal of personality and social psychology*, 83(1):44.
- Osgood, C. E., G. J. Suci, and P. H. Tannenbaum. 1957. *The Measurement of Meaning*. University of Illinois Press.
- Pennington, J., R. Socher, and C. D. Manning. 2014. [GloVe: Global vectors for word representation](#). *EMNLP*.
- Peterson, J. C., D. Chen, and T. L. Griffiths. 2020. Parallelograms revisited: Exploring the limitations of vector space models for simple analogies. *Cognition*, 205.
- Pilehvar, M. T. and J. Camacho-Collados. 2019. [WiC: the word-in-context dataset for evaluating context-sensitive meaning representations](#). *NAACL HLT*.
- Rehder, B., M. E. Schreiner, M. B. W. Wolfe, D. Laham, T. K. Landauer, and W. Kintsch. 1998. Using Latent Semantic Analysis to assess knowledge: Some technical considerations. *Discourse Processes*, 25(2-3):337–354.
- Rohde, D. L. T., L. M. Gonnerman, and D. C. Plaut. 2006. An improved model of semantic similarity based on lexical co-occurrence. *CACM*, 8:627–633.
- Rumelhart, D. E. and A. A. Abrahamson. 1973. A model for analogical reasoning. *Cognitive Psychology*, 5(1):1–28.
- Salton, G. 1971. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice Hall.
- Schluter, N. 2018. [The word analogy testing caveat](#). *NAACL HLT*.
- Schone, P. and D. Jurafsky. 2000. [Knowledge-free induction of morphology using latent semantic analysis](#). *CoNLL*.
- Schone, P. and D. Jurafsky. 2001a. [Is knowledge-free induction of multiword unit dictionary headwords a solved problem?](#) *EMNLP*.
- Schone, P. and D. Jurafsky. 2001b. [Knowledge-free induction of inflectional morphologies](#). *NAACL*.
- Schütze, H. 1992. [Dimensions of meaning](#). *Proceedings of Supercomputing '92*. IEEE Press.
- Schütze, H. 1997. *Ambiguity Resolution in Language Learning – Computational and Cognitive Models*. CSLI, Stanford, CA.
- Schütze, H., D. A. Hull, and J. Pedersen. 1995. [A comparison of classifiers and document representations for the routing problem](#). *SIGIR-95*.
- Schütze, H. and J. Pedersen. 1993. A vector model for syntagmatic and paradigmatic relatedness. *9th Annual Conference of the UW Centre for the New OED and Text Research*.
- Sparck Jones, K. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.
- Sparck Jones, K. 1986. *Synonymy and Semantic Classification*. Edinburgh University Press, Edinburgh. Republication of 1964 PhD Thesis.
- Switzer, P. 1965. Vector images in document retrieval. *Statistical Association Methods For Mechanized Documentation. Symposium Proceedings*. Washington, D.C., USA, March 17, 1964. <https://nvlpubs.nist.gov/nistpubs/Legacy/MP/nbsmiscellaneouspub269.pdf>.
- Tian, Y., V. Kulkarni, B. Perozzi, and S. Skiena. 2016. On the convergent properties of word embedding methods. ArXiv preprint arXiv:1605.03956.
- Turian, J., L. Ratinov, and Y. Bengio. 2010. [Word representations: a simple and general method for semi-supervised learning](#). *ACL*.
- Turney, P. D. and M. L. Littman. 2005. Corpus-based learning of analogies and semantic relations. *Machine Learning*, 60(1-3):251–278.
- van der Maaten, L. and G. E. Hinton. 2008. [Visualizing high-dimensional data using t-SNE](#). *JMLR*, 9:2579–2605.
- Wierzbicka, A. 1992. *Semantics, Culture, and Cognition: University Human Concepts in Culture-Specific Configurations*. Oxford University Press.
- Wierzbicka, A. 1996. *Semantics: Primes and Universals*. Oxford University Press.
- Wittgenstein, L. 1953. *Philosophical Investigations*. (Translated by Anscombe, G.E.M.). Blackwell.
- Zhao, J., T. Wang, M. Yatskar, V. Ordonez, and K.-W. Chang. 2017. [Men also like shopping: Reducing gender bias amplification using corpus-level constraints](#). *EMNLP*.
- Zhao, J., Y. Zhou, Z. Li, W. Wang, and K.-W. Chang. 2018. [Learning gender-neutral word embeddings](#). *EMNLP*.

Language Modeling

Introduction to N-grams

Probabilistic Language Models

Today's goal: assign a probability to a sentence

- Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
- Spell Correction

Why?

- 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.!!

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

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

More variables:

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

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

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

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

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

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

How to estimate these probabilities

Could we just count and divide?

$P(\text{the lits water is so transparent that}) =$

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

No! Too many possible sentences!

We'll never see enough data for estimating these

Markov Assumption



Andrei Markov

Simplifying assumption:

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

Or maybe

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

Markov Assumption

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

In other words, we approximate each component in the product

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

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 which I had just put into the machine room
on the fifth floor crashed.”

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

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

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

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

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

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

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

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

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

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

$= .000031$

What kinds of knowledge?

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

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

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

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

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

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

$P(\text{i} \mid \langle s \rangle) = .25$

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 indispensible 40
serve as the individual 234

Google Book N-grams

<http://ngrams.googlecode.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.

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 _____

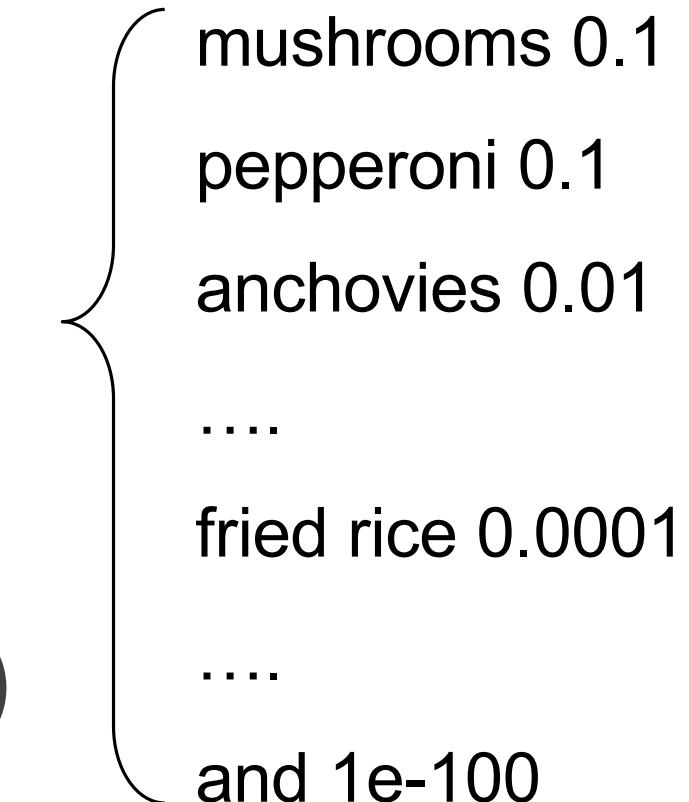
The 33rd President of the US was _____

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



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

The Shannon Game intuition for perplexity

From Josh Goodman

Perplexity is weighted equivalent branching factor

How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9'

- Perplexity 10

How hard is recognizing (30,000) names at Microsoft.

- Perplexity = 30,000

Let's imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)
- What is the perplexity? Next slide

The Shannon Game intuition for perplexity

Josh Goodman: imagine a call-routing phone system gets 120K calls and has to recognize

- "Operator" (let's say this occurs 1 in 4 calls)
- "Sales" (1 in 4)
- "Technical Support" (1 in 4)
- 30,000 different names (each name occurring 1 time in the 120K calls)

We get the perplexity of this sequence of length 120K by first multiplying 120K probabilities (90K of which are 1/4 and 30K of which are 1/120K), and then taking the inverse 120,000th root:

$$\text{Perp} = (\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \dots * \frac{1}{120K} * \frac{1}{120K} * \dots)^{(-1/120K)}$$

But this can be arithmetically simplified to just $N = 4$: the operator (1/4), the sales (1/4), the tech support (1/4), and the 30,000 names (1/120,000):

$$\text{Perplexity} = ((\frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4})^{(-1/4)} = 52.6$$

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?

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

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

$\langle s \rangle$ I
I want
want to
to eat
eat Chinese
Chinese food
food $\langle /s \rangle$

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 training set author of the LM that generated 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:

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

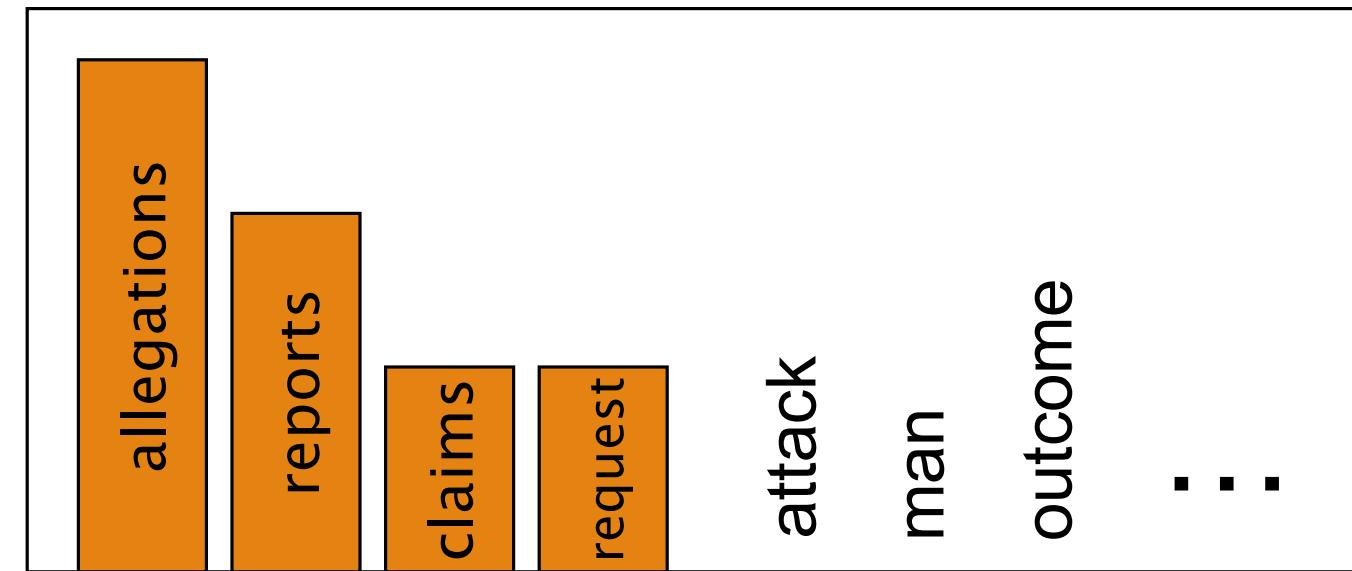
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

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

2.5 allegations

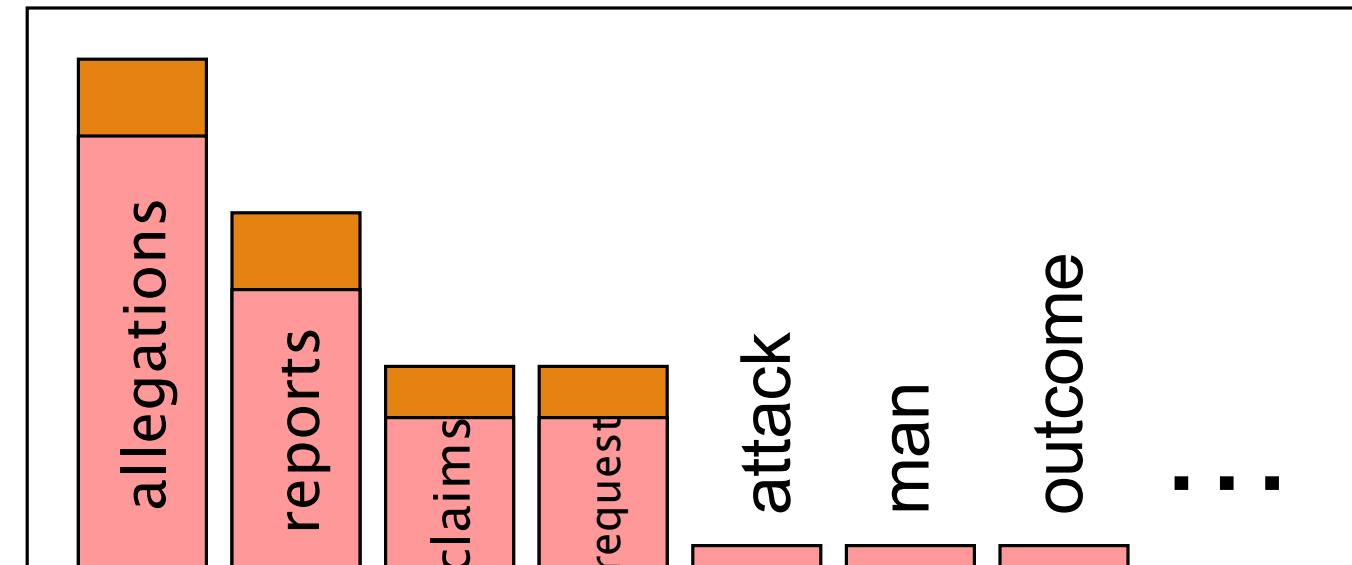
1.5 reports

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:

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

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:

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

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 turned out to 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!

It sure looks like $c^* = (c - .75)$

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)!

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

discounted bigram Interpolation weight

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 _____* *Kongses* ?
- “Kong” turns out to be more common than “glasses”
- ... but “Kong” always follows “Hong”

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

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

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

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 (Kong) occurring in only one context (Hong) 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

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

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

Continuation count = Number of unique single word contexts for \bullet

Language Modeling

Advanced:
Kneser-Ney Smoothing

Spelling Correction: Edit Distance

Pawan Goyal

CSE, IITKGP

Week 2: Lecture 1

Spelling Correction

NPTEL

Spelling Correction

I am writing this email on behaf of ...

NPTEL

Spelling Correction

I am writing this email on behaf of ...

The user typed ‘behaf’.

Which are some close words?

Spelling Correction

I am writing this email on behaf of ...

The user typed ‘behaf’.

Which are some close words?

- behalf
- behave
-

Spelling Correction

I am writing this email on behaf of ...

The user typed ‘behaf’.

Which are some close words?

- behalf
- behave
-

Isolated word error correction

- Pick the one that is closest to ‘behaf’

Spelling Correction

I am writing this email on behaf of ...

The user typed ‘behaf’.

Which are some close words?

- behalf
- behave
-

Isolated word error correction

- Pick the one that is closest to ‘behaf’
- How to define ‘closest’?

Spelling Correction

I am writing this email on behaf of ...

The user typed ‘behaf’.

Which are some close words?

- behalf
- behave
-

Isolated word error correction

- Pick the one that is closest to ‘behaf’
- How to define ‘closest’?
- Need a **distance metric**

Spelling Correction

I am writing this email on behaf of ...

The user typed 'behaf'.

Which are some close words?

- behalf
- behave
-

Isolated word error correction

- Pick the one that is closest to 'behaf'
- How to define 'closest'?
- Need a **distance metric**
- The simplest metric: **edit distance**

Edit Distance

- The minimum edit distance between two strings

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
 - ▶ Insertion
 - ▶ Deletion
 - ▶ Substitution

Minimum Edit Distance

Example

Edit distance from ‘intention’ to ‘execution’

Minimum Edit Distance

Example

Edit distance from ‘intention’ to ‘execution’

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has a cost of 1 (Levenshtein)
 - ▶ Distance between these is 5

Minimum Edit Distance

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N
d	s	s		i	s				

- If each operation has a cost of 1 (Levenshtein)
 - ▶ Distance between these is 5
- If substitution costs 2 (alternate version)
 - ▶ Distance between these is 8

How to find the Minimum Edit Distance?

NPTEL

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming
- **Operators:** insert, delete, substitute

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming
- **Operators:** insert, delete, substitute
- **Goal state:** the word we are trying to get to

How to find the Minimum Edit Distance?

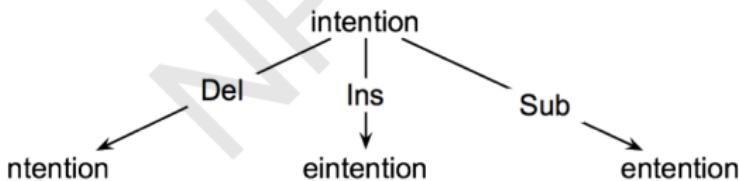
Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming
- **Operators:** insert, delete, substitute
- **Goal state:** the word we are trying to get to
- **Path cost:** what we want to minimize: the number of edits

How to find the Minimum Edit Distance?

Searching for a path (sequence of edits) from the *start string* to the *final string*:

- **Initial state:** the word we are transforming
- **Operators:** insert, delete, substitute
- **Goal state:** the word we are trying to get to
- **Path cost:** what we want to minimize: the number of edits



Minimum Edit as Search

How to navigate?

- The space of all edit sequences is huge

Minimum Edit as Search

How to navigate?

- The space of all edit sequences is huge
- Lot of distinct paths end up at the same state

Minimum Edit as Search

How to navigate?

- The space of all edit sequences is huge
- Lot of distinct paths end up at the same state
- Don't have to keep track of all of them

Minimum Edit as Search

How to navigate?

- The space of all edit sequences is huge
- Lot of distinct paths end up at the same state
- Don't have to keep track of all of them
- Keep track of the shortest path to each state

Defining Minimum Edit Distance Matrix

For two strings

- X of length n
- Y of length m

Defining Minimum Edit Distance Matrix

For two strings

- X of length n
- Y of length m

We define $D(i,j)$

- the edit distance between $X[1..i]$ and $Y[1..j]$
- i.e., the first i characters of X and the first j characters of Y

Defining Minimum Edit Distance Matrix

For two strings

- X of length n
- Y of length m

We define $D(i,j)$

- the edit distance between $X[1..i]$ and $Y[1..j]$
- i.e., the first i characters of X and the first j characters of Y

Thus, the edit distance between X and Y is $D(n,m)$

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n, m)$

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n, m)$
- Solving problems by combining solutions to subproblems

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n, m)$
- Solving problems by combining solutions to subproblems
- Bottom-up

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems
- Bottom-up
 - ▶ Compute $D(i,j)$ for small i,j

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems
- Bottom-up
 - ▶ Compute $D(i,j)$ for small i,j
 - ▶ Compute larger $D(i,j)$ based on previously computed smaller values

Computing Minimum Edit Distance

Dynamic Programming

- A tabular computation of $D(n,m)$
- Solving problems by combining solutions to subproblems
- Bottom-up
 - ▶ Compute $D(i,j)$ for small i,j
 - ▶ Compute larger $D(i,j)$ based on previously computed smaller values
 - ▶ Compute $D(i,j)$ for all i and j till you get to $D(n,m)$

Dynamic Programming Algorithm

Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

Recurrence Relation:

For each $i = 1 \dots M$

For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases}$$

Termination:

$D(N, M)$ is distance

The Edit Distance Table

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

The Edit Distance Table

N	9										
O	8										
I	7										
T	6										
N	5										
E	4										
T	3										
N	2										
I	1										
#	0	1	2	3	4	5	6	7	8	9	
	#	E	X	E	C	U	T	I	O	N	

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Computing Alignments

NPTEL

Computing Alignments

- Computing edit distance may not be sufficient for some applications

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - ▶ We often need to align characters of the two strings to each other

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - ▶ We often need to align characters of the two strings to each other
- We do this by keeping a “backtrace”

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - ▶ We often need to align characters of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - ▶ We often need to align characters of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,

Computing Alignments

- Computing edit distance may not be sufficient for some applications
 - ▶ We often need to align characters of the two strings to each other
- We do this by keeping a “backtrace”
- Every time we enter a cell, remember where we came from
- When we reach the end,
 - ▶ Trace back the path from the upper right corner to read off the alignment

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

The Edit Distance Table

N	9									
O	8									
I	7									
T	6									
N	5									
E	4	3	4							
T	3	4	5							
N	2	3	4							
I	1	2	3							
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

Minimum Edit with Backtrace

n	9	↓ 8	↙ ↘ 9	↙ ↘ 10	↙ ↘ 11	↙ ↘ 12	↓ 11	↓ 10	↓ 9	↙ 8	
o	8	↓ 7	↙ ↘ 8	↙ ↘ 9	↙ ↘ 10	↙ ↘ 11	↓ 10	↓ 9	↙ 8	← 9	
i	7	↓ 6	↙ ↘ 7	↙ ↘ 8	↙ ↘ 9	↙ ↘ 10	↓ 9	↙ 8	← 9	← 10	
t	6	↓ 5	↙ ↘ 6	↙ ↘ 7	↙ ↘ 8	↙ ↘ 9	↙ 8	← 9	← 10	← 11	
n	5	↓ 4	↙ ↘ 5	↙ ↘ 6	↙ ↘ 7	↙ ↘ 8	↙ ↘ 9	↙ ↘ 10	↙ ↘ 11	↙ ↘ 10	
e	4	↙ 3	← 4	↙ ← 5	← 6	← 7	← 8	↙ ↘ 9	↙ ↘ 10	↓ 9	
t	3	↙ ↘ 4	↙ ↘ 5	↙ ↘ 6	↙ ↘ 7	↙ ↘ 8	↙ 7	← 8	↙ ↘ 9	↓ 8	
n	2	↙ ↘ 3	↙ ↘ 4	↙ ↘ 5	↙ ↘ 6	↙ ↘ 7	↙ ↘ 8	↓ 7	↙ ↘ 8	↙ 7	
i	1	↙ ↘ 2	↙ ↘ 3	↙ ↘ 4	↙ ↘ 5	↙ ↘ 6	↙ ↘ 7	↙ 6	← 7	← 8	
#	0	1	2	3	4	5	6	7	8	9	
	#	e	x	e	c	u	t	i	o	n	

Adding Backtrace to Minimum Edit

Base conditions:

$$D(i, 0) = i$$

$$D(0, j) = j$$

Termination:

$D(N, M)$ is distance

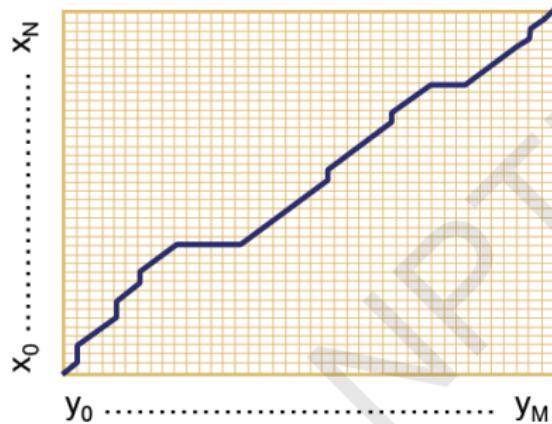
Recurrence Relation:

For each $i = 1 \dots M$

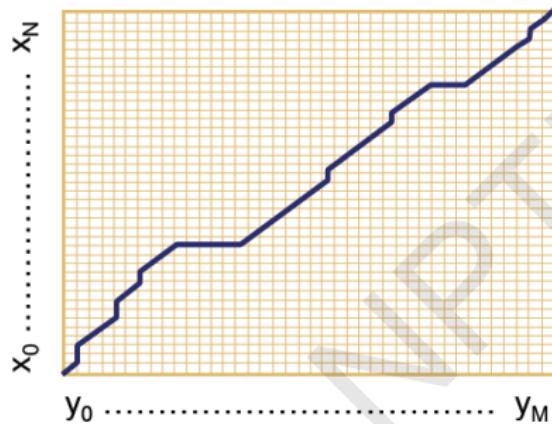
For each $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + 2; & \begin{cases} \text{if } X(i) \neq Y(j) \\ 0; \quad \text{if } X(i) = Y(j) \end{cases} \\ \text{substitution} \end{cases}$$
$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

The distance matrix

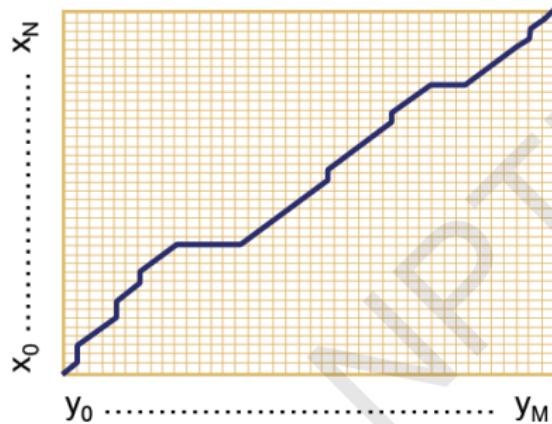


The distance matrix



Every non-decreasing path from (0,0) to (M,N) corresponds to an alignment of two sequences.

The distance matrix



Every non-decreasing path from $(0,0)$ to (M,N) corresponds to an alignment of two sequences.

An optimal alignment is composed of optimal sub-alignments.

Result of Backtrace

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

Performance

Time

NPTEL

Performance

Time

$O(nm)$

Space

Performance

Time

$O(nm)$

Space

$O(nm)$

Backtrace

Performance

Time

$O(nm)$

Space

$O(nm)$

Backtrace

$O(n + m)$

Weighted Edit Distance, Other variations

Pawan Goyal

CSE, IITKGP

Week 2: Lecture 2

Weighted Edit Distance

Why to add weights to the computation?

- Some letters are more likely to be mistyped.

Confusion Matrix for Spelling Errors

sub[X, Y] = Substitution of X (incorrect) for Y (correct)
Y (correct)

X	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	8	3	0	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

Keyboard Design



Weighted Minimum Edit Distance

Initialization:

$$D(0,0) = 0$$

$$D(i,0) = D(i-1,0) + \text{del}[x(i)]; \quad 1 < i \leq N$$

$$D(0,j) = D(0,j-1) + \text{ins}[y(j)]; \quad 1 < j \leq M$$

Recurrence Relation:

$$D(i,j) = \min \begin{cases} D(i-1,j) + \text{del}[x(i)] \\ D(i,j-1) + \text{ins}[y(j)] \\ D(i-1,j-1) + \text{sub}[x(i), y(j)] \end{cases}$$

Termination:

$D(N,M)$ is distance

How to modify the algorithm with transpose?

Transpose

- $\text{transpose}(x, y) = (y, x)$
- Also known as metathesis

How to modify the algorithm with transpose?

Transpose

- $\text{transpose}(x, y) = (y, x)$
- Also known as metathesis

Modification to the dynamic programmic algorithm

$$D[i][j] = \min \begin{cases} D(i-1, j) + 1 & (\text{deletion}) \\ D(i, j-1) + 1 & (\text{insertion}) \\ D(i-1, j-1) + \begin{cases} 1 & \text{if } (x[i] \neq y[j]) \text{ (substitution)} \\ 0 & \text{otherwise} \end{cases} & \\ D(i-2, j-2) + 1 & (x[i] = y[j-1] \text{ and } x[i-1] = y[j]) \\ & (\text{transposition}) \end{cases}$$

How to find dictionary entries with smallest edit distance?

NPTEL

How to find dictionary entries with smallest edit distance?

Naïve Method

Compute edit ditance from the query term to each dictionary term – an exhaustive search

How to find dictionary entries with smallest edit distance?

Naïve Method

Compute edit distance from the query term to each dictionary term – an exhaustive search

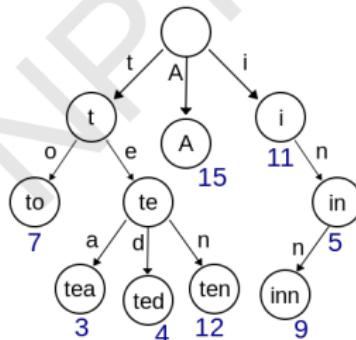
Can be made efficient if we do it over a trie structure

How to find dictionary entries with smallest edit distance?

Naïve Method

Compute edit ditance from the query term to each dictionary term – an exhaustive search

Can be made efficient if we do it over a trie structure



How to find dictionary entries with smallest edit distance?

NPTEL

How to find dictionary entries with smallest edit distance?

- Generate all possible terms with an edit distance ≤ 2 (deletion + transpose + substitution + insertion) from the query term and search them in the dictionary.

How to find dictionary entries with smallest edit distance?

- Generate all possible terms with an edit distance ≤ 2 (deletion + transpose + substitution + insertion) from the query term and search them in the dictionary.
- For a word of length 9, alphabet of size 36, this will lead to 114,324 terms to search for

How to find dictionary entries with smallest edit distance?

- Generate all possible terms with an edit distance ≤ 2 (deletion + transpose + substitution + insertion) from the query term and search them in the dictionary.
- For a word of length 9, alphabet of size 36, this will lead to 114,324 terms to search for
- For Chinese alphabet size is 70,000 (Unicode Han Characters)

How to find dictionary entries with smallest edit distance?

Symmetric Delete Spelling Correction

- Generate terms with an edit distance ≤ 2 (deletes) from each dictionary term (offline)
- Generate terms with an edit distance ≤ 2 (deletes) from the input terms and search in dictionary

How to find dictionary entries with smallest edit distance?

Symmetric Delete Spelling Correction

- Generate terms with an edit distance ≤ 2 (deletes) from each dictionary term (offline)
- Generate terms with an edit distance ≤ 2 (deletes) from the input terms and search in dictionary

Number of deletes within edit distance ≤ 2 for a word of length 9 will be 45

How to find dictionary entries with smallest edit distance?

Symmetric Delete Spelling Correction

- Generate terms with an edit distance ≤ 2 (deletes) from each dictionary term (offline)
- Generate terms with an edit distance ≤ 2 (deletes) from the input terms and search in dictionary

Number of deletes within edit distance ≤ 2 for a word of length 9 will be 45

A further check is required to remove the false positives

Spelling Correction

NPTEL

Spelling Correction

Types of spelling errors: Non-word Errors

- behaf → behalf

Spelling Correction

Types of spelling errors: Non-word Errors

- behaf → behalf

Types of spelling errors: Real-word Errors

- **Typographical errors:** three → there
- **Cognitive errors (homophones):** piece → peace, too → two

Non-word spelling errors

Non-word spelling error detection

- Any word not in a dictionary is an error
- The larger the dictionary the better

Non-word spelling errors

Non-word spelling error detection

- Any word not in a dictionary is an error
- The larger the dictionary the better

Non-word spelling error correction

- Generate candidates: real words that are similar to the error word
- Choose the best one:
 - ▶ Shortest weighted edit distance
 - ▶ Highest noisy channel probability

Real word spelling errors

For each word w , generate candidate set

- Find candidate words with similar pronunciations
- Find candidate words with similar spelling
- Include w in candidate set

Real word spelling errors

For each word w , generate candidate set

- Find candidate words with similar pronunciations
- Find candidate words with similar spelling
- Include w in candidate set

Choosing best candidate

- Noisy Channel

Noisy Channel Model for Spelling Correction

Pawan Goyal

CSE, IITKGP

Week 2: Lecture 3

Noisy Channel

We see an observation x of the misspelled word

Find the correct word w

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

Noisy Channel

We see an observation x of the misspelled word

Find the correct word w

$$\begin{aligned}\hat{w} &= \arg \max_{w \in V} P(w|x) \\ &= \arg \max_{w \in V} \frac{P(x|w)P(w)}{P(x)}\end{aligned}$$

Noisy Channel

We see an observation x of the misspelled word

Find the correct word w

$$\begin{aligned}\hat{w} &= \arg \max_{w \in V} P(w|x) \\ &= \arg \max_{w \in V} \frac{P(x|w)P(w)}{P(x)} \\ &= \arg \max_{w \in V} P(x|w)P(w)\end{aligned}$$

*Non-word spelling error: *acress**

Words with similar spelling

Small edit distance to error

Words with similar pronunciation

Small edit distance of pronunciation to error

*Non-word spelling error: *acress**

Words with similar spelling

Small edit distance to error

Words with similar pronunciation

Small edit distance of pronunciation to error

Damerau-Levenshtein edit distance

Minimum edit distance, where edits are:

*Non-word spelling error: *acress**

Words with similar spelling

Small edit distance to error

Words with similar pronunciation

Small edit distance of pronunciation to error

Damerau-Levenshtein edit distance

Minimum edit distance, where edits are:

Insertion, Deletion, Substitution,

*Non-word spelling error: *acress**

Words with similar spelling

Small edit distance to error

Words with similar pronunciation

Small edit distance of pronunciation to error

Damerau-Levenshtein edit distance

Minimum edit distance, where edits are:

Insertion, Deletion, Substitution,

Transposition of two adjacent letters

Words within edit distance 1 of acress

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	-	deletion
acress	cress	-	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	-	s	insertion
acress	acres	-	s	insertion

Candidate generation

- 80% of errors are within edit distance 1
 - Almost all errors within edit distance 2

Candidate generation

- 80% of errors are within edit distance 1
- Almost all errors within edit distance 2

Allow deletion of space or hyphen

- thisidea → this idea
- inlaw → in-law

Computing error probability: confusion matrix

- $\text{del}[x,y]$: count (xy typed as x)
- $\text{ins}[x,y]$: count (x typed as xy)
- $\text{sub}[x,y]$: count (x typed as y)
- $\text{trans}[x,y]$: count(xy typed as yx)

Computing error probability: confusion matrix

- $\text{del}[x,y]$: count (xy typed as x)
- $\text{ins}[x,y]$: count (x typed as xy)
- $\text{sub}[x,y]$: count (x typed as y)
- $\text{trans}[x,y]$: count(xy typed as yx)

Insertion and deletion are conditioned on previous character

Channel model

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Channel model for across

Candidate Correction	Correct Letter	Error Letter	x w	P(x word)
actress	t	-	c ct	.000117
cress	-	a	a #	.00000144
caress	ca	ac	ac ca	.00000164
access	c	r	r c	.000000209
across	o	e	e o	.0000093
acres	-	s	es e	.0000321
acres	-	s	ss s	.0000342

Noisy channel probability for *acress*

Candidate Correction	Correct Letter	Error Letter	x w	P(x word)	P(word)	$10^9 * P(x w)P(w)$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac ca	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss s	.0000342	.0000318	1.0

Using a bigram language model

- “ ... versatile across whose ... ”

Using a bigram language model

- “ ... versatile across whose ...”
 - Counts from the Corpus of Contemporary American English with add-1 smoothing

Using a bigram language model

- “ ... versatile across whose ... ”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = 0.000021$, $P(\text{across}|\text{versatile}) = 0.000021$

Using a bigram language model

- “ ... versatile across whose ... ”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = 0.000021$, $P(\text{across}|\text{versatile}) = 0.000021$
- $P(\text{whose}|\text{actress}) = 0.0010$, $P(\text{whose}|\text{across}) = 0.000006$

Using a bigram language model

- “ ... versatile across whose ... ”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = 0.000021$, $P(\text{across}|\text{versatile}) = 0.000021$
- $P(\text{whose}|\text{actress}) = 0.0010$, $P(\text{whose}|\text{across}) = 0.000006$
- $P(\text{"versatile actress whose"}) = 0.000021 * 0.0010 = 210 \times 10^{-10}$

Using a bigram language model

- “ ... versatile across whose ... ”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress}|\text{versatile}) = 0.000021, P(\text{across}|\text{versatile}) = 0.000021$
- $P(\text{whose}|\text{actress}) = 0.0010, P(\text{whose}|\text{across}) = 0.000006$
- $P(\text{"versatile actress whose"}) = 0.000021 * 0.0010 = 210 \times 10^{-10}$
- $P(\text{"versatile across whose"}) = 0.000021 * 0.000006 = 1 \times 10^{-10}$

Real-word spelling errors

- The study was conducted mainly **be** John Black
 - The design **an** construction of the system ...

Real-word spelling errors

- The study was conducted mainly **be** John Black
- The design **an** construction of the system ...

25-40% of spelling errors are real words

Noisy channel for real-word spell correction

Given a sentence $X = w_1, w_2, w_3 \dots, w_n$

- Candidate (w_1) = $\{w_1, w'_1, w''_1, w'''_1, \dots\}$
- Candidate (w_2) = $\{w_2, w'_2, w''_2, w'''_2, \dots\}$
- Candidate (w_3) = $\{w_3, w'_3, w''_3, w'''_3, \dots\}$

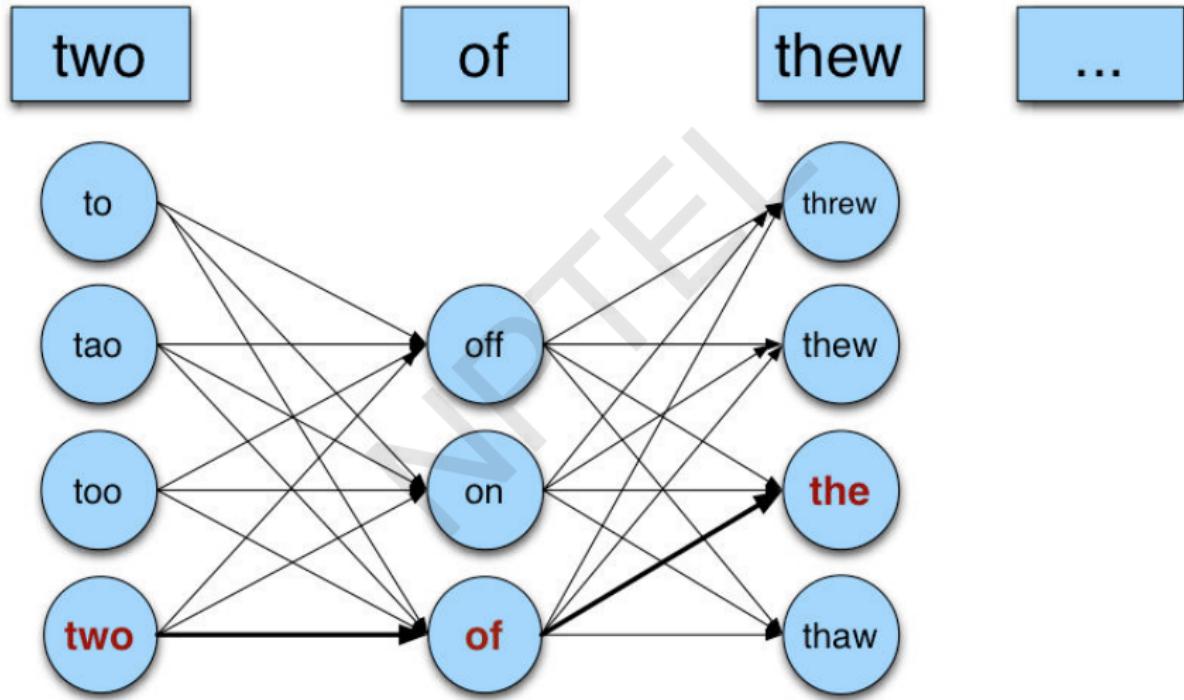
Noisy channel for real-word spell correction

Given a sentence $X = w_1, w_2, w_3 \dots, w_n$

- Candidate (w_1) = $\{w_1, w'_1, w''_1, w'''_1, \dots\}$
- Candidate (w_2) = $\{w_2, w'_2, w''_2, w'''_2, \dots\}$
- Candidate (w_3) = $\{w_3, w'_3, w''_3, w'''_3, \dots\}$

Choose the sequence W that maximizes $P(W|X)$

Noisy channel for real-world spell correction



Simplification: One error per sentence

Choose among all possible sentences with one word replaced

two of thew

- w_1, w''_2, w_3 two **off** thew
- w_1, w_2, w'_3 two of **the**
- w'''_1, w_2, w_3 **too** of thew

Simplification: One error per sentence

Choose among all possible sentences with one word replaced

two of thew

- w_1, w''_2, w_3 two **off** thew
- w_1, w_2, w'_3 two of **the**
- w'''_1, w_2, w_3 **too** of thew

Choose the sequence W that maximizes $P(W|X)$

Getting the probability values

Noisy Channel

$$\hat{W} = \arg \max_{W \in S} P(W|X)$$

where X is the observed sentence and S is the set of all the possible sequences from the candidate set

Getting the probability values

Noisy Channel

$$\hat{W} = \arg \max_{W \in S} P(W|X)$$

where X is the observed sentence and S is the set of all the possible sequences from the candidate set

$$= \arg \max_{W \in S} P(X|W)P(W)$$

Getting the probability values

Noisy Channel

$$\hat{W} = \arg \max_{W \in S} P(W|X)$$

where X is the observed sentence and S is the set of all the possible sequences from the candidate set

$$= \arg \max_{W \in S} P(X|W)P(W)$$

$P(X|W)$

- Same as for non-word spelling correction

Getting the probability values

Noisy Channel

$$\hat{W} = \arg \max_{W \in S} P(W|X)$$

where X is the observed sentence and S is the set of all the possible sequences from the candidate set

$$= \arg \max_{W \in S} P(X|W)P(W)$$

$P(X|W)$

- Same as for non-word spelling correction
- Also require probability for no error $P(w|w)$

Probability of no error

What is the probability for a correctly typed word? $P(\text{"the"}|\text{"the"})$

Probability of no error

What is the probability for a correctly typed word? $P(\text{"the"}|\text{"the"})$

It may depend on the source text under consideration

- 1 error in 10 words → 0.9
- 1 error in 100 words → 0.99

$P(W)$

Use Language Model

- Unigram
- Bigram
- ...

N-gram Language Models

Pawan Goyal

CSE, IITKGP

Week 2: Lecture 4

Context Sensitive Spelling Correction

NPTEL

Context Sensitive Spelling Correction

The office is about fifteen minuets from my house

Context Sensitive Spelling Correction

The office is about fifteen minuets from my house

min·u·et  noun \min-yə-'wet\

: a slow, graceful dance that was popular in the 17th and
18th centuries

: the music for a minuet

Context Sensitive Spelling Correction

The office is about fifteen minuets from my house

min·u·et  noun \min-yə-'wet\

: a slow, graceful dance that was popular in the 17th and 18th centuries

: the music for a minuet

Use a Language Model

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

Probabilistic Language Models: Applications

Speech Recognition

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

Probabilistic Language Models: Applications

Speech Recognition

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

Machine Translation

Which sentence is more plausible in the target language?

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

Probabilistic Language Models: Applications

Speech Recognition

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

Machine Translation

Which sentence is more plausible in the target language?

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

Other Applications

- Context Sensitive Spelling Correction
- Natural Language Generation
- ...

Completion Prediction

- Language model also supports predicting the completion of a sentence.
 - ▶ Please turn off your cell ...
 - ▶ Your program does not ...

Completion Prediction

- Language model also supports predicting the completion of a sentence.
 - ▶ Please turn off your cell ...
 - ▶ Your program does not ...
- *Predictive text input* systems can guess what you are typing and give choices on how to complete it.

Probabilistic Language Modeling

- **Goal:** Compute the probability of a sentence or sequence of words:

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

Probabilistic Language Modeling

- **Goal:** Compute the probability of a sentence or sequence of words:

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

- **Related Task:** probability of an upcoming word:

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

Probabilistic Language Modeling

- **Goal:** Compute the probability of a sentence or sequence of words:

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

- **Related Task:** probability of an upcoming word:

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

- A model that computes either of these is called a **language model**

$P(W)$

How to compute the joint probability

$P(\text{about, fifteen, minutes, from})$

$P(W)$

How to compute the joint probability

$P(\text{about, fifteen, minutes, from})$

Basic Idea

Rely on the Chain Rule of Probability

The Chain Rule

Conditional Probabilities

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

The Chain Rule

Conditional Probabilities

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

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

The Chain Rule

Conditional Probabilities

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

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

More Variables

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

The Chain Rule

Conditional Probabilities

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

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

More Variables

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

The Chain Rule in General

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

Probability of words in sentences

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

P("about fifteen minutes from") =

Probability of words in sentences

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

P("about fifteen minutes from") =

P(about) x P(fifteen | about) x P(minutes | about fifteen) x P(from | about fifteen minutes)

Estimating These Probability Values

Count and divide

$$P(\text{office} \mid \text{about fifteen minutes from}) = \frac{\text{Count}(\text{about fifteen minutes from office})}{\text{Count}(\text{about fifteen minutes from})}$$

Estimating These Probability Values

Count and divide

$$P(\text{office} \mid \text{about fifteen minutes from}) = \frac{\text{Count}(\text{about fifteen minutes from office})}{\text{Count}(\text{about fifteen minutes from})}$$

What is the problem

We may never see enough data for estimating these

Markov Assumption

Simplifying Assumption: Use only the previous word

$$P(\text{office} \mid \text{about fifteen minutes from}) \approx P(\text{office} \mid \text{from})$$

Markov Assumption

Simplifying Assumption: Use only the previous word

$$P(\text{office} \mid \text{about fifteen minutes from}) \approx P(\text{office} \mid \text{from})$$

Or the couple previous words

$$P(\text{office} \mid \text{about fifteen minutes from}) \approx P(\text{office} \mid \text{minutes from})$$

Markov Assumption

More Formally: kth order Markov Model

Chain Rule:

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

Markov Assumption

More Formally: kth order Markov Model

Chain Rule:

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

Using Markov Assumption: only k previous words

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

Markov Assumption

More Formally: kth order Markov Model

Chain Rule:

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

Using Markov Assumption: only k previous words

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

We approximate each component in the product

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

N-Gram Models

$P(\text{office} \mid \text{about fifteen minutes from})$

An N -gram model uses only $N - 1$ words of prior context.

N-Gram Models

$P(\text{office} \mid \text{about fifteen minutes from})$

An N -gram model uses only $N - 1$ words of prior context.

- Unigram: $P(\text{office})$
- Bigram: $P(\text{office} \mid \text{from})$
- Trigram: $P(\text{office} \mid \text{minutes from})$

N-Gram Models

$P(\text{office} \mid \text{about fifteen minutes from})$

An N -gram model uses only $N - 1$ words of prior context.

- Unigram: $P(\text{office})$
- Bigram: $P(\text{office} \mid \text{from})$
- Trigram: $P(\text{office} \mid \text{minutes from})$

Markov model and Language Model

N-Gram Models

$P(\text{office} \mid \text{about fifteen minutes from})$

An N -gram model uses only $N - 1$ words of prior context.

- Unigram: $P(\text{office})$
- Bigram: $P(\text{office} \mid \text{from})$
- Trigram: $P(\text{office} \mid \text{minutes from})$

Markov model and Language Model

An N -gram model is an $N - 1$ -order Markov Model

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- In general, an insufficient model of language:

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- In general, an insufficient model of language:
language has long-distance dependencies:
“The computer which I had just put into the machine room on the fifth floor **crashed**.”

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- In general, an insufficient model of language:
language has long-distance dependencies:
“The computer which I had just put into the machine room on the fifth floor **crashed.**”
- In most of the applications, we can get away with N-gram models

Estimating N-grams probabilities

NPTEL

Estimating N-grams probabilities

Maximum Likelihood Estimate

Value that makes the observed data the “most probable”

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

Estimating N-grams probabilities

Maximum Likelihood Estimate

Value that makes the observed data the “most probable”

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

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

An Example

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

<s>I am here </s>
<s>who am I </s>
<s>I would like to know </s>

An Example

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

<s>I am here </s>
<s>who am I </s>
<s>I would like to know </s>

Estimating bigrams

P(I|<s>) =

P(</s>|here) =

P(would | I) =

P(here | am) =

P(know | like) =

An Example

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

<s>I am here </s>
<s>who am I </s>
<s>I would like to know </s>

Estimating bigrams

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

$$P(\text{</s>} | \text{here}) = 1$$

$$P(\text{would} | I) = 1/3$$

$$P(\text{here} | \text{am}) = 1/2$$

$$P(\text{know} | \text{like}) = 0$$

Bigram counts from 9222 Restaurant 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

Computing bigram probabilities

Normalize by unigrams

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

Computing bigram probabilities

Normalize by unigrams

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

Bigram Probabilities

	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

Computing Sentence Probabilities

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

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

Computing Sentence Probabilities

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

$$\begin{aligned} &= P(I | < s >) \times P(\text{want} | I) \times P(\text{english} | \text{want}) \times P(\text{food} | \text{english}) \times P(</ s > | \text{food}) \\ &= 0.000031 \end{aligned}$$

What knowledge does n-gram represent?

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

Practical Issues

Everything in log space

- Avoids underflow

Practical Issues

Everything in log space

- Avoids underflow
- Adding is faster than multiplying

Practical Issues

Everything in log space

- Avoids underflow
- Adding is faster than multiplying

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

Handling zeros

Use smoothing

Language Modeling Toolkit

SRILM

<http://www.speech.sri.com/projects/srilm/>

Google N-grams

Number of tokens: 1,024,908,267,229

Number of sentences: 95,119,665,584

Number of unigrams: 13,588,391

Number of bigrams: 314,843,401

Number of trigrams: 977,069,902

Number of fourgrams: 1,313,818,354

Number of fivegrams: 1,176,470,663

[http://googleresearch.blogspot.in/2006/08/
all-our-n-gram-are-belong-to-you.html](http://googleresearch.blogspot.in/2006/08/all-our-n-gram-are-belong-to-you.html)

Example from the 4-gram data

serve as the inspector 66

serve as the inspiration 1390

serve as the installation 136

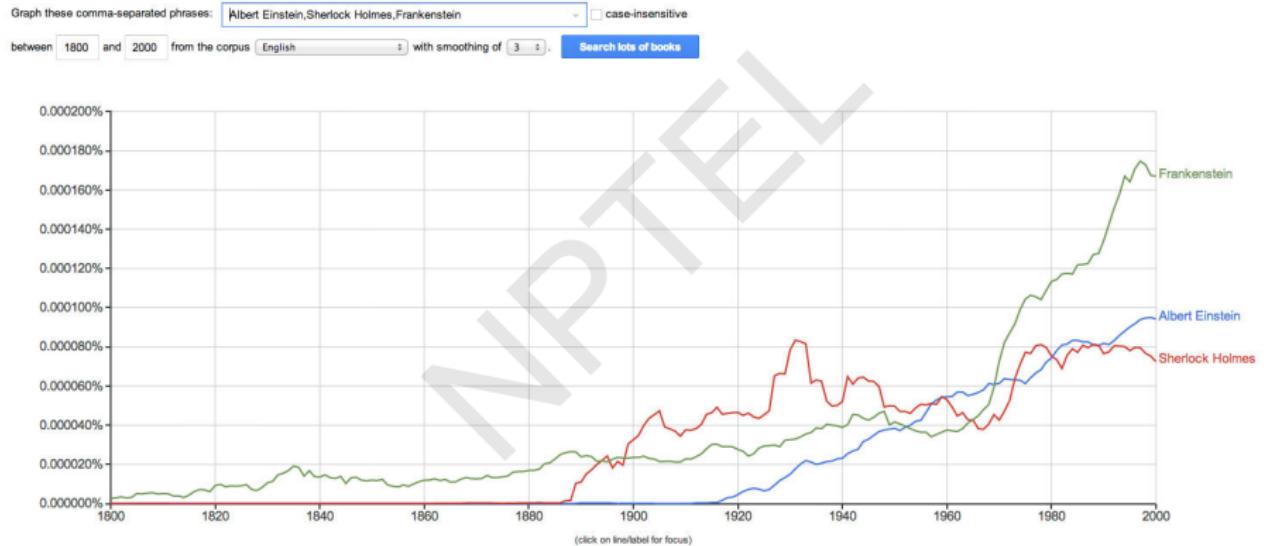
serve as the institute 187

serve as the institution 279

serve as the institutional 461

Google books Ngram Data

Google books Ngram Viewer



Evaluation of Language Models, Basic Smoothing

Pawan Goyal

CSE, IITKGP

Week 2: Lecture 5

Evaluating Language Model

Does it prefer good sentences to bad sentences?

Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Evaluating Language Model

Does it prefer good sentences to bad sentences?

Assign higher probability to real (or frequently observed) sentences than ungrammatical (or rarely observed) ones

Training and Test Corpora

- Parameters of the model are trained on a large corpus of text, called **training set**.
- Performance is tested on a disjoint (held-out) **test data** using an **evaluation metric**

Extrinsic evaluation of N-grams models

Comparison of two models, A and B

- Use each model for one or more tasks: *spelling corrector, speech recognizer, machine translation*
- Get accuracy values for A and B
- Compare accuracy for A and B

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Unigram model doesn't work for this game.

Intrinsic evaluation: Perplexity

Intuition: The Shannon Game

How well can we predict the next word?

- I always order pizza with cheese and ...
- The president of India is ...
- I wrote a ...

Unigram model doesn't work for this game.

A better model of text

is one which assigns a higher probability to the actual word

Perplexity

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

Perplexity ($PP(W)$)

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

Perplexity

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

Perplexity ($PP(W)$)

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

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

Perplexity

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

Perplexity ($PP(W)$)

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

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

Applying chain Rule

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_1 \dots w_{i-1})} \right)^{\frac{1}{N}}$$

Perplexity

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

Perplexity ($PP(W)$)

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

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

Applying chain Rule

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_1 \dots w_{i-1})} \right)^{\frac{1}{N}}$$

For bigrams

$$PP(W) = \left(\prod \frac{1}{P(w_i | w_{i-1})} \right)^{\frac{1}{N}}$$

Example: A Simple Scenario

- Consider a sentence consisting of N random digits

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

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

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

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

Example: A Simple Scenario

- Consider a sentence consisting of N random digits
- Find the perplexity of this sentence as per a model that assigns a probability $p = 1/10$ to each digit.

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

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Lower perplexity = better model

WSJ Corpus

Training: 38 million words

Test: 1.5 million words

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Unigram perplexity: 962?

The model is as confused on test data as if it had to choose uniformly and independently among 962 possibilities for each word.

The Shannon Visualization Method

Use the language model to generate word sequences

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram
 $(\langle s \rangle, w)$ as per its probability

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram
 $(\langle s \rangle, w)$ as per its probability
- Choose a random bigram
 (w, x) as per its probability

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram
 $(\langle s \rangle, w)$ as per its probability
- Choose a random bigram
 (w, x) as per its probability
- And so on until we choose
 $\langle /s \rangle$

The Shannon Visualization Method

Use the language model to generate word sequences

- Choose a random bigram ($< s >$, w) as per its probability
- Choose a random bigram (w, x) as per its probability
- And so on until we choose $</ s >$

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

Shakespeare as Corpus

- $N = 884,647$ tokens, $V = 29,066$
 - Shakespeare produced 300,000 bigram types out of $V^2 = 844$ million possible bigrams.

Approximating Shakespeare

Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
Every enter now severally so, let
Hill he late speaks; or! a more to leg less first you enter
Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like

Bigram

What means, sir. I confess she? then all sorts, he is trim, captain.
Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.
This shall forbid it should be branded, if renown made it empty.
Indeed the duke; and had a very good friend.
Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

Quadrigram

King Henry.What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
Will you not tell me who I am?
It cannot be but so.
Indeed the short and the long. Marry, 'tis a noble Lepidus.

Problems with simple MLE estimate: zeros

NPTEL

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Zero probability n-grams

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

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Zero probability n-grams

- $P(\text{offer} \mid \text{denied the}) = 0$
- The test set will be assigned a probability 0

Problems with simple MLE estimate: zeros

Training set

- ... denied the allegations
- ... denied the reports
- ... denied the claims
- ... denied the request

Test Data

- ... denied the offer
- ... denied the loan

Zero probability n-grams

- $P(\text{offer} \mid \text{denied the}) = 0$
- The test set will be assigned a probability 0
- And the perplexity can't be computed

Language Modeling: Smoothing

NPTEL

Language Modeling: Smoothing

With sparse statistics

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

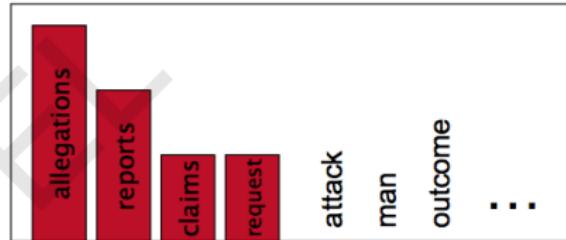
3 allegations

2 reports

1 claims

1 request

7 total



Language Modeling: Smoothing

With sparse statistics

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

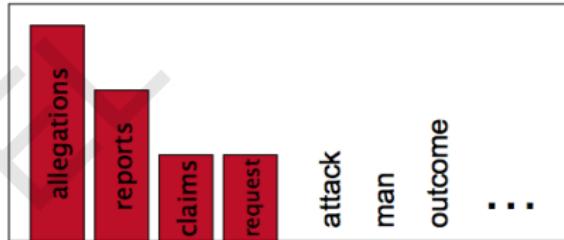
3 allegations

2 reports

1 claims

1 request

7 total



Steal probability mass to generalize better

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

2.5 allegations

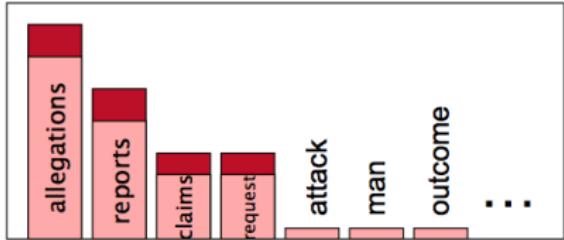
1.5 reports

0.5 claims

0.5 request

2 other

7 total



Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time than we actually did

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time than we actually did
- Just add one to all the counts!

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time than we actually did
- Just add one to all the counts!
- MLE estimate for bigram: $P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$

Laplace Smoothing (Add-one estimation)

- Pretend as if we saw each word (N-gram) one more time than we actually did
- Just add one to all the counts!
- MLE estimate for bigram: $P_{MLE}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$
- Add-1 estimate: $P_{Add-1}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$

Reconstituted counts as effect of smoothing

Effective bigram count ($c^*(w_{n-1}w_n)$)

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

Comparing with bigrams: Restaurant corpus

	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

Comparing with bigrams: Restaurant corpus

	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

More general formulations: Add-k

$$P_{Add-k}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

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

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

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

Unigram prior smoothing:

$$P_{UnigramPrior}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

More general formulations: Add-k

$$P_{Add-k}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + k}{c(w_{i-1}) + kV}$$

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

Unigram prior smoothing:

$$P_{UnigramPrior}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i) + mP(w_i)}{c(w_{i-1}) + m}$$

A good value of k or m ?

Can be optimized on held-out set

Language Modelling: Advanced Smoothing Models

Pawan Goyal

CSE, IITKGP

Week 3: Lecture 1

Advanced smoothing algorithms

Some Examples

- Good-Turing
- Kneser-Ney

Advanced smoothing algorithms

Some Examples

- Good-Turing
- Kneser-Ney

Good-Turing: Basic Intuition

Use the count of things we have seen once

- to help estimate the count of things we have never seen

N_c : Frequency of frequency c

Example Sentences

<s>I am here </s>

<s>who am I </s>

<s>I would like </s>

N_c : Frequency of frequency c

Example Sentences

<s>I am here </s>

<s>who am I </s>

<s>I would like </s>

Computing N_c

I	3	
am	2	
here	1	$N_1 = 4$
who	1	$N_2 = 1$
would	1	$N_3 = 1$
like	1	

Good Turing Estimation

Idea

- Reallocate the probability mass of n -grams that occur $r + 1$ times in the training data to the n -grams that occur r times
- In particular, reallocate the probability mass of n -grams that were seen once to the n -grams that were never seen

Adjusted count

For each count c , an adjusted count c^* is computed as:

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

where N_c is the number of n -grams seen exactly c times

Good Turing Estimation

Good Turing Smoothing

$$P_{GT}^*(\text{things with frequency } c) = \frac{c^*}{N}$$

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

Good Turing Estimation

Good Turing Smoothing

$$P_{GT}^*(\text{things with frequency } c) = \frac{c^*}{N}$$

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

What if $c = 0$

$P_{GT}^*(\text{things with frequency } c) = \frac{N_1}{N}$ where N denotes the total number of bigrams that actually occur in training

Complications

What about words with high frequency?

- For small c , $N_c > N_{c+1}$
- For large c , too jumpy

Complications

What about words with high frequency?

- For small c , $N_c > N_{c+1}$
- For large c , too jumpy

Simple Good-Turing

Replace empirical N_k with a best-fit power law once counts get unreliable

Good-Turing numbers: Example

22 million words of AP Neswire

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

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Good-Turing numbers: Example

22 million words of AP Neswire

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

It looks like $c^* = c - 0.75$

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

Absolute Discounting Interpolation

Why don't we just subtract 0.75 (or some d)?

Absolute Discounting Interpolation

Why don't we just subtract 0.75 (or some d)?

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

Absolute Discounting Interpolation

Why don't we just subtract 0.75 (or some d)?

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

We may keep some more values of d for counts 1 and 2

Absolute Discounting Interpolation

Why don't we just subtract 0.75 (or some d)?

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

We may keep some more values of d for counts 1 and 2
But can we do better than using the regular unigram correct?

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*
- “Francisco” more common than “glasses”

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*
- “Francisco” more common than “glasses”
- *But “Francisco” mostly follows “San”*

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*
- “Francisco” more common than “glasses”
- *But “Francisco” mostly follows “San”*

$P(w)$: “How likely is w ? ”

Instead, $P_{continuation}(w)$: “How likely is w to appear as a novel continuation?”

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*
- “Francisco” more common than “glasses”
- *But “Francisco” mostly follows “San”*

$P(w)$: “How likely is w ? ”

Instead, $P_{continuation}(w)$: “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

Kneser-Ney Smoothing

Intuition

- Shannon game: *I can't see without my reading ...: glasses/Francisco?*
- “Francisco” more common than “glasses”
- *But “Francisco” mostly follows “San”*

$P(w)$: “How likely is w ? ”

Instead, $P_{continuation}(w)$: “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

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

Kneser-Ney Smoothing

How many times does w appear as a novel continuation?

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

Kneser-Ney Smoothing

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

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

Kneser-Ney Smoothing

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

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

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

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

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

Kneler-Ney Smoothing

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

Kneser-Ney Smoothing

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

λ is a normalizing constant

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

Model Combination

As N increases

- The power (expressiveness) of an N-gram model increases

Model Combination

As N increases

- The power (expressiveness) of an N-gram model increases
- But the ability to estimate accurate parameters from sparse data decreases (i.e. the smoothing problem gets worse).

As N increases

- The power (expressiveness) of an N-gram model increases
- But the ability to estimate accurate parameters from sparse data decreases (i.e. the smoothing problem gets worse).

A general approach is to combine the results of multiple N-gram models.

Backoff and Interpolation

It might help to use less context

Backoff and Interpolation

It might help to use less context

when you haven't learned much about larger contexts

Backoff and Interpolation

It might help to use less context

when you haven't learned much about larger contexts

Backoff

- use trigram if you have good evidence
- otherwise bigram, otherwise unigram

Backoff and Interpolation

It might help to use less context

when you haven't learned much about larger contexts

Backoff

- use trigram if you have good evidence
- otherwise bigram, otherwise unigram

Interpolation

mix unigram, bigram, trigram

Backoff

Estimating $P(w_i|w_{i-2}w_{i-1})$

- If we do not have counts to compute $P(w_i|w_{i-2}w_{i-1})$ estimate this using the bigram probability $P(w_i|w_{i-1})$

Backoff

Estimating $P(w_i|w_{i-2}w_{i-1})$

- If we do not have counts to compute $P(w_i|w_{i-2}w_{i-1})$ estimate this using the bigram probability $P(w_i|w_{i-1})$
- If we do not have counts to compute $P(w_i|w_{i-1})$, estimate this using the unigram probability $P(w_i)$

Backoff

Estimating $P(w_i|w_{i-2}w_{i-1})$

- If we do not have counts to compute $P(w_i|w_{i-2}w_{i-1})$ estimate this using the bigram probability $P(w_i|w_{i-1})$
- If we do not have counts to compute $P(w_i|w_{i-1})$, estimate this using the unigram probability $P(w_i)$

$$P_{bo}(w_i|w_{i-2}w_{i-1}) =$$

- $\hat{P}(w_i|w_{i-2}w_{i-1})$, if $c(w_{i-2}w_{i-1}w_i) > 0$
- $\lambda(w_{i-1}w_{i-2})P_{bo}(w_i|w_{i-1})$, otherwise

$$\text{where } P_{bo}(w_i|w_{i-1}) =$$

- $\hat{P}(w_i|w_{i-1})$ if $c(w_{i-1}w_i) > 0$
- $\lambda(w_{n-1})\hat{P}(w_n)$, otherwise

Example Problem

In a corpus, suppose there are 4 words, a , b , c , and d . You are provided with the following counts.

n-gram	count	n-gram	count	n-gram	count
aba	4	ba	5	a	8
abb	0	bb	3	b	9
abc	0	bc	0	c	8
abd	0	bd	0	d	7

Use the recursive definition of backoff smoothing to obtain the probability distribution, $P_{backoff}(w_n | w_{n-2} w_{n-1})$, where $w_{n-1} = b$ and $w_{n-2} = a$. Also assume that $\hat{P}(x) = P(x) - 1/8$.

Linear Interpolation

Simple Interpolation

$$\tilde{P}(w_n | w_{n-1} w_{n-2}) = \lambda_1 P(w_n | w_{n-1} w_{n-2}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

Linear Interpolation

Simple Interpolation

$$\tilde{P}(w_n | w_{n-1} w_{n-2}) = \lambda_1 P(w_n | w_{n-1} w_{n-2}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

Linear Interpolation

Simple Interpolation

$$\tilde{P}(w_n | w_{n-1} w_{n-2}) = \lambda_1 P(w_n | w_{n-1} w_{n-2}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$$

$$\sum_i \lambda_i = 1$$

Lambdas conditional on context

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

Setting the lambda values

Use a held-out corpus

Choose λ s to maximize the probability of held-out data:

- Find the N-gram probabilities on the training data
- Search for λ s that give the largest probability to held-out data

Computational Morphology

Pawan Goyal

CSE, IITKGP

Week 3: Lecture 2

Morphology

Morphology studies the internal structure of words, how words are built up from smaller meaningful units called **morphemes**

NPTEL

Morphology

Morphology studies the internal structure of words, how words are built up from smaller meaningful units called **morphemes**

dogs

- 2 morphemes, 'dog' and 's'
- 's' is a plural marker on nouns

Morphology

Morphology studies the internal structure of words, how words are built up from smaller meaningful units called **morphemes**

dogs

- 2 morphemes, 'dog' and 's'
- 's' is a plural marker on nouns

unladylike

3 morphemes

- un- 'not'
- lady 'well-behaved woman'
- -like 'having the characteristic of'

Allomorphs

Variants of the same morpheme, but cannot be replaced by one another

Example

- opposite: un-happy, in-comprehensible, im-possible, ir-rational

Bound and Free Morphemes

Bound

Cannot appear as a word by itself.

-s (*dog-s*), -ly (*quick-ly*), -ed (*walk-ed*)

Bound and Free Morphemes

Bound

Cannot appear as a word by itself.

-s (*dog-s*), -ly (*quick-ly*), -ed (*walk-ed*)

Free

Can appear as a word by itself; often can combine with other morphemes too.

house (*house-s*), *walk* (*walk-ed*), *of*, *the*, or

Stems and Affixes

Stems and Affixes

- Stems (roots): The core meaning bearing units
- Affixes: Bits and pieces adhering to stems to change their meanings and grammatical functions

Stems and Affixes

Stems and Affixes

- Stems (roots): The core meaning bearing units
- Affixes: Bits and pieces adhering to stems to change their meanings and grammatical functions

Mostly, stems are free morphemes and affixes are bound morphemes

Types of affixes

NPTEL

Types of affixes

- Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
un-happy, pre-existing

Types of affixes

- Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
un-happy, pre-existing
- Suffix: -ity, -ation, etc (-taa, -ke, -ka etc.)
talk-ing, quick-ly

Types of affixes

- Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
un-happy, pre-existing
- Suffix: -ity, -ation, etc (-taa, -ke, -ka etc.)
talk-ing, quick-ly
- Infix: 'n' in '*vindati*' (he knows), as contrasted with *vid* (to know).
Philippines: basa 'read' → b-um-asa 'read'
English:

Types of affixes

- Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
un-happy, pre-existing
- Suffix: -ity, -ation, etc (-taa, -ke, -ka etc.)
talk-ing, quick-ly
- Infix: 'n' in '*vindati*' (he knows), as contrasted with *vid* (to know).
Philippines: basa 'read' → b-um-asa 'read'
English: abso-bloody-lutely (emphasis)

Types of affixes

- Prefix: un-, anti-, etc (a-, ati-, pra- etc.)
un-happy, pre-existing
- Suffix: -ity, -ation, etc (-taa, -ke, -ka etc.)
talk-ing, quick-ly
- Infix: 'n' in '*vindati*' (he knows), as contrasted with *vid* (to know).
Philippines: basa 'read' → b-um-asa 'read'
English: abso-bloody-lutely (emphasis)
- Circumfixes - precedes and follows the stem
Dutch: berg 'mountain', ge-berg-te 'mountains'

Content and functional morphemes

Content morphemes

Carry some semantic content

car, -able, un-

Content and functional morphemes

Content morphemes

Carry some semantic content

car, -able, un-

Functional morphemes

Provide grammatical information

-s (plural), -s (3rd singular)

Inflectional and Derivational Morphology

Two different kind of relationship among words

NPTEL

Inflectional and Derivational Morphology

Two different kind of relationship among words

Inflectional morphology

Grammatical: number, tense, case, gender

Creates new forms of the same word: *bring, brought, brings, bringing*

Inflectional and Derivational Morphology

Two different kind of relationship among words

Inflectional morphology

Grammatical: number, tense, case, gender

Creates new forms of the same word: *bring, brought, brings, bringing*

Derivational morphology

Creates new words by changing part-of-speech: *logic, logical, illogical, illogicality, logician*

Inflectional and Derivational Morphology

Two different kind of relationship among words

Inflectional morphology

Grammatical: number, tense, case, gender

Creates new forms of the same word: *bring, brought, brings, bringing*

Derivational morphology

Creates new words by changing part-of-speech: *logic, logical, illogical, illogicality, logician*

Fairly systematic but some derivations missing: *sincere - sincerity, scarce - scarcity, curious - curiosity, fierce - fiercity?*

Morphological processes

Concatenation

Adding continuous affixes - the most common process:

- hope+less, un+happy, anti+capital+ist+s

Morphological processes

Concatenation

Adding continuous affixes - the most common process:

- hope+less, un+happy, anti+capital+ist+s

Often, there are phonological/graphemic changes on morpheme boundaries:

- book + s [s], shoe + s [z]
- happy +er → happier

Morphological processes

Reduplication: part of the word or the entire word is doubled

Morphological processes

Reduplication: part of the word or the entire word is doubled

- Nama: 'go' (look), 'go-go' (examine with attention)

Morphological processes

Reduplication: part of the word or the entire word is doubled

- Nama: 'go' (look), 'go-go' (examine with attention)
- Tagalog: 'basa' (read), 'ba-basa'(will read)

Morphological processes

Reduplication: part of the word or the entire word is doubled

- Nama: 'go' (look), 'go-go' (examine with attention)
- Tagalog: 'basa' (read), 'ba-basa' (will read)
- Sanskrit: 'pac' (cook), 'papāca' (perfect form, cooked)

Morphological processes

Reduplication: part of the word or the entire word is doubled

- Nama: 'go' (look), 'go-go' (examine with attention)
- Tagalog: 'basa' (read), 'ba-basa' (will read)
- Sanskrit: 'pac' (cook), 'papāca' (perfect form, cooked)
- Phrasal reduplication (Telugu): *pillavādu naḍustū naḍustū paḍi pōyādu*
(The child fell down while walking)

Morphological processes

Suppletion

'irregular' relation between the words
go - went, good - better

Morphological processes

Suppletion

'irregular' relation between the words
go - went, good - better

Morpheme internal changes

The word changes internally
sing - sang - sung, man - men, goose - geese

Word Formation

Compounding

Words formed by combining two or more words

Example in English:

- Adj + Adj → Adj: bitter-sweet
- N + N → N: rain-bow
- V + N → V: pick-pocket
- P + V → V: over-do

Word Formation

Compounding

Words formed by combining two or more words

Example in English:

- Adj + Adj → Adj: bitter-sweet
- N + N → N: rain-bow
- V + N → V: pick-pocket
- P + V → V: over-do

Particular to languages

room-temperature: Hindi translation?

Word Formation

Compounding

Words formed by combining two or more words

Example in English:

- Adj + Adj → Adj: bitter-sweet
- N + N → N: rain-bow
- V + N → V: pick-pocket
- P + V → V: over-do

Particular to languages

room-temperature: Hindi translation?

Word Formation

Acronyms

laser: Light Amplification by Simulated Emission of Radiation

Word Formation

Acronyms

laser: Light Amplification by Simulated Emission of Radiation

Blending

Parts of two different words are combined

- breakfast + lunch → brunch
- smoke + fog → smog
- motor + hotel → motel

Word Formation

Acronyms

laser: Light Amplification by Simulated Emission of Radiation

Blending

Parts of two different words are combined

- breakfast + lunch → brunch
- smoke + fog → smog
- motor + hotel → motel

Clipping

Longer words are shortened

Word Formation

Acronyms

laser: Light Amplification by Simulated Emission of Radiation

Blending

Parts of two different words are combined

- breakfast + lunch → brunch
- smoke + fog → smog
- motor + hotel → motel

Clipping

Longer words are shortened

doctor, laboratory, advertisement, dormitory, examination, bicycle, refrigerator

Processing morphology

- Lemmatization: word → lemma
 $\text{saw} \rightarrow \{\text{see}, \text{saw}\}$

Processing morphology

- Lemmatization: word → lemma
 $\text{saw} \rightarrow \{\text{see}, \text{saw}\}$
- Morphological analysis : word → setOf(lemma +tag)
 $\text{saw} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle, \langle \text{saw}, \text{noun.sg} \rangle \}$

Processing morphology

- Lemmatization: word → lemma
 $\text{saw} \rightarrow \{\text{see}, \text{saw}\}$
- Morphological analysis : word → setOf(lemma +tag)
 $\text{saw} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle, \langle \text{saw}, \text{noun.sg} \rangle \}$
- Tagging: word → tag, considers context
 $\text{Peter saw her} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle \}$

Processing morphology

- Lemmatization: word → lemma
 $\text{saw} \rightarrow \{\text{see}, \text{saw}\}$
- Morphological analysis : word → setOf(lemma +tag)
 $\text{saw} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle, \langle \text{saw}, \text{noun.sg} \rangle \}$
- Tagging: word → tag, considers context
 $\text{Peter saw her} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle \}$
- Morpheme segmentation: de-nation-al-iz-ation

Processing morphology

- Lemmatization: word → lemma
 $\text{saw} \rightarrow \{\text{see}, \text{saw}\}$
- Morphological analysis : word → setOf(lemma +tag)
 $\text{saw} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle, \langle \text{saw}, \text{noun.sg} \rangle \}$
- Tagging: word → tag, considers context
 $\text{Peter saw her} \rightarrow \{ \langle \text{see}, \text{verb.past} \rangle \}$
- Morpheme segmentation: de-nation-al-iz-ation
- Generation: see + verb.past → saw

What are the applications?

- Text-to-speech synthesis:
lead:

What are the applications?

- Text-to-speech synthesis:
lead: verb or noun?
read:

What are the applications?

- Text-to-speech synthesis:
lead: verb or noun?
read: present or past?
 - Search and information retrieval
 - Machine translation, grammar correction

Morphological Analysis

Input	Morphological Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	(goose +N +SG) or (goose +V)
gooses	goose +V +3SG
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Morphological Analysis

Input	Morphological Parsed Output
cats	cat +N +PL
cat	cat +N +SG
cities	city +N +PL
geese	goose +N +PL
goose	(goose +N +SG) or (goose +V)
gooses	goose +V +3SG
merging	merge +V +PRES-PART
caught	(catch +V +PAST-PART) or (catch +V +PAST)

Goal

To take input forms like those in the first column and produce output forms like those in the second column.

Output contains stem and additional information; +N for noun, +SG for singular, +PL for plural, +V for verb etc.

Issues involved

boy → boys

Issues involved

boy → boys

fly → flies → flies (y→ i rule)

Issues involved

boy → boys

fly → flys → flies (y→ i rule)

Toiling → toil

Issues involved

boy → boys

fly → flys → flies (y→ i rule)

Toiling → toil

Duckling → duckl?

Issues involved

boy → boys

fly → flys → flies (y→ i rule)

Toiling → toil

Duckling → duckl?

- Getter → get + er
- Doer → do + er

Issues involved

boy → boys

fly → flys → flies (y→ i rule)

Toiling → toil

Duckling → duckl?

- Getter → get + er
- Doer → do + er
- Beer → be + er?

Knowledge Required

Knowledge of stems or roots

Duck is a possible root, not duckl.

We need a dictionary (lexicon)

Knowledge Required

Knowledge of stems or roots

Duck is a possible root, not *duck!*.

We need a dictionary (lexicon)

Morphotactics

Which class of morphemes follow other classes of morphemes inside the word?

Ex: plural morpheme follows the noun

Knowledge Required

Knowledge of stems or roots

Duck is a possible root, not *duck!*.

We need a dictionary (lexicon)

Morphotactics

Which class of morphemes follow other classes of morphemes inside the word?

Ex: plural morpheme follows the noun

Only some endings go on some words

- *Do+er*: ok
- *Be+er*: not so

Knowledge Required

Knowledge of stems or roots

Duck is a possible root, not *duckl*.

We need a dictionary (lexicon)

Morphotactics

Which class of morphemes follow other classes of morphemes inside the word?

Ex: plural morpheme follows the noun

Only some endings go on some words

- *Do+er*: ok
- *Be+er*: not so

Spelling change rules

Adjust the surface form using spelling change rules

- *Get + er → getter*

Why can't this be put in a big lexicon?

- English: just 317,477 forms from 90,196 lexical entries, a ratio of 3.5:1

Why can't this be put in a big lexicon?

- English: just 317,477 forms from 90,196 lexical entries, a ratio of 3.5:1
- Sanskrit: 11 million forms from a lexicon of 170,000 entries, a ratio of 64.7:1

Why can't this be put in a big lexicon?

- English: just 317,477 forms from 90,196 lexical entries, a ratio of 3.5:1
- Sanskrit: 11 million forms from a lexicon of 170,000 entries, a ratio of 64.7:1
- New forms can be created, compounding etc.

One of the most common methods is finite-state-machines

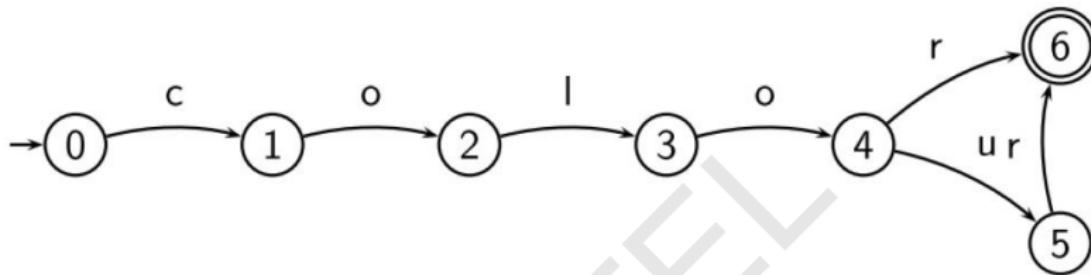
Finite-state methods for morphology

Pawan Goyal

CSE, IITKGP

Week 3: Lecture 3

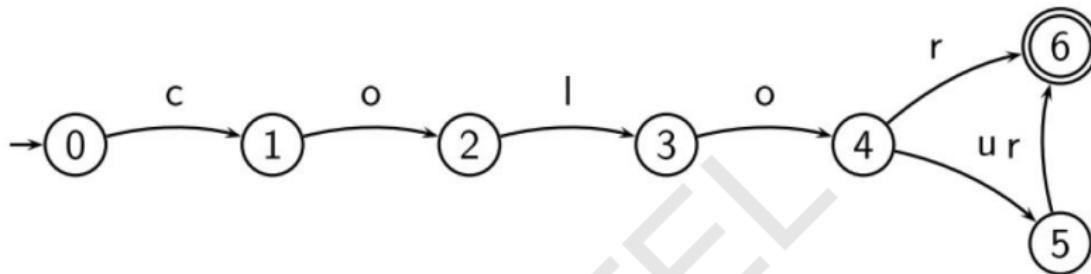
Finite State Automaton (FSA)



What is FSA?

- A kind of directed graph

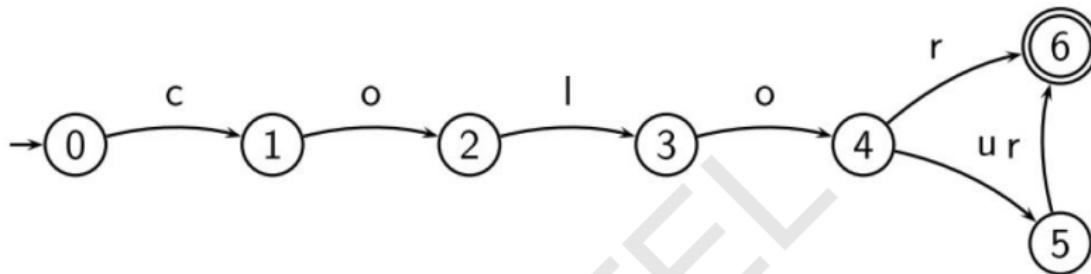
Finite State Automaton (FSA)



What is FSA?

- A kind of directed graph
- Nodes are called states, edges are labeled with symbols (possibly empty ϵ)

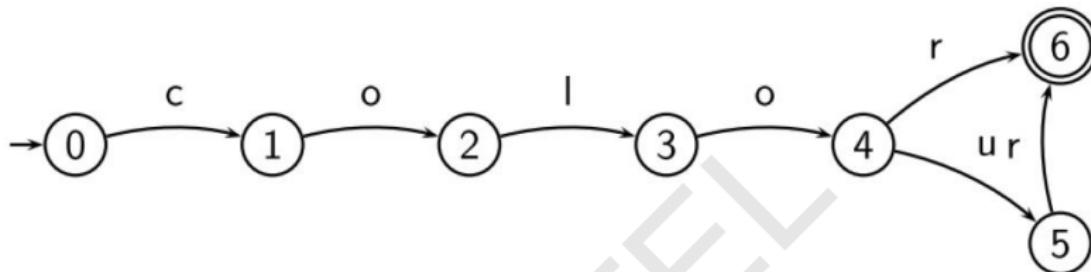
Finite State Automaton (FSA)



What is FSA?

- A kind of directed graph
- Nodes are called states, edges are labeled with symbols (possibly empty ϵ)
- Start state and accepting states

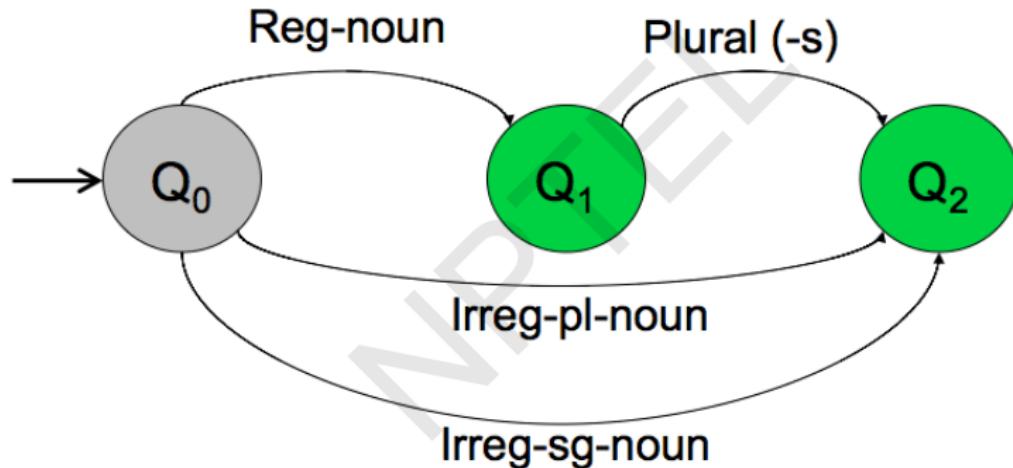
Finite State Automaton (FSA)



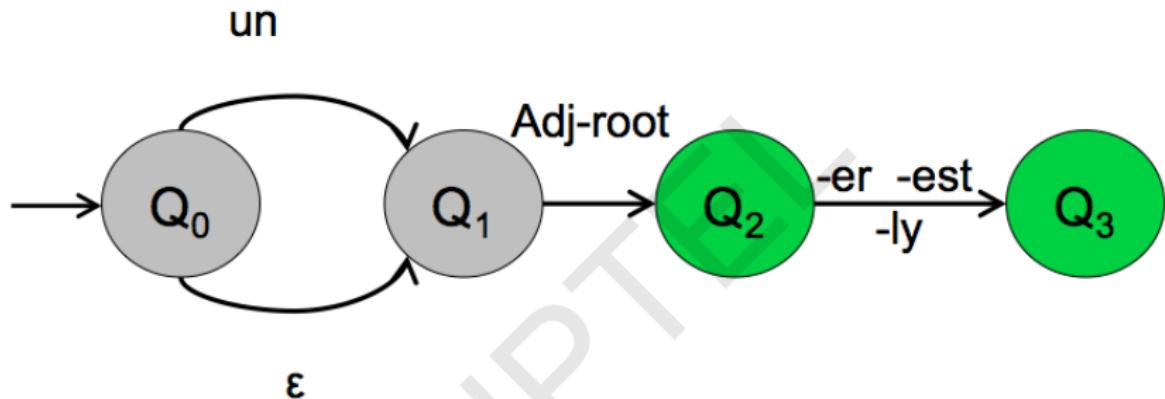
What is FSA?

- A kind of directed graph
- Nodes are called states, edges are labeled with symbols (possibly empty ϵ)
- Start state and accepting states
- Recognizes regular languages, i.e., languages specified by regular expressions

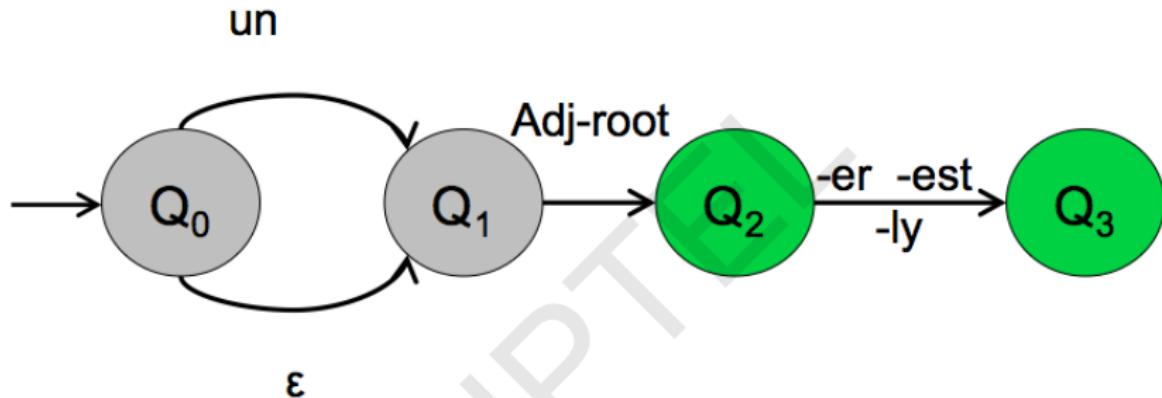
FSA for nominal inflection in English



FSA for English Adjectives



FSA for English Adjectives



Word modeled

happy, happier, happiest, real, unreal, cool, coolly, clear, clearly, unclear,
unclearly, ...

Morphotactics

- The last two examples model some parts of the English morphotactics
- But what about the information about regular and irregular roots?

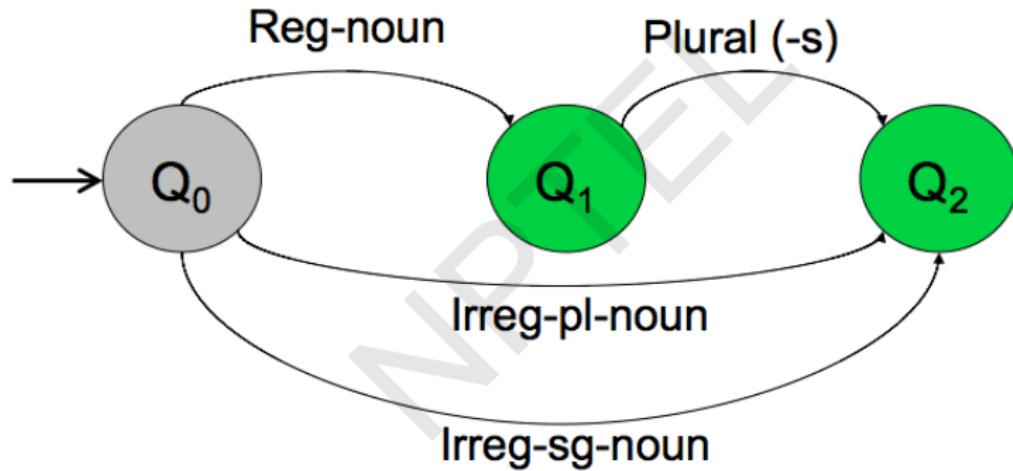
Morphotactics

- The last two examples model some parts of the English morphotactics
- But what about the information about regular and irregular roots?

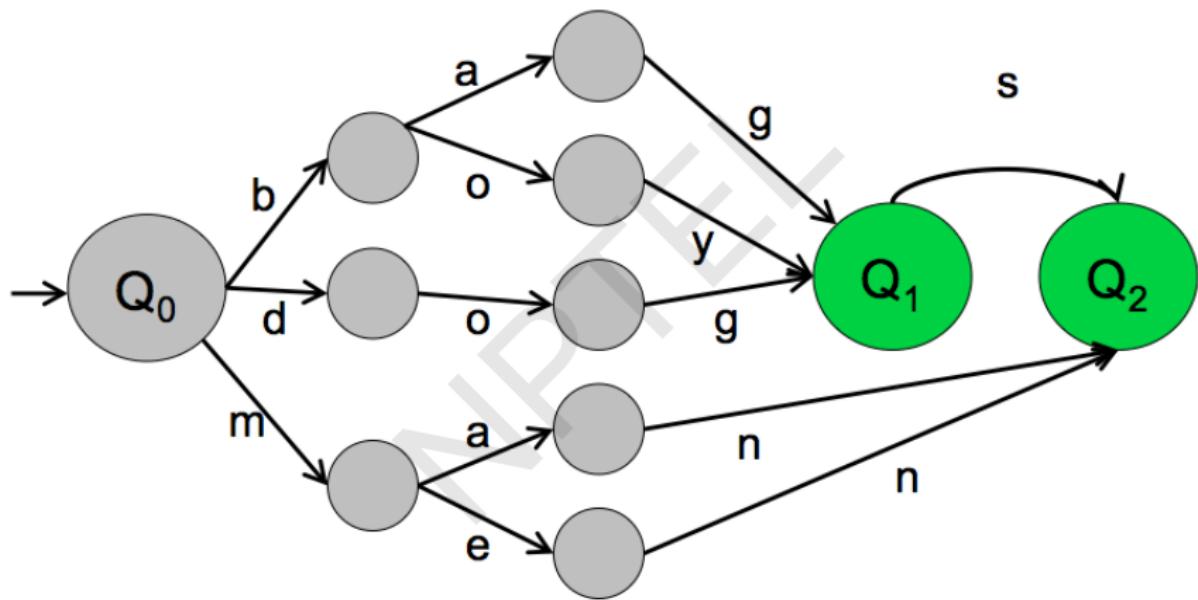
Lexicon

Can we include the lexicon in the FSA?

FSA for nominal inflection in English



After adding a mini-lexicon



Some properties of FSAs: Elegance

- Recognizing problem can be solved in linear time (independent of the size of the automaton)
- There is an algorithm to transform each automaton into a unique equivalent automaton with the least number of states
- An FSA is deterministic iff it has no empty (ϵ) transition and for each state and each symbol, there is at most one applicable transition
- Every non-deterministic automaton can be transformed into a deterministic one

But ...

FSAs are language recognizers/generators.

But ...

FSAs are language recognizers/generators.

We need transducers to build Morphological Analyzers

But ...

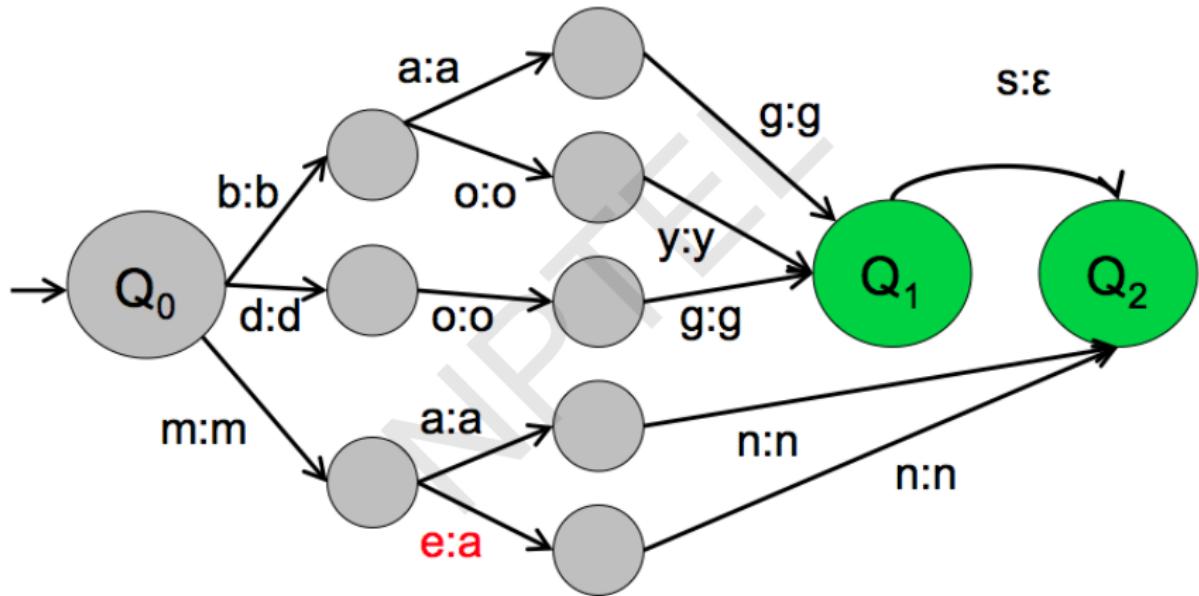
FSA are language recognizers/generators.

We need transducers to build Morphological Analyzers

Finite State Transducers

- Translate strings from one language to strings in another language
- Like FSA, but each edge is associated with two strings

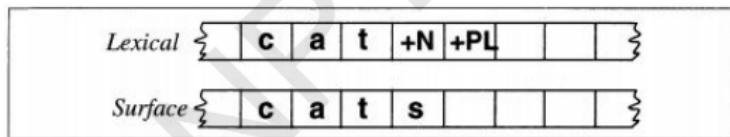
An example FST



Two-level morphology

Given the input *cats*, we would like to output *cat+N+PL*, telling us that cat is a plural noun.

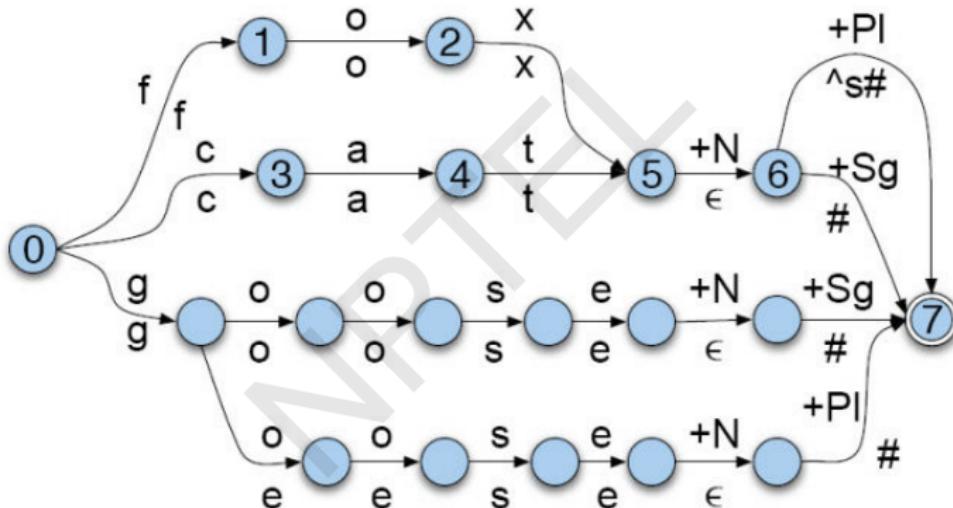
We do this via a version of **two-level morphology**, a correspondence between a lexical level (morphemes and features) to a surface level (actual spelling).



Intermediate tape for Spelling change rules

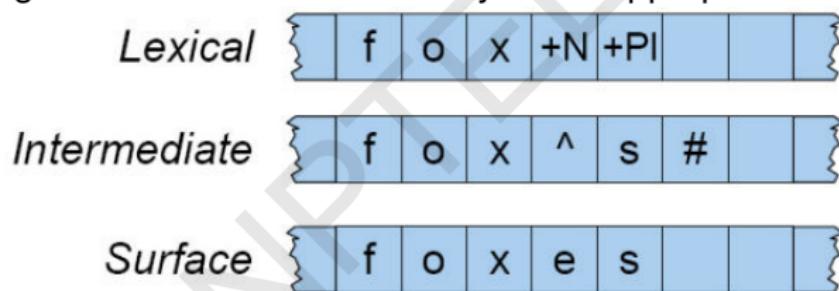


English Nominal Inflection FST



Spelling Handling

A spelling change rule would insert an e only in the appropriate environment.



Rule Handling

Rule Notation

$a \rightarrow b/c_d$: “rewrite a as b when it occurs between c and d .”

Morphological Analysis: Approaches

Two different ways to address phonological/graphemic variations

- Linguistic approach: A phonological component accompanying the simple concatenative process of attaching an ending
- Engineering approach: Phonological changes and irregularities are factored into endings and a higher number of paradigms

Different Approaches: Example from Czech

	woman	owl	draft	iceberg	vapor	fly
S1	žen-a	sov-a	skic-a	kr-a	pár-a	mouch-a
S2	žen-y	sov-y	skic-i	kr-y	pář-y	mouch-y
S3	žen-ě	sov-ě	skic-e	kř-e	pář-e	mouš-e
:						
P2	žen-0	sov-0	skic-0	ker-0	par-0	much-0

A linguistic approach

$$\begin{array}{llllll} \text{žen} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{sov} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{skic} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{kr} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{pár} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{mouch} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} \end{array}$$

An engineering approach

$$\begin{array}{llllll} \text{žen} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{sov} + \begin{cases} \text{a} \\ \text{y} \\ \check{\text{e}} \\ 0 \end{cases} & \text{skic} + \begin{cases} \text{a} \\ \text{i} \\ \text{e} \\ 0 \end{cases} & \text{k} + \begin{cases} \text{ra} \\ \text{ry} \\ \check{\text{e}}\text{r} \\ \text{er} \end{cases} & \text{p} + \begin{cases} \text{ára} \\ \text{áry} \\ \check{\text{e}}\text{r} \\ \text{ar} \end{cases} & \text{m} + \begin{cases} \text{oucha} \\ \text{ouchy} \\ \text{ouše} \\ \text{uch} \end{cases} \end{array}$$

Tools Available

- AT&T FSM Library and Lextools

<http://www2.research.att.com/~fsmtools/fsm/>

- OpenFST (Google and NYU)

<http://www.openfst.org/>

Introduction to POS Tagging

Pawan Goyal

CSE, IITKGP

Week 3: Lecture 4

Part-of-Speech (POS) tagging

NPTEL

Part-of-Speech (POS) tagging

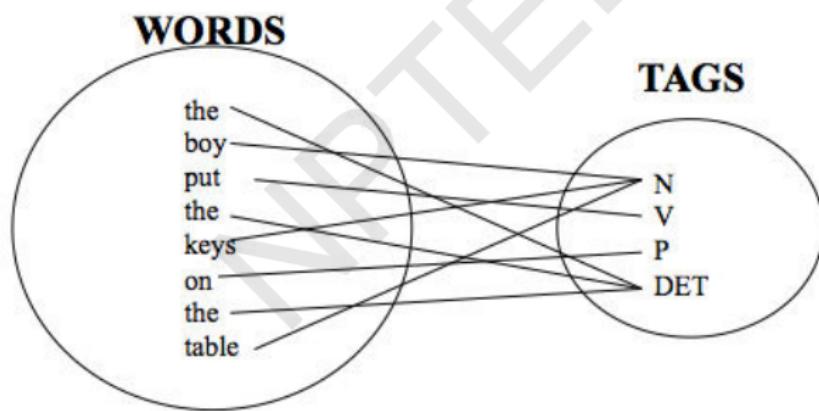
Task

Given a text of English, identify the parts of speech of each word

Part-of-Speech (POS) tagging

Task

Given a text of English, identify the parts of speech of each word



Parts of Speech: How many?

Open class words (content words)

- nouns, verbs, adjectives, adverbs
- mostly content-bearing: they refer to objects, actions, and features in the world
- *open class*, since new words are added all the time

Parts of Speech: How many?

Open class words (content words)

- nouns, verbs, adjectives, adverbs
- mostly content-bearing: they refer to objects, actions, and features in the world
- *open class*, since new words are added all the time

Closed class words

- pronouns, determiners, prepositions, connectives, ...
- there is a limited number of these
- *mostly functional*: to tie the concepts of a sentence together

POS examples

- N noun chair, bandwidth, pacing
- V verb study, debate, munch
- ADJ adj purple, tall, ridiculous
- ADV adverb unfortunately, slowly,
- P preposition of, by, to
- PRO pronoun I, me, mine
- DET determiner the, a, that, those

POS tagging: Choosing a tagset

- To do POS tagging, a standard set needs to be chosen

POS tagging: Choosing a tagset

- To do POS tagging, a standard set needs to be chosen
- Could pick very coarse tagsets
N, V, Adj, Adv

POS tagging: Choosing a tagset

- To do POS tagging, a standard set needs to be chosen
- Could pick very coarse tagsets
 N, V, Adj, Adv
- More commonly used set is finer grained, “UPenn TreeBank tagset”, 45 tags

POS tagging: Choosing a tagset

- To do POS tagging, a standard set needs to be chosen
- Could pick very coarse tagsets
 N , V , Adj , Adv
- More commonly used set is finer grained, “UPenn TreeBank tagset”, 45 tags

A Nice Tutorial on POS tags

<https://sites.google.com/site/partofspeechhelp/>

UPenn TreeBank POS tag set

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	<i>and, but, or</i>	SYM	Symbol	+%, &
CD	Cardinal number	<i>one, two, three</i>	TO	"to"	<i>to</i>
DT	Determiner	<i>a, the</i>	UH	Interjection	<i>ah, oops</i>
EX	Existential 'there'	<i>there</i>	VB	Verb, base form	<i>eat</i>
FW	Foreign word	<i>mea culpa</i>	VBD	Verb, past tense	<i>ate</i>
IN	Preposition/sub-conj	<i>of, in, by</i>	VBG	Verb, gerund	<i>eating</i>
JJ	Adjective	<i>yellow</i>	VBN	Verb, past participle	<i>eaten</i>
JJR	Adj., comparative	<i>bigger</i>	VBP	Verb, non-3sg pres	<i>eat</i>
JJS	Adj., superlative	<i>wildest</i>	VBZ	Verb, 3sg pres	<i>eats</i>
LS	List item marker	<i>1, 2, One</i>	WDT	Wh-determiner	<i>which, that</i>
MD	Modal	<i>can, should</i>	WP	Wh-pronoun	<i>what, who</i>
NN	Noun, sing. or mass	<i>llama</i>	WP\$	Possessive wh-	<i>whose</i>
NNS	Noun, plural	<i>llamas</i>	WRB	Wh-adverb	<i>how, where</i>
NNP	Proper noun, singular	<i>IBM</i>	\$	Dollar sign	\$
NNPS	Proper noun, plural	<i>Carolinas</i>	#	Pound sign	#
PDT	Predeterminer	<i>all, both</i>	"	Left quote	(“ or “)
POS	Possessive ending	<i>'s</i>	"	Right quote	(‘ or ’)
PRP	Personal pronoun	<i>I, you, he</i>	(Left parenthesis	([, (, {, <)
PRP\$	Possessive pronoun	<i>your, one's</i>)	Right parenthesis	(],), }, >)
RB	Adverb	<i>quickly, never</i>	,	Comma	,
RBR	Adverb, comparative	<i>faster</i>	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	<i>fastest</i>	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	<i>up, off</i>			

Using the UPenn tagset

Example Sentence

The grand jury commented on a number of other topics.

Using the UPenn tagset

Example Sentence

The grand jury commented on a number of other topics.

POS tagged sentence

The/DT grand/JJ jury/NN commmented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

Why is POS tagging hard?

NPTEL

Why is POS tagging hard?

Words often have more than one POS: back

- The back door:

Why is POS tagging hard?

Words often have more than one POS: back

- The back door: *back/JJ*
- On my back:

Why is POS tagging hard?

Words often have more than one POS: back

- The back door: *back/JJ*
- On my back: *back/NN*
- Win the voters back:

Why is POS tagging hard?

Words often have more than one POS: back

- The back door: *back/JJ*
- On my back: *back/NN*
- Win the voters back: *back/RB*
- Promised to back the bill:

Why is POS tagging hard?

Words often have more than one POS: back

- The back door: *back/JJ*
- On my back: *back/NN*
- Win the voters back: *back/RB*
- Promised to back the bill: *back/VB*

Why is POS tagging hard?

Words often have more than one POS: back

- The back door: *back/JJ*
- On my back: *back/NN*
- Win the voters back: *back/RB*
- Promised to back the bill: *back/VB*

POS tagging problem

To determine the POS tag for a particular instance of a word

Ambiguous word types in the Brown Corpus

Ambiguity in the Brown corpus

- 40% of word tokens are ambiguous
- 12% of word types are ambiguous

Ambiguous word types in the Brown Corpus

Ambiguity in the Brown corpus

- 40% of word tokens are ambiguous
- 12% of word types are ambiguous
- Breakdown of ambiguous word types:

Unambiguous (1 tag)	35,340
Ambiguous (2–7 tags)	4,100
2 tags	3,760
3 tags	264
4 tags	61
5 tags	12
6 tags	2
7 tags	1 ("still")

How bad is the ambiguity problem?

- One tag is usually more likely than the others.

How bad is the ambiguity problem?

- One tag is usually more likely than the others.

In the Brown corpus, *race* is a noun 98% of the time, and a verb 2% of the time

How bad is the ambiguity problem?

- One tag is usually more likely than the others.
In the Brown corpus, *race* is a noun 98% of the time, and a verb 2% of the time
- A tagger for English that simply chooses the most likely tag for each word can achieve good performance

How bad is the ambiguity problem?

- One tag is usually more likely than the others.
In the Brown corpus, *race* is a noun 98% of the time, and a verb 2% of the time
- A tagger for English that simply chooses the most likely tag for each word can achieve good performance
- Any new approach should be compared against the unigram baseline (assigning each token to its most likely tag)

Deciding the correct POS

Can be difficult even for people

- Mrs./NNP Shaefer/NNP never/RB got/VBD around/_ to/TO joining/VBG.
- All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/_ the/DT corner/NN.
- Chateau/NNP Petrus/NNP costs/VBZ around/_ 2500/CD.

Deciding the correct POS

Can be difficult even for people

- Mrs./NNP Shaefer/NNP never/RB got/VBD around/RP to/TO joining/VBG.
- All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB around/IN the/DT corner/NN.
- Chateau/NNP Petrus/NNP costs/VBZ around/RB 2500/CD.

Relevant knowledge for POS tagging

The word itself

- Some words may only be nouns, e.g. *arrow*
- Some words are ambiguous, e.g. *like, flies*
- Probabilities may help, if one tag is more likely than another

Relevant knowledge for POS tagging

The word itself

- Some words may only be nouns, e.g. *arrow*
- Some words are ambiguous, e.g. *like, flies*
- Probabilities may help, if one tag is more likely than another

Local context

- Two determiners rarely follow each other
- Two base form verbs rarely follow each other
- Determiner is almost always followed by adjective or noun

POS tagging: Two approaches

Rule-based Approach

- Assign each word in the input a list of potential POS tags
- Then winnow down this list to a single tag using hand-written rules

POS tagging: Two approaches

Rule-based Approach

- Assign each word in the input a list of potential POS tags
- Then winnow down this list to a single tag using hand-written rules

Statistical tagging

- Get a training corpus of tagged text, learn the transformation rules from the most frequent tags (TBL tagger)
- Probabilistic: Find the most likely sequence of tags T for a sequence of words W

TBL Tagger

Label the training set with most frequent tags

- The can was rusted.

TBL Tagger

Label the training set with most frequent tags

- The can was rusted.
- The/DT can/MD was/VBD rusted/VBD.

TBL Tagger

Label the training set with most frequent tags

- The can was rusted.
- The/DT can/MD was/VBD rusted/VBD.

Add transformation rules to reduce training mistakes

- MD →NN: DT_
- VBD→VBN: VBD_

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Different instances of d and c

- **Part-of-Speech Tagging:**

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Different instances of d and c

- **Part-of-Speech Tagging:** words are observed and tags are hidden.
- **Text Classification:**

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Different instances of d and c

- **Part-of-Speech Tagging:** words are observed and tags are hidden.
- **Text Classification:** sentences/documents are observed and the category is hidden.

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Different instances of d and c

- **Part-of-Speech Tagging:** words are observed and tags are hidden.
- **Text Classification:** sentences/documents are observed and the category is hidden.
Categories can be positive/negative for sentiments ..
sports/politics/business for documents ...

Probabilistic Tagging: Two different families of models

Problem at hand

We have some data $\{(d, c)\}$ of paired observations d and hidden classes c .

Different instances of d and c

- **Part-of-Speech Tagging:** words are observed and tags are hidden.
- **Text Classification:** sentences/documents are observed and the category is hidden.
Categories can be positive/negative for sentiments ..
sports/politics/business for documents ...

What gives rise to the two families?

Whether they generate the observed data from hidden stuff or the hidden structure given the data?

Generative vs. Conditional Models

Generative (Joint) Models

Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: $P(d, c)$ in terms of $P(d|c)$

Generative vs. Conditional Models

Generative (Joint) Models

Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: $P(d, c)$ in terms of $P(d|c)$
e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.

Generative vs. Conditional Models

Generative (Joint) Models

Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: $P(d, c)$ in terms of $P(d|c)$
e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.

Discriminative (Conditional) Models

Take the data as given, and put a probability over hidden structure given the data: $P(c|d)$

Generative vs. Conditional Models

Generative (Joint) Models

Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: $P(d, c)$ in terms of $P(d|c)$
e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.

Discriminative (Conditional) Models

Take the data as given, and put a probability over hidden structure given the data: $P(c|d)$
e.g. Logistic regression, maximum entropy models, conditional random fields

Generative vs. Conditional Models

Generative (Joint) Models

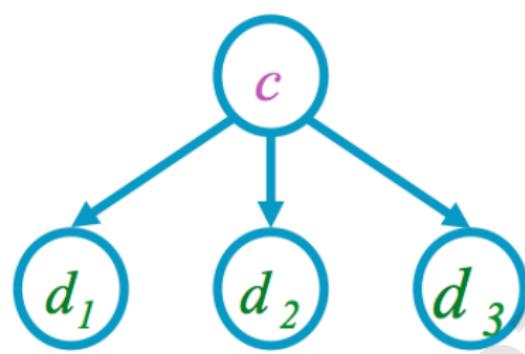
Generate the observed data from hidden stuff, i.e. put a probability over the observations given the class: $P(d, c)$ in terms of $P(d|c)$
e.g. Naïve Bayes' classifiers, Hidden Markov Models etc.

Discriminative (Conditional) Models

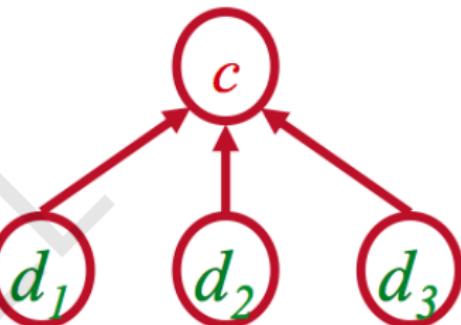
Take the data as given, and put a probability over hidden structure given the data: $P(c|d)$
e.g. Logistic regression, maximum entropy models, conditional random fields

SVMs, perceptron, etc. are discriminative classifiers but not directly probabilistic

Generative vs. Discriminative Models

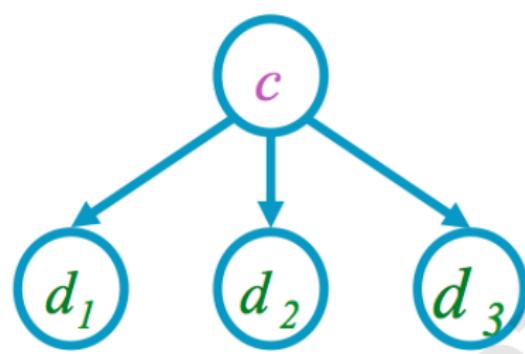


Naive Bayes

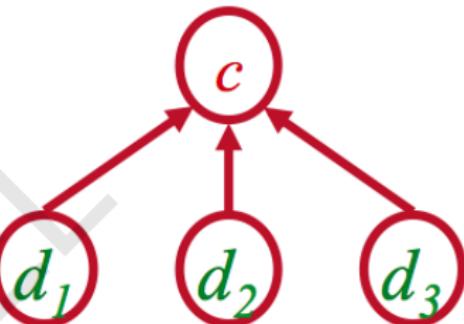


Logistic Regression

Generative vs. Discriminative Models



Naive Bayes



Logistic Regression

Joint vs. conditional likelihood

- A *joint* model gives probabilities $P(d, c)$ and tries to maximize this joint likelihood.
- A *conditional* model gives probabilities $P(c|d)$, taking the data as given and modeling only the conditional probability of the class.

Hidden Markov Models for POS Tagging

Pawan Goyal

CSE, IITKGP

Week 3: Lecture 5

Probabilistic Tagging

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Probabilistic Tagging

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tagging: Probabilistic View (Generative Model)

Find

$$\hat{T} = \operatorname{argmax}_T P(T|W)$$

Probabilistic Tagging

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tagging: Probabilistic View (Generative Model)

Find

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \frac{P(W|T)P(T)}{P(W)}\end{aligned}$$

Probabilistic Tagging

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tagging: Probabilistic View (Generative Model)

Find

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \frac{P(W|T)P(T)}{P(W)} \\ &= \operatorname{argmax}_T P(W|T)P(T)\end{aligned}$$

Probabilistic Tagging

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tagging: Probabilistic View (Generative Model)

Find

$$\begin{aligned}\hat{T} &= \operatorname{argmax}_T P(T|W) \\ &= \operatorname{argmax}_T \frac{P(W|T)P(T)}{P(W)} \\ &= \operatorname{argmax}_T P(W|T)P(T) \\ &= \operatorname{argmax}_T \prod_i P(w_i|w_1 \dots w_{i-1}, t_1 \dots t_i)P(t_i|t_1 \dots t_{i-1})\end{aligned}$$

Further simplifications

$$\hat{T} = \operatorname{argmax}_T \prod_i P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) P(t_i | t_1 \dots t_{i-1})$$

- The probability of a word appearing depends only on its own POS tag

Further simplifications

$$\hat{T} = \operatorname{argmax}_T \prod_i P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) P(t_i | t_1 \dots t_{i-1})$$

- The probability of a word appearing depends only on its own POS tag
 $P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) \approx P(w_i | t_i)$
- Bigram assumption: the probability of a tag appearing depends only on the previous tag

Further simplifications

$$\hat{T} = \operatorname{argmax}_T \prod_i P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) P(t_i | t_1 \dots t_{i-1})$$

- The probability of a word appearing depends only on its own POS tag
 $P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) \approx P(w_i | t_i)$
- Bigram assumption: the probability of a tag appearing depends only on the previous tag
 $P(t_i | t_1 \dots t_{i-1}) \approx P(t_i | t_{i-1})$

Further simplifications

$$\hat{T} = \operatorname{argmax}_T \prod_i P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) P(t_i | t_1 \dots t_{i-1})$$

- The probability of a word appearing depends only on its own POS tag
 $P(w_i | w_1 \dots w_{i-1}, t_1 \dots t_i) \approx P(w_i | t_i)$
- Bigram assumption: the probability of a tag appearing depends only on the previous tag
 $P(t_i | t_1 \dots t_{i-1}) \approx P(t_i | t_{i-1})$
- Using these simplifications:

$$\hat{T} = \operatorname{argmax}_T \prod_i P(w_i | t_i) P(t_i | t_{i-1})$$

Computing the probability values

Tag Transition probabilities $p(t_i|t_{i-1})$

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = 0.49$$

Computing the probability values

Tag Transition probabilities $p(t_i|t_{i-1})$

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

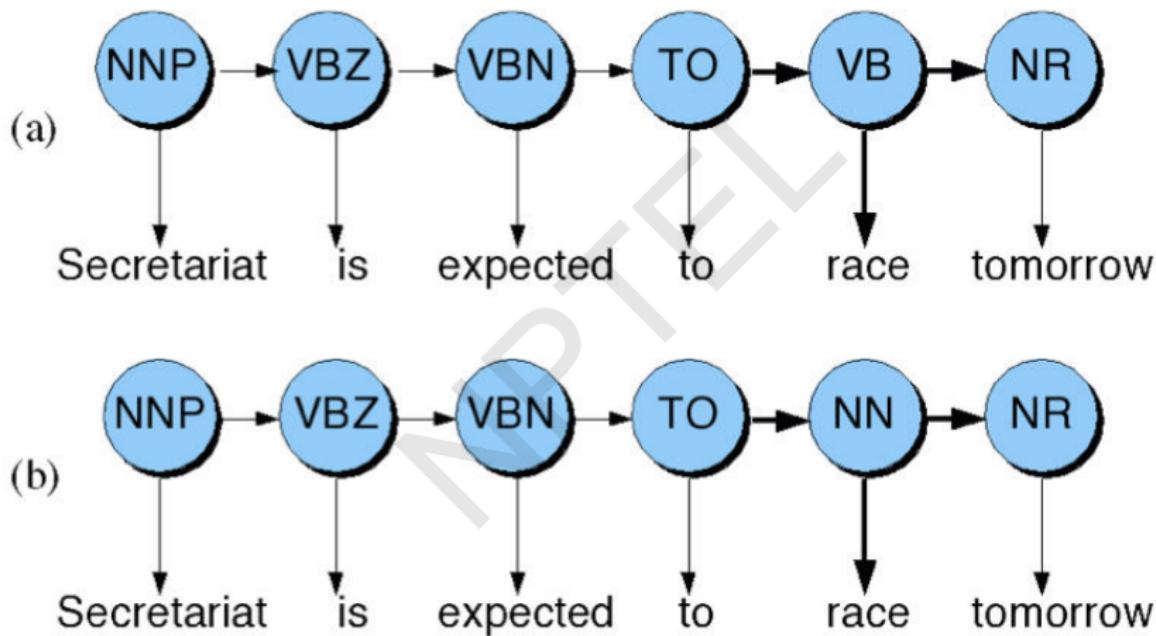
$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = 0.49$$

Word Likelihood probabilities $p(w_i|t_i)$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = 0.47$$

Disambiguating “race”



Disambiguating “race”

Difference in probability due to

- $P(VB|TO)$ vs. $P(NN|TO)$
- $P(race|VB)$ vs. $P(race|NN)$
- $P(NR|VB)$ vs. $P(NR|NN)$

Disambiguating “race”

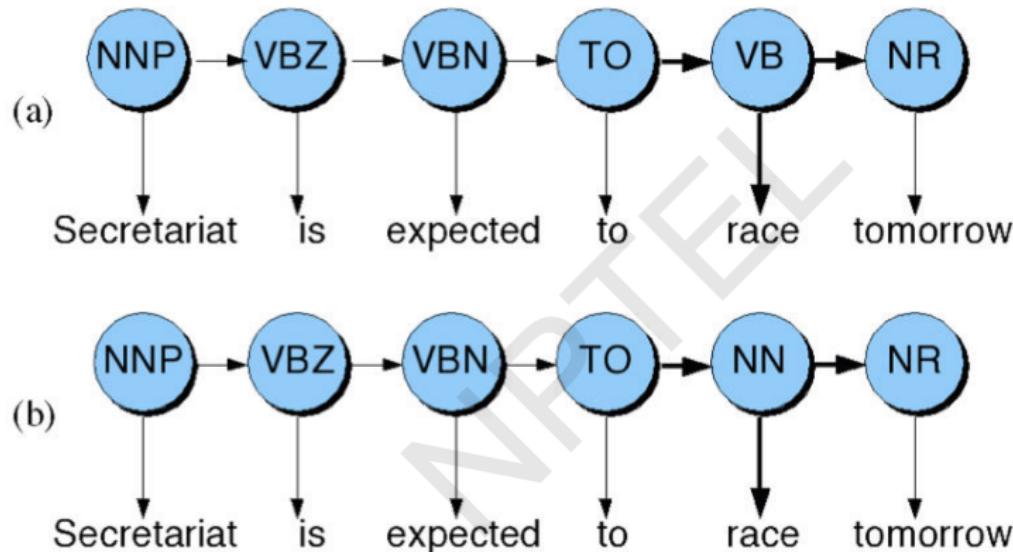
Difference in probability due to

- $P(VB|TO)$ vs. $P(NN|TO)$
- $P(race|VB)$ vs. $P(race|NN)$
- $P(NR|VB)$ vs. $P(NR|NN)$

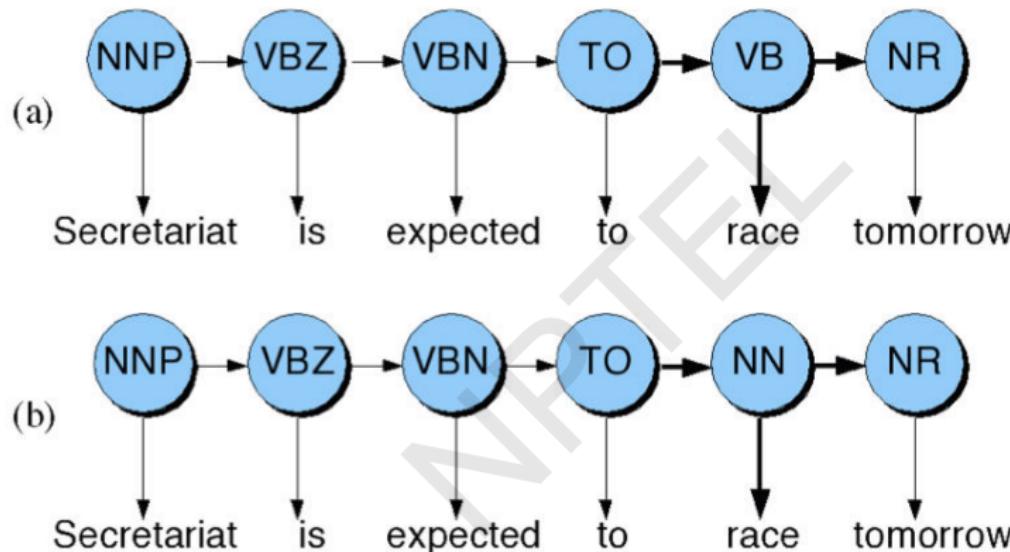
After computing the probabilities

- $P(NN|TO)P(NR|NN)P(race|NN) = 0.0047 \times 0.0012 \times 0.00057 = 0.00000000032$
- $P(VB|TO)P(NR|VB)P(race|VB) = 0.83 \times 0.0027 \times 0.00012 = 0.00000027$

What is this model?



What is this model?



This is a Hidden Markov Model

Hidden Markov Models

- Tag Transition probabilities $p(t_i|t_{i-1})$
- Word Likelihood probabilities (emissions) $p(w_i|t_i)$

Hidden Markov Models

- Tag Transition probabilities $p(t_i|t_{i-1})$
- Word Likelihood probabilities (emissions) $p(w_i|t_i)$
- What we have described with these probabilities is a hidden markov model.
- Let us quickly introduce the Markov Chain, or observable Markov Model.

Markov Chain = First-order Markov Model

Weather example

- Three types of weather: *sunny, rainy, foggy*

Markov Chain = First-order Markov Model

Weather example

- Three types of weather: *sunny, rainy, foggy*
- q_n : variable denoting the weather on the n^{th} day

Markov Chain = First-order Markov Model

Weather example

- Three types of weather: *sunny, rainy, foggy*
- q_n : variable denoting the weather on the n^{th} day
- We want to find the following conditional probabilities:

$$P(q_n|q_{n-1}, q_{n-2}, \dots, q_1)$$

Markov Chain = First-order Markov Model

Weather example

- Three types of weather: *sunny, rainy, foggy*
- q_n : variable denoting the weather on the n^{th} day
- We want to find the following conditional probabilities:

$$P(q_n|q_{n-1}, q_{n-2}, \dots, q_1)$$

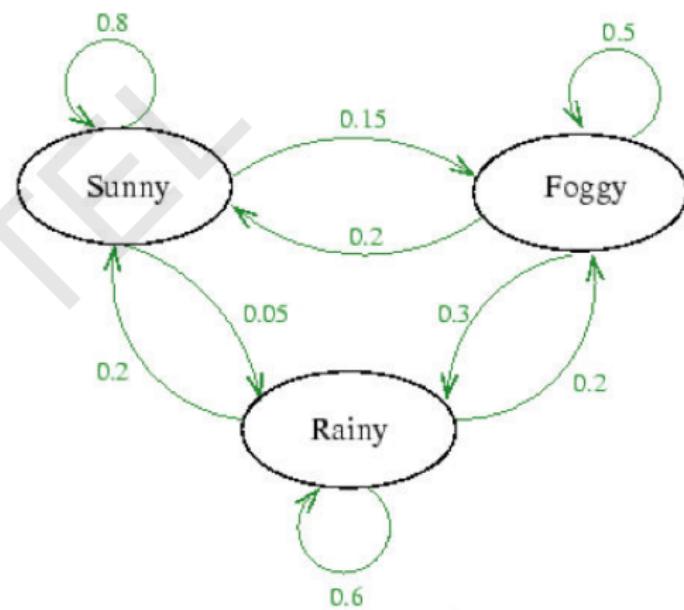
First-order Markov Assumption

$$P(q_n|q_{n-1}, q_{n-2}, \dots, q_1) = P(q_n|q_{n-1})$$

Markov Chain Transition Table

Table 1: Probabilities $p(q_{n+1}|q_n)$ of tomorrow's weather based on today's weather

		Tomorrow's weather		
		Sunny	Rainy	Foggy
Today's weather				
Sunny		0.8	0.05	0.15
Rainy		0.2	0.6	0.2
Foggy		0.2	0.3	0.5



Using Markov Chain

- Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

Using Markov Chain

- Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

$$P(q_2 = \text{sunny}, q_3 = \text{rainy} | q_1 = \text{sunny})$$

Using Markov Chain

- Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

$$P(q_2 = \text{sunny}, q_3 = \text{rainy} | q_1 = \text{sunny})$$

$$= P(q_3 = \text{rainy} | q_2 = \text{sunny}, q_1 = \text{sunny}) \times P(q_2 = \text{sunny} | q_1 = \text{sunny})$$

Using Markov Chain

- Given that today the weather is sunny, what is the probability that tomorrow is sunny and day after is rainy?

$$P(q_2 = \text{sunny}, q_3 = \text{rainy} | q_1 = \text{sunny})$$

$$\begin{aligned}&= P(q_3 = \text{rainy} | q_2 = \text{sunny}, q_1 = \text{sunny}) \times P(q_2 = \text{sunny} | q_1 = \text{sunny}) \\&= P(q_3 = \text{rainy} | q_2 = \text{sunny}) \times P(q_2 = \text{sunny} | q_1 = \text{sunny}) \\&= 0.05 \times 0.8 \\&= 0.04\end{aligned}$$

Hidden Markov Model

- For Markov chains, the output symbols are the same as the states
'sunny' weather is both observable and state

Hidden Markov Model

- For Markov chains, the output symbols are the same as the states
'sunny' weather is both observable and state
- But in POS tagging

Hidden Markov Model

- For Markov chains, the output symbols are the same as the states
'sunny' weather is both observable and state
- But in POS tagging
The output symbols are words

Hidden Markov Model

- For Markov chains, the output symbols are the same as the states
'sunny' weather is both observable and state
- But in POS tagging
The output symbols are words
But the hidden states are POS tags
- A Hidden Markov Model is an extension of a Markov chain in which the output symbols are not the same as the states
- We don't know which state we are in

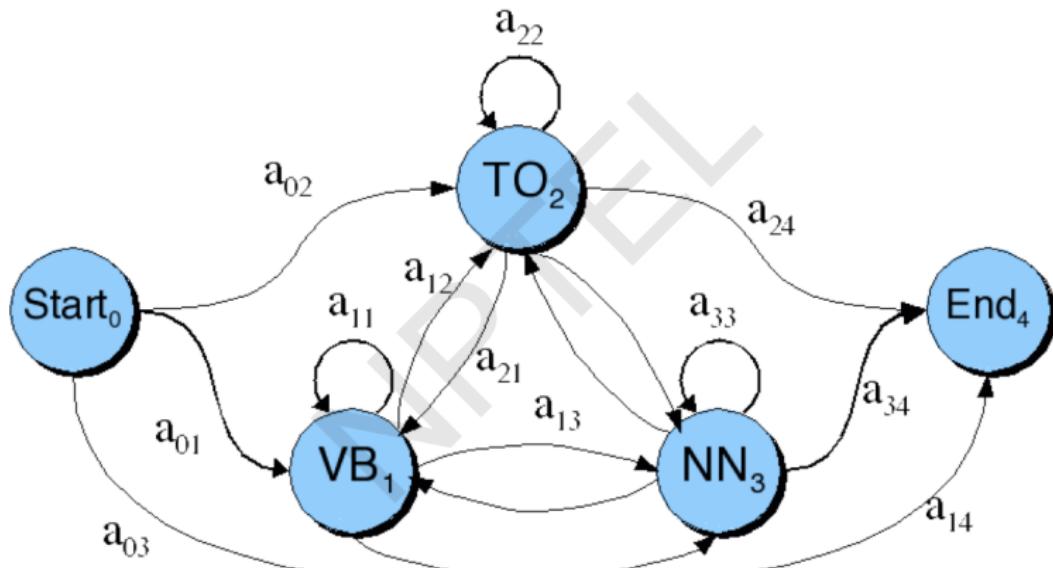
Hidden Markov Models (HMMs)

Elements of an HMM model

- A set of states (here: the tags)
- An output alphabet (here: words)
- Initial state (here: beginning of sentence)
- State transition probabilities (here $p(t_n|t_{n-1})$)
- Symbol emission probabilities (here $p(w_i|t_i)$)

Graphical Representation

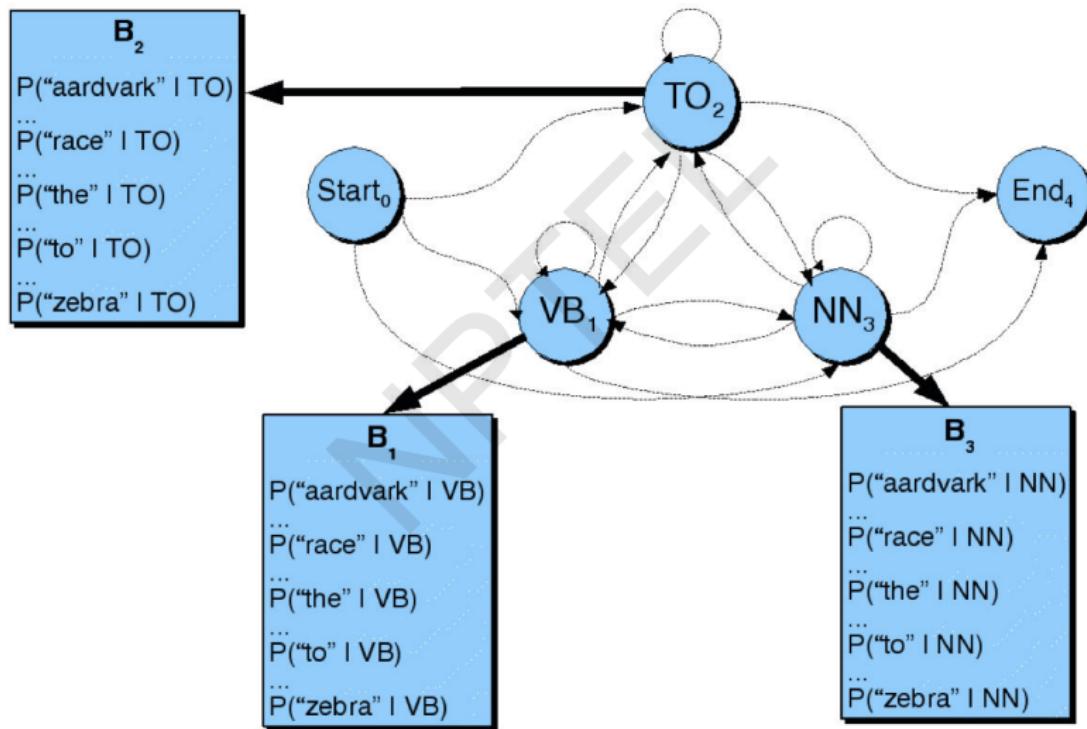
When tagging a sentence, we are walking through the state graph:



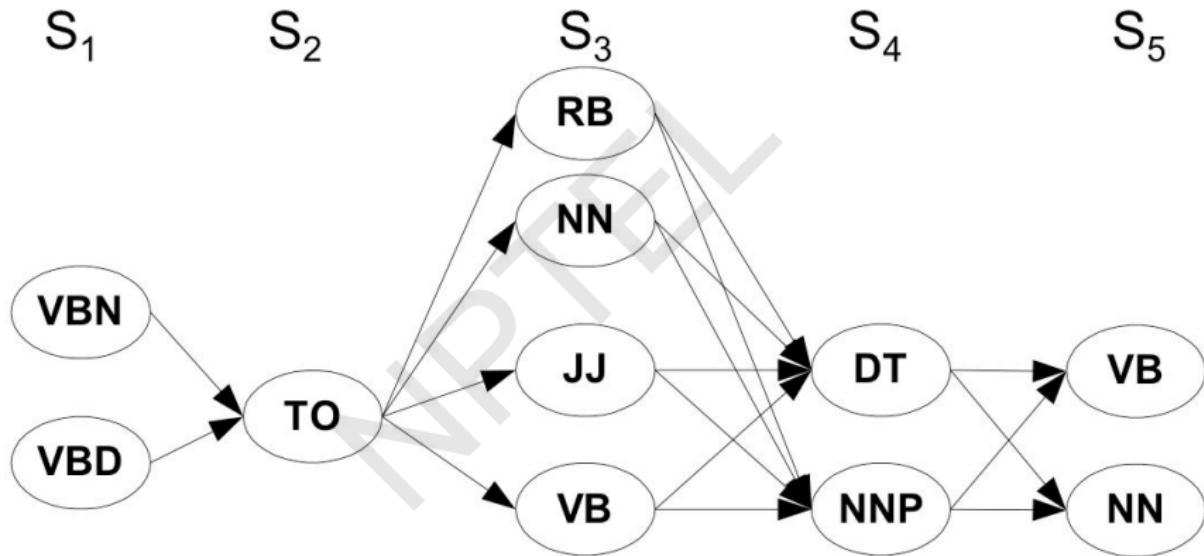
Edges are labeled with the state transition probabilities: $p(t_n|t_{n-1})$

Graphical Representation

At each state we emit a word: $P(w_n | t_n)$

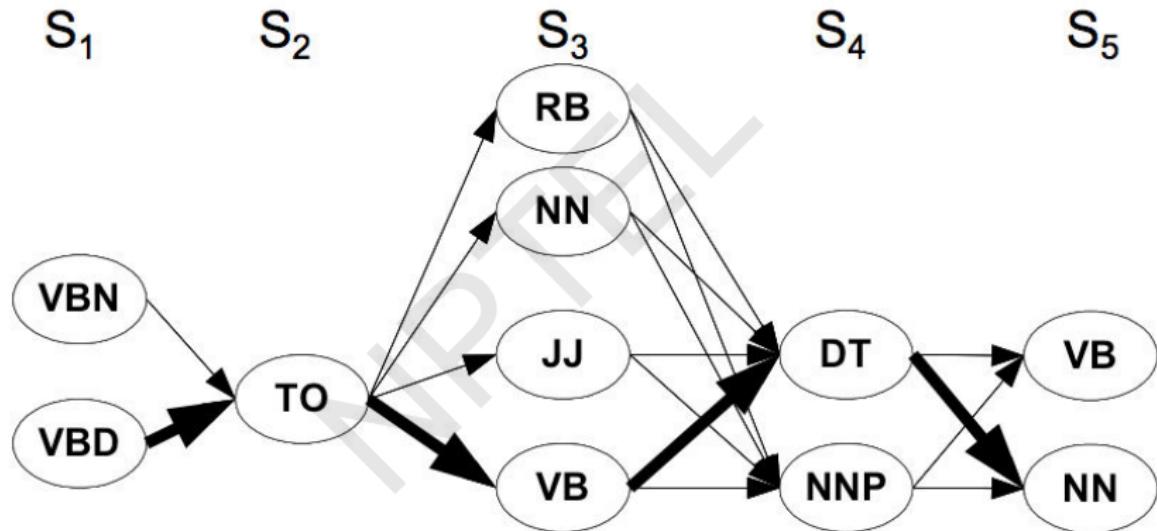


Walking through the states: best path



promised to back the bill

Walking through the states: best path



promised to back the bill

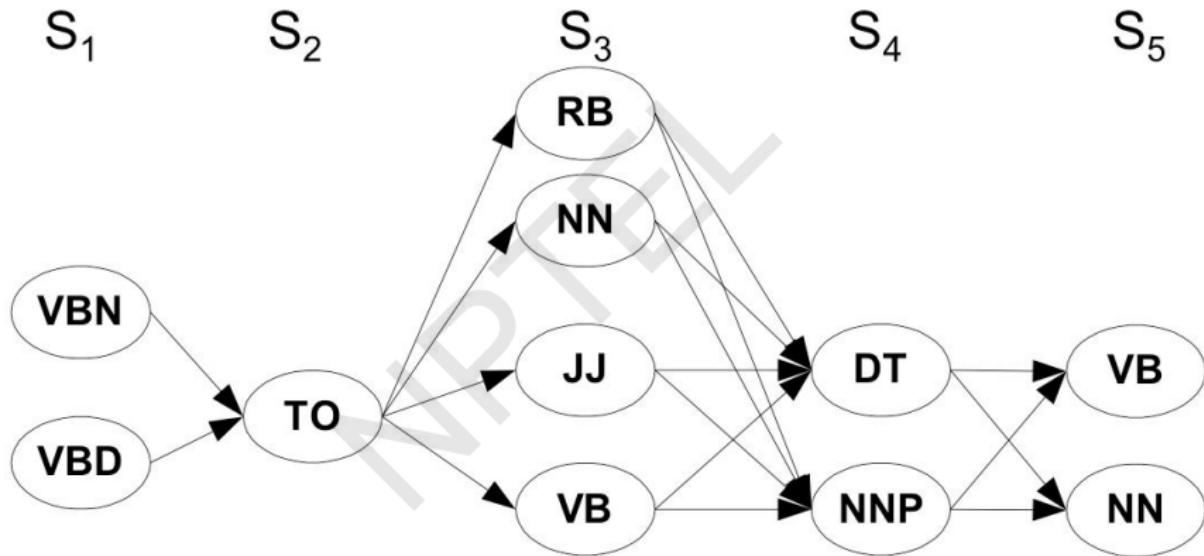
Viterbi Decoding for HMM, Parameter Learning

Pawan Goyal

CSE, IIT Kharagpur

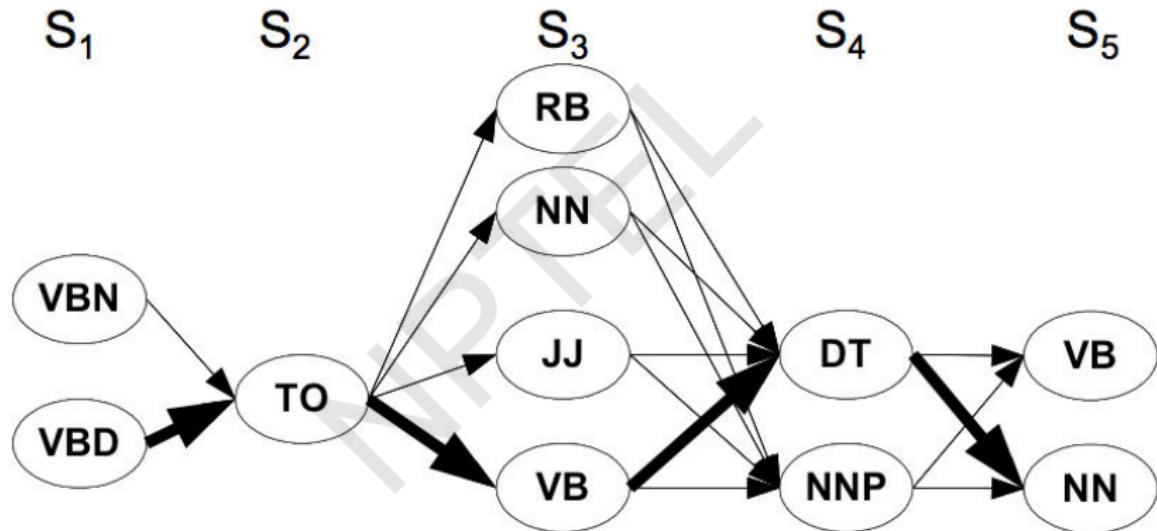
Week 4, Lecture 1

Walking through the states: best path



promised to back the bill

Walking through the states: best path



promised to back the bill

Finding the best path: Viterbi Algorithm

Intuition

Optimal path for each state can be recorded. We need

- Cheapest cost to state j at step s : $\delta_j(s)$
- Backtrace from that state to best predecessor $\psi_j(s)$

Finding the best path: Viterbi Algorithm

Intuition

Optimal path for each state can be recorded. We need

- Cheapest cost to state j at step s : $\delta_j(s)$
- Backtrace from that state to best predecessor $\psi_j(s)$

Computing these values

- $\delta_j(s+1) = \max_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_{s+1}|t_j)$

Finding the best path: Viterbi Algorithm

Intuition

Optimal path for each state can be recorded. We need

- Cheapest cost to state j at step s : $\delta_j(s)$
- Backtrace from that state to best predecessor $\psi_j(s)$

Computing these values

- $\delta_j(s+1) = \max_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_{s+1}|t_j)$
- $\psi_j(s+1) = \operatorname{argmax}_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_{s+1}|t_j)$

Finding the best path: Viterbi Algorithm

Intuition

Optimal path for each state can be recorded. We need

- Cheapest cost to state j at step s : $\delta_j(s)$
- Backtrace from that state to best predecessor $\psi_j(s)$

Computing these values

- $\delta_j(s+1) = \max_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_{s+1}|t_j)$
- $\psi_j(s+1) = \operatorname{argmax}_{1 \leq i \leq N} \delta_i(s) p(t_j|t_i) p(w_{s+1}|t_j)$

Best final state is $\operatorname{argmax}_{1 \leq i \leq N} \delta_i(|S|)$, we can backtrack from there

Practice Question

- Suppose you want to use a HMM tagger to tag the phrase, “the light book”, where we have the following probabilities:
- $P(\text{the}|\text{Det}) = 0.3$, $P(\text{the}|\text{Noun}) = 0.1$, $P(\text{light}|\text{Noun}) = 0.003$, $P(\text{light}|\text{Adj}) = 0.002$, $P(\text{light}|\text{Verb}) = 0.06$, $P(\text{book}|\text{Noun}) = 0.003$, $P(\text{book}|\text{Verb}) = 0.01$
- $P(\text{Verb}|\text{Det}) = 0.00001$, $P(\text{Noun}|\text{Det}) = 0.5$, $P(\text{Adj}|\text{Det}) = 0.3$,
 $P(\text{Noun}|\text{Noun}) = 0.2$, $P(\text{Adj}|\text{Noun}) = 0.002$, $P(\text{Noun}|\text{Adj}) = 0.2$,
 $P(\text{Noun}|\text{Verb}) = 0.3$, $P(\text{Verb}|\text{Noun}) = 0.3$, $P(\text{Verb}|\text{Adj}) = 0.001$,
 $P(\text{Verb}|\text{Verb}) = 0.1$
- Work out in details the steps of the Viterbi algorithm. You can use a Table to show the steps. Assume all other conditional probabilities, not mentioned to be zero. Also, assume that all tags have the same probabilities to appear in the beginning of a sentence.

Learning the Parameters

Two Scenarios

- A labeled dataset is available, with the POS category of individual words in a corpus
- Only the corpus is available, but not labeled with the POS categories

Learning the Parameters

Two Scenarios

- A labeled dataset is available, with the POS category of individual words in a corpus
- Only the corpus is available, but not labeled with the POS categories

Methods for these scenarios

- For the first scenario, parameters can be directly estimated using maximum likelihood estimate from the labeled dataset
- For the second scenario, *Baum-Welch Algorithm* is used to estimate the parameters of the hidden markov model.

Baum Welch Algorithm

Pawan Goyal

CSE, IIT Kharagpur

Week 4, Lecture 2

Baum Welch Algorithm

Uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden markov model

NPTEL

Baum Welch Algorithm

Uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden markov model

Parameters of HMM

Let X_t be the random variable denoting hidden state at time t , and Y_t be the observation variable at time T . HMM parameters are given by $\theta = (A, B, \pi)$ where

- $A = \{a_{ij}\} = P(X_t = j | X_{t-1} = i)$ is the state transition matrix
- $\pi = \{\pi_i\} = P(X_1 = i)$ is the initial state distribution
- $B = \{b_j(y_t)\} = P(Y_t = y_t | X_t = j)$ is the emission matrix

Baum Welch Algorithm

Uses the well-known EM algorithm to find the maximum likelihood estimate of the parameters of a hidden markov model

Parameters of HMM

Let X_t be the random variable denoting hidden state at time t , and Y_t be the observation variable at time T . HMM parameters are given by $\theta = (A, B, \pi)$ where

- $A = \{a_{ij}\} = P(X_t = j | X_{t-1} = i)$ is the state transition matrix
- $\pi = \{\pi_i\} = P(X_1 = i)$ is the initial state distribution
- $B = \{b_j(y_t)\} = P(Y_t = y_t | X_t = j)$ is the emission matrix

Given observation sequences $Y = (Y_1 = y_1, Y_2 = y_2, \dots, Y_T = y_T)$, the algorithm tries to find the parameters θ that maximise the probability of the observation.

The Algorithm

The basic idea is to start with some random initial conditions on the parameters θ , estimate best values of state paths X_t using these, then re-estimate the parameters θ using the just-computed values of X_t , iteratively.

The Algorithm

The basic idea is to start with some random initial conditions on the parameters θ , estimate best values of state paths X_t using these, then re-estimate the parameters θ using the just-computed values of X_t , iteratively.

Intuition

- Choose some initial values for $\theta = (A, B, \pi)$.
- *Repeat the following step until convergence:*
- Determine probable (state) paths ... $X_{t-1} = i, X_t = j \dots$
- Count the expected number of transitions a_{ij} as well as the expected number of times, various emissions $b_j(y_t)$ are made
- Re-estimate $\theta = (A, B, \pi)$ using a_{ij} and $b_j(y_t)$ s.

A forward-backward algorithm is used for finding probable paths.

Forward-Backward Algorithm

Forward Procedure

$\alpha_i(t) = P(Y_1 = y_1, \dots, Y_t = y_t, X_t = i | \theta)$ be the probability of seeing y_1, \dots, y_t and being in state i at time t . Found recursively using:

- $\alpha_i(1) = \pi_i b_i(y_1)$

Forward-Backward Algorithm

Forward Procedure

$\alpha_i(t) = P(Y_1 = y_1, \dots, Y_t = y_t, X_t = i | \theta)$ be the probability of seeing y_1, \dots, y_t and being in state i at time t . Found recursively using:

- $\alpha_i(1) = \pi_i b_i(y_1)$
- $\alpha_j(t+1) = b_j(y_{t+1}) \sum_{i=1}^N \alpha_i(t) a_{ij}$

Forward-Backward Algorithm

Forward Procedure

$\alpha_i(t) = P(Y_1 = y_1, \dots, Y_t = y_t, X_t = i | \theta)$ be the probability of seeing y_1, \dots, y_t and being in state i at time t . Found recursively using:

- $\alpha_i(1) = \pi_i b_i(y_1)$
- $\alpha_j(t+1) = b_j(y_{t+1}) \sum_{i=1}^N \alpha_i(t) a_{ij}$

Backward Procedure

$\beta_i(t) = P(Y_{t+1} = y_{t+1}, \dots, Y_T = y_T | X_t = i, \theta)$ be the probability of ending partial sequence y_{t+1}, \dots, y_T given starting state i at time t . $\beta_i(t)$ is computed recursively as:

- $\beta_i(T) = 1$

Forward-Backward Algorithm

Forward Procedure

$\alpha_i(t) = P(Y_1 = y_1, \dots, Y_t = y_t, X_t = i | \theta)$ be the probability of seeing y_1, \dots, y_t and being in state i at time t . Found recursively using:

- $\alpha_i(1) = \pi_i b_i(y_1)$
- $\alpha_j(t+1) = b_j(y_{t+1}) \sum_{i=1}^N \alpha_i(t) a_{ij}$

Backward Procedure

$\beta_i(t) = P(Y_{t+1} = y_{t+1}, \dots, Y_T = y_T | X_t = i, \theta)$ be the probability of ending partial sequence y_{t+1}, \dots, y_T given starting state i at time t . $\beta_i(t)$ is computed recursively as:

- $\beta_i(T) = 1$
- $\beta_i(t) = \sum_{j=1}^N \beta_j(t+1) a_{ij} b_j(y_{t+1})$

Finding probabilities of paths

We compute the following variables:

- Probability of being in state i at time t given the observation Y and parameters θ

$$\gamma_i(t) = P(X_t = i|Y, \theta) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^N \alpha_j(t)\beta_j(t)}$$

- Probability of being in state i and j at time t and $t+1$ respectively given the observation Y and parameters θ

$$\zeta_{ij}(t) = P(X_t = i, X_{t+1} = j|Y, \theta) = \frac{\alpha_i(t)a_{ij}\beta_j(t+1)b_j(y_{t+1})}{\sum_{i=1}^N \sum_{j=1}^N \alpha_i(t)a_{ij}\beta_j(t+1)b_j(y_{t+1})}$$

Updating the parameters

- $\pi_i = \gamma_i(1)$, expected number of times state i was seen at time 1
- $a_{ij} = \frac{\sum_{t=1}^T \zeta_{ij}(t)}{\sum_{t=1}^T \gamma_i(t)}$, expected number of transitions from state i to state j , compared to the total number of transitions away from state i
- $b_i(v_k) = \frac{\sum_{t=1}^T 1_{y_t=v_k} \gamma_i(t)}{\sum_{t=1}^T \gamma_i(t)}$ with $1_{y_t=v_k}$ being an indicator function, is the expected number of times the output observations are v_k while being in state i compared to the expected total number of times in state i .

Maximum Entropy Models

Pawan Goyal

CSE, IIT Kharagpur

Week 4, Lecture 3

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Possible solutions:

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Possible solutions:

- Use morphological cues (capitalization, suffix) to assign a more calculated guess

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Possible solutions:

- Use morphological cues (capitalization, suffix) to assign a more calculated guess

Limited Context

- “is clearly **marked**” → verb, past participle
- “he clearly **marked**” → verb, past tense

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Possible solutions:

- Use morphological cues (capitalization, suffix) to assign a more calculated guess

Limited Context

- “is clearly **marked**” → verb, past participle
- “he clearly **marked**” → verb, past tense

Possible solution:

Issues with Markov Model Tagging

Unknown Words

We do not have the required probabilities.

Possible solutions:

- Use morphological cues (capitalization, suffix) to assign a more calculated guess

Limited Context

- “is clearly **marked**” → verb, past participle
- “he clearly **marked**” → verb, past tense

Possible solution: Use higher order model, combine various n-gram models to avoid sparseness problem

Maximum Entropy Modeling: Discriminative Model

NPTEL

Maximum Entropy Modeling: Discriminative Model

- We may identify a heterogeneous set of features which contribute in some way to the choice of POS tag of the current word.

Maximum Entropy Modeling: Discriminative Model

- We may identify a heterogeneous set of features which contribute in some way to the choice of POS tag of the current word.
 - ▶ Whether it is the first word in the article

Maximum Entropy Modeling: Discriminative Model

- We may identify a heterogeneous set of features which contribute in some way to the choice of POS tag of the current word.
 - ▶ Whether it is the first word in the article
 - ▶ Whether the next word is *to*

Maximum Entropy Modeling: Discriminative Model

- We may identify a heterogeneous set of features which contribute in some way to the choice of POS tag of the current word.
 - ▶ Whether it is the first word in the article
 - ▶ Whether the next word is *to*
 - ▶ Whether one of the last 5 words is a preposition, etc.
- MaxEnt combines these features in a probabilistic model

Maximum Entropy: The Model

$$p_{\lambda}(y|x) = \frac{1}{Z_{\lambda}(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

NPTEL

Maximum Entropy: The Model

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

where

- $Z_\lambda(x)$ is a normalizing constant given by

$$Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

Maximum Entropy: The Model

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

where

- $Z_\lambda(x)$ is a normalizing constant given by

$$Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x,y)\right)$$

- λ_i is a weight given to a feature f_i

Maximum Entropy: The Model

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

where

- $Z_\lambda(x)$ is a normalizing constant given by

$$Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

- λ_i is a weight given to a feature f_i
- x denotes an observed datum and y denotes a class

What is the form of the features?

Features in Maximum Entropy Models

- Features encode elements of the context x for predicting tag y
- Context x is taken around the word w , for which a tag y is to be predicted

Features in Maximum Entropy Models

- Features encode elements of the context x for predicting tag y
- Context x is taken around the word w , for which a tag y is to be predicted
- Features are binary values functions, e.g.,

$$f(x,y) = \begin{cases} 1 & \text{if } isCapitalized(w) \& y = NNP \\ 0 & \text{otherwise} \end{cases}$$

Example Features

Example: Named Entities

- LOCATION (in Arcadia)
- LOCATION (in Québec)
- DRUG (taking Zantac)
- PERSON (saw Sue)

Example Features

Example: Named Entities

- LOCATION (in Arcadia)
- LOCATION (in Québec)
- DRUG (taking Zantac)
- PERSON (saw Sue)

Example Features

- $f_1(x, y) = [y = \text{LOCATION} \wedge w_{-1} = \text{"in"} \wedge \text{isCapitalized}(w)]$
- $f_2(x, y) = [y = \text{LOCATION} \wedge \text{hasAccentedLatinChar}(w)]$
- $f_3(x, y) = [y = \text{DRUG} \wedge \text{ends}(w, "c")]$

Tagging with Maximum Entropy Model

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tagging with Maximum Entropy Model

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tag sequence candidate $\{t_1, \dots, t_n\}$ has conditional probability:

$$P(t_1, \dots, t_n | w_1, \dots, w_n) = \prod_{i=1}^n p(t_i | x_i)$$

Tagging with Maximum Entropy Model

- $W = w_1 \dots w_n$ - words in the corpus (observed)
- $T = t_1 \dots t_n$ - the corresponding tags (unknown)

Tag sequence candidate $\{t_1, \dots, t_n\}$ has conditional probability:

$$P(t_1, \dots, t_n | w_1, \dots, w_n) = \prod_{i=1}^n p(t_i | x_i)$$

- The context x_i also includes previously assigned tags for a fixed history.
- Beam search is used to find the most probable sequence

Beam Inference

Beam Inference

- At each position, keep the top k complete sequences
- Extend each sequence in each local way
- The extensions compete for the k slots at the next position

Beam Inference

Beam Inference

- At each position, keep the top k complete sequences
- Extend each sequence in each local way
- The extensions compete for the k slots at the next position

But what is a MaxEnt model?

Let's go to the basics now!

Maximum Entropy Model

Intuitive Principle

Model all that is known and assume nothing about that which is unknown.

Maximum Entropy Model

Intuitive Principle

Model all that is known and assume nothing about that which is unknown.

Given a collection of facts, choose a model which is consistent with all the facts, but otherwise as uniform as possible.

Maximum Entropy: Overview

- Suppose we wish to model an expert translator's decisions concerning the proper French rendering of the English word 'in'.

Maximum Entropy: Overview

- Suppose we wish to model an expert translator's decisions concerning the proper French rendering of the English word 'in'.
- Each French word or phrase f is assigned an estimate $p(f)$, probability that the expert would choose f as a translation of 'in'.

Maximum Entropy: Overview

- Suppose we wish to model an expert translator's decisions concerning the proper French rendering of the English word 'in'.
- Each French word or phrase f is assigned an estimate $p(f)$, probability that the expert would choose f as a translation of 'in'.
- Collect a large sample of instances of the expert's decisions

Maximum Entropy: Overview

- Suppose we wish to model an expert translator's decisions concerning the proper French rendering of the English word 'in'.
- Each French word or phrase f is assigned an estimate $p(f)$, probability that the expert would choose f as a translation of 'in'.
- Collect a large sample of instances of the expert's decisions
- **Goal:** extract a set of facts about the decision-making process (first task) that will aid in constructing a model of this process (second task)

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.
- First constraint: $p(\text{dans})+p(\text{en})+p(\text{\'a})+p(\text{au cours de})+p(\text{pendant}) = 1$.

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.
- First constraint: $p(\text{dans})+p(\text{en})+p(\text{\'a})+p(\text{au cours de})+p(\text{pendant}) = 1$.
- Infinite number of models p for which this identity holds, the most intuitive model?

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.
- First constraint: $p(\text{dans})+p(\text{en})+p(\text{\'a})+p(\text{au cours de})+p(\text{pendant}) = 1$.
- Infinite number of models p for which this identity holds, the most intuitive model?
- *allocate the total probability evenly among the five possible phrases* → most uniform model subject to our knowledge.

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.
- First constraint: $p(\text{dans})+p(\text{en})+p(\text{\'a})+p(\text{au cours de})+p(\text{pendant}) = 1$.
- Infinite number of models p for which this identity holds, the most intuitive model?
- *allocate the total probability evenly among the five possible phrases* → most uniform model subject to our knowledge.
- *Is it the most uniform model overall?*

Maximum Entropy Model: Overview

First clue: list of allowed translations

- Suppose the translator always chooses among {dans, en, á, au cours de, pendant}.
- First constraint: $p(\text{dans})+p(\text{en})+p(\text{\'a})+p(\text{au cours de})+p(\text{pendant}) = 1$.
- Infinite number of models p for which this identity holds, the most intuitive model?
- *allocate the total probability evenly among the five possible phrases* → most uniform model subject to our knowledge.
- *Is it the most uniform model overall?* → No, that would grant an equal probability to every possible French phrase.

Maximum Entropy Model: Overview

More clues from the expert's decision

- **Second clue:** Suppose the expert chose either ‘dans’ or ‘en’ 30% of the time.

Maximum Entropy Model: Overview

More clues from the expert's decision

- **Second clue:** Suppose the expert chose either ‘dans’ or ‘en’ 30% of the time.
- **Third clue:** In half of the cases, the expert chose either ‘dans’ or ‘á’

Maximum Entropy Model: Overview

More clues from the expert's decision

- **Second clue:** Suppose the expert chose either ‘dans’ or ‘en’ 30% of the time.
- **Third clue:** In half of the cases, the expert chose either ‘dans’ or ‘a’

How do we measure uniformity of a model?

As we add complexity to the model, we face two difficulties:

- What exactly is meant by “uniform”?
- How can one measure the uniformity of a model?

Maximum Entropy Modeling

Entropy: measures the uncertainty of a distribution.

Quantifying uncertainty (“surprise”)

- Event x
- Probability p_x
- Surprise: $\log(1/p_x)$

Maximum Entropy Modeling

Entropy: measures the uncertainty of a distribution.

Quantifying uncertainty (“surprise”)

- Event x
- Probability p_x
- Surprise: $\log(1/p_x)$

Entropy: expected surprise (over p)

$$H(p) = E_p \left[\log_2 \frac{1}{p_x} \right] = - \sum_x p_x \log_2 p_x$$

Maximum Entropy Modeling

Entropy: measures the uncertainty of a distribution.

Quantifying uncertainty (“surprise”)

- Event x
- Probability p_x
- Surprise: $\log(1/p_x)$

Entropy: expected surprise (over p)

$$H(p) = E_p \left[\log_2 \frac{1}{p_x} \right] = - \sum_x p_x \log_2 p_x$$

Coin Tossing

Maximum Entropy Modeling

Distribution required

- Minimize commitment = maximize entropy
- Resemble some reference distribution

Maximum Entropy Modeling

Distribution required

- Minimize commitment = maximize entropy
- Resemble some reference distribution

Solution

Maximize entropy H , subject to feature-based constraints:

$$E_p[f_i] = E_{\tilde{p}}[f_i]$$

Maximum Entropy Modeling

Distribution required

- Minimize commitment = maximize entropy
- Resemble some reference distribution

Solution

Maximize entropy H , subject to feature-based constraints:

$$E_p[f_i] = E_{\tilde{p}}[f_i]$$

Adding constraints

- Lowers maximum entropy
- Brings the distribution further from uniform and closer to data

Maximum Entropy Principle

Given n feature functions f_i , we would like p to lie in the subset C of P defined by

$$C = \{p \in P | p(f_i) = \tilde{p}(f_i), i \in \{1, 2, \dots, n\}\}$$

Maximum Entropy Principle

Given n feature functions f_i , we would like p to lie in the subset C of P defined by

$$C = \{p \in P | p(f_i) = \tilde{p}(f_i), i \in \{1, 2, \dots, n\}\}$$

Empirical count (expectation) of a feature

$$\tilde{p}(f_i) = \sum_{x,y} \tilde{p}(x,y) f_i(x,y)$$

Maximum Entropy Principle

Given n feature functions f_i , we would like p to lie in the subset C of P defined by

$$C = \{p \in P | p(f_i) = \tilde{p}(f_i), i \in \{1, 2, \dots, n\}\}$$

Empirical count (expectation) of a feature

$$\tilde{p}(f_i) = \sum_{x,y} \tilde{p}(x,y) f_i(x,y)$$

Model expectation of a feature

$$p(f_i) = \sum_{x,y} \tilde{p}(x) p(y|x) f_i(x,y)$$

Select the distribution which is most uniform (conditional probability):

$$p^* = \operatorname{argmax}_{p \in C} H(p) = H(Y|X) \approx - \sum_{x,y} \tilde{p}(x) p(y|x) \log p(y|x)$$

Maximum Entropy Principle

$$p^* = \operatorname{argmax}_{p \in C} H(p)$$

NPTEL

Maximum Entropy Principle

$$p^* = \operatorname{argmax}_{p \in C} H(p)$$

Constraint Optimization

Introduce a parameter λ_i for each feature f_i . Lagrangian is given by

$$\Lambda(p, \lambda) = H(p) + \sum_i \lambda_i (p(f_i) - \tilde{p}(f_i))$$

Solving, we get

$$p_\lambda(y|x) = \frac{1}{Z_\lambda(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

where $Z_\lambda(x)$ is a normalizing constant given by

$$Z_\lambda(x) = \sum_y \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

Maximum Entropy Models

Pawan Goyal

CSE, IIT Kharagpur

Week 4, Lecture 4

Practice Question

Consider the maximum entropy model for POS tagging, where you want to estimate $P(\text{tag}|\text{word})$. In a hypothetical setting, assume that *tag* can take the values *D*, *N* and *V* (short forms for Determiner, Noun and Verb). The variable *word* could be any member of a set V of possible words, where V contains the words *a*, *man*, *sleeps*, as well as additional words. The distribution should give the following probabilities

- $P(D|a) = 0.9$
- $P(N|\text{man}) = 0.9$
- $P(V|\text{sleeps}) = 0.9$
- $P(D|\text{word}) = 0.6$ for any word other than *a*, *man* or *sleeps*
- $P(N|\text{word}) = 0.3$ for any word other than *a*, *man* or *sleeps*
- $P(V|\text{word}) = 0.1$ for any word other than *a*, *man* or *sleeps*

It is assumed that all other probabilities, not defined above could take any values such that $\sum_{\text{tag}} P(\text{tag}|\text{word}) = 1$ is satisfied for any word in V .

- Define the features of your maximum entropy model that can model this distribution. Mark your features as f_1, f_2 and so on. Each feature should have the same format as explained in the class.
[Hint: 6 Features should make the analysis easier]
- For each feature f_i , assume a weight λ_i . Now, write expression for the following probabilities in terms of your model parameters
 - ▶ $P(D|\text{cat})$
 - ▶ $P(N|\text{laughs})$
 - ▶ $P(D|\text{man})$
- What value do the parameters in your model take to give the distribution as described above. (i.e. $P(D|a) = 0.9$ and so on. You may leave the final answer in terms of equations)

Features for POS Tagging (Ratnaparakhi, 1996)

The specific word and tag context available to a feature is

$$h_i = \{w_i, w_{i+1}, w_{i+2}, w_{i-1}, w_{i-2}, t_{i-1}, t_{i-2}\}$$

Features for POS Tagging (Ratnaparakhi, 1996)

The specific word and tag context available to a feature is

$$h_i = \{w_i, w_{i+1}, w_{i+2}, w_{i-1}, w_{i-2}, t_{i-1}, t_{i-2}\}$$

Example: $f_j(h_i, t_i) = 1$ if $\text{suffix}(w_i) = \text{"ing"}$ & $t_i = \text{VBG}$

Example Features

Condition	Features
w_i is not rare	$w_i = X$ & $t_i = T$
w_i is rare	X is prefix of w_i , $ X \leq 4$ & $t_i = T$
	X is suffix of w_i , $ X \leq 4$ & $t_i = T$
	w_i contains number & $t_i = T$
	w_i contains uppercase character & $t_i = T$
	w_i contains hyphen & $t_i = T$
$\forall w_i$	$t_{i-1} = X$ & $t_i = T$
	$t_{i-2}t_{i-1} = XY$ & $t_i = T$
	$w_{i-1} = X$ & $t_i = T$
	$w_{i-2} = X$ & $t_i = T$
	$w_{i+1} = X$ & $t_i = T$
	$w_{i+2} = X$ & $t_i = T$

Example Features

<i>Word:</i>	the	stories	about	well-heeled	communities	and	developers
<i>Tag:</i>	DT	NNS	IN	JJ	NNS	CC	NNS
<i>Position:</i>	1	2	3	4	5	6	7

Example Features

Word:	the	stories	about	well-heeled	communities	and	developers
Tag:	DT	NNS	IN	JJ	NNS	CC	NNS
Position:	1	2	3	4	5	6	7

$w_i = \text{about}$ & $t_i = \text{IN}$
 $w_{i-1} = \text{stories}$ & $t_i = \text{IN}$
 $w_{i-2} = \text{the}$ & $t_i = \text{IN}$
 $w_{i+1} = \text{well-heeled}$ & $t_i = \text{IN}$
 $w_{i+2} = \text{communities}$ & $t_i = \text{IN}$
 $t_{i-1} = \text{NNS}$ & $t_i = \text{IN}$
 $t_{i-2}t_{i-1} = \text{DT NNS}$ & $t_i = \text{IN}$

$w_{i-1} = \text{about}$ & $t_i = \text{JJ}$
 $w_{i-2} = \text{stories}$ & $t_i = \text{JJ}$
 $w_{i+1} = \text{communities}$ & $t_i = \text{JJ}$
 $w_{i+2} = \text{and}$ & $t_i = \text{JJ}$
 $t_{i-1} = \text{IN}$ & $t_i = \text{JJ}$
 $t_{i-2}t_{i-1} = \text{NNS IN}$ & $t_i = \text{JJ}$
 $\text{prefix}(w_i) = w$ & $t_i = \text{JJ}$
 $\text{prefix}(w_i) = we$ & $t_i = \text{JJ}$
 $\text{prefix}(w_i) = wel$ & $t_i = \text{JJ}$
 $\text{prefix}(w_i) = well$ & $t_i = \text{JJ}$
 $\text{suffix}(w_i) = d$ & $t_i = \text{JJ}$
 $\text{suffix}(w_i) = ed$ & $t_i = \text{JJ}$
 $\text{suffix}(w_i) = led$ & $t_i = \text{JJ}$
 $\text{suffix}(w_i) = eled$ & $t_i = \text{JJ}$
 $w_i \text{ contains hyphen}$ & $t_i = \text{JJ}$

Search Algorithm

Conditional Probability

Given a sentence $\{w_1, \dots, w_n\}$, a tag sequence candidate $\{t_1, \dots, t_n\}$ has conditional probability:

$$P(t_1, \dots, t_n | w_1, \dots, w_n) = \prod_{i=1}^n p(t_i | x_i)$$

Search Algorithm

Conditional Probability

Given a sentence $\{w_1, \dots, w_n\}$, a tag sequence candidate $\{t_1, \dots, t_n\}$ has conditional probability:

$$P(t_1, \dots, t_n | w_1, \dots, w_n) = \prod_{i=1}^n p(t_i | x_i)$$

A *Tag Dictionary* is used, which, for each known word, lists the tags that it has appeared with in the training set.

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search description

- Generate tags for w_1 , find top N , set $s_{1j}, 1 \leq j \leq N$, accordingly.

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search description

- Generate tags for w_1 , find top N , set $s_{1j}, 1 \leq j \leq N$, accordingly.
- Initialize $i = 2$
 - ▶ Initialize $j = 1$
 - ▶ Generate tags for w_i , given $s_{(i-1)j}$ as previous tag context, and append each tag to $s_{(i-1)j}$ to make a new sequence
 - ▶ $j = j + 1$, repeat if $j \leq N$

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search description

- Generate tags for w_1 , find top N , set $s_{1j}, 1 \leq j \leq N$, accordingly.
- Initialize $i = 2$
 - ▶ Initialize $j = 1$
 - ▶ Generate tags for w_i , given $s_{(i-1)j}$ as previous tag context, and append each tag to $s_{(i-1)j}$ to make a new sequence
 - ▶ $j = j + 1$, repeat if $j \leq N$
- Find N highest probability sequences generated by above loop, set s_{ij} accordingly

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search description

- Generate tags for w_1 , find top N , set $s_{1j}, 1 \leq j \leq N$, accordingly.
- Initialize $i = 2$
 - ▶ Initialize $j = 1$
 - ▶ Generate tags for w_i , given $s_{(i-1)j}$ as previous tag context, and append each tag to $s_{(i-1)j}$ to make a new sequence
 - ▶ $j = j + 1$, repeat if $j \leq N$
- Find N highest probability sequences generated by above loop, set s_{ij} accordingly
- $i = i + 1$, repeat if $i \leq n$

Search Algorithm

Let $W = \{w_1, \dots, w_n\}$ be a test sentence, s_{ij} be the j th highest probability tag sequence up to and including word w_i .

Search description

- Generate tags for w_1 , find top N , set $s_{1j}, 1 \leq j \leq N$, accordingly.
- Initialize $i = 2$
 - ▶ Initialize $j = 1$
 - ▶ Generate tags for w_i , given $s_{(i-1)j}$ as previous tag context, and append each tag to $s_{(i-1)j}$ to make a new sequence
 - ▶ $j = j + 1$, repeat if $j \leq N$
- Find N highest probability sequences generated by above loop, set s_{ij} accordingly
- $i = i + 1$, repeat if $i \leq n$
- Return highest probability sequence s_{n1}

A Good Reference

Berger et al., *A Maximum Entropy Approach to Natural Language Processing*,
Computational Linguistics, Vol. 22, No. 1.

Conditional Random Fields

Pawan Goyal

CSE, IIT Kharagpur

Week 4, Lecture 5

Practice Question

Suppose you want to use a MaxEnt tagger to tag the sentence, “the light book”. We know that the top 2 POS tags for the words *the*, *light* and *book* are {*Det*, *Noun*}, {*Verb*, *Adj*} and {*Verb*, *Noun*}, respectively. Assume that the MaxEnt model uses the following history h_i (context) for a word w_i :

$$h_i = \{w_i, w_{i-1}, w_{i+1}, t_{i-1}\}$$

where w_{i-1} and w_{i+1} correspond to the previous and next words and t_{i-1} corresponds to the tag of the previous word. Accordingly, the following features are being used by the MaxEnt model:

- $f_1: t_{i-1} = \text{Det}$ and $t_i = \text{Adj}$
- $f_2: t_{i-1} = \text{Noun}$ and $t_i = \text{Verb}$
- $f_3: t_{i-1} = \text{Adj}$ and $t_i = \text{Noun}$
- $f_4: w_{i-1} = \text{the}$ and $t_i = \text{Adj}$
- $f_5: w_{i-1} = \text{the} \& w_{i+1} = \text{book}$ and $t_i = \text{Adj}$
- $f_6: w_{i-1} = \text{light}$ and $t_i = \text{Noun}$
- $f_7: w_{i+1} = \text{light}$ and $t_i = \text{Det}$
- $f_8: w_{i-1} = \text{NULL}$ and $t_i = \text{Noun}$

Assume that each feature has a uniform weight of 1.0.

Use Beam search algorithm with a beam-size of 2 to identify the highest probability tag sequence for the sentence.

Problem with Maximum Entropy Models

Per-state normalization

All the mass that arrives at a state must be distributed among the possible successor states

Problem with Maximum Entropy Models

Per-state normalization

All the mass that arrives at a state must be distributed among the possible successor states

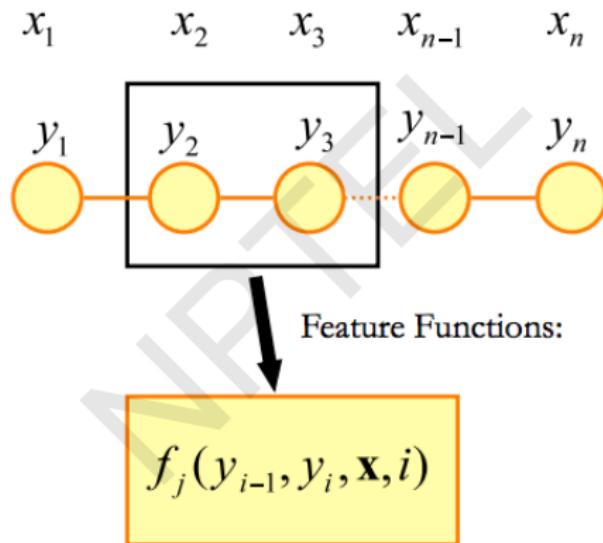
This gives a 'label bias' problem

Let's see the intuition (on paper)

Conditional Random Fields

- CRFs are conditionally trained, undirected graphical models.
- Let's look at the linear chain structure

Conditional Random Fields: Feature Functions



Feature Functions

Express some characteristic of the empirical distribution
that we wish to hold in the model distribution

$$f_j(y_{i-1}, y_i, \mathbf{x}, i)$$

1 if $y_{i-1} = IN$ and
 $y_i = NNP$ and
 $x_i = September$

0 otherwise

Conditional Random Fields: Distribution

Label sequence modelled as a normalized product of feature functions:

$$P(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{Z(\mathbf{x})} \exp \sum_{i=1}^n \sum_j \lambda_j f_j(y_{i-1}, y_i, \mathbf{x}, i)$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}} \sum_{i=1}^n \sum_j \lambda_j f_j(y_{i-1}, y_i, \mathbf{x}, i)$$

- Have the advantages of MEMM but avoid the label bias problem
- CRFs are globally normalized, whereas MEMMs are locally normalized.
- Widely used and applied. CRFs have been (close to) state-of-the-art in many sequence labeling tasks.

Syntax - Introduction

Pawan Goyal

CSE, IIT Kharagpur

Week 5: Lecture 1

What is Syntax?

NPTEL

What is Syntax?

- Refers to the way words are arranged together, and the relationship between them.

NPTEL

What is Syntax?

- Refers to the way words are arranged together, and the relationship between them.
- **Language Models:** Importance of modeling word order

What is Syntax?

- Refers to the way words are arranged together, and the relationship between them.
- **Language Models:** Importance of modeling word order
- **POS categories:** An equivalence class for words

What is Syntax?

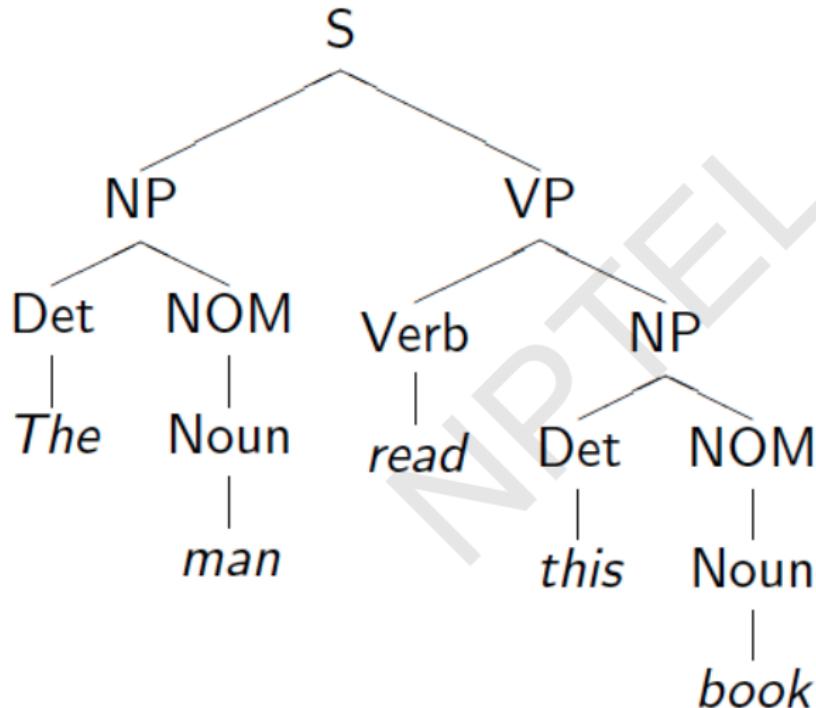
- Refers to the way words are arranged together, and the relationship between them.
- **Language Models:** Importance of modeling word order
- **POS categories:** An equivalence class for words
- More complex notions: constituency, grammatical relations, subcategorization etc.

What is Syntax?

- Refers to the way words are arranged together, and the relationship between them.
- **Language Models:** Importance of modeling word order
- **POS categories:** An equivalence class for words
- More complex notions: constituency, grammatical relations, subcategorization etc.



Syntax Tree: Example



Defining the notions: Constituency

Constituent

A group of words acts as a single unit - phrases, clauses etc.

Defining the notions: Constituency

Constituent

A group of words acts as a single unit - phrases, clauses etc.

Part of Speech - “Substitution Test”

The {sad, intelligent, green, fat, ...} one is in the corner.

Defining the notions: Constituency

Constituent

A group of words acts as a single unit - phrases, clauses etc.

Part of Speech - "Substitution Test"

The {sad, intelligent, green, fat, ...} one is in the corner.

Constituency: Noun Phrase

- *Kermit the frog*
- *they*
- *December twenty-sixth*
- *the reason he is running for president*

Constituent Phrases

Usually named based on the word that heads the constituent:

<i>the man from Amherst</i>	is a Noun Phrase (NP) because the head man is a noun
<i>extremely clever</i>	is an Adjective Phrase (AP) because the head clever is an adjective
<i>down the river</i>	is a Prepositional Phrase (PP) because the head down is a preposition
<i>killed the rabbit</i>	is a Verb Phrase (VP) because the head killed is a verb

Constituent Phrases

Usually named based on the word that heads the constituent:

- | | |
|-----------------------------|---|
| <i>the man from Amherst</i> | is a Noun Phrase (NP) because the head man is a noun |
| <i>extremely clever</i> | is an Adjective Phrase (AP) because the head clever is an adjective |
| <i>down the river</i> | is a Prepositional Phrase (PP) because the head down is a preposition |
| <i>killed the rabbit</i> | is a Verb Phrase (VP) because the head killed is a verb |

Words can also act as phrases

Joe grew potatoes

Constituent Phrases

Usually named based on the word that heads the constituent:

- | | |
|-----------------------------|---|
| <i>the man from Amherst</i> | is a Noun Phrase (NP) because the head man is a noun |
| <i>extremely clever</i> | is an Adjective Phrase (AP) because the head clever is an adjective |
| <i>down the river</i> | is a Prepositional Phrase (PP) because the head down is a preposition |
| <i>killed the rabbit</i> | is a Verb Phrase (VP) because the head killed is a verb |

Words can also act as phrases

Joe grew potatoes

Joe and *potatoes* are both nouns and noun phrases

Constituent Phrases

Usually named based on the word that heads the constituent:

- | | |
|-----------------------------|---|
| <i>the man from Amherst</i> | is a Noun Phrase (NP) because the head man is a noun |
| <i>extremely clever</i> | is an Adjective Phrase (AP) because the head clever is an adjective |
| <i>down the river</i> | is a Prepositional Phrase (PP) because the head down is a preposition |
| <i>killed the rabbit</i> | is a Verb Phrase (VP) because the head killed is a verb |

Words can also act as phrases

Joe grew potatoes

Joe and *potatoes* are both nouns and noun phrases

Compare with: *The man from Amherst grew beautiful russet potatoes.*

Constituent Phrases

Usually named based on the word that heads the constituent:

- | | |
|-----------------------------|---|
| <i>the man from Amherst</i> | is a Noun Phrase (NP) because the head man is a noun |
| <i>extremely clever</i> | is an Adjective Phrase (AP) because the head clever is an adjective |
| <i>down the river</i> | is a Prepositional Phrase (PP) because the head down is a preposition |
| <i>killed the rabbit</i> | is a Verb Phrase (VP) because the head killed is a verb |

Words can also act as phrases

Joe grew potatoes

Joe and *potatoes* are both nouns and noun phrases

Compare with: *The man from Amherst grew beautiful russet potatoes.*

Joe appears in a place that a larger noun phrase could have been.

Evidence that constituency exists

They appear in similar environments

NPTEL

Evidence that constituency exists

They appear in similar environments

Kermit the frog comes on stage

They come to Massachusetts every summer

December twenty-sixth comes after Christmas

The reason he is running for president comes out only now.

But not each individual word in the constituent

*The comes our... *is comes out... *for comes out...

Evidence that constituency exists

They appear in similar environments

Kermit the frog comes on stage

They come to Massachusetts every summer

December twenty-sixth comes after Christmas

The reason he is running for president comes out only now.

But not each individual word in the constituent

*The comes our... *is comes out... *for comes out...

Can be placed in a number of different locations

Evidence that constituency exists

They appear in similar environments

Kermit the frog comes on stage

They come to Massachusetts every summer

December twenty-sixth comes after Christmas

The reason he is running for president comes out only now.

But not each individual word in the constituent

*The comes our... *is comes out... *for comes out...

Can be placed in a number of different locations

Constituent = Prepositional phrase: *On December twenty-sixth*

On December twenty-sixth I'd like to fly to Florida.

I'd like to fly on December twenty-sixth to Florida.

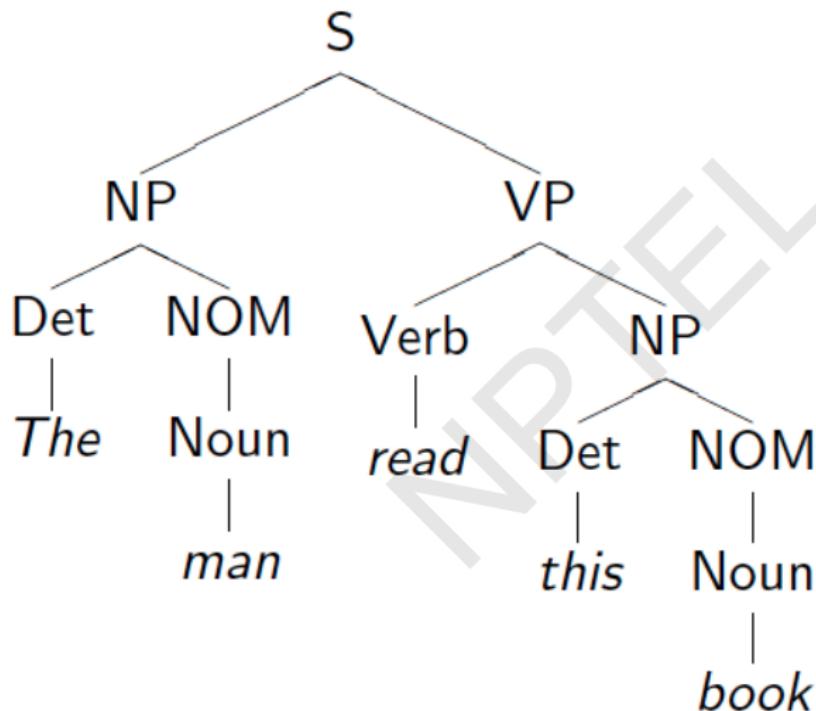
I'd like to fly to Florida on December twenty-sixth.

But not split apart

*On December I'd like to fly twenty-sixth to Florida.

*On I'd like to fly December twenty-sixth to Florida.

Modeling Constituency: what tool do we need?



Modeling Constituency

NPTEL

Modeling Constituency

Context-free grammar

The most common way of modeling constituency

Modeling Constituency

Context-free grammar

The most common way of modeling constituency

Consists of production Rules

These rules express the ways in which the symbols of the language can be grouped and ordered together

Modeling Constituency

Context-free grammar

The most common way of modeling constituency

Consists of production Rules

These rules express the ways in which the symbols of the language can be grouped and ordered together

Example

Noun phrase can be composed of either a ProperNoun or a determiner (Det) followed by a Nominal; a Nominal can be more than one nouns

Modeling Constituency

Context-free grammar

The most common way of modeling constituency

Consists of production Rules

These rules express the ways in which the symbols of the language can be grouped and ordered together

Example

Noun phrase can be composed of either a ProperNoun or a determiner (Det) followed by a Nominal; a Nominal can be more than one nouns

$NP \rightarrow \text{Det Nominal}$

$NP \rightarrow \text{ProperNoun}$

$\text{Nominal} \rightarrow \text{Noun} \mid \text{Noun Nominal}$

CFG for Languages

NPTEL

CFG for Languages

CFG: $G = (T, N, S, R)$

- T : set of terminals
- N : set of non-terminals
 - ▶ For NLP, we distinguish out a set $P \subset N$ of pre-terminals, which always rewrite as terminals
- S : start symbol
- R : Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)^*$

CFG for Languages

CFG: $G = (T, N, S, R)$

- T : set of terminals
- N : set of non-terminals
 - ▶ For NLP, we distinguish out a set $P \subset N$ of pre-terminals, which always rewrite as terminals
- S : start symbol
- R : Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)^*$

Terminals and pre-terminals

Terminals mainly correspond to words in the language while pre-terminals mainly correspond to POS categories

CFG for Languages

CFG: $G = (T, N, S, R)$

- T : set of terminals
- N : set of non-terminals
 - ▶ For NLP, we distinguish out a set $P \subset N$ of pre-terminals, which always rewrite as terminals
- S : start symbol
- R : Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)^*$

Terminals and pre-terminals

Terminals mainly correspond to words in the language while pre-terminals mainly correspond to POS categories

CFG for Languages

Example

$NP \rightarrow \text{Det Nominal}$

$NP \rightarrow \text{ProperNoun}$

$\text{Nominal} \rightarrow \text{Noun} \mid \text{Noun Nominal}$

CFG for Languages

Example

$NP \rightarrow \text{Det Nominal}$

$NP \rightarrow \text{ProperNoun}$

$\text{Nominal} \rightarrow \text{Noun} \mid \text{Noun Nominal}$

Now, these can be combined with other rules, that express facts about a lexicon.

CFG for Languages

Example

$NP \rightarrow \text{Det Nominal}$

$NP \rightarrow \text{ProperNoun}$

$\text{Nominal} \rightarrow \text{Noun} \mid \text{Noun Nominal}$

Now, these can be combined with other rules, that express facts about a lexicon.

$\text{Det} \rightarrow \text{a}$

$\text{Det} \rightarrow \text{the}$

$\text{Noun} \rightarrow \text{flight}$

CFG for Languages

Example

$NP \rightarrow \text{Det Nominal}$

$NP \rightarrow \text{ProperNoun}$

$\text{Nominal} \rightarrow \text{Noun} \mid \text{Noun Nominal}$

Now, these can be combined with other rules, that express facts about a lexicon.

$\text{Det} \rightarrow \text{a}$

$\text{Det} \rightarrow \text{the}$

$\text{Noun} \rightarrow \text{flight}$

Can you identify the terminal, non-terminals and preterminals?

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

→ Det Noun

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

→ Det Noun → a Noun

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

→ Det Noun → a Noun → a flight

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

→ Det Noun → a Noun → a flight

- Thus a CFG can be used to randomly generate a series of strings

CFG as a generator

NP → Det Nominal

NP → ProperNoun

Nominal → Noun | Noun Nominal

Det → a

Det → the

Noun → flight

Generating ‘a flight’:

NP → Det Nominal

→ Det Noun → a Noun → a flight

- Thus a CFG can be used to randomly generate a series of strings
- This sequence of rule expansions is called a derivation of the string of words, usually represented as a tree

CFGs and Grammaticality

A CFG defines a formal language = set of all sentences (string of words) that can be derived by the grammar

CFGs and Grammaticality

A CFG defines a formal language = set of all sentences (string of words) that can be derived by the grammar

- Sentences in this set are said to be **grammatical**
- Sentences outside this set are said to be **ungrammatical**

Recursive Definition

- $\text{PP} \rightarrow \text{Prep NP}$
- $\text{NP} \rightarrow \text{Noun PP}$

Recursive Definition

- $PP \rightarrow Prep\ NP$
- $NP \rightarrow Noun\ PP$

Example Sentence

[_SThe mailman ate his [_{NP} lunch [_{PP} with his friend [_{PP} from the cleaning staff [_{PP} of the building [_{PP} at the intersection [_{PP} on the north end [_{PP} of town]]]]]]].

What does Context stand for in CFG?

NPTEL

What does Context stand for in CFG?

- The notion of *context* has nothing to do with the ordinary meaning of word context in language

What does Context stand for in CFG?

- The notion of *context* has nothing to do with the ordinary meaning of word context in language
- All it really means is that the non-terminal on the left-hand side of a rule is out there all by itself (free of context)

What does Context stand for in CFG?

- The notion of *context* has nothing to do with the ordinary meaning of word context in language
- All it really means is that the non-terminal on the left-hand side of a rule is out there all by itself (free of context)

$A \rightarrow BC$

- I can rewrite A as B followed by C regardless of the context in which A is found
- Or when I see a B followed by a C , I can infer an A regardless of the surrounding context

Syntax -Parsing I

Pawan Goyal

CSE, IIT Kharagpur

Week 5: Lecture 2

Grammar Rewrite Rules

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a \mid the$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid man$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid read$
$NP \rightarrow Det\ NOM$	$Aux \rightarrow does$
$NOM \rightarrow Noun$	
$NOM \rightarrow Noun\ NOM$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	

Grammar Rewrite Rules

$S \rightarrow NP\ VP$	Det \rightarrow <i>that</i> <i>this</i> <i>a</i> <i>the</i>
$S \rightarrow Aux\ NP\ VP$	Noun \rightarrow <i>book</i> <i>flight</i> <i>meal</i> <i>man</i>
$S \rightarrow VP$	Verb \rightarrow <i>book</i> <i>include</i> <i>read</i>
$NP \rightarrow Det\ NOM$	Aux \rightarrow <i>does</i>
$NOM \rightarrow Noun$	
$NOM \rightarrow Noun\ NOM$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	

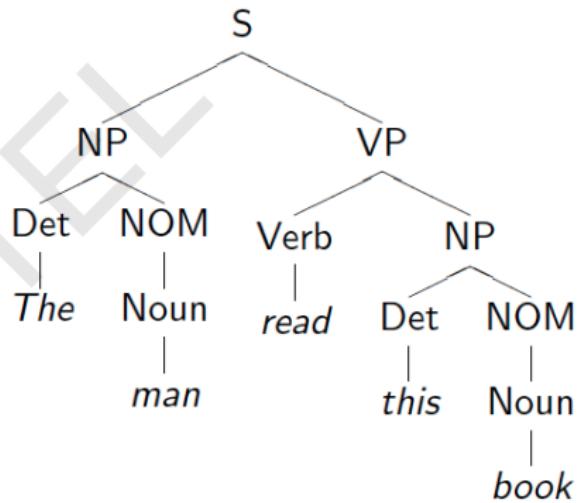
$S \rightarrow NP\ VP$
 $\rightarrow Det\ NOM\ VP$
 $\rightarrow The\ NOM\ VP$
 $\rightarrow The\ Noun\ VP$
 $\rightarrow The\ man\ VP$
 $\rightarrow The\ man\ Verb\ NP$
 $\rightarrow The\ man\ read\ NP$
 $\rightarrow The\ man\ read\ Det\ NOM$
 $\rightarrow The\ man\ read\ this\ NOM$
 $\rightarrow The\ man\ read\ this\ Noun$
 $\rightarrow The\ man\ read\ this\ book$

Parse Tree

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

Parse Tree

$S \rightarrow NP\ VP$
 $\rightarrow Det\ NOM\ VP$
 $\rightarrow The\ NOM\ VP$
 $\rightarrow The\ Noun\ VP$
 $\rightarrow The\ man\ VP$
 $\rightarrow The\ man\ Verb\ NP$
 $\rightarrow The\ man\ read\ NP$
 $\rightarrow The\ man\ read\ Det\ NOM$
 $\rightarrow The\ man\ read\ this\ NOM$
 $\rightarrow The\ man\ read\ this\ Noun$
 $\rightarrow The\ man\ read\ this\ book$



What is Parsing?

- The process of taking a string and a grammar and returning all possible parse trees for that string

What is Parsing?

- The process of taking a string and a grammar and returning all possible parse trees for that string
- That is, find all trees, whose root is the start symbol S , which cover exactly the words in the input

What is Parsing?

- The process of taking a string and a grammar and returning all possible parse trees for that string
- That is, find all trees, whose root is the start symbol S , which cover exactly the words in the input

What are the constraints? “book that flight”

- There must be three leaves, *book*, *that* and *flight*
- The tree must have one root, the start symbol S

What is Parsing?

- The process of taking a string and a grammar and returning all possible parse trees for that string
- That is, find all trees, whose root is the start symbol S , which cover exactly the words in the input

What are the constraints? “book that flight”

- There must be three leaves, *book*, *that* and *flight*
- The tree must have one root, the start symbol S
- Give rise to two search strategies: *top-down* (goal-oriented) and *bottom-up* (data-directed)

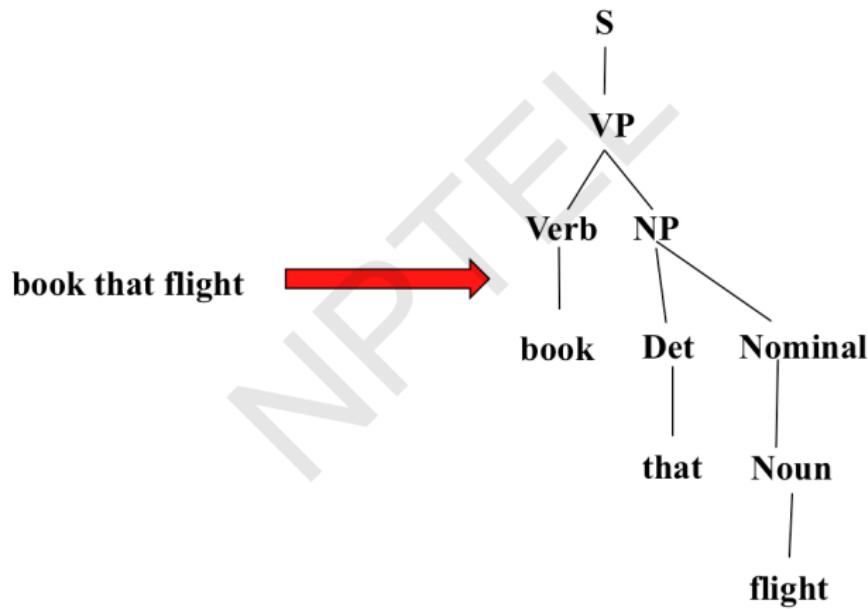
Grammar

$S \rightarrow NP\ VP$
 $S \rightarrow Aux\ NP\ VP$
 $S \rightarrow VP$
 $NP \rightarrow Pronoun$
 $NP \rightarrow Proper-Noun$
 $NP \rightarrow Det\ Nominal$
 $Nominal \rightarrow Noun$
 $Nominal \rightarrow Nominal\ Noun$
 $Nominal \rightarrow Nominal\ PP$
 $VP \rightarrow Verb$
 $VP \rightarrow Verb\ NP$
 $VP \rightarrow VP\ PP$
 $PP \rightarrow Prep\ NP$

Lexicon

$Det \rightarrow the\ | a\ | that\ | this$
 $Noun \rightarrow book\ | flight\ | meal\ | money$
 $Verb \rightarrow book\ | include\ | prefer$
 $Pronoun \rightarrow I\ | he\ | she\ | me$
 $Proper-Noun \rightarrow Houston\ | NWA$
 $Aux \rightarrow does$
 $Prep \rightarrow from\ | to\ | on\ | near\ | through$

Parsing



Top-Down Parsing

- Searches for a parse tree by trying to build upon the root node S down to the leaves

Top-Down Parsing

- Searches for a parse tree by trying to build upon the root node S down to the leaves
- Start by assuming that the input can be derived by the designated start symbol S

Top-Down Parsing

- Searches for a parse tree by trying to build upon the root node S down to the leaves
- Start by assuming that the input can be derived by the designated start symbol S
- Find all trees that can start with S , by looking at the grammar rules with S on the left-hand side

Top-Down Parsing

- Searches for a parse tree by trying to build upon the root node S down to the leaves
- Start by assuming that the input can be derived by the designated start symbol S
- Find all trees that can start with S , by looking at the grammar rules with S on the left-hand side
- Trees are grown downward until they eventually reach the POS categories at the bottom

Top-Down Parsing

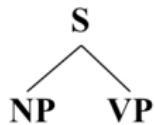
- Searches for a parse tree by trying to build upon the root node S down to the leaves
- Start by assuming that the input can be derived by the designated start symbol S
- Find all trees that can start with S , by looking at the grammar rules with S on the left-hand side
- Trees are grown downward until they eventually reach the POS categories at the bottom
- Trees whose leaves fail to match the words in the input can be rejected

Top-Down Parsing

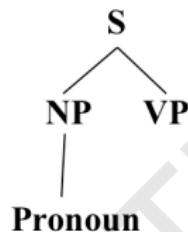
S

NPTEL

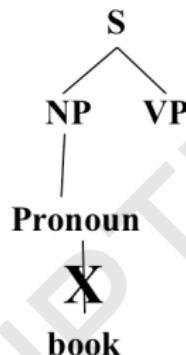
Top-Down Parsing



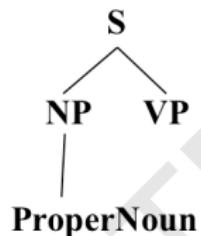
Top-Down Parsing



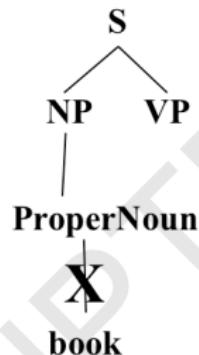
Top-Down Parsing



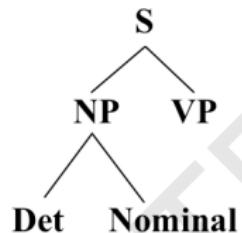
Top-Down Parsing



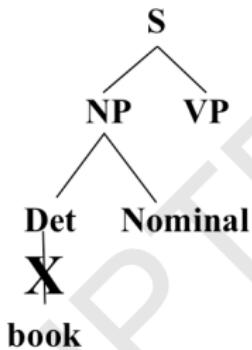
Top-Down Parsing



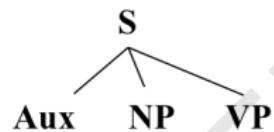
Top-Down Parsing



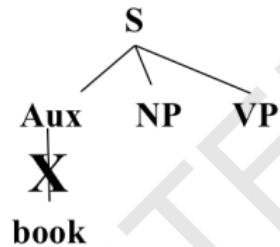
Top-Down Parsing



Top-Down Parsing



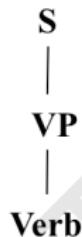
Top-Down Parsing



Top-Down Parsing

S
|
VP

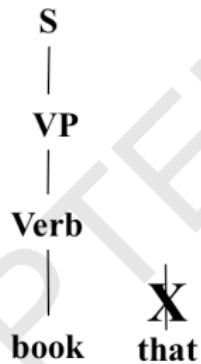
Top-Down Parsing



Top-Down Parsing



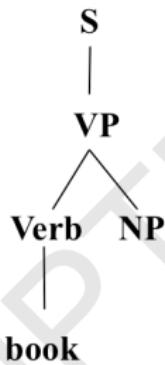
Top-Down Parsing



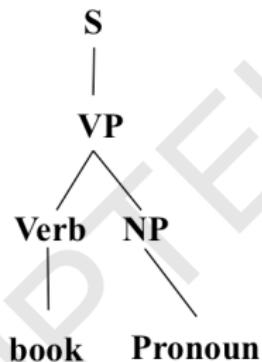
Top-Down Parsing



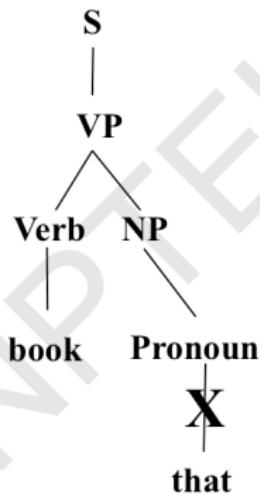
Top-Down Parsing



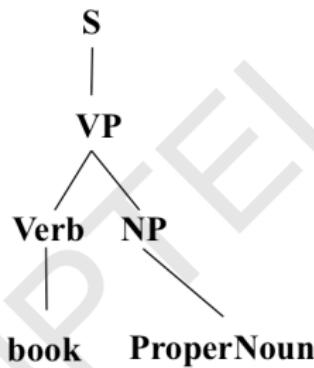
Top-Down Parsing



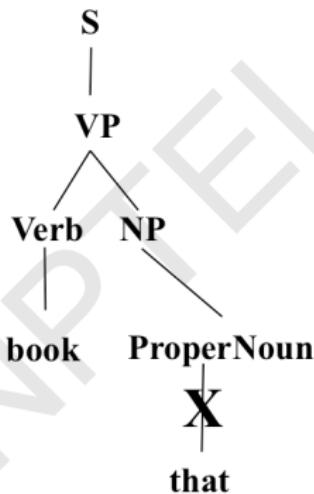
Top-Down Parsing



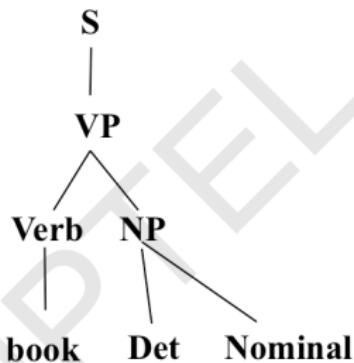
Top-Down Parsing



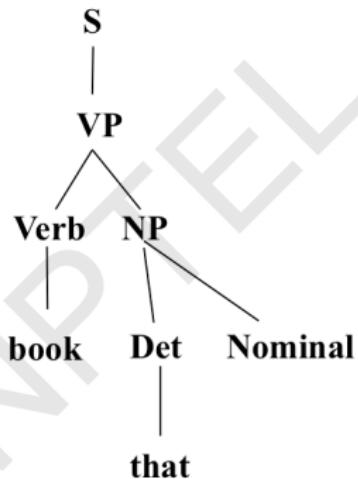
Top-Down Parsing



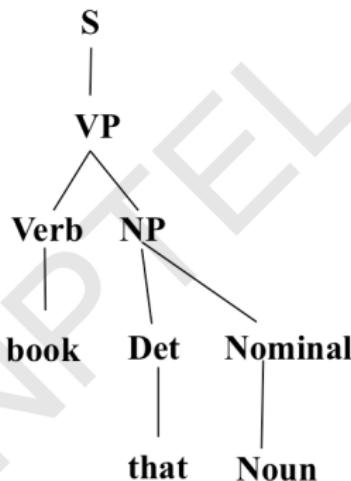
Top-Down Parsing



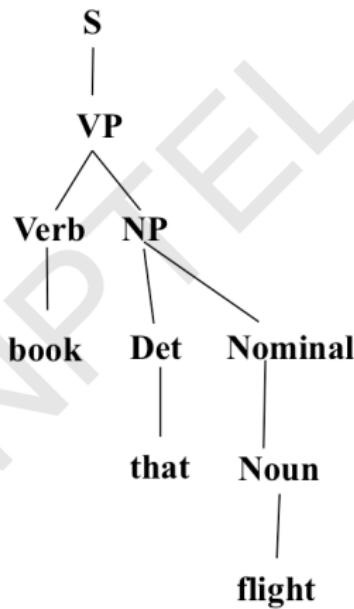
Top-Down Parsing



Top-Down Parsing



Top-Down Parsing



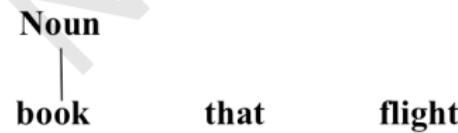
Bottom-Up Parsing

- The parser starts with the words of the input, and tries to build trees from the words up, by applying rules from the grammar one at a time
- Parser looks for the places in the parse-in-progress where the right-hand-side of some rule might fit.

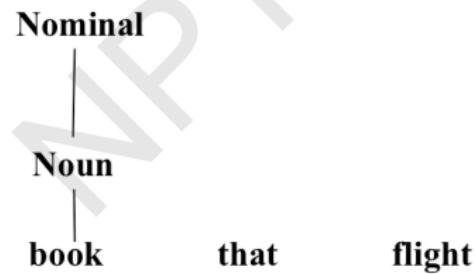
Bottom-Up Parsing

book that flight

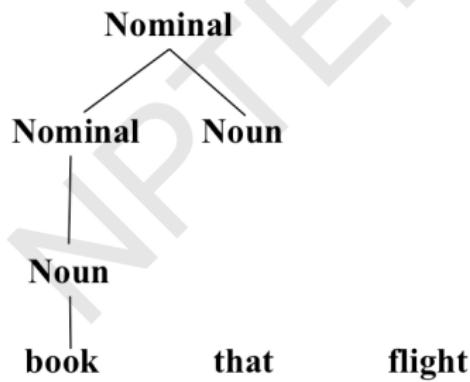
Bottom-Up Parsing



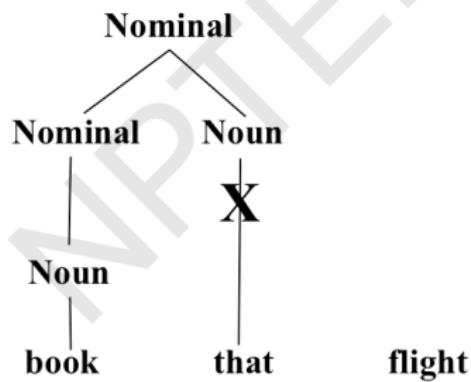
Bottom-Up Parsing



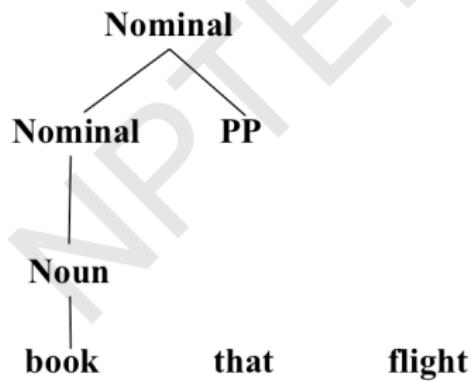
Bottom-Up Parsing



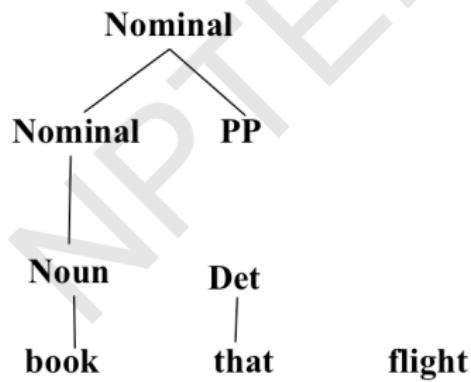
Bottom-Up Parsing



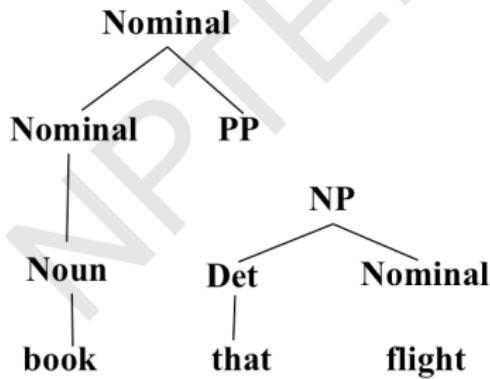
Bottom-Up Parsing



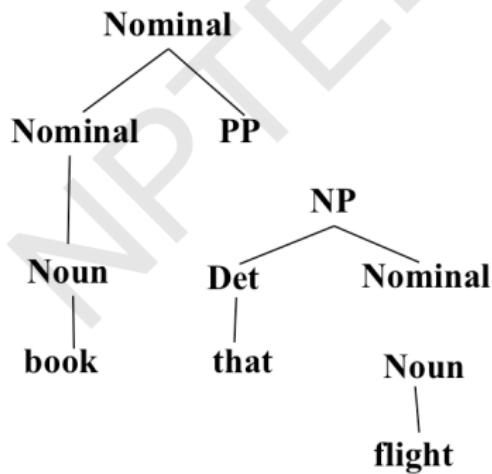
Bottom-Up Parsing



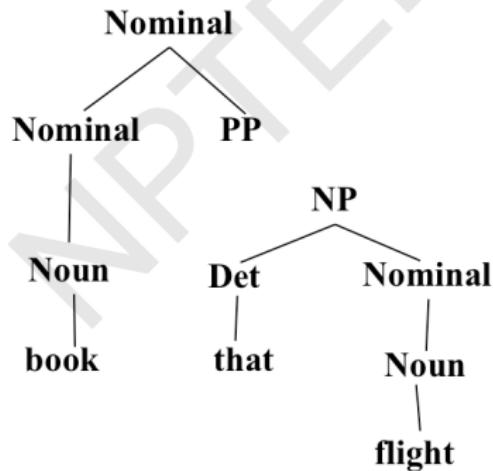
Bottom-Up Parsing



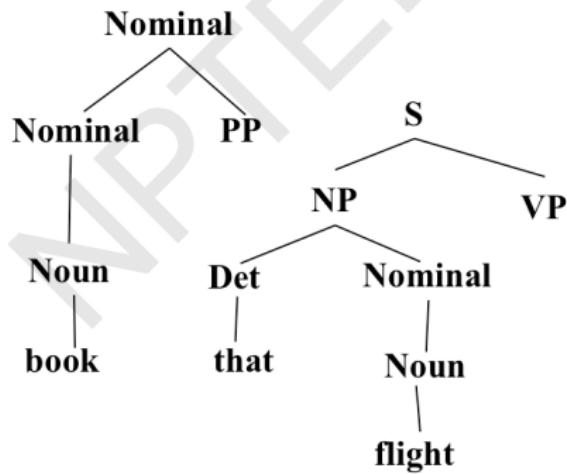
Bottom-Up Parsing



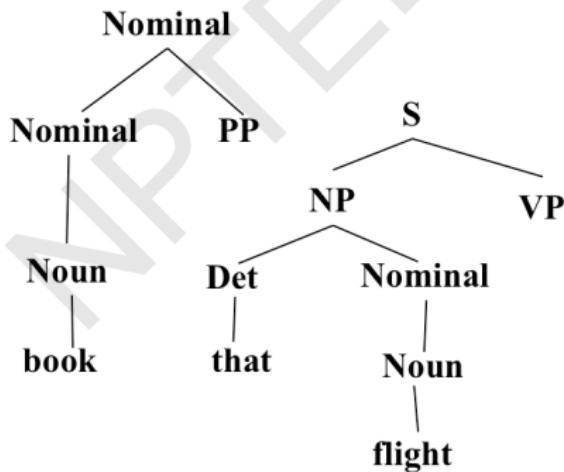
Bottom-Up Parsing



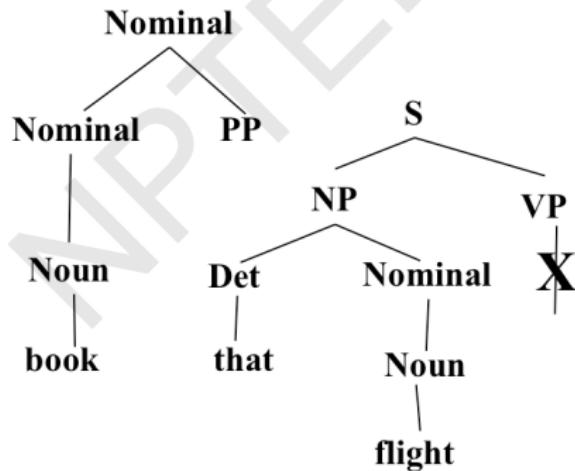
Bottom-Up Parsing



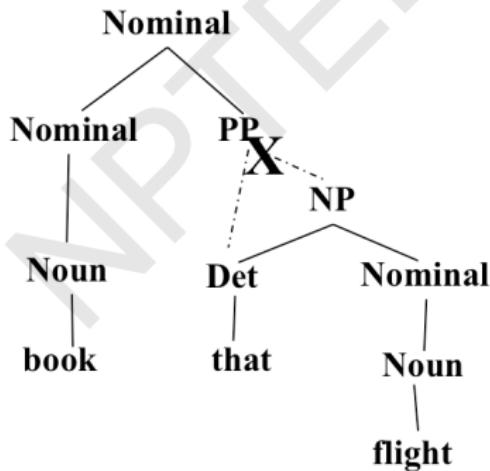
Bottom-Up Parsing



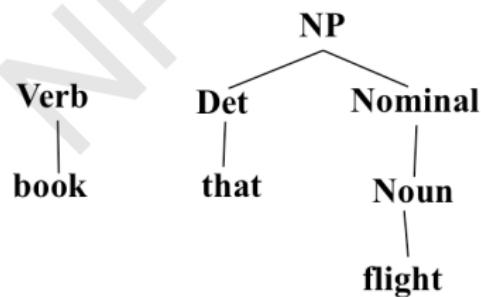
Bottom-Up Parsing



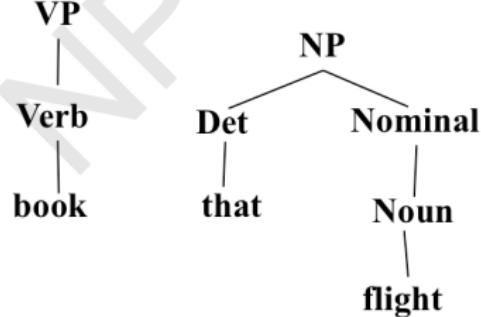
Bottom-Up Parsing



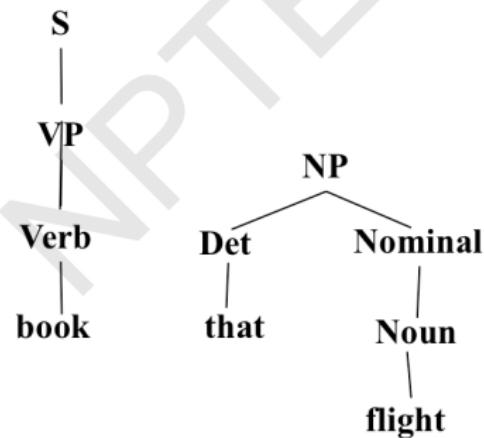
Bottom-Up Parsing



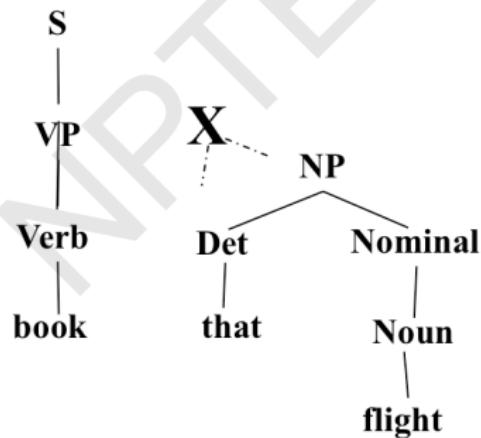
Bottom-Up Parsing



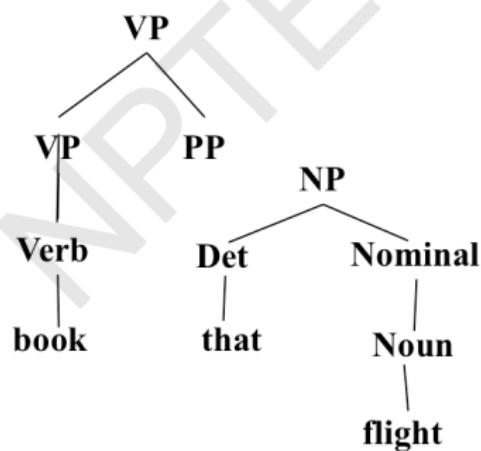
Bottom-Up Parsing



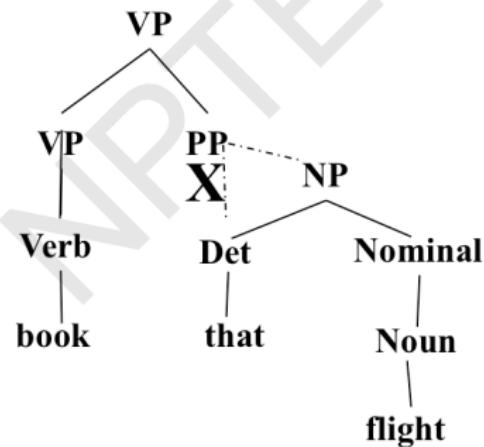
Bottom-Up Parsing



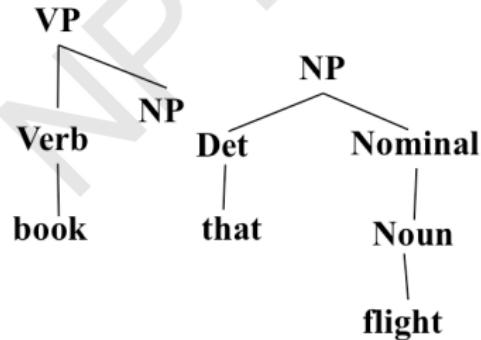
Bottom-Up Parsing



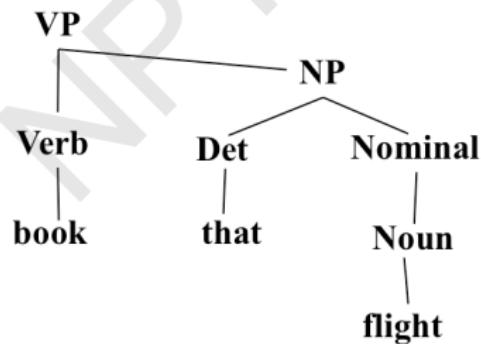
Bottom-Up Parsing



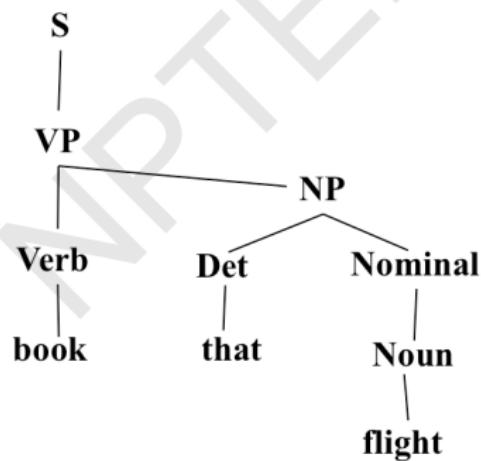
Bottom-Up Parsing



Bottom-Up Parsing



Bottom-Up Parsing



Top-Down vs. Bottom-Up

NPTEL

Top-Down vs. Bottom-Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.

Top-Down vs. Bottom-Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.

Top-Down vs. Bottom-Up

- Top down never explores options that will not lead to a full parse, but can explore many options that never connect to the actual sentence.
- Bottom up never explores options that do not connect to the actual sentence but can explore options that can never lead to a full parse.
- Relative amounts of wasted search depend on how much the grammar branches in each direction.

Dynamic Programming Parsing

NPTEL

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memoizing) critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.

Dynamic Programming Parsing

- To avoid extensive repeated work, must cache intermediate results, i.e. completed phrases.
- Caching (memoizing) critical to obtaining a polynomial time parsing (recognition) algorithm for CFGs.
- Dynamic programming algorithms based on both top-down and bottom-up search can achieve $O(n^3)$ recognition time where n is the length of the input string.

Dynamic Programming Parsing Methods

- CKY (Cocke-Kasami-Younger) algorithm: bottom-up, requires normalizing the grammar

Dynamic Programming Parsing Methods

- CKY (Cocke-Kasami-Younger) algorithm: bottom-up, requires normalizing the grammar
- Earley Parser - top-down, does not require normalizing grammar, more complex

Dynamic Programming Parsing Methods

- CKY (Cocke-Kasami-Younger) algorithm: bottom-up, requires normalizing the grammar
- Earley Parser - top-down, does not require normalizing grammar, more complex
- More generally, *chart parsers* retain completed phrases in a chart and can combine top-down and bottom-up searches.

CKY Algorithm

- Grammar must be converted to Chomsky normal form (CNF) in which all productions must have
 - ▶ Either, exactly two non-terminals on the RHS
 - ▶ Or, 1 terminal symbol on the RHS

CKY Algorithm

- Grammar must be converted to Chomsky normal form (CNF) in which all productions must have
 - ▶ Either, exactly two non-terminals on the RHS
 - ▶ Or, 1 terminal symbol on the RHS
- Parse bottom-up storing phrases formed from all substrings in a triangular table (chart)

Converting to CNF

Original Grammar

S → NP VP

S → Aux NP VP

S → VP

NP → Pronoun

NP → Proper-Noun

NP → Det Nominal

Nominal → Noun

Nominal → Nominal Noun

Nominal → Nominal PP

VP → Verb

VP → Verb NP

VP → VP PP

PP → Prep NP

Pronoun → I | he | she | me

Noun → book | flight | meal | money

Verb → book | include | prefer

Proper-Noun → Houston | NWA

Converting to CNF

Original Grammar

$S \rightarrow NP\ VP$
 $S \rightarrow Aux\ NP\ VP$
 $S \rightarrow VP$
 $NP \rightarrow Pronoun$
 $NP \rightarrow Proper-Noun$
 $NP \rightarrow Det\ Nominal$
 $Nominal \rightarrow Noun$
 $Nominal \rightarrow Nominal\ Noun$
 $Nominal \rightarrow Nominal\ PP$
 $VP \rightarrow Verb$
 $VP \rightarrow Verb\ NP$
 $VP \rightarrow VP\ PP$
 $PP \rightarrow Prep\ NP$
 $Pronoun \rightarrow I\ | he\ | she\ | me$
 $Noun \rightarrow book\ | flight\ | meal\ | money$
 $Verb \rightarrow book\ | include\ | prefer$
 $Proper-Noun \rightarrow Houston\ | NWA$

Chomsky Normal Form

$S \rightarrow NP\ VP$
 $S \rightarrow X1\ VP$
 $X1 \rightarrow Aux\ NP$
 $S \rightarrow book\ | include\ | prefer$
 $S \rightarrow Verb\ NP$
 $S \rightarrow VP\ PP$
 $NP \rightarrow I\ | he\ | she\ | me$
 $NP \rightarrow Houston\ | NWA$
 $NP \rightarrow Det\ Nominal$
 $Nominal \rightarrow book\ | flight\ | meal\ | money$
 $Nominal \rightarrow Nominal\ Noun$
 $Nominal \rightarrow Nominal\ PP$
 $VP \rightarrow book\ | include\ | prefer$
 $VP \rightarrow Verb\ NP$
 $VP \rightarrow VP\ PP$
 $PP \rightarrow Prep\ NP$
 $Pronoun \rightarrow I\ | he\ | she\ | me$
 $Noun \rightarrow book\ | flight\ | meal\ | money$
 $Verb \rightarrow book\ | include\ | prefer$
 $Proper-Noun \rightarrow Houston\ | NWA$

Syntax -CKY, PCFGs

Pawan Goyal

CSE, IIT Kharagpur

Week 5: Lecture 3

CKY Algorithm

- Let n be the number of words in the input. Think about $n + 1$ lines separating them, numbered 0 to n .
- x_{ij} will denote the words between line i and j
- We build a table so that x_{ij} contains all the possible non-terminal spanning for words between line i and j .
- We build the Table bottom-up.

CKY Algorithm

- Let n be the number of words in the input. Think about $n + 1$ lines separating them, numbered 0 to n .
- x_{ij} will denote the words between line i and j
- We build a table so that x_{ij} contains all the possible non-terminal spanning for words between line i and j .
- We build the Table bottom-up.

Home Exercise

Use CKY algorithm to find the parse tree for “Book the flight through Houston” using the CNF form shown in the previous slide.

CKY for CFG

a 1	pilot 2	likes 3	flying 4	planes 5

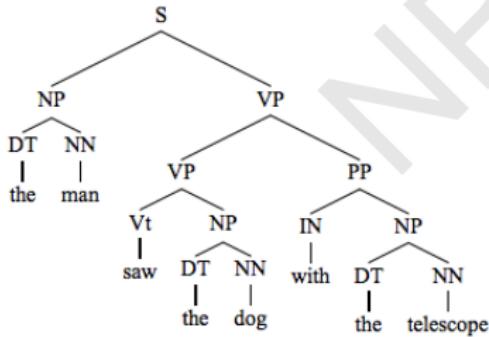
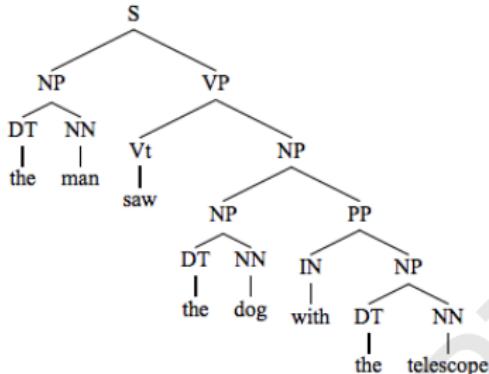
$S \rightarrow NP\ VP$
 $VP \rightarrow VBG\ NNS$
 $VP \rightarrow VBZ\ VP$
 $VP \rightarrow VBZ\ NP$
 $NP \rightarrow DT\ NN$
 $NP \rightarrow JJ\ NNS$
 $DT \rightarrow a$
 $NN \rightarrow pilot$
 $VBZ \rightarrow likes$
 $VBG \rightarrow flying$
 $JJ \rightarrow flying$
 $NNS \rightarrow planes$

CKY for CFG

a 1	pilot 2	likes 3	flying 4	planes 5
DT	NP	-	-	S S
	NN	-	-	-
		VBZ	-	VP VP
			JJ VBG	NP VP
				NNS

$S \rightarrow NP\ VP$
 $VP \rightarrow VBG\ NNS$
 $VP \rightarrow VBZ\ VP$
 $VP \rightarrow VBZ\ NP$
 $NP \rightarrow DT\ NN$
 $NP \rightarrow JJ\ NNS$
 $DT \rightarrow a$
 $NN \rightarrow pilot$
 $VBZ \rightarrow likes$
 $VBG \rightarrow flying$
 $JJ \rightarrow flying$
 $NNS \rightarrow planes$

What about Ambiguities?



Probabilistic Context-free grammars (PCFGs)

PCFG: $G = (T, N, S, R, P)$

- T : set of terminals
- N : set of non-terminals
 - ▶ For NLP, we distinguish out a set $P \subset N$ of pre-terminals, which always rewrite as terminals
- S : start symbol
- R : Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)^*$

Probabilistic Context-free grammars (PCFGs)

PCFG: $G = (T, N, S, R, P)$

- T : set of terminals
- N : set of non-terminals
 - ▶ For NLP, we distinguish out a set $P \subset N$ of pre-terminals, which always rewrite as terminals
- S : start symbol
- R : Rules/productions of the form $X \rightarrow \gamma$, $X \in N$ and $\gamma \in (T \cup N)^*$
- $P(R)$ gives the probability of each rule.

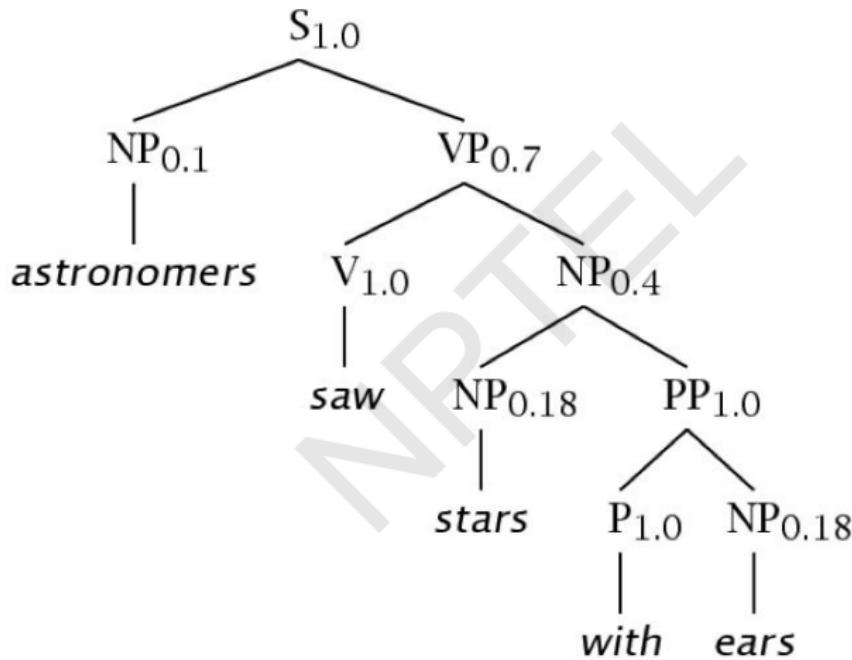
$$\forall X \in N, \sum_{X \rightarrow \gamma \in R} P(X \rightarrow \gamma) = 1$$

A Simple PCFG (in CNF)

S	→	NP VP	1.0	NP	→	NP PP	0.4
VP	→	V NP	0.7	NP	→	<i>astronomers</i>	0.1
VP	→	VP PP	0.3	NP	→	<i>ears</i>	0.18
PP	→	P NP	1.0	NP	→	<i>saw</i>	0.04
P	→	<i>with</i>	1.0	NP	→	<i>stars</i>	0.18
V	→	<i>saw</i>	1.0	NP	→	<i>telescope</i>	0.1

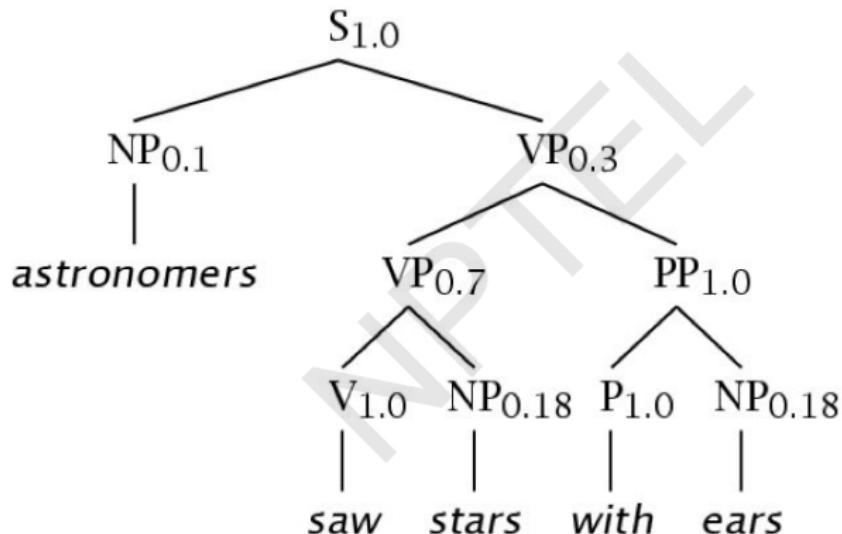
Example Trees

$t_1:$



Example Trees

t_2 :



Probability of trees and strings

- $P(t)$: The probability of tree is the product of the probabilities of the rules used to generate it
- $P(w_{1n})$: The probability of the string is the sum of the probabilities of the trees which have that string as their yield

Tree and String probabilities

NPTEL

Tree and String probabilities

w_{15} = *astronomers saw stars with ears*

$$\begin{aligned} P(t_1) &= 1.0 * 0.1 * 0.7 * 1.0 * 0.4 * 0.18 \\ &\quad * 1.0 * 1.0 * 0.18 \end{aligned}$$

$$= 0.0009072$$

$$\begin{aligned} P(t_2) &= 1.0 * 0.1 * 0.3 * 0.7 * 1.0 * 0.18 \\ &\quad * 1.0 * 1.0 * 0.18 \end{aligned}$$

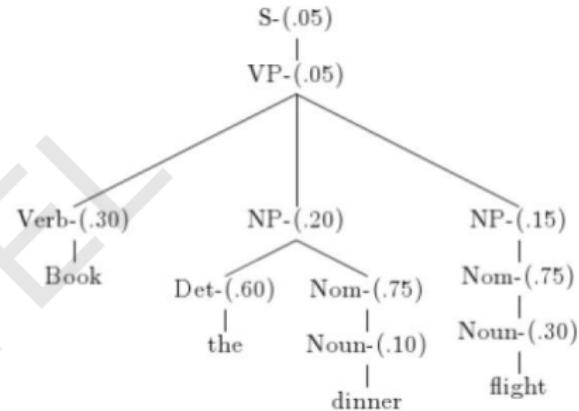
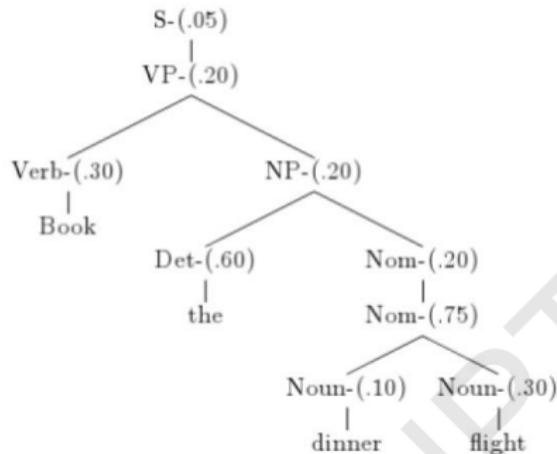
$$= 0.0006804$$

$$\begin{aligned} P(w_{15}) &= P(t_1) + P(t_2) \\ &= 0.0009072 + 0.0006804 \\ &= 0.0015876 \end{aligned}$$

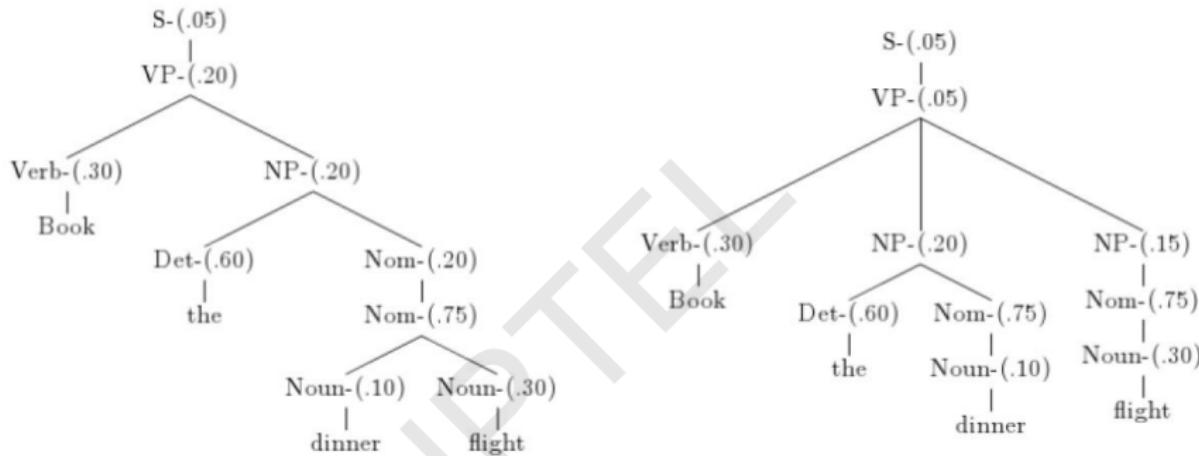
“Book the dinner flight”

NPTEL

“Book the dinner flight”



“Book the dinner flight”



Probabilities

- Parse tree 1: $.05 \times .20 \times .30 \times .20 \times .60 \times .20 \times .75 \times .10 \times .30 = 1.62 \times 10^{-6}$
- Parse tree 2: $.05 \times .05 \times .30 \times .20 \times .60 \times .75 \times .10 \times .15 \times .75 \times .30 = 2.28 \times 10^{-7}$

Features of PCFGs

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse

Features of PCFGs

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse
- *But* the probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence. Thus, PCFG does not give a very good idea of the plausibility of the sentence.

Features of PCFGs

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse
- *But* the probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence. Thus, PCFG does not give a very good idea of the plausibility of the sentence.
- Real text tends to have grammatical mistakes. PCFG avoids this problem by ruling out nothing, but by giving implausible sentences a low probability

Features of PCFGs

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse
- *But* the probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence. Thus, PCFG does not give a very good idea of the plausibility of the sentence.
- Real text tends to have grammatical mistakes. PCFG avoids this problem by ruling out nothing, but by giving implausible sentences a low probability
- In practice, a PCFG is a worse language model for English than an n-gram model

Features of PCFGs

- As the number of possible trees for a given input grows, a PCFG gives some idea of the plausibility of a particular parse
- *But* the probability estimates are based purely on structural factors, and do not factor in lexical co-occurrence. Thus, PCFG does not give a very good idea of the plausibility of the sentence.
- Real text tends to have grammatical mistakes. PCFG avoids this problem by ruling out nothing, but by giving implausible sentences a low probability
- In practice, a PCFG is a worse language model for English than an n-gram model
- All else being equal, the probability of a smaller tree is greater than a larger tree

Important Questions?

Let W_{1m} be a sentence, G a grammar, t a parse tree

Important Questions?

Let W_{1m} be a sentence, G a grammar, t a parse tree

- What is the most likely parse of sentence?

$$\operatorname{argmax}_t P(t|w_{1m}, G)$$

Important Questions?

Let W_{1m} be a sentence, G a grammar, t a parse tree

- What is the most likely parse of sentence?

$$\operatorname{argmax}_t P(t|w_{1m}, G)$$

- What is the probability of a sentence?

$$P(w_{1m}|G)$$

Important Questions?

Let W_{1m} be a sentence, G a grammar, t a parse tree

- What is the most likely parse of sentence?

$$\operatorname{argmax}_t P(t|w_{1m}, G)$$

- What is the probability of a sentence?

$$P(w_{1m}|G)$$

- How to learn the rule probabilities in the grammar G ?

PCFGs - Inside-outside probabilities

Pawan Goyal

CSE, IIT Kharagpur

Week 5: Lecture 4

How to find the most likely parse?: CKY for PCFG

NPTEL

How to find the most likely parse?: CKY for PCFG

a 1	pilot 2	likes 3	flying 4	planes 5

$S \rightarrow NP\ VP$ [1.0]
 $VP \rightarrow VBG\ NNS$ [0.1]
 $VP \rightarrow VBZ\ VP$ [0.1]
 $VP \rightarrow VBZ\ NP$ [0.3]
 $NP \rightarrow DT\ NN$ [0.3]
 $NP \rightarrow JJ\ NNS$ [0.4]
 $DT \rightarrow a$ [0.3]
 $NN \rightarrow pilot$ [0.1]
 $VBZ \rightarrow likes$ [0.4]
 $VBG \rightarrow flying$ [0.5]
 $JJ \rightarrow flying$ [0.1]
 $NNS \rightarrow planes$ [.34]

CKY for PCFG

a 1	pilot 2	likes 3	flying 4	planes 5
DT [0.3]	NP [.009]	-	-	S $[1.4688 \times 10^{-5}]$ S $[6.12 \times 10^{-6}]$
	NN [0.1]	-	-	-
		VBZ [0.4]	-	VP [.001632] VP [.00068]
			JJ [0.1] VBG [0.5]	NP [.0136] VP [.017]
				NNS [.34]

$S \rightarrow NP VP$ [1.0]
 $VP \rightarrow VBG NNS$ [0.1]
 $VP \rightarrow VBZ VP$ [0.1]
 $VP \rightarrow VBZ NP$ [0.3]
 $NP \rightarrow DT NN$ [0.3]
 $NP \rightarrow JJ NNS$ [0.4]
 $DT \rightarrow a$ [0.3]
 $NN \rightarrow pilot$ [0.1]
 $VBZ \rightarrow likes$ [0.4]
 $VBG \rightarrow flying$ [0.5]
 $JJ \rightarrow flying$ [0.1]
 $NNS \rightarrow planes$ [.34]

$$0.009 \times 0.00068 \times 1.0 = 6.12 \times 10^{-6}$$

Probability of a String

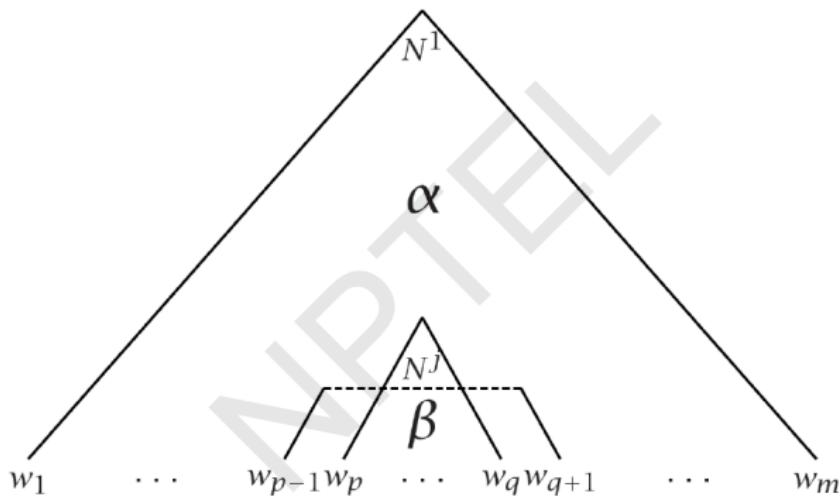
$$P(w_{1m}|G)$$

Probability of a String

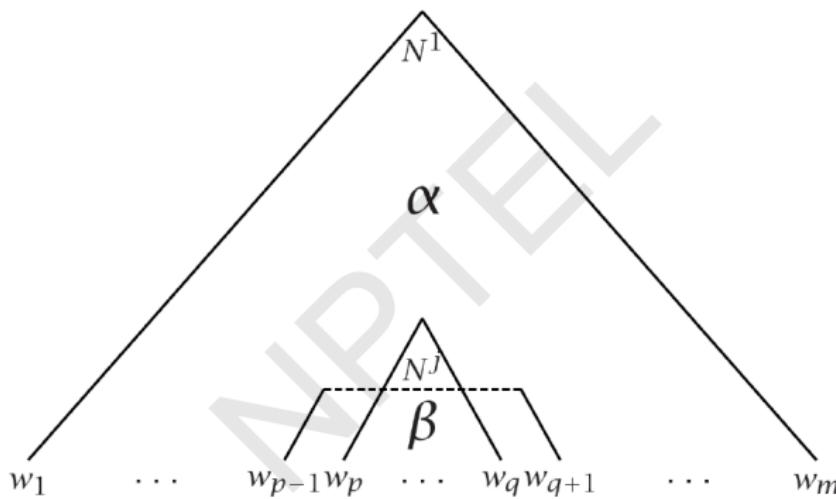
$$P(w_{1m}|G)$$

- In general, simply summing the probabilities of all possible parse trees is not an efficient way to calculate the string probability
- We use *inside algorithm*, a dynamic programming algorithm based on inside probabilities.

Inside and Outside Probabilities



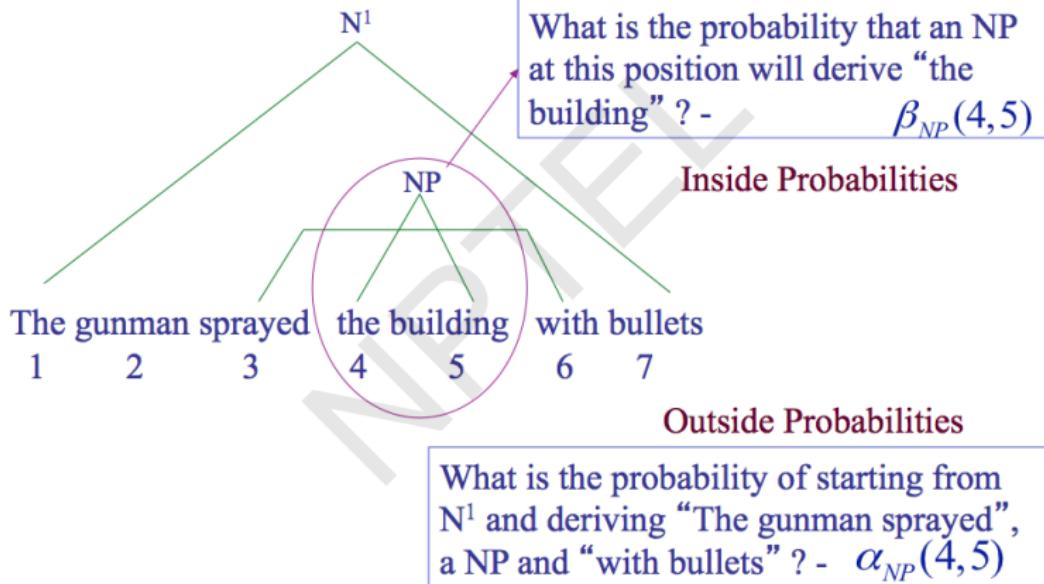
Inside and Outside Probabilities



Outside: $\alpha_j(p, q) = P(w_{1(p-1)}, N^j_{pq}, w_{(q+1)m} | G)$

Inside: $\beta_j(p, q) = P(w_{pq} | N^j_{pq}, G)$

Inside-outside probabilities

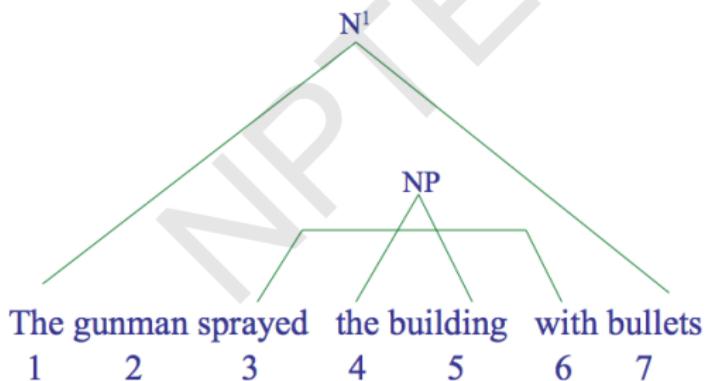


Inside-outside probabilities

$\alpha_{NP}(4,5)$ for "the building"

$= P(\text{The gunman sprayed}, NP_{4,5}, \text{with bullets} | G)$

$\beta_{NP}(4,5)$ for "the building" $= P(\text{the building} | NP_{4,5}, G)$



Inside Probabilities: Base Step

$$\beta_j(p, q) = P(w_{pq} | N_{pq}^j, G)$$

NPTEL

Inside Probabilities: Base Step

$$\beta_j(p, q) = P(w_{pq} | N^j_{pq}, G)$$

Base case

$$\begin{aligned}\beta_j(k, k) &= P(w_{kk} | N^j_{kk}, G) \\ &= P(N^j \rightarrow w_k | G)\end{aligned}$$

Base case for pre-terminals only

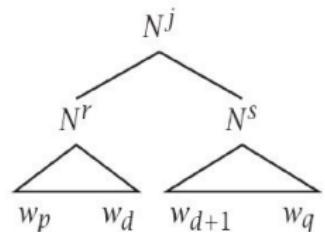
E.g., suppose $N^j = NN$ is being considered and $NN \rightarrow building$ is one of the rules with probability 0.5

$$\beta_{NN}(5, 5) = P(building | NN_{5,5}, G) = P(NN_{5,5} \rightarrow building | G)$$

Inside Probabilities: Induction Step

Assuming Chomsky Normal Form, the first rule must be of the form $N^j \rightarrow N^r N^s$

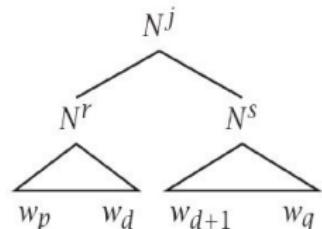
$$\beta_j(p, q) = \sum_{r,s} \sum_{d=p}^{q-1} P(N^j \rightarrow N^r N^s) \beta_r(p, d) \beta_s(d+1, q)$$



Inside Probabilities: Induction Step

Assuming Chomsky Normal Form, the first rule must be of the form $N^j \rightarrow N^r N^s$

$$\beta_j(p, q) = \sum_{r,s} \sum_{d=p}^{q-1} P(N^j \rightarrow N^r N^s) \beta_r(p, d) \beta_s(d+1, q)$$



- Consider different splits of the words - indicated by d
E.g., *the huge building*
- Consider different non-terminals to be used in the rule:
E.g., $NP \rightarrow DT\ NN$, $NP \rightarrow DT\ NNS$

Calculation of inside probabilities

astronomers	saw	stars	with	ears	

$S \rightarrow NP VP \quad 1.0$ $NP \rightarrow NP PP \quad 0.4$
 $PP \rightarrow P NP \quad 1.0$ $NP \rightarrow \text{astronomers} \quad 0.1$
 $VP \rightarrow V NP \quad 0.7$ $NP \rightarrow \text{ears} \quad 0.18$
 $VP \rightarrow VP PP \quad 0.3$ $NP \rightarrow \text{saw} \quad 0.04$
 $P \rightarrow \text{with} \quad 1.0$ $NP \rightarrow \text{stars} \quad 0.18$
 $V \rightarrow \text{saw} \quad 1.0$ $NP \rightarrow \text{telescopes} \quad 0.1$

Calculation of inside probabilities

	1	2	3	4	5
1	$\beta_{NP} = 0.1$		$\beta_S = 0.0126$		$\beta_S = 0.0015876$
2		$\beta_{NP} = 0.04$ $\beta_V = 1.0$	$\beta_{VP} = 0.126$		$\beta_{VP} = 0.015876$
3			$\beta_{NP} = 0.18$		$\beta_{NP} = 0.01296$
4				$\beta_P = 1.0$	$\beta_{PP} = 0.18$
5					$\beta_{NP} = 0.18$
	<i>astronomers</i>	<i>saw</i>	<i>stars</i>	<i>with</i>	<i>ears</i>

Outside Probabilities

Compute top-down (after inside probabilities)

NPTEL

Outside Probabilities

Compute top-down (after inside probabilities)

Base Case

Outside Probabilities

Compute top-down (after inside probabilities)

Base Case

$$\alpha_1(1, m) = 1$$

$$\alpha_j(1, m) = 0, j \neq 1$$

Outside Probabilities

Compute top-down (after inside probabilities)

Base Case

$$\alpha_1(1, m) = 1$$

$$\alpha_j(1, m) = 0, j \neq 1$$

Induction

Outside Probabilities

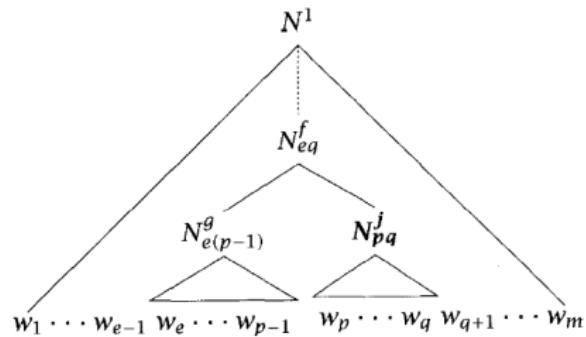
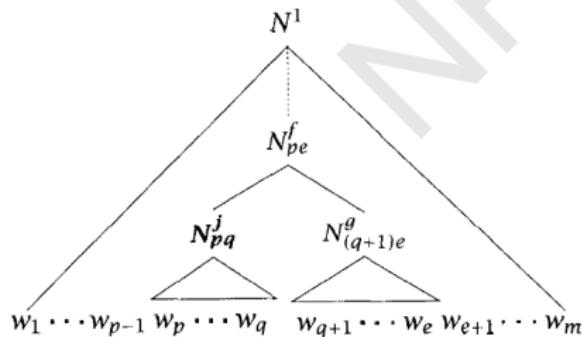
Compute top-down (after inside probabilities)

Base Case

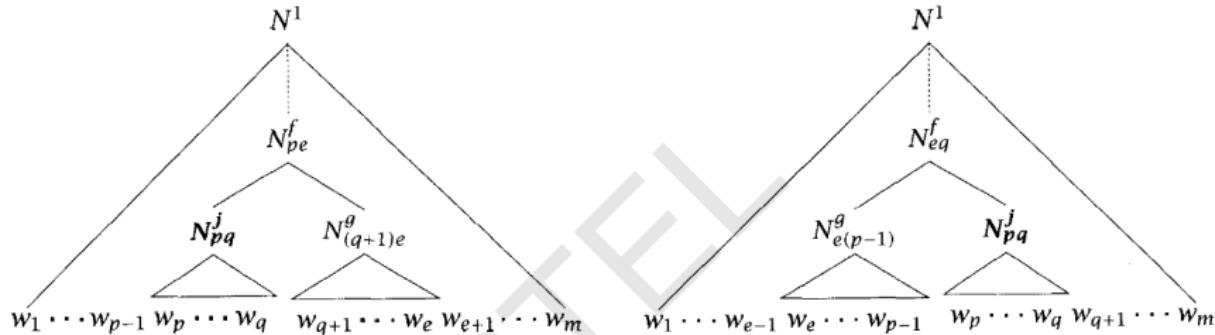
$$\alpha_1(1, m) = 1$$

$$\alpha_j(1, m) = 0, j \neq 1$$

Induction



Outside Probabilities: Induction



$$\begin{aligned}\alpha_j(p, q) &= \sum_{f,g} \sum_{e=q+1}^m \alpha_f(p, e) P(N^f \rightarrow N^j N^g) \beta_g(q+1, e) \\ &\quad + \sum_{f,g} \sum_{e=1}^{p-1} \alpha_f(e, q) P(N^f \rightarrow N^g N^j) \beta_g(e, p-1)\end{aligned}$$

Product of inside-outside probabilities

$$\alpha_j(p, q) \beta_j(p, q) = P(w_{1(p-1)}, N_{pq}^j | G) P(w_{pq} | N_{pq}^j, G) = P(w_{1m}, N_{pq}^j | G)$$

NPTEL

Product of inside-outside probabilities

$$\alpha_j(p, q)\beta_j(p, q) = P(w_{1(p-1)}, N_{pq}^j, w_{(q+1)m}|G)P(w_{pq}|N_{pq}^j, G) = P(w_{1m}, N_{pq}^j|G)$$

The probability of the sentence and that there is some consistent spanning from word p to q is given by:

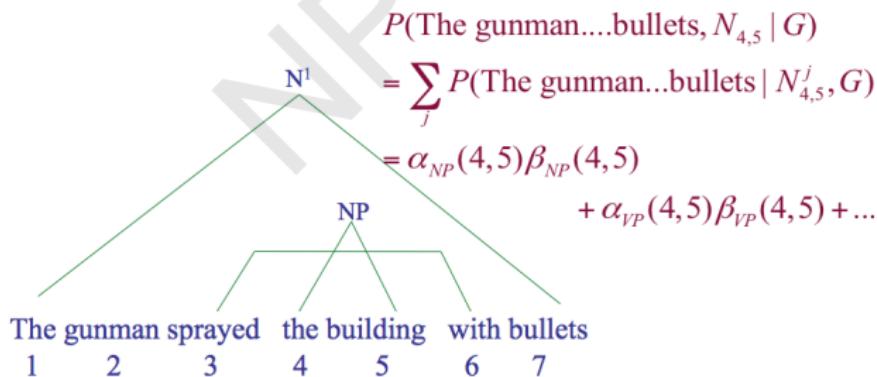
$$P(w_{1m}, N_{pq}|G) = \sum \alpha_j(p, q)\beta_j(p, q) = P(N_1 \rightarrow w_{1m}, N_{pq} \rightarrow w_{pq}|G)$$

Product of inside-outside probabilities

$$\alpha_j(p, q)\beta_j(p, q) = P(w_{1(p-1)}, N_{pq}^j, w_{(q+1)m} | G)P(w_{pq} | N_{pq}^j, G) = P(w_{1m}, N_{pq}^j | G)$$

The probability of the sentence and that there is some consistent spanning from word p to q is given by:

$$P(w_{1m}, N_{pq} | G) = \sum \alpha_j(p, q)\beta_j(p, q) = P(N_1 \rightarrow w_{1m}, N_{pq} \rightarrow w_{pq} | G)$$



Inside-outside probabilities

Pawan Goyal

CSE, IIT Kharagpur

Week 5: Lecture 5

How to get the rule probabilities

Parsed Training Data

You can count!

$$\hat{P}(N^j \rightarrow \delta) = \frac{C(N^j \rightarrow \delta)}{\sum_{\gamma} C(N^j \rightarrow \gamma)}$$

How to get the rule probabilities

Parsed Training Data

You can count!

$$\hat{P}(N^j \rightarrow \delta) = \frac{C(N^j \rightarrow \delta)}{\sum_{\gamma} C(N^j \rightarrow \gamma)}$$

But what if the training data is not available?

i.e. gold standard parse is not known.

How to get the rule probabilities

Parsed Training Data

You can count!

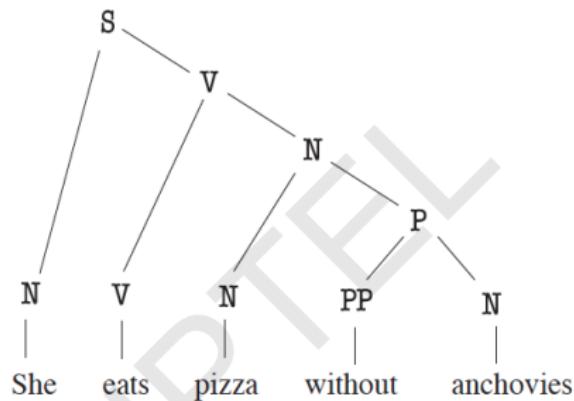
$$\hat{P}(N^j \rightarrow \delta) = \frac{C(N^j \rightarrow \delta)}{\sum_{\gamma} C(N^j \rightarrow \gamma)}$$

But what if the training data is not available?

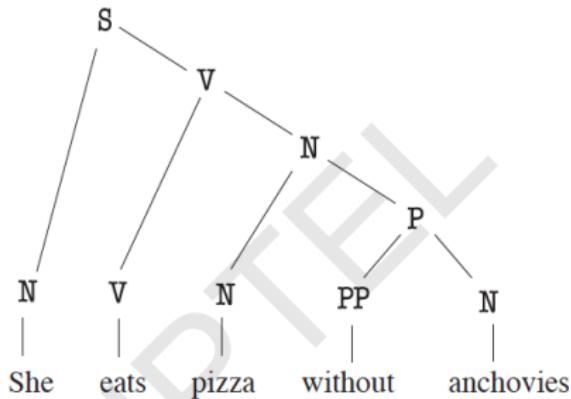
i.e. gold standard parse is not known.

- Underlying CFG is known and we are given a set of sentences
- For each sentence, we can find out all the possible parses
- *Maximize the likelihood of the sentences in the data under the PCFG constraints*

Example data



Example data



Rules of the form $A \rightarrow BC$

$$S \rightarrow NV$$

$$V \rightarrow VN$$

$$N \rightarrow NP$$

$$P \rightarrow PP\ N.$$

Rules of the form $A \rightarrow w$

$$N \rightarrow \text{She}$$

$$V \rightarrow \text{eats}$$

$$N \rightarrow \text{pizza}$$

$$PP \rightarrow \text{without}$$

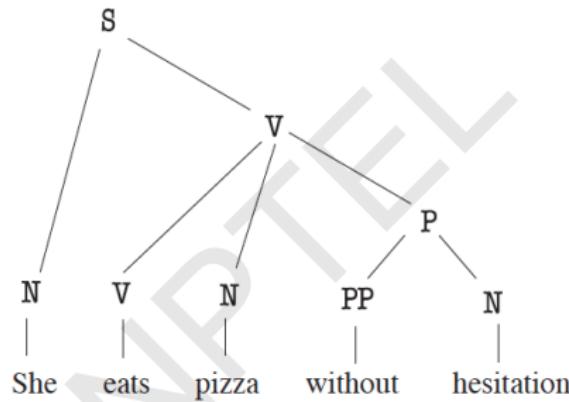
$$N \rightarrow \text{anchovies}.$$

Example data

Is any other parse possible for *She eats pizza without anchovies* syntactically?

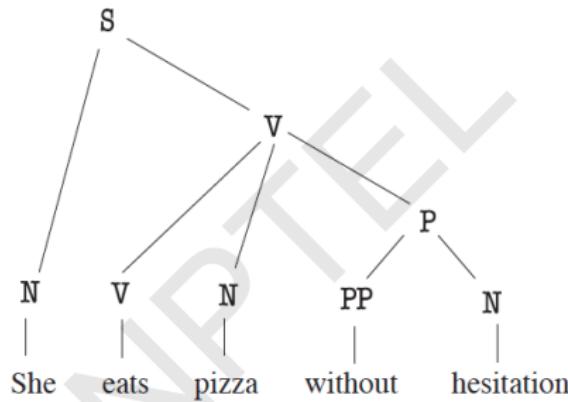
Example data

Is any other parse possible for *She eats pizza without anchovies* syntactically?
Consider *She eats pizza without hesitation*



Example data

Is any other parse possible for *She eats pizza without anchovies* syntactically?
Consider *She eats pizza without hesitation*



New Context-free rules:

$$V \rightarrow V N P$$

$$N \rightarrow \text{hesitation} .$$

Estimating the model parameters

We need to find probabilities such as

- $\phi(S \rightarrow N \ V)$
- $\phi(N \rightarrow pizza)$

Estimating the model parameters

We need to find probabilities such as

- $\phi(S \rightarrow N V)$
- $\phi(N \rightarrow pizza)$

Requirements

For each non-terminal A , the derivation probabilities sum up to 1

$$\sum_{\alpha} \phi(A \rightarrow \alpha) = 1$$

Estimating the model parameters

We need to find probabilities such as

- $\phi(S \rightarrow N V)$
- $\phi(N \rightarrow pizza)$

Requirements

For each non-terminal A , the derivation probabilities sum up to 1

$$\sum_{\alpha} \phi(A \rightarrow \alpha) = 1$$

For the example grammar:

$$\begin{aligned}\phi(N \rightarrow N P) + \phi(N \rightarrow pizza) + \phi(N \rightarrow anchovies) &+ \\ + \phi(N \rightarrow hesitation) + \phi(N \rightarrow She) &= 1 \\ \phi(V \rightarrow V N) + \phi(V \rightarrow V N P) + \phi(V \rightarrow eats) &= 1\end{aligned}$$

$$\begin{aligned}\phi(S \rightarrow N V) &= 1 \\ \phi(P \rightarrow PP N) &= 1 \\ \phi(PP \rightarrow without) &= 1\end{aligned}$$

Likelihood computation

W_1 = “She eats pizza without anchovies”

W_2 = “She eats pizza without hesitation”.

Likelihood computation

W_1 = “She eats pizza without anchovies”

W_2 = “She eats pizza without hesitation”.

$$\begin{aligned} P_\phi(W_1, T_1) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N)\ \phi(N \rightarrow N\ P)\ \times \\ &\quad \times\ \phi(P \rightarrow PP\ N)\ \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats})\ \times \\ &\quad \times\ \phi(N \rightarrow \text{pizza})\ \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{anchovies}) \end{aligned}$$

$$\begin{aligned} P_\phi(W_2, T_1) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N\ P)\ \phi(P \rightarrow P\ PP)\ \times \\ &\quad \times\ \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats})\ \phi(N \rightarrow \text{pizza})\ \times \\ &\quad \times\ \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{hesitation}) \end{aligned}$$

Likelihood computation

$$\begin{aligned} P_\phi(W_1, T_2) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N\ P)\ \phi(P \rightarrow P\ PP) \times \\ &\quad \times \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats})\ \phi(N \rightarrow \text{pizza}) \times \\ &\quad \times \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{anchovies}) \end{aligned}$$

$$\begin{aligned} P_\phi(W_2, T_1) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N)\ \phi(N \rightarrow N\ P) \times \\ &\quad \times \phi(P \rightarrow PP\ N)\ \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats}) \times \\ &\quad \times \phi(N \rightarrow \text{pizza})\ \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{hesitation}) \end{aligned}$$

Likelihood computation

$$\begin{aligned} P_\phi(W_1, T_2) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N\ P)\ \phi(P \rightarrow P\ PP) \times \\ &\quad \times \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats})\ \phi(N \rightarrow \text{pizza}) \times \\ &\quad \times \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{anchovies}) \end{aligned}$$

$$\begin{aligned} P_\phi(W_2, T_1) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N)\ \phi(N \rightarrow N\ P) \times \\ &\quad \times \phi(P \rightarrow PP\ N)\ \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats}) \times \\ &\quad \times \phi(N \rightarrow \text{pizza})\ \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{hesitation}) \end{aligned}$$

Likelihood of the corpus

Probability of a sentence W : $P_\phi(W) = \sum_T P_\phi(W, T)$

Likelihood computation

$$\begin{aligned} P_\phi(W_1, T_2) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N\ P)\ \phi(P \rightarrow P\ PP) \times \\ &\quad \times \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats})\ \phi(N \rightarrow \text{pizza}) \times \\ &\quad \times \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{anchovies}) \end{aligned}$$

$$\begin{aligned} P_\phi(W_2, T_1) &= \phi(S \rightarrow N\ V)\ \phi(V \rightarrow V\ N)\ \phi(N \rightarrow N\ P) \times \\ &\quad \times \phi(P \rightarrow PP\ N)\ \phi(N \rightarrow \text{She})\ \phi(V \rightarrow \text{eats}) \times \\ &\quad \times \phi(N \rightarrow \text{pizza})\ \phi(PP \rightarrow \text{without})\ \phi(N \rightarrow \text{hesitation}) \end{aligned}$$

Likelihood of the corpus

Probability of a sentence W : $P_\phi(W) = \sum_T P_\phi(W, T)$

If the training data comprises of sentences W_1, W_2, \dots, W_N , then the likelihood is

$$L(\phi) = P_\phi(W_1)P_\phi(W_2) \cdots P_\phi(W_N)$$

Likelihood maximization

Approach

Starting at some initial parameters ϕ , re-estimate to obtain new parameters ϕ' for which $L(\phi') \geq L(\phi)$. *Repeat until convergence*

Parameter Estimation

Given some rule probabilities ϕ and training corpus $W_1, W_2 \dots W_n$, the new parameters are obtained as:

$$\phi'(\text{A} \rightarrow \text{B C}) = \frac{\text{count}(\text{A} \rightarrow \text{B C})}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

$$\phi'(\text{A} \rightarrow w) = \frac{\text{count}(\text{A} \rightarrow w)}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

What is $\text{count}(.)$?

Parameter Estimation

Given some rule probabilities ϕ and training corpus $W_1, W_2 \dots W_n$, the new parameters are obtained as:

$$\phi'(\text{A} \rightarrow \text{B C}) = \frac{\text{count}(\text{A} \rightarrow \text{B C})}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

$$\phi'(\text{A} \rightarrow w) = \frac{\text{count}(\text{A} \rightarrow w)}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

What is $\text{count}(.)$?

$$\text{count}(\text{A} \rightarrow \text{B C}) = \sum_{i=1}^N c_{\phi}(\text{A} \rightarrow \text{B C}, W_i)$$

$$\text{count}(\text{A} \rightarrow w) = \sum_{i=1}^N c_{\phi}(\text{A} \rightarrow w, W_i)$$

Parameter Estimation

Given some rule probabilities ϕ and training corpus $W_1, W_2 \dots W_n$, the new parameters are obtained as:

$$\phi'(\text{A} \rightarrow \text{B C}) = \frac{\text{count}(\text{A} \rightarrow \text{B C})}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

$$\phi'(\text{A} \rightarrow w) = \frac{\text{count}(\text{A} \rightarrow w)}{\sum_{\alpha} \text{count}(\text{A} \rightarrow \alpha)}$$

What is $\text{count}(.)$?

$$\text{count}(\text{A} \rightarrow \text{B C}) = \sum_{i=1}^N c_{\phi}(\text{A} \rightarrow \text{B C}, W_i)$$

$$\text{count}(\text{A} \rightarrow w) = \sum_{i=1}^N c_{\phi}(\text{A} \rightarrow w, W_i)$$

$c_{\phi}(A \rightarrow \alpha, W_i)$ is the expected number of times $(A \rightarrow \alpha)$ is used in generating the sentence W_i , when the rule probabilities are given by ϕ

Computing Expected counts

Inside probabilities

The nonterminal A derives the string of words $w_i, \dots w_j$ in the sentence :

$$\beta_{ij}(A) = P_\phi(A \Rightarrow^* w_i \dots w_j)$$

Computing Expected counts

Inside probabilities

The nonterminal A derives the string of words $w_i, \dots w_j$ in the sentence :

$$\beta_{ij}(A) = P_\phi(A \Rightarrow^* w_i \dots w_j)$$

Outside probabilities

Beginning with the start symbol S we can derive the string

$$w_1 \dots w_{i-1} A w_{j+1} \dots w_n : \alpha_{ij}(A) = P_\phi(S \Rightarrow^* w_1 \dots w_{i-1} A w_{j+1} \dots w_n)$$

Computing Expected counts

Inside probabilities

The nonterminal A derives the string of words $w_i, \dots w_j$ in the sentence :

$$\beta_{ij}(A) = P_\phi(A \Rightarrow^* w_i \dots w_j)$$

Outside probabilities

Beginning with the start symbol S we can derive the string

$$w_1 \dots w_{i-1} A w_{j+1} \dots w_n : \alpha_{ij}(A) = P_\phi(S \Rightarrow^* w_1 \dots w_{i-1} A w_{j+1} \dots w_n)$$

Expected count

$$c_\phi(A \rightarrow BC, W) = \frac{\phi(A \rightarrow BC)}{P_\phi(W)} \sum_{1 \leq i \leq j \leq k \leq n} \alpha_{ik}(A) \beta_{ij}(B) \beta_{j+1,k}(C)$$

$$c_\phi(A \rightarrow w, W) = \frac{\phi(A \rightarrow w)}{P_\phi(W)} \sum_{1 \leq i \leq n} \alpha_{ii}(A)$$

And how to compute inside-outside probabilities

Inductively, as discussed earlier

$$\beta_{ii}(A) = \phi(A \rightarrow w_i)$$

$$\alpha_{1n}(S) = 1$$

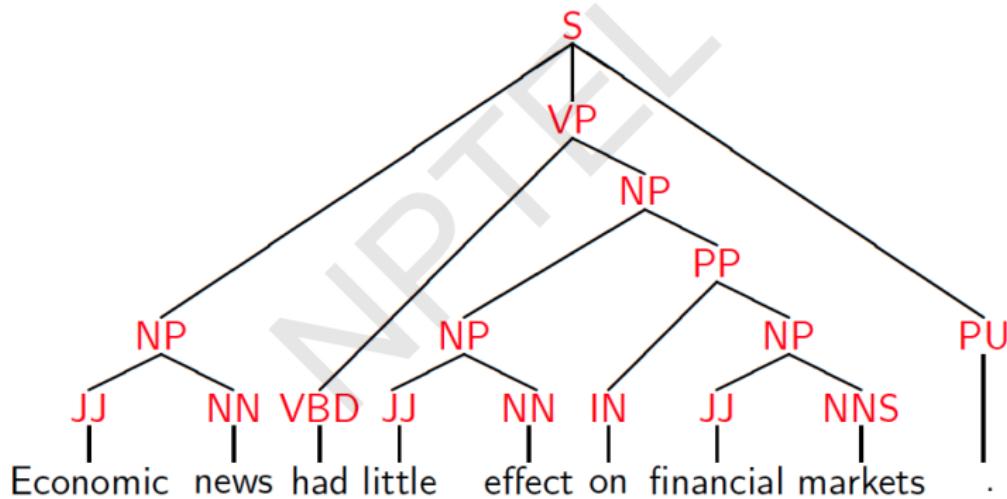
Dependency Grammars and Parsing - Introduction

Pawan Goyal

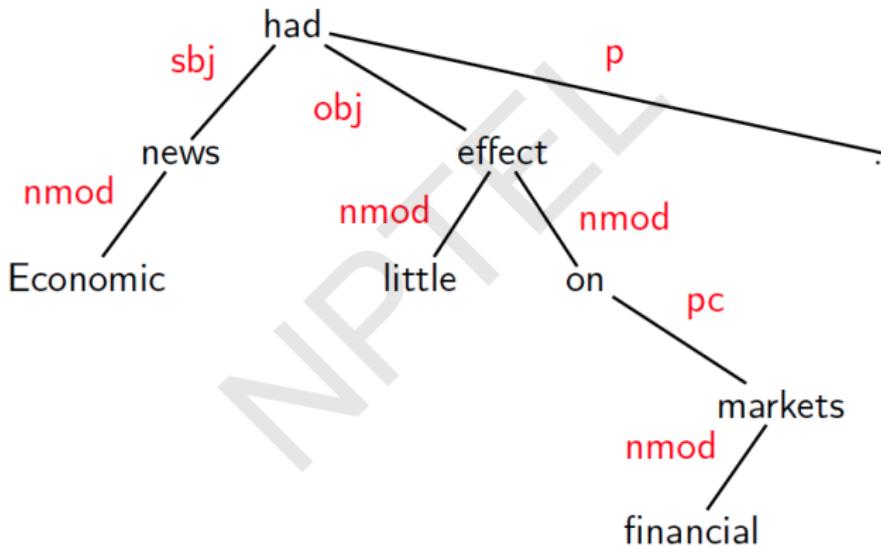
CSE, IIT Kharagpur

Week 6, Lecture 1

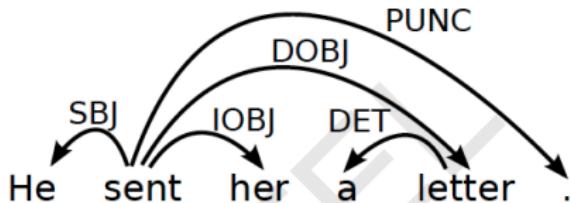
Phrase Structure



Dependency Structure Representation

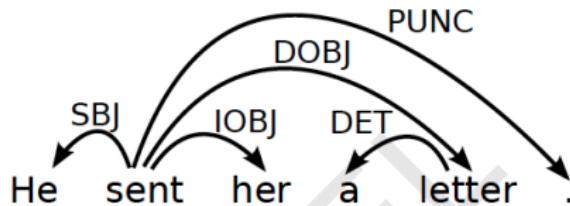


Dependency Structure



- Connects the words in a sentence by putting arrows between the words.
- Arrows show relations between the words and are typed by some grammatical relations.
- Arrows connect a head (governor, superior, regent) with a dependent (modifier, inferior, subordinate).
- Usually dependencies form a tree.

Criteria for Heads and Dependents

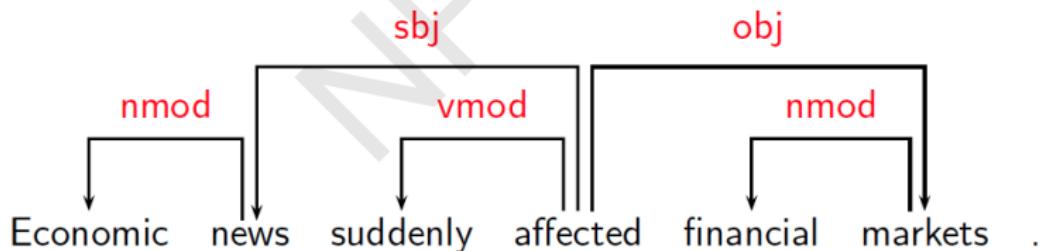


Criteria for a syntactic relation between a head H and a dependent D in a construction C

- H determines the syntactic category of C ; H can replace C .
- D specifies H .
- H is obligatory; D may be optional.
- H selects D and determines whether D is obligatory.
- The form of D depends on H (agreement or government).
- The linear position of D is specified with reference to H .

Some Clear Cases

Construction	Head	Dependent
Exocentric	Verb	Subject (sbj)
	Verb	Object (obj)
Endocentric	Verb	Adverbial (vmod)
	Noun	Attribute (nmod)



Comparison

Phrase structures explicitly represent

- Phrases (nonterminal nodes)
- Structural categories (nonterminal labels)

Comparison

Phrase structures explicitly represent

- Phrases (nonterminal nodes)
- Structural categories (nonterminal labels)

Dependency structures explicitly represent

- Head-dependent relations (directed arcs)
- Functional categories (arc labels)

Dependency Graphs

- A dependency structure can be defined as a directed graph G , consisting of
 - ▶ a set V of nodes,
 - ▶ a set A of arcs (edges),

Dependency Graphs

- A dependency structure can be defined as a directed graph G , consisting of
 - a set V of nodes,
 - a set A of arcs (edges),
- Labeled graphs:
 - Nodes in V are labeled with word forms (and annotation).
 - Arcs in A are labeled with dependency types.

Dependency Graphs

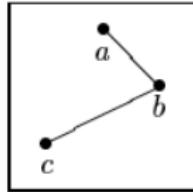
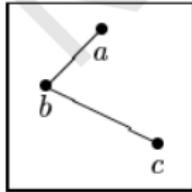
- A dependency structure can be defined as a directed graph G , consisting of
 - a set V of nodes,
 - a set A of arcs (edges),
- Labeled graphs:
 - Nodes in V are labeled with word forms (and annotation).
 - Arcs in A are labeled with dependency types.
- Notational convention:
 - Arc (w_i, d, w_j) links head w_i to dependent w_j with label d
 - $w_i \xrightarrow{d} w_j \Leftrightarrow (w_i, d, w_j) \in A$
 - $i \rightarrow j \equiv (i, j) \in A$
 - $i \rightarrow^* j \equiv i = j \vee \exists k : i \rightarrow k, k \rightarrow^* j$

Formal conditions on Dependency Graphs

- G is connected:
 - ▶ For every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$.
- G is acyclic:
 - ▶ if $i \rightarrow j$ then not $j \rightarrow^* i$.
- G obeys the single head constraint:
 - ▶ if $i \rightarrow j$ then not $k \rightarrow j$, for any $k \neq i$.
- G is projective:
 - ▶ if $i \rightarrow j$ then $j \rightarrow^* k$, for any k such that both j and k lie on the same side of i .

Formal conditions on Dependency Graphs

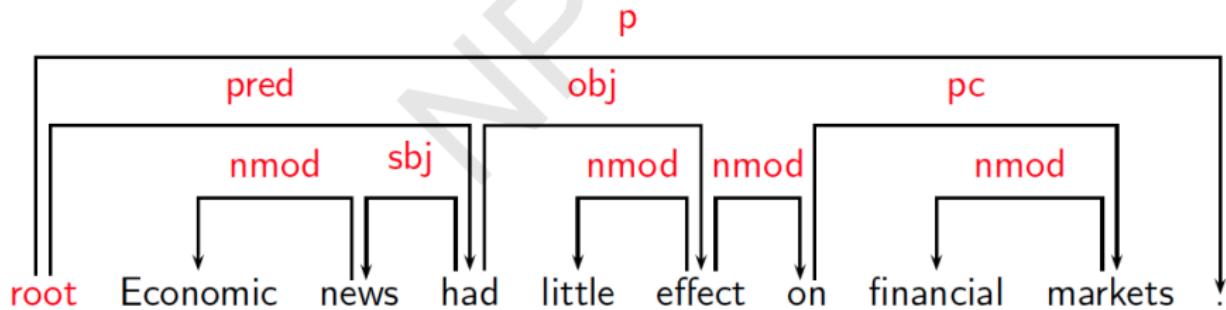
- G is connected:
 - ▶ For every node i there is a node j such that $i \rightarrow j$ or $j \rightarrow i$.
- G is acyclic:
 - ▶ if $i \rightarrow j$ then not $j \rightarrow^* i$.
- G obeys the single head constraint:
 - ▶ if $i \rightarrow j$ then not $k \rightarrow j$, for any $k \neq i$.
- G is projective:
 - ▶ if $i \rightarrow j$ then $j \rightarrow^* k$, for any k such that both j and k lie on the same side of i .



Formal Conditions: Basic Intuitions

Connectedness, Acyclicity and Single-Head

- **Connectedness:** Syntactic structure is complete.
- **Acyclicity:** Syntactic structure is hierarchical.
- **Single-Head:** Every word has at most one syntactic head.
- **Projectivity:** No crossing of dependencies.



Dependency Parsing

Dependency Parsing

- **Input:** Sentence $x = w_1, \dots, w_n$
- **Output:** Dependency graph G

Parsing Methods

- Deterministic Parsing
- Maximum Spanning Tree Based
- Constraint Propagation Based

Transition Based Parsing: Formulation

Pawan Goyal

CSE, IIT Kharagpur

Week 6, Lecture 2

Deterministic Parsing

Basic idea

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

Deterministic Parsing

Basic idea

Derive a single syntactic representation (dependency graph) through a deterministic sequence of elementary parsing actions

Configurations

A parser configuration is a triple $c = (S, B, A)$, where

- S : a stack $[\dots, w_i]_S$ of partially processed words,
- B : a buffer $[w_j, \dots]_B$ of remaining input words,
- A : a set of labeled arcs (w_i, d, w_j) .

Stack

$[\text{sent}, \text{her}, \text{a}]_S$

Buffer

$[\text{letter}, \cdot]_B$

Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$

Transition System

A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where

- C is a set of configurations,
- T is a set of transitions, such that $t : C \rightarrow C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

Transition System

A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where

- C is a set of configurations,
- T is a set of transitions, such that $t : C \rightarrow C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

A transition sequence for a sentence x is a set of configurations

$C_{0,m} = (c_0, c_1, \dots, c_m)$ such that

$c_0 = c_s(x)$, $c_m \in C_t$, $c_i = t(c_{i-1})$ for some $t \in T$

Transition System

A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where

- C is a set of configurations,
- T is a set of transitions, such that $t : C \rightarrow C$,
- c_s is an initialization function
- $C_t \subseteq C$ is a set of terminal configurations.

A transition sequence for a sentence x is a set of configurations

$C_{0,m} = (c_0, c_1, \dots, c_m)$ such that

$c_0 = c_s(x)$, $c_m \in C_t$, $c_i = t(c_{i-1})$ for some $t \in T$

Initialization: $([], [w_1, \dots, w_n]_B, \{\})$

Termination: $(S, [], A)$

Transitions for Arc-Eager Parsing

$$\text{Left-Arc}(d) \frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots]_S, [w_j, \dots]_B, A \cup \{(w_j, d, w_i)\})} \neg \text{HEAD}(w_i)$$

$$\text{Right-Arc}(d) \frac{([\dots, w_i]_S, [w_j, \dots]_B, A)}{([\dots, w_i, w_j]_S, [\dots]_B, A \cup \{(w_i, d, w_j)\})}$$

$$\text{Reduce} \frac{([\dots, w_i]_S, B, A)}{([\dots]_S, B, A)} \text{HEAD}(w_i)$$

$$\text{Shift} \frac{([\dots]_S, [w_i, \dots]_B, A)}{([\dots, w_i]_S, [\dots]_B, A)}$$

Example

Parse Example

Transitions:

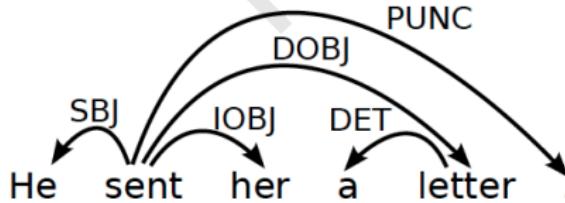
Stack

[]_s

Buffer

[He, sent, her, a, letter, .]__B

Arcs



Example

Parse Example

Transitions: SH

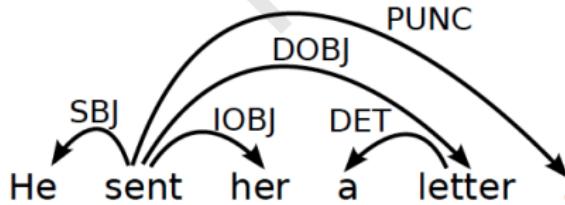
Stack

[He]_S

Buffer

[sent, her, a, letter, .]_B

Arcs



Example

Parse Example

Transitions: SH-LA

Stack

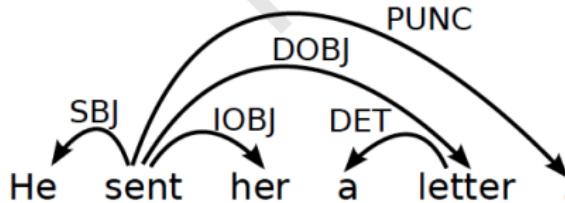
[]s

Buffer

[sent, her, a, letter, .]B

Arcs

He $\xleftarrow{\text{SBJ}}$ sent



Example

Parse Example

Transitions: SH-LA-SH

Stack

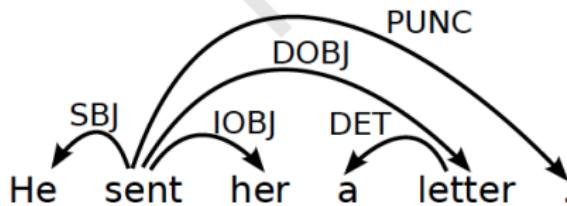
$[\text{sent}]_S$

Buffer

$[\text{her, a, letter, .}]_B$

Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$



Example

Parse Example

Transitions: SH-LA-SH-RA

Stack

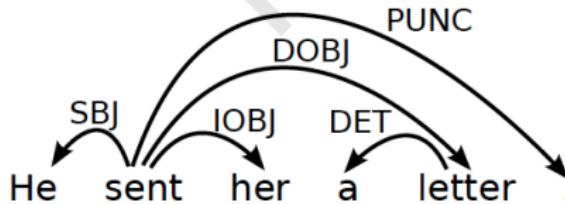
[sent, her]_S

Buffer

[a, letter, .]_B

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her



Example

Parse Example

Transitions: SH-LA-SH-RA-SH

Stack

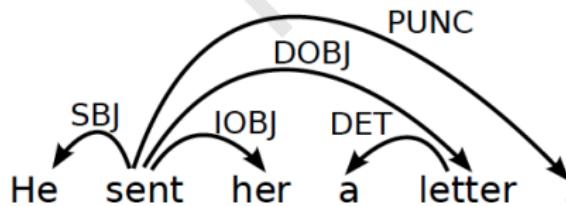
[sent, her, a]_S

Buffer

[letter, .]_B

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her



Example

Parse Example

Transitions: SH-LA-SH-RA-SH-LA

Stack

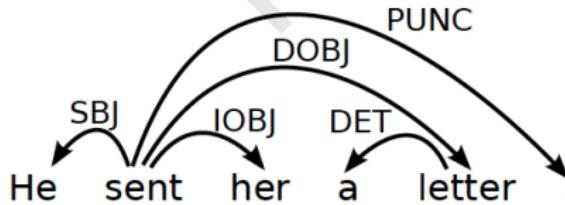
[sent, her]_S

Buffer

[letter, .]_B

Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her
a $\xleftarrow{\text{DET}}$ letter



Example

Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE

Stack

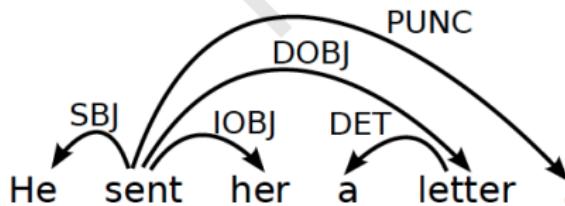
$[\text{sent}]_S$

Buffer

$[\text{letter}, .]_B$

Arcs

$\text{He} \xleftarrow{\text{SBJ}} \text{sent}$
 $\text{sent} \xrightarrow{\text{IOBJ}} \text{her}$
 $\text{a} \xleftarrow{\text{DET}} \text{letter}$



Example

Parse Example

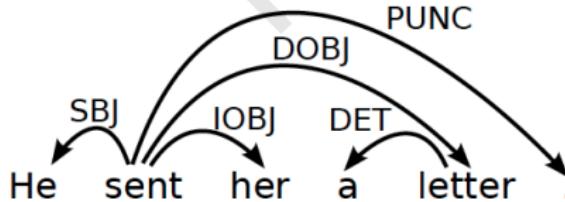
Transitions: SH-LA-SH-RA-SH-LA-RE-RA

Stack

[sent, letter]_S

Buffer

[.]_B



Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her
a $\xleftarrow{\text{DET}}$ letter
sent $\xrightarrow{\text{DOBJ}}$ letter

Example

Parse Example

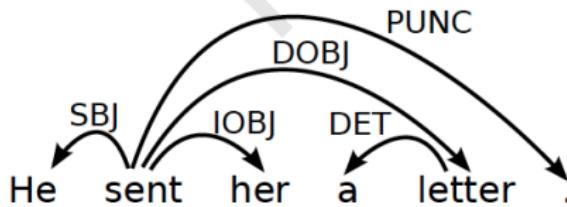
Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE

Stack

$[\text{sent}]_S$

Buffer

$[.]_B$



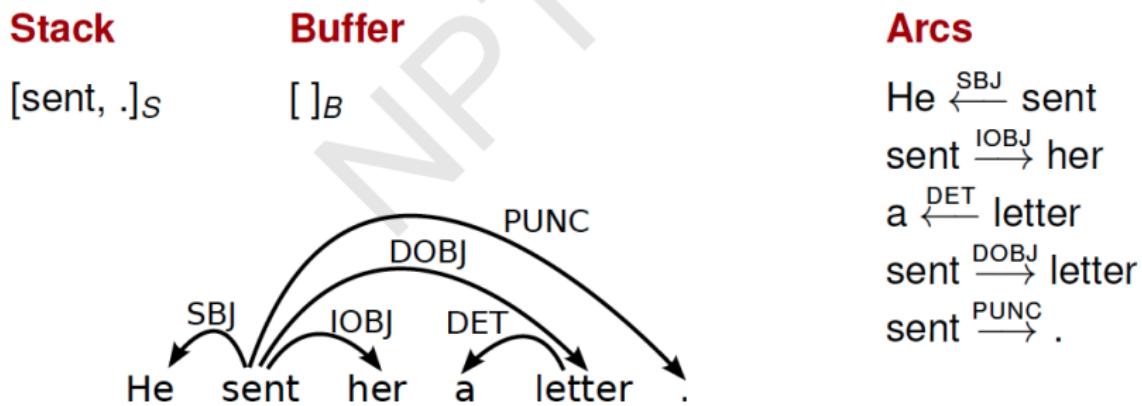
Arcs

He $\xleftarrow{\text{SBJ}}$ sent
sent $\xrightarrow{\text{IOBJ}}$ her
a $\xleftarrow{\text{DET}}$ letter
sent $\xrightarrow{\text{DOBJ}}$ letter

Example

Parse Example

Transitions: SH-LA-SH-RA-SH-LA-RE-RA-RE-RA



Transition Based Parsing: Learning

Pawan Goyal

CSE, IIT Kharagpur

Week 6, Lecture 3

Classifier-Based Parsing

Data-driven deterministic parsing:

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

Classifier-Based Parsing

Data-driven deterministic parsing:

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

Learning Problem

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

Classifier-Based Parsing

Data-driven deterministic parsing:

- Deterministic parsing requires an **oracle**.
- An oracle can be approximated by a **classifier**.
- A classifier can be trained using **treebank** data.

Learning Problem

Approximate a function from **configurations**, represented by feature vectors to **transitions**, given a training set of gold standard **transition sequences**.

Three issues

- How to represent configurations by feature vectors?
- How to derive training data from treebanks?
- How to learn classifiers?

Feature Models

A feature representation $f(c)$ of a configuration c is a vector of simple features $f_i(c)$.

Typical Features

- Nodes:

- ▶ Target nodes (top of S , head of B)
- ▶ Linear context (neighbors in S and B)
- ▶ Structural context (parents, children, siblings in G)

Feature Models

A feature representation $f(c)$ of a configuration c is a vector of simple features $f_i(c)$.

Typical Features

- Nodes:
 - ▶ Target nodes (top of S , head of B)
 - ▶ Linear context (neighbors in S and B)
 - ▶ Structural context (parents, children, siblings in G)
- Attributes:
 - ▶ Word form (and lemma)
 - ▶ Part-of-speech (and morpho-syntactic features)
 - ▶ Dependency type (if labeled)
 - ▶ Distance (between target tokens)

Deterministic Parsing

To guide the parser, a linear classifier can be used:

$$t^* = \arg \max_t w.f(c, t)$$

Weight vector w learned from treebank data.

Using classifier at run-time

- ```
PARSE(w_1, \dots, w_n)
1 $c \leftarrow ([], [w_1, \dots, w_n]_B, \{\})$
2 while $B_c \neq []$
3 $t^* \leftarrow \arg \max_t w.f(c, t)$
4 $c \leftarrow t^*(c)$
5 return $T = (\{w_1, \dots, w_n\}, A_c)$
```

# *Training data*

- Training instances have the form  $(f(c), t)$ , where
  - ▶  $f(c)$  is a feature representation of a configuration  $c$ ,
  - ▶  $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).

## Training data

- Training instances have the form  $(f(c), t)$ , where
  - ▶  $f(c)$  is a feature representation of a configuration  $c$ ,
  - ▶  $t$  is the correct transition out of  $c$  (i.e.,  $o(c) = t$ ).
- Given a dependency treebank, we can sample the oracle function  $o$  as follows:
  - ▶ For each sentence  $x$  with gold standard dependency graph  $G_x$ , construct a transition sequence  $C_{0,m} = (c_0, c_1, \dots, c_m)$  such that
$$c_0 = c_s(x), \\ G_{c_m} = G_x$$
  - ▶ For each configuration  $c_i (i < m)$ , we construct a training instance  $(f(c_i), t_i)$ , where  $t_i(c_i) = c_{i+1}$ .

# *Standard Oracle for Arc-Eager Parsing*

$o(c, T) =$

- **Left-Arc** if  $\text{top}(S_c) \leftarrow \text{first}(B_c)$  in  $T$
- **Right-Arc** if  $\text{top}(S_c) \rightarrow \text{first}(B_c)$  in  $T$
- **Reduce** if  $\exists w < \text{top}(S_c) : w \leftrightarrow \text{first}(B_c)$  in  $T$
- **Shift** otherwise

# Online Learning with an Oracle

```
LEARN({ $T_1, \dots, T_N\}$)
1 $w \leftarrow 0.0$
2 for i in $1..K$
3 for j in $1..N$
4 $c \leftarrow ([], [w_1, \dots, w_{n_j}]_B, \{\})$
5 while $B_c \neq []$
6 $t^* \leftarrow \arg \max_t w.f(c, t)$
7 $t_o \leftarrow o(c, T_i)$
8 if $t^* \neq t_o$
9 $w \leftarrow w + f(c, t_o) - f(c, t^*)$
10 $c \leftarrow t_o(c)$
11 return w
```

Oracle  $o(c, T_i)$  returns the optimal transition of  $c$  and  $T_i$

## Example

Consider the sentence, ‘John saw Mary’.

- Draw a dependency graph for this sentence.
- Assume that you are learning a classifier for the data-driven deterministic parsing and the above sentence is a gold-standard parse in your training data. You are also given that *John* and *Mary* are ‘Nouns’, while the POS tag of *saw* is ‘Verb’. Assume that your features correspond to the following conditions:
  - ▶ The stack is empty
  - ▶ Top of stack is Noun and Top of buffer is Verb
  - ▶ Top of stack is Verb and Top of buffer is Noun

Initialize the weights of all your features to 5.0, except that in all of the above cases, you give a weight of 5.5 to *Left-Arc*. Define your feature vector and the initial weight vector.

- Use this gold standard parse during online learning and report the weights after completing one full iteration of Arc-Eager parsing over this sentence.

# *MST-based Dependency Parsing*

Pawan Goyal

CSE, IIT Kharagpur

Week 6, Lecture 4

# *Maximum Spanning Tree Based*

NPTEL

# *Maximum Spanning Tree Based*

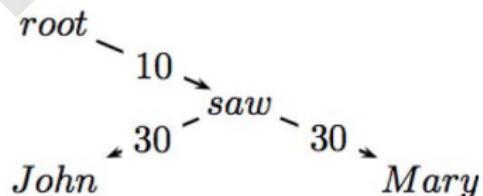
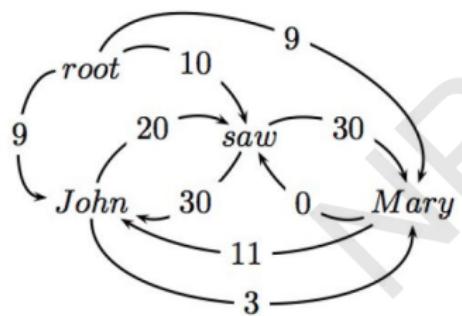
## *Basic Idea*

Starting from all possible connections, find the maximum spanning tree.

# Maximum Spanning Tree Based

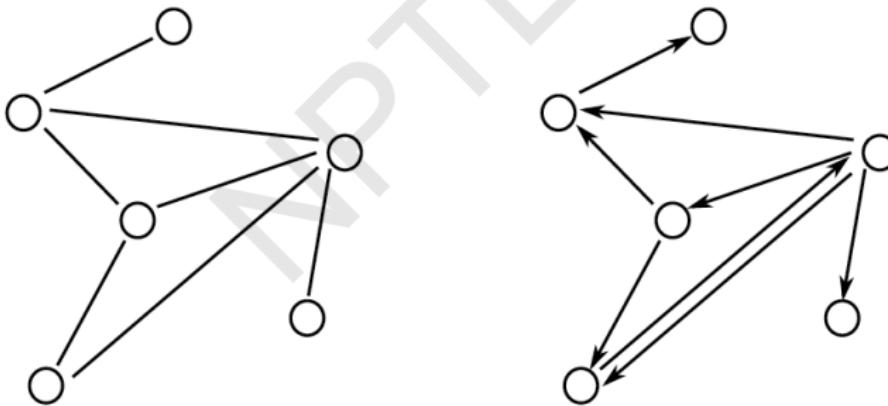
## Basic Idea

Starting from all possible connections, find the maximum spanning tree.



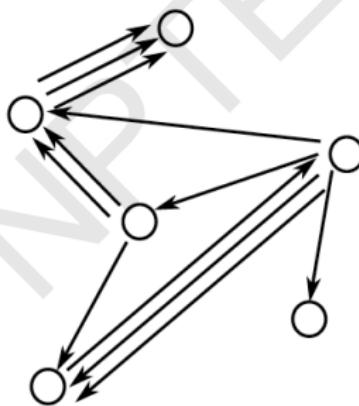
# Some Graph Theory Reminders

- A graph  $G = (V, A)$  is a set of vertices  $V$  and arcs  $(i, j) \in A$  where  $i, j \in V$ .
- Undirected graphs:  $(i, j) \in A \Leftrightarrow (j, i) \in A$
- Directed graphs (digraphs) :  $(i, j) \in A \Rightarrow (j, i) \notin A$



# Multi-Digraphs

- A multi-digraph is a digraph where multiple arcs between vertices are possible
- $(i,j,k) \in A$  represents the  $k^{th}$  arc from vertex  $i$  to vertex  $j$ .

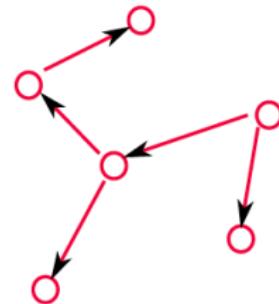
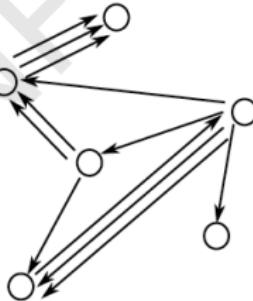
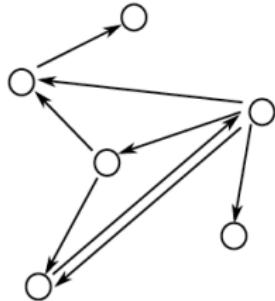


# *Directed Spanning Trees*

- A directed spanning tree of a (multi-)digraph  $G = (V, A)$  is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)

# Directed Spanning Trees

- A directed spanning tree of a (multi-)digraph  $G = (V, A)$  is a subgraph  $G' = (V', A')$  such that:
  - ▶  $V' = V$
  - ▶  $A' \subseteq A$ , and  $|A'| = |V'| - 1$
  - ▶  $G'$  is a tree (acyclic)
- A spanning tree of the following (multi-)digraphs



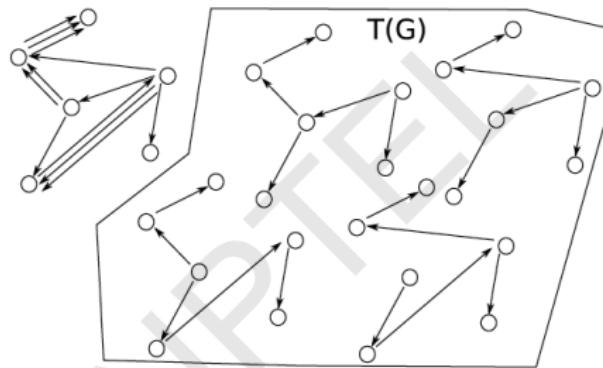
# *Weighted Directed Spanning Trees*

- Assume we have a weight function for each arc in a multi-digraph  $G = (V, A)$ .
- Define  $w_{ij}^k \geq 0$  to be the weight of  $(i, j, k) \in A$  for a multi-digraph
- Define the weight of directed spanning tree  $G'$  of graph  $G$  as

$$w(G') = \sum_{(i,j,k) \in G'} w_{ij}^k$$

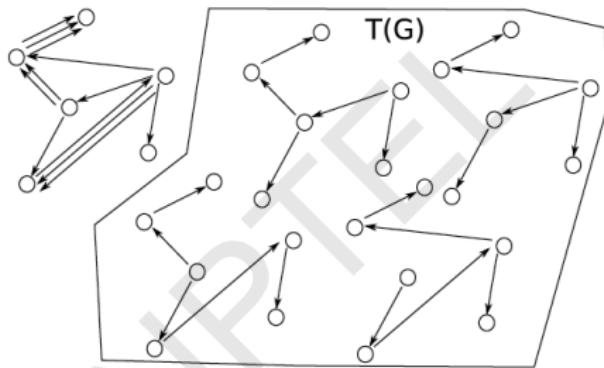
# Maximum Spanning Trees (MST)

Let  $T(G)$  be the set of all spanning trees for graph  $G$



# Maximum Spanning Trees (MST)

Let  $T(G)$  be the set of all spanning trees for graph  $G$



## The MST problem

Find the spanning tree  $G'$  of the graph  $G$  that has the highest weight

$$G' = \arg \max_{G' \in T(G)} w(G') = \arg \max_{G' \in T(G)} \sum_{(i,j,k) \in G'} w_{ij}^k$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

# Finding MST

## Directed Graph

For each sentence  $x$ , define the directed graph  $G_x = (V_x, E_x)$  given by

$$V_x = \{x_0 = \text{root}, x_1, \dots, x_n\}$$

$$E_x = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\}$$

$G_x$  is a graph with

- the sentence words and the dummy root symbol as vertices and
- a directed edge between every pair of distinct words and
- a directed edge from the root symbol to every word

# *Chu-Liu-Edmonds Algorithm*

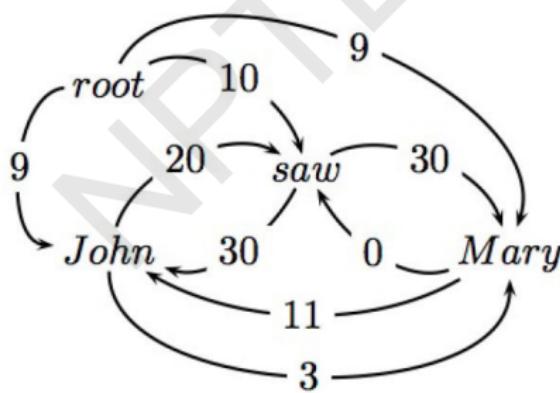
## *Chu-Liu-Edmonds Algorithm*

- Each vertex in the graph greedily selects the incoming edge with the highest weight.
- If a tree results, it must be a maximum spanning tree.
- If not, there must be a cycle.
  - ▶ Identify the cycle and contract it into a single vertex.
  - ▶ Recalculate edge weights going into and out of the cycle.

# *Chu-Liu-Edmonds Algorithm*

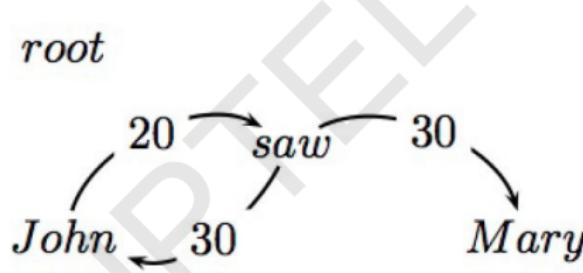
$x = \text{John saw Mary}$

- Build the directed graph



## *Chu-Liu-Edmonds Algorithm*

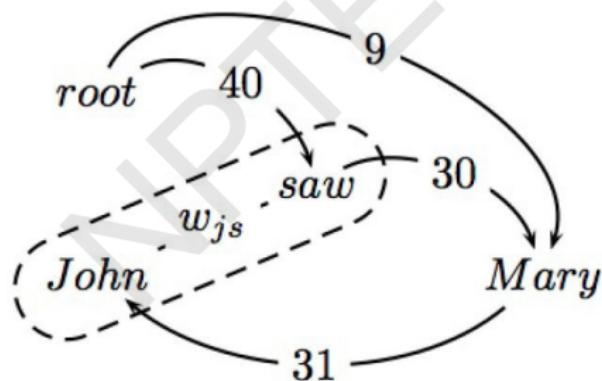
- Find the highest scoring incoming arc for each vertex



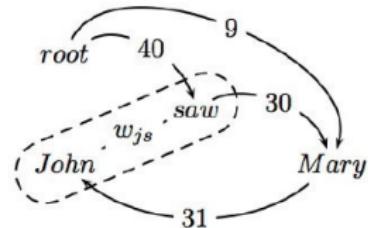
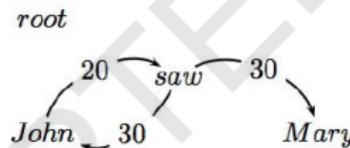
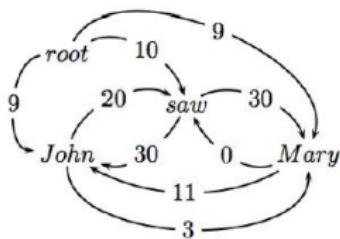
- If this is a tree, then we have found MST.

# *Chu-Liu-Edmonds Algorithm*

- If not a tree, identify cycle and contract
- Recalculate arc weights into and out-of cycle



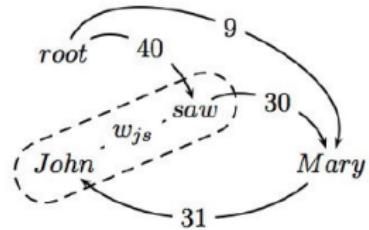
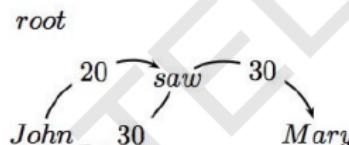
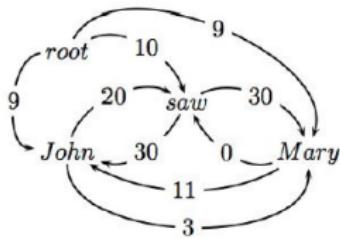
# Chu-Liu-Edmonds Algorithm



## Outgoing arc weights

- Equal to the max of outgoing arc over all vertices in cycle
- e.g., John → Mary is 3 and saw → Mary is 30.

# Chu-Liu-Edmonds Algorithm

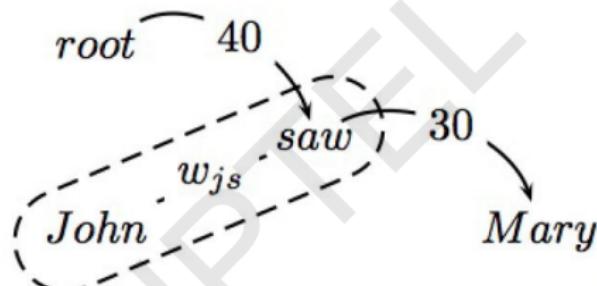


## Incoming arc weights

- Equal to the weight of best spanning tree that includes head of incoming arc and all nodes in cycle
- $\text{root} \rightarrow \text{saw} \rightarrow \text{John}$  is 40
- $\text{root} \rightarrow \text{John} \rightarrow \text{saw}$  is 29

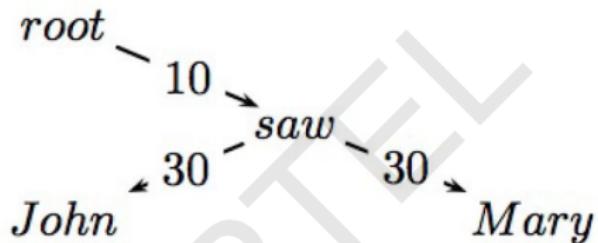
# *Chu-Liu-Edmonds Algorithm*

Calling the algorithm again on the contracted graph:



- This is a tree and the MST for the contracted graph
- Go back up the recursive call and reconstruct final graph

# *Chu-Liu-Edmonds Algorithm*



- The edge from  $w_{js}$  to *Mary* was from *saw*
- The edge from *root* to  $w_{js}$  represented a tree from *root* to *saw* to *John*.

# *MST-based Dependency Parsing: Learning*

Pawan Goyal

CSE, IIT Kharagpur

Week 6, Lecture 5

# *Arc weights as linear classifiers*

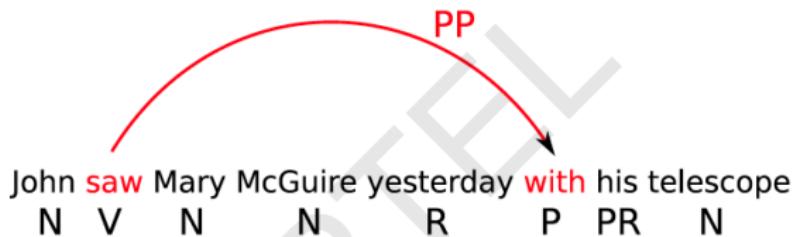
$$w_{ij}^k = w.f(i, j, k)$$

# *Arc weights as linear classifiers*

$$w_{ij}^k = w.f(i,j,k)$$

- Arc weights are a linear combination of features of the arc  $f(i,j,k)$  and a corresponding weight vector  $w$
- What arc features?

# Arc Features $f(i,j,k)$

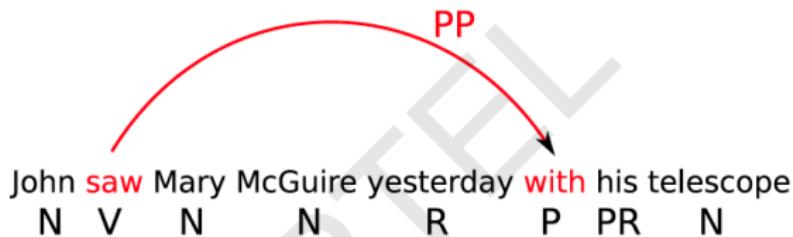


## Features

Identities of the words  $w_i$  and  $w_j$  for a label  $l_k$

head = saw & dependent=with

# Arc Features $f(i,j,k)$

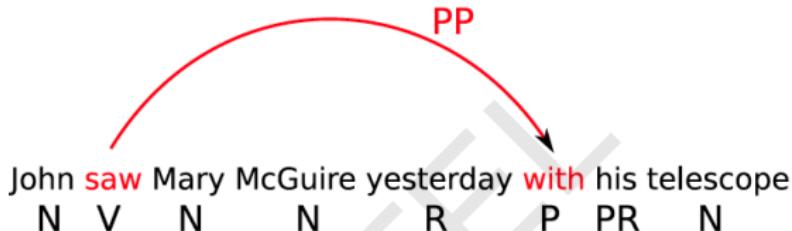


## Features

Part-of-speech tags of the words  $w_i$  and  $w_j$  for a label  $l_k$

head-pos = Verb & dependent-pos=Preposition

# Arc Features $f(i,j,k)$



## Features

Part-of-speech of words surrounding and between  $w_i$  and  $w_j$

inbetween-pos = Noun

inbetween-pos = Adverb

dependent-pos-right = Pronoun

head-pos-left=Noun

# Arc Features $f(i,j,k)$



## Features

Number of words between  $w_i$  and  $w_j$ , and their orientation

arc-distance = 3  
arc-direction = right

# Arc Features $f(i,j,k)$



## Features

- Combinations
  - head-pos=Verb & dependent-pos=Preposition & arc-label=PP
  - head-pos=Verb & dependent=with & arc-distance=3
- No limit : any feature over arc  $(i,j,k)$  or input  $x$

# Learning the parameters

- Re-write the inference problem

$$\begin{aligned} G &= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} {w_{ij}}^k \\ &= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) \\ &= \arg \max_{G \in T(G_x)} w \cdot f(G) \end{aligned}$$

# Learning the parameters

- Re-write the inference problem

$$\begin{aligned} G &= \arg \max_{G \in T(G_x)} \sum_{(i,j,k) \in G} {w_{ij}}^k \\ &= \arg \max_{G \in T(G_x)} w \cdot \sum_{(i,j,k) \in G} f(i,j,k) \\ &= \arg \max_{G \in T(G_x)} w \cdot f(G) \end{aligned}$$

- Which can be plugged into learning algorithms

# Inference-based Learning

Training data:  $T = \{(x_t, G_t)\}_{t=1}^{|T|}$

1.  $w^{(0)} = 0; i = 0$
2. for  $n : 1..N$
3. for  $t : 1..|T|$
4.     Let  $G' = \operatorname{argmax}_{G'} w^{(i)}.f(G')$
5.     if  $G' \neq G_t$
6.          $w^{(i+1)} = w^{(i)} + f(G_t) - f(G')$
7.          $i = i + 1$
8. return  $w^i$

## Example

Suppose you are training MST Parser for dependency and the sentence, “John saw Mary” occurs in the training set. Also, for simplicity, assume that there is only one dependency relation, “rel”. Thus, for every arc from word  $w_i$  to  $w_j$ , your features may be simplified to depend only on words  $w_i$  and  $w_j$  and not on the relation label.

Below is the set of features

- $f_1: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_2: \text{pos}(w_i) = \text{Verb}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_3: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Verb}$
- $f_4: w_i = \text{Root}$  and  $\text{pos}(w_j) = \text{Noun}$
- $f_5: w_i = \text{Root}$  and  $w_j$  occurs at the end of sentence
- $f_6: w_i$  occurs before  $w_j$  in the sentence
- $f_7: \text{pos}(w_i) = \text{Noun}$  and  $\text{pos}(w_j) = \text{Verb}$

The feature weights before the start of the iteration are: {3, 20, 15, 12, 1, 10, 20}. Determine the weights after an iteration over this example.

# *Distributional Semantics - Introduction*

Pawan Goyal

CSE, IIT Kharagpur

Week 7, Lecture 1

# *Introduction*

*What is Semantics?*

# *Introduction*

*What is Semantics?*

**The study of meaning:** Relation between symbols and their denotata.

# *Introduction*

*What is Semantics?*

**The study of meaning:** Relation between symbols and their denotata.  
John told Mary that the train moved out of the station at 3 o'clock.

# *Computational Semantics*

NPTEL

# *Computational Semantics*

## *Computational Semantics*

The study of how to automate the process of constructing and reasoning with meaning representations of natural language expressions.

# *Computational Semantics*

## *Computational Semantics*

The study of how to automate the process of constructing and reasoning with meaning representations of natural language expressions.

*Methods in Computational Semantics generally fall in two categories:*

- **Formal Semantics:** Construction of precise mathematical models of the relations between expressions in a natural language and the world.

# *Computational Semantics*

## *Computational Semantics*

The study of how to automate the process of constructing and reasoning with meaning representations of natural language expressions.

*Methods in Computational Semantics generally fall in two categories:*

- **Formal Semantics:** Construction of precise mathematical models of the relations between expressions in a natural language and the world.  
*John chases a bat*  $\rightarrow \exists x[bat(x) \wedge chase(john, x)]$

# *Computational Semantics*

## *Computational Semantics*

The study of how to automate the process of constructing and reasoning with meaning representations of natural language expressions.

*Methods in Computational Semantics generally fall in two categories:*

- **Formal Semantics:** Construction of precise mathematical models of the relations between expressions in a natural language and the world.  
*John chases a bat* →  $\exists x[bat(x) \wedge chase(john, x)]$
- **Distributional Semantics:** The study of statistical patterns of human word usage to extract semantics.

# *Distributional Hypothesis*

NPTEL

# *Distributional Hypothesis*

## *Distributional Hypothesis: Basic Intuition*

*“The meaning of a word is its use in language.” (Wittgenstein, 1953)*

*“You know a word by the company it keeps.” (Firth, 1957)*

# *Distributional Hypothesis*

## *Distributional Hypothesis: Basic Intuition*

*“The meaning of a word is its use in language.” (Wittgenstein, 1953)*

*“You know a word by the company it keeps.” (Firth, 1957)*

→ Word meaning (whatever it might be) is reflected in linguistic distributions.

# *Distributional Hypothesis*

## *Distributional Hypothesis: Basic Intuition*

*“The meaning of a word is its use in language.” (Wittgenstein, 1953)*

*“You know a word by the company it keeps.” (Firth, 1957)*

→ Word meaning (whatever it might be) is reflected in linguistic distributions.

*“Words that occur in the same contexts tend to have similar meanings.” (Zellig Harris, 1968)*

# *Distributional Hypothesis*

## *Distributional Hypothesis: Basic Intuition*

*“The meaning of a word is its use in language.” (Wittgenstein, 1953)*

*“You know a word by the company it keeps.” (Firth, 1957)*

→ Word meaning (whatever it might be) is reflected in linguistic distributions.

*“Words that occur in the same contexts tend to have similar meanings.” (Zellig Harris, 1968)*

→ Semantically similar words tend to have similar distributional patterns.

# *Distributional Semantics: a linguistic perspective*

*"If linguistics is to deal with meaning, it can only do so through distributional analysis." (Zellig Harris)*

# *Distributional Semantics: a linguistic perspective*

*“If linguistics is to deal with meaning, it can only do so through distributional analysis.” (Zellig Harris)*

*“If we consider words or morphemes A and B to be more different in meaning than A and C, then we will often find that the distributions of A and B are more different than the distributions of A and C. In other words, difference in meaning correlates with difference of distribution.” (Zellig Harris, “Distributional Structure”)*

# *Distributional Semantics: a linguistic perspective*

*“If linguistics is to deal with meaning, it can only do so through distributional analysis.” (Zellig Harris)*

*“If we consider words or morphemes A and B to be more different in meaning than A and C, then we will often find that the distributions of A and B are more different than the distributions of A and C. In other words, difference in meaning correlates with difference of distribution.” (Zellig Harris, “Distributional Structure”)*

**Differential and not referential**

# *Distributional Semantics: a cognitive perspective*

## *Contextual representation*

A word's contextual representation is an abstract cognitive structure that accumulates from encounters with the word in various linguistic contexts.

# *Distributional Semantics: a cognitive perspective*

## *Contextual representation*

A word's contextual representation is an abstract cognitive structure that accumulates from encounters with the word in various linguistic contexts.

## *We learn new words based on contextual cues*

# *Distributional Semantics: a cognitive perspective*

## *Contextual representation*

A word's contextual representation is an abstract cognitive structure that accumulates from encounters with the word in various linguistic contexts.

## *We learn new words based on contextual cues*

He filled the **wampimuk** with the substance, passed it around and we all drunk some.

# *Distributional Semantics: a cognitive perspective*

## *Contextual representation*

A word's contextual representation is an abstract cognitive structure that accumulates from encounters with the word in various linguistic contexts.

## *We learn new words based on contextual cues*

He filled the **wampimuk** with the substance, passed it around and we all drunk some.

We found a little **wampimuk** sleeping behind the tree.

# *Distributional Semantic Models (DSMs)*

- Computational models that build contextual semantic representations from corpus data

# *Distributional Semantic Models (DSMs)*

- Computational models that build contextual semantic representations from corpus data
- DSMs are models for semantic representations
  - ▶ The semantic content is represented by a vector
  - ▶ Vectors are obtained through the statistical analysis of the linguistic contexts of a word

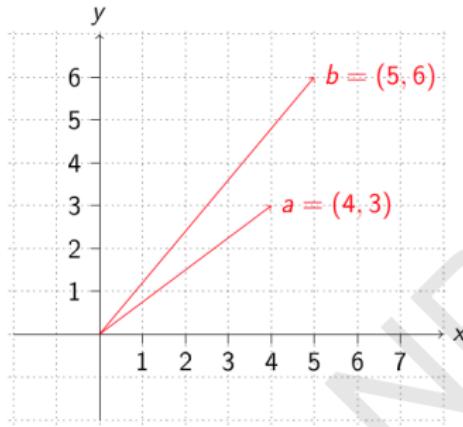
# *Distributional Semantic Models (DSMs)*

- Computational models that build contextual semantic representations from corpus data
- DSMs are models for semantic representations
  - ▶ The semantic content is represented by a vector
  - ▶ Vectors are obtained through the statistical analysis of the linguistic contexts of a word
- Alternative names
  - ▶ corpus-based semantics
  - ▶ statistical semantics
  - ▶ geometrical models of meaning
  - ▶ vector semantics
  - ▶ word space models

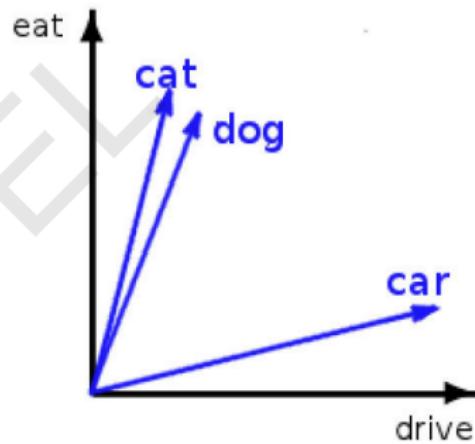
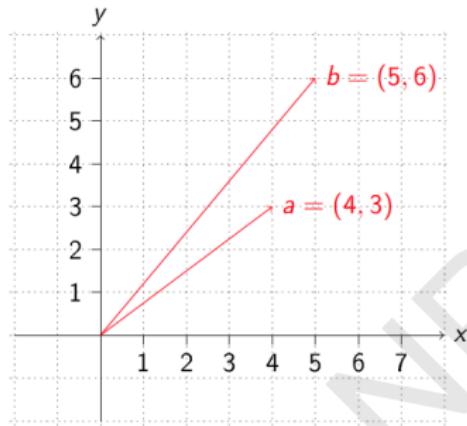
# *Distributional Semantics: The general intuition*

- **Distributions** are vectors in a multidimensional semantic space, that is, objects with a magnitude and a direction.
- The **semantic space** has dimensions which correspond to possible contexts, as gathered from a given corpus.

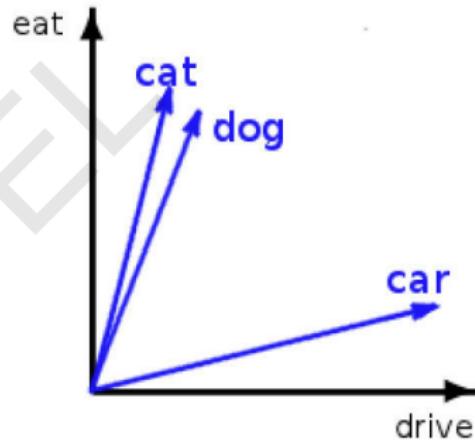
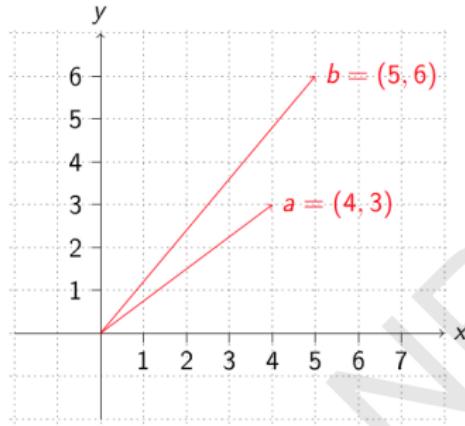
# Vector Space



# Vector Space



# Vector Space



In practice, many more dimensions are used.

$cat = [\dots \text{dog } 0.8, \text{ eat } 0.7, \text{ joke } 0.01, \text{ mansion } 0.2, \dots]$

# Word Space

## Small Dataset

*An automobile is a wheeled motor vehicle used for transporting passengers .*

*A car is a form of transport , usually with four wheels and the capacity to carry around five passengers .*

*Transport for the London games is limited , with spectators strongly advised to avoid the use of cars .*

*The London 2012 soccer tournament began yesterday , with plenty of goals in the opening matches .*

*Giggs scored the first goal of the football tournament at Wembley , North London .*

*Bellamy was largely a passenger in the football match , playing no part in either goal .*

*Target words: <automobile, car, soccer, football>*

*Term vocabulary: <wheel, transport, passenger, tournament, London, goal, match>*

# *Constructing Word spaces*

Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**

# *Constructing Word spaces*

Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**
- Define a **context window**, number of words surrounding target word

# *Constructing Word spaces*

Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**
- Define a **context window**, number of words surrounding target word
  - ▶ The context can in general be defined in terms of documents, paragraphs or sentences.

# *Constructing Word spaces*

Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**
- Define a **context window**, number of words surrounding target word
  - ▶ The context can in general be defined in terms of documents, paragraphs or sentences.
- Count number of times the target word co-occurs with the context words:  
**co-occurrence matrix**

# *Constructing Word spaces*

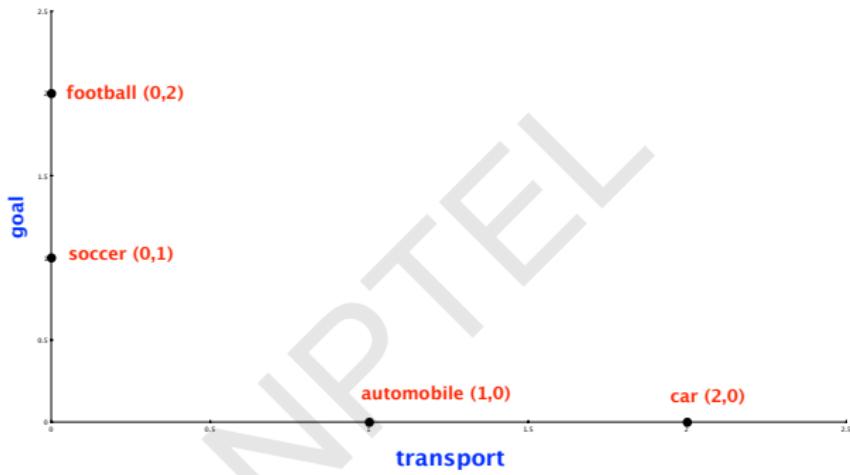
Informal algorithm for constructing word spaces

- Pick the words you are interested in: **target words**
- Define a **context window**, number of words surrounding target word
  - ▶ The context can in general be defined in terms of documents, paragraphs or sentences.
- Count number of times the target word co-occurs with the context words: **co-occurrence matrix**
- Build vectors out of (a function of) these co-occurrence counts

# Constructing Word spaces: distributional vectors

distributional matrix = targets X contexts

|            | wheel | transport | passenger | tournament | London | goal | match |
|------------|-------|-----------|-----------|------------|--------|------|-------|
| automobile | 1     | 1         | 1         | 0          | 0      | 0    | 0     |
| car        | 1     | 2         | 1         | 0          | 1      | 0    | 0     |
| soccer     | 0     | 0         | 0         | 1          | 1      | 1    | 1     |
| football   | 0     | 0         | 1         | 1          | 1      | 2    | 1     |



# Computing similarity

|            | wheel | transport | passenger | tournament | London | goal | match |
|------------|-------|-----------|-----------|------------|--------|------|-------|
| automobile | 1     | 1         | 1         | 0          | 0      | 0    | 0     |
| car        | 1     | 2         | 1         | 0          | 1      | 0    | 0     |
| soccer     | 0     | 0         | 0         | 1          | 1      | 1    | 1     |
| football   | 0     | 0         | 1         | 1          | 1      | 2    | 1     |

## Using simple vector product

$$\text{automobile} \cdot \text{car} = 4$$

$$\text{automobile} \cdot \text{soccer} = 0$$

$$\text{automobile} \cdot \text{football} = 1$$

$$\text{car} \cdot \text{soccer} = 1$$

$$\text{car} \cdot \text{football} = 2$$

$$\text{soccer} \cdot \text{football} = 5$$

# *Distributional Models of Semantics*

Pawan Goyal

CSE, IIT Kharagpur

Week 7, Lecture 2

# *Vector Space Model without distributional similarity*

Words are treated as atomic symbols

# Vector Space Model without distributional similarity

Words are treated as atomic symbols

## One-hot representation

motel [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND  
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0] = 0

# *Distributional Similarity Based Representations*

*You know a word by the company it keeps*

# *Distributional Similarity Based Representations*

*You know a word by the company it keeps*

government debt problems turning into banking crises as has happened in  
saying that Europe needs unified banking regulation to replace the hodgepodge

# *Distributional Similarity Based Representations*

*You know a word by the company it keeps*

government debt problems turning into banking crises as has happened in  
saying that Europe needs unified banking regulation to replace the hodgepodge

*These words will represent banking*

# *Building a DSM step-by-step*

## *The “linguistic” steps*

Pre-process a corpus (to define targets and contexts)



Select the targets and the contexts

# *Building a DSM step-by-step*

## *The “linguistic” steps*

Pre-process a corpus (to define targets and contexts)



Select the targets and the contexts

## *The “mathematical” steps*

Count the target-context co-occurrences



Weight the contexts (optional)



Build the distributional matrix



Reduce the matrix dimensions (optional)



Compute the vector distances on the (reduced) matrix

# *Many design choices*

| Matrix type             | Weighting             | Dimensionality reduction | Vector comparison |
|-------------------------|-----------------------|--------------------------|-------------------|
| word × document         | probabilities         | LSA                      | Euclidean         |
| word × word             | length normalization  | PLSA                     | Cosine            |
| word × search proximity | TF-IDF                | X LDA                    | X Dice            |
| adj. × modified noun    | PMI                   | PCA                      | Jaccard           |
| word × dependency rel.  | Positive PMI          | IS                       | KL                |
| verb × arguments        | PPMI with discounting | DCA                      | KL with skew      |
| ⋮                       | ⋮                     | ⋮                        | ⋮                 |

# *Many design choices*

| Matrix type             | Weighting             | Dimensionality reduction | Vector comparison |
|-------------------------|-----------------------|--------------------------|-------------------|
| word × document         | probabilities         | LSA                      | Euclidean         |
| word × word             | length normalization  | PLSA                     | Cosine            |
| word × search proximity | TF-IDF                | X LDA                    | X Dice            |
| adj. × modified noun    | PMI                   | PCA                      | Jaccard           |
| word × dependency rel.  | Positive PMI          | IS                       | KL                |
| verb × arguments        | PPMI with discounting | DCA                      | KL with skew      |
| ⋮                       | ⋮                     | ⋮                        | ⋮                 |

## *General Questions*

- How do the rows (words, ...) relate to each other?
- How do the columns (contexts, documents, ...) relate to each other?

# *The parameter space*

*A number of parameters to be fixed*

- Which type of context?
- Which weighting scheme?
- Which similarity measure?
- ...

A specific parameter setting determines a particular type of DSM (e.g. LSA, HAL, etc.)

## *Documents as context: Word × document*

|         | d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8 | d9 | d10 |
|---------|----|----|----|----|----|----|----|----|----|-----|
| against | 0  | 0  | 0  | 1  | 0  | 0  | 3  | 2  | 3  | 0   |
| age     | 0  | 0  | 0  | 1  | 0  | 3  | 1  | 0  | 4  | 0   |
| agent   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| ages    | 0  | 0  | 0  | 0  | 0  | 2  | 0  | 0  | 0  | 0   |
| ago     | 0  | 0  | 0  | 2  | 0  | 0  | 0  | 0  | 3  | 0   |
| agree   | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| ahead   | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0   |
| ain't   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| air     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| aka     | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0   |

# *Words as context: Word × Word*

|         | against | age  | agent | ages | ago  | agree | ahead | ain.t | air | aka | al  |
|---------|---------|------|-------|------|------|-------|-------|-------|-----|-----|-----|
| against | 2003    | 90   | 39    | 20   | 88   | 57    | 33    | 15    | 58  | 22  | 24  |
| age     | 90      | 1492 | 14    | 39   | 71   | 38    | 12    | 4     | 18  | 4   | 39  |
| agent   | 39      | 14   | 507   | 2    | 21   | 5     | 10    | 3     | 9   | 8   | 25  |
| ages    | 20      | 39   | 2     | 290  | 32   | 5     | 4     | 3     | 6   | 1   | 6   |
| ago     | 88      | 71   | 21    | 32   | 1164 | 37    | 25    | 11    | 34  | 11  | 38  |
| agree   | 57      | 38   | 5     | 5    | 37   | 627   | 12    | 2     | 16  | 19  | 14  |
| ahead   | 33      | 12   | 10    | 4    | 25   | 12    | 429   | 4     | 12  | 10  | 7   |
| ain't   | 15      | 4    | 3     | 3    | 11   | 2     | 4     | 166   | 0   | 3   | 3   |
| air     | 58      | 18   | 9     | 6    | 34   | 16    | 12    | 0     | 746 | 5   | 11  |
| aka     | 22      | 4    | 8     | 1    | 11   | 19    | 10    | 3     | 5   | 261 | 9   |
| al      | 24      | 39   | 25    | 6    | 38   | 14    | 7     | 3     | 11  | 9   | 861 |

# *Words as contexts*

## *Parameters*

- Window size
- Window shape - rectangular/triangular/other

# *Words as contexts*

## *Parameters*

- Window size
- Window shape - rectangular/triangular/other

## *Consider the following passage*

*Suspected communist rebels on 4 July 1989 killed Col. Herminio Taylo, police chief of Makati, the Philippines major financial center, in an escalation of street violence sweeping the Capitol area. The gunmen shouted references to the rebel New People's Army. They fled in a commandeered passenger jeep. The military says communist rebels have killed up to 65 soldiers and police in the Capitol region since January.*

# *Words as contexts*

## *Parameters*

- Window size
- Window shape - rectangular/triangular/other

*5 words window (unfiltered): 2 words either side of the target word*

*Suspected communist rebels on 4 July 1989 killed Col. Herminio Taylo, police chief of Makati, the Philippines major financial center, in an escalation of street violence sweeping the Capitol area. The gunmen shouted references to the rebel New People's Army. They fled in a commandeered passenger jeep. The military says communist rebels have killed up to 65 soldiers and police in the Capitol region since January.*

# *Words as contexts*

## *Parameters*

- Window size
- Window shape - rectangular/triangular/other

*5 words window (filtered): 2 words either side of the target word*

*Suspected communist rebels on 4 July 1989 killed Col. Herminio Taylo, police chief of Makati, the Philippines major financial center, in an escalation of street violence sweeping the Capitol area. The gunmen shouted references to the rebel New People's Army. They fled in a commandeered passenger jeep. The military says communist rebels have killed up to 65 soldiers and police in the Capitol region since January.*

# *Context weighting: documents as context*

NPTEL

# *Context weighting: documents as context*

## *Indexing function F: Essential factors*

- **Word frequency ( $f_{ij}$ ):** How many times a word appears in the document?

$$F \propto f_{ij}$$

- **Document length ( $|D_i|$ ):** How many words appear in the document?

$$F \propto \frac{1}{|D_i|}$$

- **Document frequency ( $N_j$ ):** Number of documents in which a word

appears.  $F \propto \frac{1}{N_j}$

# Context weighting: documents as context

## Indexing function $F$ : Essential factors

- **Word frequency ( $f_{ij}$ ):** How many times a word appears in the document?

$$F \propto f_{ij}$$

- **Document length ( $|D_i|$ ):** How many words appear in the document?

$$F \propto \frac{1}{|D_i|}$$

- **Document frequency ( $N_j$ ):** Number of documents in which a word appears.  $F \propto \frac{1}{N_j}$

## Indexing Weight: $tf\text{-}Idf$

- $f_{ij} * \log(\frac{N}{N_j})$  for each term, normalize the weight in a document with respect to  $L_2$ -norm.

# *Context weighting: words as context*

NPTEL

# *Context weighting: words as context*

## *basic intuition*

| word1 | word2        | freq(1,2) | freq(1) | freq(2) |
|-------|--------------|-----------|---------|---------|
| dog   | small        | 855       | 33,338  | 490,580 |
| dog   | domesticated | 29        | 33,338  | 918     |

# *Context weighting: words as context*

## *basic intuition*

| word1 | word2        | freq(1,2) | freq(1) | freq(2) |
|-------|--------------|-----------|---------|---------|
| dog   | small        | 855       | 33,338  | 490,580 |
| dog   | domesticated | 29        | 33,338  | 918     |

**Association measures** are used to give more weight to contexts that are more significantly associated with a target word.

# *Context weighting: words as context*

## *basic intuition*

| word1 | word2        | freq(1,2) | freq(1) | freq(2) |
|-------|--------------|-----------|---------|---------|
| dog   | small        | 855       | 33,338  | 490,580 |
| dog   | domesticated | 29        | 33,338  | 918     |

**Association measures** are used to give more weight to contexts that are more significantly associated with a target word.

- The less frequent the target and context element are, the higher the weight given to their co-occurrence count should be.

# Context weighting: words as context

## basic intuition

| word1 | word2        | freq(1,2) | freq(1) | freq(2) |
|-------|--------------|-----------|---------|---------|
| dog   | small        | 855       | 33,338  | 490,580 |
| dog   | domesticated | 29        | 33,338  | 918     |

**Association measures** are used to give more weight to contexts that are more significantly associated with a target word.

- The less frequent the target and context element are, the higher the weight given to their co-occurrence count should be.  
⇒ Co-occurrence with frequent context element *small* is less informative than co-occurrence with rarer *domesticated*.

# Context weighting: words as context

## basic intuition

| word1 | word2        | freq(1,2) | freq(1) | freq(2) |
|-------|--------------|-----------|---------|---------|
| dog   | small        | 855       | 33,338  | 490,580 |
| dog   | domesticated | 29        | 33,338  | 918     |

**Association measures** are used to give more weight to contexts that are more significantly associated with a target word.

- The less frequent the target and context element are, the higher the weight given to their co-occurrence count should be.  
⇒ Co-occurrence with frequent context element *small* is less informative than co-occurrence with rarer *domesticated*.
- different measures - e.g., Mutual information, Log-likelihood ratio

# *Pointwise Mutual Information (PMI)*

$$PMI(w_1, w_2) = \log_2 \frac{P_{corpus}(w_1, w_2)}{P_{ind}(w_1, w_2)}$$

# *Pointwise Mutual Information (PMI)*

$$PMI(w_1, w_2) = \log_2 \frac{P_{corpus}(w_1, w_2)}{P_{ind}(w_1, w_2)}$$

$$PMI(w_1, w_2) = \log_2 \frac{P_{corpus}(w_1, w_2)}{P_{corpus}(w_1)P_{corpus}(w_2)}$$

# *Pointwise Mutual Information (PMI)*

$$PMI(w_1, w_2) = \log_2 \frac{P_{corpus}(w_1, w_2)}{P_{ind}(w_1, w_2)}$$

$$PMI(w_1, w_2) = \log_2 \frac{P_{corpus}(w_1, w_2)}{P_{corpus}(w_1)P_{corpus}(w_2)}$$

$$P_{corpus}(w_1, w_2) = \frac{freq(w_1, w_2)}{N}$$

$$P_{corpus}(w) = \frac{freq(w)}{N}$$

# *PMI: Issues and Variations*

## *Positive PMI*

All PMI values less than zero are replaced with zero.

# *PMI: Issues and Variations*

## *Positive PMI*

All PMI values less than zero are replaced with zero.

## *Bias towards infrequent events*

Consider  $w_j$  having the maximum association with  $w_i$ ,  
 $P_{corpus}(w_i) \approx P_{corpus}(w_j) \approx P_{corpus}(w_i, w_j)$

# *PMI: Issues and Variations*

## *Positive PMI*

All PMI values less than zero are replaced with zero.

## *Bias towards infrequent events*

Consider  $w_j$  having the maximum association with  $w_i$ ,

$$P_{corpus}(w_i) \approx P_{corpus}(w_j) \approx P_{corpus}(w_i, w_j)$$

PMI increases as the probability of  $w_i$  decreases.

## *Positive PMI*

All PMI values less than zero are replaced with zero.

## *Bias towards infrequent events*

Consider  $w_j$  having the maximum association with  $w_i$ ,

$$P_{\text{corpus}}(w_i) \approx P_{\text{corpus}}(w_j) \approx P_{\text{corpus}}(w_i, w_j)$$

PMI increases as the probability of  $w_i$  decreases.

Also, consider a word  $w_j$  that occurs once in the corpus, also in the context of  $w_i$ .

# *PMI: Issues and Variations*

## *Positive PMI*

All PMI values less than zero are replaced with zero.

## *Bias towards infrequent events*

Consider  $w_j$  having the maximum association with  $w_i$ ,

$$P_{corpus}(w_i) \approx P_{corpus}(w_j) \approx P_{corpus}(w_i, w_j)$$

PMI increases as the probability of  $w_i$  decreases.

Also, consider a word  $w_j$  that occurs once in the corpus, also in the context of  $w_i$ . A discounting factor proposed by Pantel and Lin:

$$\delta_{ij} = \frac{f_{ij}}{f_{ij} + 1} \frac{\min(f_i, f_j)}{\min(f_i, f_j) + 1}$$

$$PMI_{new}(w_i, w_j) = \delta_{ij} PMI(w_i, w_j)$$

# Distributional Vectors: Example

## Normalized Distributional Vectors using Pointwise Mutual Information

|                  |                                                                                                                                                                       |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>petroleum</b> | oil:0.032 gas:0.029 crude:0.029 barrels:0.028 exploration:0.027 barrel:0.026 opec:0.026 refining:0.026 gasoline:0.026 fuel:0.025 natural:0.025 exporting:0.025        |
| <b>drug</b>      | trafficking:0.029 cocaine:0.028 narcotics:0.027 fda:0.026 police:0.026 abuse:0.026 marijuana:0.025 crime:0.025 colombian:0.025 arrested:0.025 addicts:0.024           |
| <b>insurance</b> | insurers:0.028 premiums:0.028 lloyds:0.026 reinsurance:0.026 underwriting:0.025 pension:0.025 mortgage:0.025 credit:0.025 investors:0.024 claims:0.024 benefits:0.024 |
| <b>forest</b>    | timber:0.028 trees:0.027 land:0.027 forestry:0.026 environmental:0.026 species:0.026 wildlife:0.026 habitat:0.025 tree:0.025 mountain:0.025 river:0.025 lake:0.025    |
| <b>robotics</b>  | robots:0.032 automation:0.029 technology:0.028 engineering:0.026 systems:0.026 sensors:0.025 welding:0.025 computer:0.025 manufacturing:0.025 automated:0.025         |

# *Distributional Semantics: Applications, Structured Models*

Pawan Goyal

CSE, IIT Kharagpur

Week 7, Lecture 3

# *Application to Query Expansion: Addressing Term Mismatch*

## *Term Mismatch Problem in Information Retrieval*

- Stems from the word independence assumption during document indexing.
- User query: *insurance cover which pays for long term care*.
- A relevant document may contain terms different from the actual user query.
- Some relevant words concerning this query:  $\{medicare, premiums, insurers\}$

# *Application to Query Expansion: Addressing Term Mismatch*

## *Term Mismatch Problem in Information Retrieval*

- Stems from the word independence assumption during document indexing.
- User query: *insurance cover which pays for long term care*.
- A relevant document may contain terms different from the actual user query.
- Some relevant words concerning this query:  $\{medicare, premiums, insurers\}$

## *Using DSMs for Query Expansion*

Given a user query, reformulate it using related terms to enhance the retrieval performance.

- The distributional vectors for the query terms are computed.
- Expanded query is obtained by a linear combination or a functional combination of these vectors.

# *Query Expansion using Unstructured DSMs*

## *TREC Topic 104: catastrophic health insurance*

**Query Representation:** surtax:1.0 hcfa:0.97 medicare:0.93 hmos:0.83  
medicaid:0.8 hmo:0.78 beneficiaries:0.75 ambulatory:0.72 premiums:0.72  
hospitalization:0.71 hhs:0.7 reimbursable:0.7 deductible:0.69

# Query Expansion using Unstructured DSMs

## TREC Topic 104: catastrophic health insurance

**Query Representation:** surtax:1.0 hcfa:0.97 medicare:0.93 hmos:0.83  
medicaid:0.8 hmo:0.78 beneficiaries:0.75 ambulatory:0.72 premiums:0.72  
hospitalization:0.71 hhs:0.7 reimbursable:0.7 deductible:0.69

- Broad expansion terms: **medicare, beneficiaries, premiums ...**
- Specific domain terms: **HCFA** (Health Care Financing Administration), **HMO** (Health Maintenance Organization), **HHS** (Health and Human Services)

# *Query Expansion using Unstructured DSMs*

## *TREC Topic 104: catastrophic health insurance*

**Query Representation:** surtax:1.0 hcfa:0.97 medicare:0.93 hmos:0.83  
medicaid:0.8 hmo:0.78 beneficiaries:0.75 ambulatory:0.72 premiums:0.72  
hospitalization:0.71 hhs:0.7 reimbursable:0.7 deductible:0.69

- Broad expansion terms: **medicare, beneficiaries, premiums** ...
- Specific domain terms: **HCFA** (Health Care Financing Administration), **HMO** (Health Maintenance Organization), **HHS** (Health and Human Services)

## *TREC Topic 355: ocean remote sensing*

**Query Representation:** radiometer:1.0 landsat:0.97 ionosphere:0.94  
cnes:0.84 altimeter:0.83 nasda:0.81 meterology:0.81 cartography:0.78  
geostationary:0.78 doppler:0.78 oceanographic:0.76

# *Query Expansion using Unstructured DSMs*

## *TREC Topic 104: catastrophic health insurance*

**Query Representation:** surtax:1.0 hcfa:0.97 medicare:0.93 hmos:0.83  
medicaid:0.8 hmo:0.78 beneficiaries:0.75 ambulatory:0.72 premiums:0.72  
hospitalization:0.71 hhs:0.7 reimbursable:0.7 deductible:0.69

- Broad expansion terms: **medicare, beneficiaries, premiums** ...
- Specific domain terms: **HCFA** (Health Care Financing Administration), **HMO** (Health Maintenance Organization), **HHS** (Health and Human Services)

## *TREC Topic 355: ocean remote sensing*

**Query Representation:** radiometer:1.0 landsat:0.97 ionosphere:0.94  
cnes:0.84 altimeter:0.83 nasda:0.81 meterology:0.81 cartography:0.78  
geostationary:0.78 doppler:0.78 oceanographic:0.76

- Broad expansion terms: **radiometer, landsat, ionosphere** ...
- Specific domain terms: **CNES** (Centre National d'Études Spatiales) and **NASDA** (National Space Development Agency of Japan)

# *Similarity Measures for Binary Vectors*

Let  $X$  and  $Y$  denote the binary distributional vectors for words  $X$  and  $Y$ .

## *Similarity Measures*

Dice coefficient :  $\frac{2|X \cap Y|}{|X| + |Y|}$

# *Similarity Measures for Binary Vectors*

Let  $X$  and  $Y$  denote the binary distributional vectors for words  $X$  and  $Y$ .

## *Similarity Measures*

$$\text{Dice coefficient : } \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\text{Jaccard Coefficient : } \frac{|X \cap Y|}{|X \cup Y|}$$

# *Similarity Measures for Binary Vectors*

Let  $X$  and  $Y$  denote the binary distributional vectors for words  $X$  and  $Y$ .

## *Similarity Measures*

$$\text{Dice coefficient : } \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\text{Jaccard Coefficient : } \frac{|X \cap Y|}{|X \cup Y|}$$

$$\text{Overlap Coefficient : } \frac{|X \cap Y|}{\min(|X|, |Y|)}$$

# Similarity Measures for Binary Vectors

Let  $X$  and  $Y$  denote the binary distributional vectors for words  $X$  and  $Y$ .

## Similarity Measures

$$\text{Dice coefficient : } \frac{2|X \cap Y|}{|X| + |Y|}$$

$$\text{Jaccard Coefficient : } \frac{|X \cap Y|}{|X \cup Y|}$$

$$\text{Overlap Coefficient : } \frac{|X \cap Y|}{\min(|X|, |Y|)}$$

Jaccard coefficient penalizes small number of shared entries, while Overlap coefficient uses the concept of inclusion.

# *Similarity Measures for Vector Spaces*

Let  $\vec{X}$  and  $\vec{Y}$  denote the distributional vectors for words  $X$  and  $Y$ .

$$\vec{X} = [x_1, x_2, \dots, x_n], \vec{Y} = [y_1, y_2, \dots, y_n]$$

# *Similarity Measures for Vector Spaces*

Let  $\vec{X}$  and  $\vec{Y}$  denote the distributional vectors for words  $X$  and  $Y$ .

$$\vec{X} = [x_1, x_2, \dots, x_n], \vec{Y} = [y_1, y_2, \dots, y_n]$$

## *Similarity Measures*

$$\text{Cosine similarity : } \cos(\vec{X}, \vec{Y}) = \frac{\vec{X} \cdot \vec{Y}}{|\vec{X}| |\vec{Y}|}$$

# Similarity Measures for Vector Spaces

Let  $\vec{X}$  and  $\vec{Y}$  denote the distributional vectors for words  $X$  and  $Y$ .

$$\vec{X} = [x_1, x_2, \dots, x_n], \vec{Y} = [y_1, y_2, \dots, y_n]$$

## Similarity Measures

$$\text{Cosine similarity : } \cos(\vec{X}, \vec{Y}) = \frac{\vec{X} \cdot \vec{Y}}{|\vec{X}| |\vec{Y}|}$$

$$\text{Euclidean distance : } |\vec{X} - \vec{Y}| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

# *Similarity Measure for Probability Distributions*

Let  $p$  and  $q$  denote the probability distributions corresponding to two distributional vectors.

# Similarity Measure for Probability Distributions

Let  $p$  and  $q$  denote the probability distributions corresponding to two distributional vectors.

## Similarity Measures

$$\text{KL-divergence} : D(p||q) = \sum_i p_i \log \frac{p_i}{q_i}$$

$$\text{Information Radius} : D(p||\frac{p+q}{2}) + D(q||\frac{p+q}{2})$$

$$L_1\text{-norm} : \sum_i |p_i - q_i|$$

# *Attributional Similarity vs. Relational Similarity*

## *Attributional Similarity*

The attributional similarity between two words  $a$  and  $b$  depends on the degree of correspondence between the properties of  $a$  and  $b$ .

*Ex: dog and wolf*

## *Relational Similarity*

Two pairs  $(a, b)$  and  $(c, d)$  are relationally similar if they have many similar relations.

*Ex: dog: bark and cat: meow*

# *Relational Similarity: Pair-pattern matrix*

## *Pair-pattern matrix*

- Row vectors correspond to pairs of words, such as *mason: stone* and *carpenter: wood*
- Column vectors correspond to the patterns in which the pairs occur, e.g. *X cuts Y* and *X works with Y*
- Compute the similarity of rows to find similar pairs

# *Relational Similarity: Pair-pattern matrix*

## *Pair-pattern matrix*

- Row vectors correspond to pairs of words, such as *mason: stone* and *carpenter: wood*
- Column vectors correspond to the patterns in which the pairs occur, e.g. *X cuts Y* and *X works with Y*
- Compute the similarity of rows to find similar pairs

## *Extended Distributional Hypothesis; Lin and Pantel*

Patterns that co-occur with similar pairs tend to have similar meanings.

# *Relational Similarity: Pair-pattern matrix*

## *Pair-pattern matrix*

- Row vectors correspond to pairs of words, such as *mason: stone* and *carpenter: wood*
- Column vectors correspond to the patterns in which the pairs occur, e.g. *X cuts Y* and *X works with Y*
- Compute the similarity of rows to find similar pairs

## *Extended Distributional Hypothesis; Lin and Pantel*

Patterns that co-occur with similar pairs tend to have similar meanings.

This matrix can also be used to measure the semantic similarity of patterns.

# *Relational Similarity: Pair-pattern matrix*

## *Pair-pattern matrix*

- Row vectors correspond to pairs of words, such as *mason: stone* and *carpenter: wood*
- Column vectors correspond to the patterns in which the pairs occur, e.g. *X cuts Y* and *X works with Y*
- Compute the similarity of rows to find similar pairs

## *Extended Distributional Hypothesis; Lin and Pantel*

Patterns that co-occur with similar pairs tend to have similar meanings.

This matrix can also be used to measure the semantic similarity of patterns.

Given a pattern such as “X solves Y”, you can use this matrix to find similar patterns, such as “Y is solved by X”, “Y is resolved in X”, “X resolves Y”.

## Basic Issue

- Words may not be the basic context units anymore
- How to capture and represent syntactic information?  
*X solves Y and Y is solved by X*

# Structured DSMs

## Basic Issue

- Words may not be the basic context units anymore
- How to capture and represent syntactic information?  
*X solves Y and Y is solved by X*

## An Ideal Formalism

- Should mirror semantic relationships as close as possible
- Incorporate word-based information and syntactic analysis
- Should be applicable to different languages

# Structured DSMs

## Basic Issue

- Words may not be the basic context units anymore
- How to capture and represent syntactic information?  
*X solves Y and Y is solved by X*

## An Ideal Formalism

- Should mirror semantic relationships as close as possible
- Incorporate word-based information and syntactic analysis
- Should be applicable to different languages

Use Dependency grammar framework

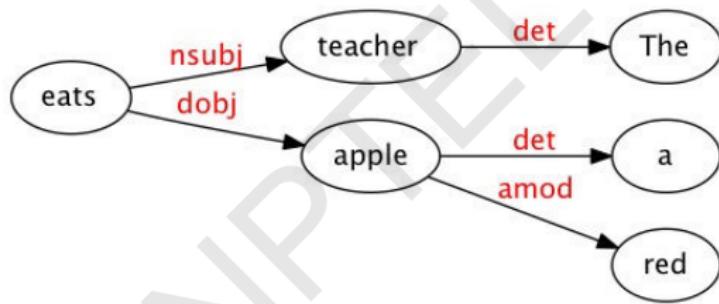
# *Structured DSMs*

NPTEL

# Structured DSMs

Using Dependency Structure: How does it help?

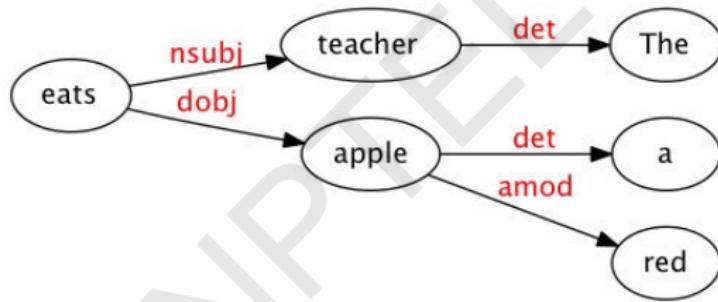
The teacher eats a red apple.



# Structured DSMs

Using Dependency Structure: How does it help?

The teacher eats a red apple.



- 'eat' is not a legitimate context for 'red'.
- The 'object' relation connecting 'eat' and 'apple' is treated as a different type of co-occurrence from the 'modifier' relation linking 'red' and 'apple'.

# *Structured DSMs*

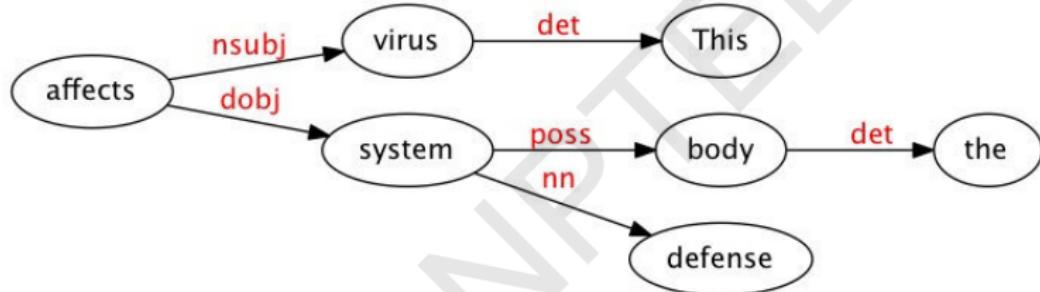
## *Structured DSMs: Words as ‘legitimate’ contexts*

- Co-occurrence statistics are collected using parser-extracted relations.
- To qualify as context of a target item, a word must be linked to it by some (interesting) lexico-syntactic relation

# Structured DSMs

## Distributional models, as guided by dependency

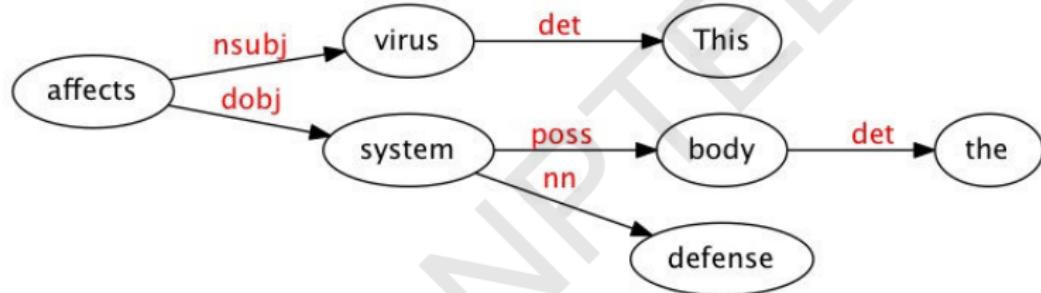
Ex: For the sentence '*This virus affects the body's defense system.*', the dependency parse is:



# Structured DSMs

## Distributional models, as guided by dependency

Ex: For the sentence '*This virus affects the body's defense system.*', the dependency parse is:



## Word vectors

<system, dobj, affects> ...

Corpus-derived ternary data can also be mapped onto a 2-way matrix

## *2-way matrix*

<system, dobj, affects>  
<virus, nsubj, affects>

*The dependency information can be dropped*

- <system, dobj, affects>  $\Rightarrow$  <system, affects>
- <virus, nsubj, affects>  $\Rightarrow$  <virus, affects>

## *2-way matrix*

<system, dobj, affects>  
<virus, nsubj, affects>

*The dependency information can be dropped*

- <system, dobj, affects> ⇒ <system, affects>
- <virus, nsubj, affects> ⇒ <virus, affects>

*Link and one word can be concatenated and treated as attributes*

- virus={nsubj-affects:0.05,...},
- system={dobj-affects:0.03,...}

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences for Verbs*

Most verbs prefer arguments of a particular type. This regularity is known as selectional preference.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences for Verbs*

Most verbs prefer arguments of a particular type. This regularity is known as selectional preference.

- From a parsed corpus, noun vectors are calculated as shown for ‘virus’ and ‘system’.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences for Verbs*

Most verbs prefer arguments of a particular type. This regularity is known as selectional preference.

- From a parsed corpus, noun vectors are calculated as shown for ‘virus’ and ‘system’.

|           | obj-carry | obj-buy | obj-drive | obj-eat | obj-store | sub-fly | ... |
|-----------|-----------|---------|-----------|---------|-----------|---------|-----|
| car       | 0.1       | 0.4     | 0.8       | 0.02    | 0.2       | 0.05    | ... |
| vegetable | 0.3       | 0.5     | 0         | 0.6     | 0.3       | 0.05    | ... |
| biscuit   | 0.4       | 0.4     | 0         | 0.5     | 0.4       | 0.02    | ... |
| ...       | ...       | ...     | ...       | ...     | ...       | ...     | ... |

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences*

- Suppose we want to compute the selectional preferences of the nouns as object of verb 'eat'.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences*

- Suppose we want to compute the selectional preferences of the nouns as object of verb ‘eat’.
- $n$  nouns having highest weight in the dimension ‘obj-eat’ are selected, let  $\{\text{vegetable, biscuit, ...}\}$  be the set of these  $n$  nouns.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences*

- Suppose we want to compute the selectional preferences of the nouns as object of verb ‘eat’.
- $n$  nouns having highest weight in the dimension ‘obj-eat’ are selected, let  $\{\text{vegetable, biscuit, ...}\}$  be the set of these  $n$  nouns.
- The complete vectors of these  $n$  nouns are used to obtain an ‘object prototype’ of the verb.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences*

- Suppose we want to compute the selectional preferences of the nouns as object of verb ‘eat’.
- $n$  nouns having highest weight in the dimension ‘obj-eat’ are selected, let  $\{\text{vegetable, biscuit, ...}\}$  be the set of these  $n$  nouns.
- The complete vectors of these  $n$  nouns are used to obtain an ‘object prototype’ of the verb.
- ‘object prototype’ will indicate various attributes such as these nouns can be consumed, bought, carried, stored etc.

# *Structured DSMs for Selectional Preferences*

## *Selectional Preferences*

- Suppose we want to compute the selectional preferences of the nouns as object of verb ‘eat’.
- $n$  nouns having highest weight in the dimension ‘obj-eat’ are selected, let  $\{\text{vegetable}, \text{biscuit}, \dots\}$  be the set of these  $n$  nouns.
- The complete vectors of these  $n$  nouns are used to obtain an ‘object prototype’ of the verb.
- ‘object prototype’ will indicate various attributes such as these nouns can be consumed, bought, carried, stored etc.
- Similarity of a noun to this ‘object prototype’ is used to denote the plausibility of that noun being an object of verb ‘eat’.

# *Word Embeddings - Part I*

Pawan Goyal

CSE, IIT Kharagpur

Week 7, Lecture 4

# *Word Vectors*

- At one level, it is simply a vector of weights.

# *Word Vectors*

- At one level, it is simply a vector of weights.
- In a simple 1-of-N (or ‘one-hot’) encoding every element in the vector is associated with a word in the vocabulary.

# Word Vectors

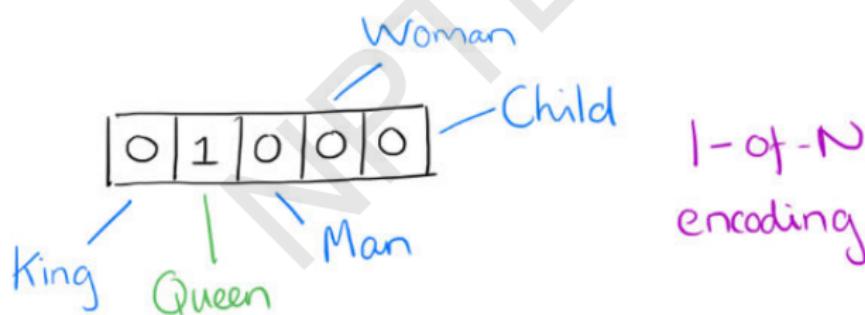
- At one level, it is simply a vector of weights.
- In a simple 1-of-N (or ‘one-hot’) encoding every element in the vector is associated with a word in the vocabulary.
- The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero.

## One-hot representation

motel [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND  
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0] = 0

# Word Vectors - One-hot Encoding

- Suppose our vocabulary has only five words: King, Queen, Man, Woman, and Child.
- We could encode the word 'Queen' as:



# *Limitations of One-hot encoding*

NPTEL

# *Limitations of One-hot encoding*

*Word vectors are not comparable*

Using such an encoding, there is no meaningful comparison we can make between word vectors other than equality testing.

# *Word2Vec – A distributed representation*

## *Distributional representation – word embedding?*

Any word  $w_i$  in the corpus is given a distributional representation by an embedding

$$w_i \in R^d$$

i.e., a  $d$ -dimensional vector, which is mostly learnt!

# *Word2Vec – A distributed representation*

## *Distributional representation – word embedding?*

Any word  $w_i$  in the corpus is given a distributional representation by an embedding

$$w_i \in R^d$$

i.e., a  $d$ -dimensional vector, which is mostly learnt!

*linguistics* =

$$\begin{Bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{Bmatrix}$$

# *Distributional Representation*

- Take a vector with several hundred dimensions (say 1000).
- Each word is represented by a distribution of weights across those elements.
- So instead of a one-to-one mapping between an element in the vector and a word, the representation of a word is spread across all of the elements in the vector, and
- Each element in the vector contributes to the definition of many words.

## Distributional Representation: Illustration

If we label the dimensions in a hypothetical word vector (there are no such pre-assigned labels in the algorithm of course), it might look a bit like this:



Such a vector comes to represent in some abstract way the ‘meaning’ of a word

# *Word Embeddings*

- $d$  typically in the range 50 to 1000
- Similar words should have similar embeddings

# Word Embeddings

- $d$  typically in the range 50 to 1000
- Similar words should have similar embeddings

*SVD can also be thought of as an embedding method*

# *Reasoning with Word Vectors*

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.

# *Reasoning with Word Vectors*

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

# *Reasoning with Word Vectors*

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

## *Case of Singular-Plural Relations*

If we denote the vector for word  $i$  as  $x_i$ , and focus on the singular/plural relation, we observe that

# Reasoning with Word Vectors

- It has been found that the learned word representations in fact capture meaningful syntactic and semantic regularities in a very simple way.
- Specifically, the regularities are observed as constant vector offsets between pairs of words sharing a particular relationship.

## Case of Singular-Plural Relations

If we denote the vector for word  $i$  as  $x_i$ , and focus on the singular/plural relation, we observe that

$$x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families} \approx x_{car} - x_{cars}$$

and so on.

# *Reasoning with Word Vectors*

Perhaps more surprisingly, we find that this is also the case for a variety of semantic relations.

*Good at answering analogy questions*

a is to b, as c is to ?

*man is to woman as uncle is to ? (aunt)*

# *Reasoning with Word Vectors*

Perhaps more surprisingly, we find that this is also the case for a variety of semantic relations.

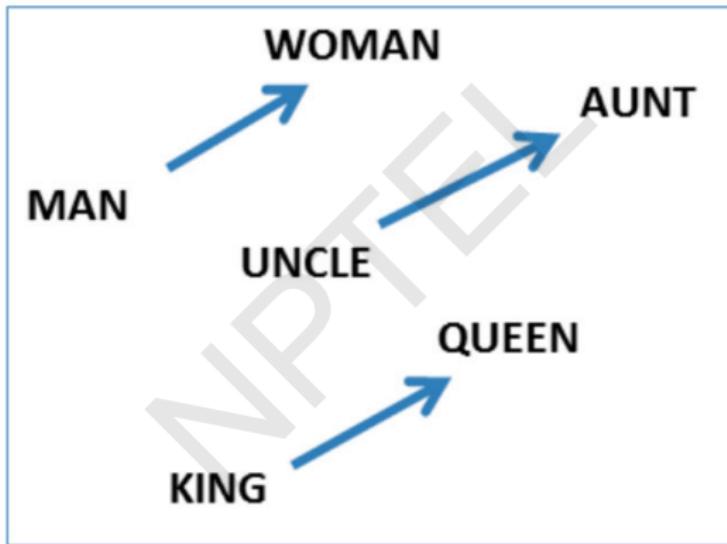
*Good at answering analogy questions*

a is to b, as c is to ?

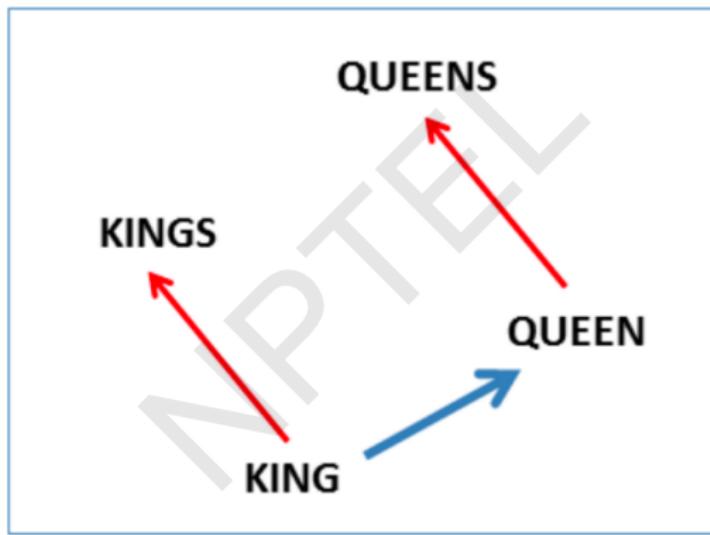
*man is to woman as uncle is to ? (aunt)*

*A simple vector offset method based on cosine distance shows the relation.*

# Vector Offset for Gender Relation



# *Vector Offset for Singular-Plural Relation*



# Encoding Other Dimensions of Similarity

## Analogy Testing

| Relationship         | Example 1           | Example 2         | Example 3            |
|----------------------|---------------------|-------------------|----------------------|
| France - Paris       | Italy: Rome         | Japan: Tokyo      | Florida: Tallahassee |
| big - bigger         | small: larger       | cold: colder      | quick: quicker       |
| Miami - Florida      | Baltimore: Maryland | Dallas: Texas     | Kona: Hawaii         |
| Einstein - scientist | Messi: midfielder   | Mozart: violinist | Picasso: painter     |
| Sarkozy - France     | Berlusconi: Italy   | Merkel: Germany   | Koizumi: Japan       |
| copper - Cu          | zinc: Zn            | gold: Au          | uranium: plutonium   |
| Berlusconi - Silvio  | Sarkozy: Nicolas    | Putin: Medvedev   | Obama: Barack        |
| Microsoft - Windows  | Google: Android     | IBM: Linux        | Apple: iPhone        |
| Microsoft - Ballmer  | Google: Yahoo       | IBM: McNealy      | Apple: Jobs          |
| Japan - sushi        | Germany: bratwurst  | France: tapas     | USA: pizza           |

# Analogy Testing

$$a:b :: c:?$$



$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

man:woman :: king:?

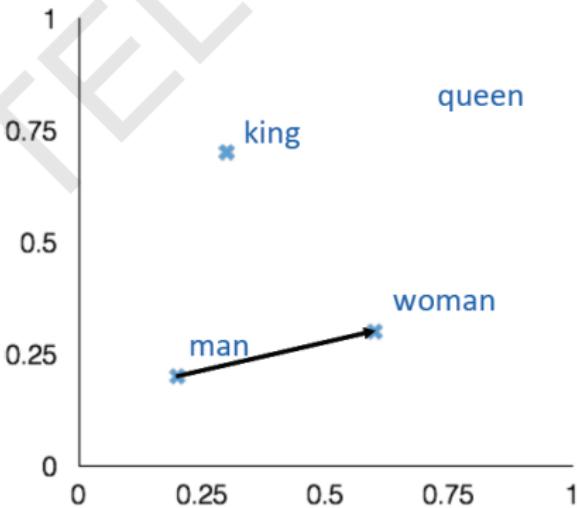
$$+ \text{ king} \quad [0.30 \ 0.70]$$

$$- \text{ man} \quad [0.20 \ 0.20]$$

$$+ \text{ woman} \quad [0.60 \ 0.30]$$

---

$$\text{queen} \quad [0.70 \ 0.80]$$



# Country-capital city relationships

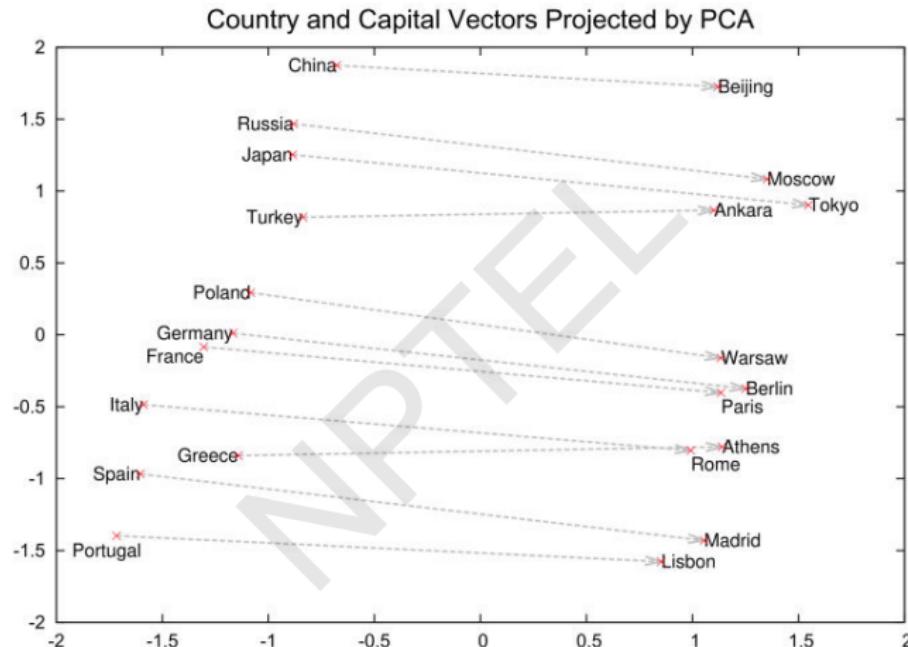


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

# More Analogy Questions

| Newspapers          |                       |               |                     |
|---------------------|-----------------------|---------------|---------------------|
| New York            | New York Times        | Baltimore     | Baltimore Sun       |
| San Jose            | San Jose Mercury News | Cincinnati    | Cincinnati Enquirer |
| NHL Teams           |                       |               |                     |
| Boston              | Boston Bruins         | Montreal      | Montreal Canadiens  |
| Phoenix             | Phoenix Coyotes       | Nashville     | Nashville Predators |
| NBA Teams           |                       |               |                     |
| Detroit             | Detroit Pistons       | Toronto       | Toronto Raptors     |
| Oakland             | Golden State Warriors | Memphis       | Memphis Grizzlies   |
| Airlines            |                       |               |                     |
| Austria             | Austrian Airlines     | Spain         | Spainair            |
| Belgium             | Brussels Airlines     | Greece        | Aegean Airlines     |
| Company executives  |                       |               |                     |
| Steve Ballmer       | Microsoft             | Larry Page    | Google              |
| Samuel J. Palmisano | IBM                   | Werner Vogels | Amazon              |

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

# Element Wise Addition

We can also use element-wise addition of vector elements to ask questions such as ‘German + airlines’ and by looking at the closest tokens to the composite vector come up with impressive answers:

| Czech + currency | Vietnam + capital | German + airlines      | Russian + river | French + actress     |
|------------------|-------------------|------------------------|-----------------|----------------------|
| koruna           | Hanoi             | airline Lufthansa      | Moscow          | Juliette Binoche     |
| Czech crown      | Ho Chi Minh City  | carrier Lufthansa      | Volga River     | Vanessa Paradis      |
| Polish zolty     | Viet Nam          | flag carrier Lufthansa | upriver         | Charlotte Gainsbourg |
| CTK              | Vietnamese        | Lufthansa              | Russia          | Cecile De            |

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

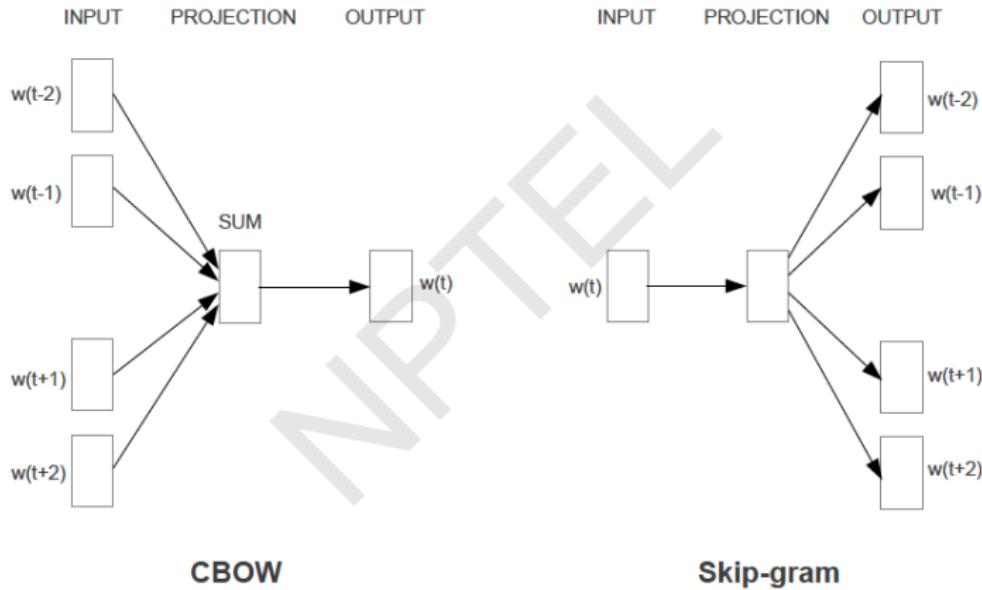
# *Learning Word Vectors*

## *Basic Idea*

*Instead of capturing co-occurrence counts directly, predict (using) surrounding words of every word.*

Code as well as word-vectors: <https://code.google.com/p/word2vec/>

## Two Variations: CBOW and Skip-grams



# *Word Embeddings - Part II*

Pawan Goyal

CSE, IIT Kharagpur

Week 7, Lecture 5

- Consider a piece of prose such as:

“The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntactic and semantic word relationships.”

- Consider a piece of prose such as:  
“The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntactic and semantic word relationships.”
- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words that precede it, and the four words that follow it:

# CBOW

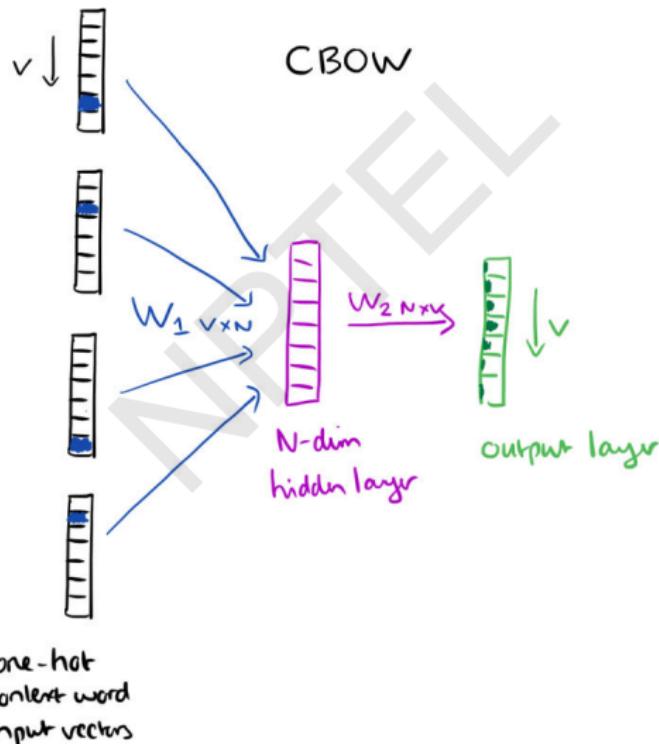
- Consider a piece of prose such as:  
“The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntactic and semantic word relationships.”
- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words that precede it, and the four words that follow it:

... an efficient method for learning high quality distributed vector ...

The diagram illustrates a sliding window context around a 'focus word'. A blue arrow points upwards from the text to the word 'method'. Below the text, two green curly braces, each labeled 'context' in green, encompass the word 'method' and the surrounding text 'an efficient method for learning high quality distributed vector ...'. The word 'method' is positioned centrally between the two contexts.

# CBOW

The context words form the input layer. Each word is encoded in one-hot form.  
A single hidden and output layer.



## *CBOW: Training Objective*

- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.
- In our example, given the input (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”), we want to maximize the probability of getting “learning” as the output.

## *CBOW: Input to Hidden Layer*

Since our input vectors are one-hot, multiplying an input vector by the weight matrix  $W_1$  amounts to simply selecting a row from  $W_1$ .

$$\begin{array}{c} \text{input} \\ 1 \times V \end{array} \quad \begin{array}{c} W_1 \\ V \times N \end{array} \quad \begin{array}{c} \text{hidden} \\ 1 \times N \end{array}$$
$$[0 \ 1 \ 0] \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}_{W_1} = [e \ f \ g \ h]$$

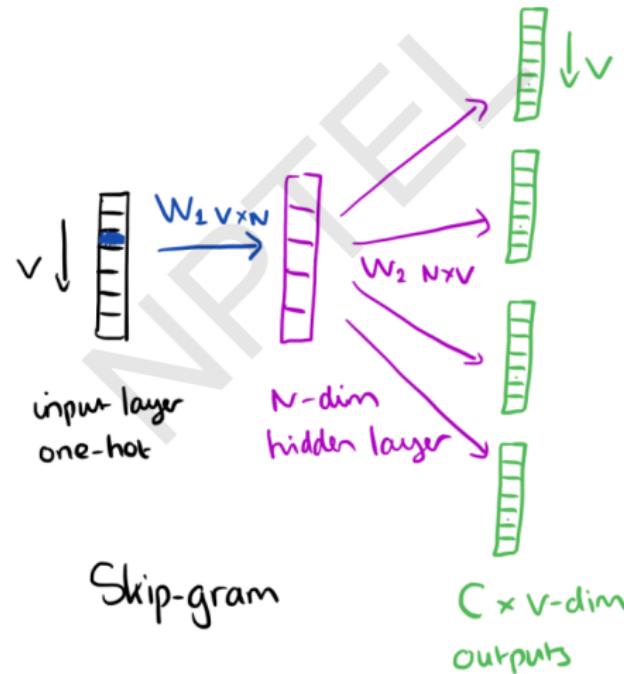
Given  $C$  input word vectors, the activation function for the hidden layer  $h$  amounts to simply summing the corresponding ‘hot’ rows in  $W_1$ , and dividing by  $C$  to take their average.

## *CBOW: Hidden to Output Layer*

From the hidden layer to the output layer, the second weight matrix  $W_2$  can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

# Skip-gram Model

The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector, and the target context words are now at the output layer:



## *Skip-gram Model: Training*

- The activation function for the hidden layer simply amounts to copying the corresponding row from the weights matrix  $W_1$  (linear) as we saw before.
- At the output layer, we now output  $C$  multinomial distributions instead of just one.
- The training objective is to minimize the summed prediction error across all context words in the output layer. In our example, the input would be “learning”, and we hope to see (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”) at the output layer.

# *Skip-gram Model*

## *Details*

Predict surrounding words in a window of length  $c$  of each word

# Skip-gram Model

## Details

Predict surrounding words in a window of length  $c$  of each word

**Objective Function:** Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

# Word Vectors

For  $p(w_{t+j}|w_t)$  the simplest first formulation is

$$p(w_O|w_I) = \frac{\exp(v'_{wO}^T v_{WI})}{\sum_{w=1}^W \exp(v'_{wI}^T v_{WI})}$$

# Word Vectors

For  $p(w_{t+j}|w_t)$  the simplest first formulation is

$$p(w_O|w_I) = \frac{\exp(v'_{wO}^T v_{WI})}{\sum_{w=1}^W \exp(v'_{wI}^T v_{WI})}$$

where  $v$  and  $v'$  are “input” and “output” vector representations of  $w$  (so every word has two vectors)

## Parameters $\theta$

With  $d$ -dimensional words and  $V$  many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ v'_{aardvark} \\ v'_a \\ \vdots \\ v'_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

# *Gradient Descent for Parameter Updates*

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

## *Two sets of vectors*

Best solution is to sum these up

$$L_{final} = L + L'$$

## *Two sets of vectors*

Best solution is to sum these up

$$L_{final} = L + L'$$

*A good tutorial to understand parameter learning:*

<https://arxiv.org/pdf/1411.2738.pdf>

## *Two sets of vectors*

Best solution is to sum these up

$$L_{final} = L + L'$$

*A good tutorial to understand parameter learning:*

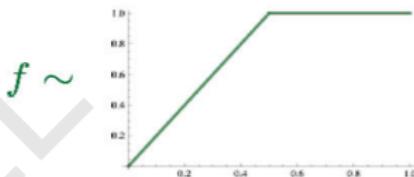
<https://arxiv.org/pdf/1411.2738.pdf>

*An interactive Demo*

<https://ronxin.github.io/wevi/>

# Glove

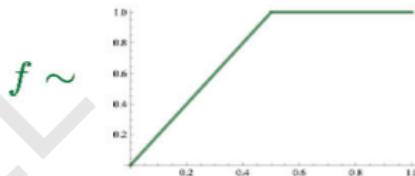
$$J = \frac{1}{2} \sum_{ij} f(P_{ij}) (w_i \cdot \tilde{w}_j - \log P_{ij})^2$$



Combine the best of both worlds – count based methods as well as direct prediction methods

# Glove

$$J = \frac{1}{2} \sum_{ij} f(P_{ij}) (w_i \cdot \tilde{w}_j - \log P_{ij})^2$$



Combine the best of both worlds – count based methods as well as direct prediction methods

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors

Code and vectors: <http://nlp.stanford.edu/projects/glove/>

# *Lexical Semantics*

Pawan Goyal

CSE, IIT Kharagpur

Week 8, Lecture 1

# *Lexical Semantics*

## *Definition*

**Lexical semantics** is concerned with the systematic meaning related connections among lexical items, and the internal meaning-related structure of individual lexical items.

## *Definition*

**Lexical semantics** is concerned with the systematic meaning related connections among lexical items, and the internal meaning-related structure of individual lexical items.

To identify the semantics of lexical items, we need to focus on the notion of **lexeme**, an individual entry in the lexicon.

# *Lexical Semantics*

## *Definition*

**Lexical semantics** is concerned with the systematic meaning related connections among lexical items, and the internal meaning-related structure of individual lexical items.

To identify the semantics of lexical items, we need to focus on the notion of **lexeme**, an individual entry in the lexicon.

## *What is a lexeme?*

**Lexeme** should be thought of as a pairing of a particular orthographic and phonological form with some sort of symbolic meaning representation.

- Orthographic form, and phonological form refer to the appropriate form part of a lexeme
- Sense refers to a lexeme's meaning counterpart.

# Example

## verge<sup>1</sup> | vərj |

noun

an edge or border: *they came down to the verge of the lake.*

- an extreme limit beyond which something specified will happen: *I was on the verge of tears.*
- Brit. a grass edging such as that by the side of a road or path.
- Architecture an edge of tiles projecting over a gable.

verb [ no obj. ] (**verge on**)

approach (something) closely; be close or similar to (something): *despair verging on the suicidal.*

ORIGIN late Middle English: via Old French from Latin *virga 'rod.'* The current verb sense dates from the late 18th cent.

## verge<sup>2</sup> | vərj |

noun

a wand or rod carried before a bishop or dean as an emblem of office.

ORIGIN late Middle English: from Latin *virga 'rod.'*

## verge<sup>3</sup> | vərj |

verb [ no obj. ]

incline in a certain direction or toward a particular state: *his style verged into the art nouveau school.*

ORIGIN early 17th cent. (in the sense '**descend (to the horizon)**') : from Latin *vergere 'to bend, incline.'*

## *Example: meaning related facts?*

*Definitions from the American Heritage Dictionary (Morris, 1985)*

- **right** *adj.* located near the right hand esp. being on the right when facing the same direction as the observer
- **left** *adj.* located near to this side of the body than the right
- **red** *n.* the color of blood or a ruby
- **blood** *n.* the red liquid that circulates in the heart, arteries and veins of animals

## *Example: meaning related facts?*

*Definitions from the American Heritage Dictionary (Morris, 1985)*

- **right** *adj.* located near the right hand esp. being on the right when facing the same direction as the observer
- **left** *adj.* located near to this side of the body than the right
- **red** *n.* the color of blood or a ruby
- **blood** *n.* the red liquid that circulates in the heart, arteries and veins of animals

- The entries are description of lexemes in terms of other lexemes
- Definitions make it clear that *right* and *left* are similar kind of lexemes that stand in some kind of alternation, or opposition, to one another
- We can glean that *red* is a color, it can be applied to both *blood* and *rubies*, and that *blood* is a liquid.

# *Relations between word meanings*

- Homonymy
- Polysemy
- Synonymy
- Antonymy
- Hypernymy
- Hyponymy
- Meronymy

# *Homonymy*

## *Definition*

**Homonymy** is defined as a relation that holds between words that have the same form with unrelated meanings.

# *Homonymy*

## *Definition*

**Homonymy** is defined as a relation that holds between words that have the same form with unrelated meanings.

## *Examples*

- Bat (wooden stick-like thing) vs Bat (flying mammal thing)
- Bank (financial institution) vs Bank (riverside)

# *Homonymy*

## *Definition*

**Homonymy** is defined as a relation that holds between words that have the same form with unrelated meanings.

## *Examples*

- Bat (wooden stick-like thing) vs Bat (flying mammal thing)
- Bank (financial institution) vs Bank (riverside)

## *homophones and homographs*

**homophones** are the words with the same pronunciation but different spellings.

- write vs right
- piece vs peace

**homographs** are the lexemes with the same orthographic form but different meaning. Ex: bass

# *Problems for NLP applications*

## *Text-to-Speech*

Same orthographic form but different phonological form

# *Problems for NLP applications*

## *Text-to-Speech*

Same orthographic form but different phonological form

## *Information Retrieval*

Different meaning but same orthographic form

# *Problems for NLP applications*

## *Text-to-Speech*

Same orthographic form but different phonological form

## *Information Retrieval*

Different meaning but same orthographic form

## *Speech Recognition*

to, two, too

*Perfect homonyms are also problematic*

# Polysemy

*Multiple related meanings within a single lexeme.*

- The *bank* was constructed in 1875 out of local red brick.
- I withdrew the money from the *bank*.

# Polysemy

*Multiple related meanings within a single lexeme.*

- The *bank* was constructed in 1875 out of local red brick.
- I withdrew the money from the *bank*.

*Are those the same sense?*

- Sense 1: “The building belonging to a financial institution”
- Sense 2: “A financial institution”

*Another example*

- Heavy snow caused the roof of the *school* to collapse.
- The *school* hired more teachers this year than ever before.

## *Polysemy: multiple related meanings*

*Often, the relationships are systematic*

E.g., building vs. organization

*school, university, hospital, church, supermarket*

# *Polysemy: multiple related meanings*

*Often, the relationships are systematic*

E.g., building vs. organization

*school, university, hospital, church, supermarket*

*More examples:*

- Author (Jane Austen wrote Emma) ↔ Works of Author (I really love Jane Austen)
- Animal (The chicken was domesticated in Asia) ↔ Meat (The chicken was overcooked)
- Tree (Plums have beautiful blossoms) ↔ Fruit (I ate a preserved plum yesterday)

# *Polysemy: multiple related meanings*

## *Zeugma test*

- Which of these flights *serve* breakfast?
- Does Midwest Express *serve* Philadelphia?

# *Polysemy: multiple related meanings*

## *Zeugma test*

- Which of these flights *serve* breakfast?
- Does Midwest Express *serve* Philadelphia?

*\*Does Midwest Express serve breakfast and San Jose?*

# *Polysemy: multiple related meanings*

## *Zeugma test*

- Which of these flights *serve* breakfast?
- Does Midwest Express *serve* Philadelphia?

\*Does Midwest Express serve breakfast and San Jose?

*Combine two separate uses of a lexeme into a single example using conjunction*

Since it sounds weird, we say that these are two different senses of *serve*.

# Synonymy

*Words that have the same meaning in some or all contexts.*

- filbert / hazelnut
- couch / sofa
- big / large
- automobile / car
- vomit / throw up
- water /  $H_2O$

Two lexemes are synonyms if they can be successfully substituted for each other in all situations.

# *Synonymy: A relation between senses*

Consider the words *big* and *large*.

*Are they synonyms?*

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

# *Synonymy: A relation between senses*

Consider the words *big* and *large*.

*Are they synonyms?*

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

*How about here?*

- Miss Nelson, for instance, became a kind of **big** sister to Benjamin.
- \*Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

# *Synonymy: A relation between senses*

Consider the words *big* and *large*.

## *Are they synonyms?*

- How **big** is that plane?
- Would I be flying on a **large** or small plane?

## *How about here?*

- Miss Nelson, for instance, became a kind of **big** sister to Benjamin.
- \*Miss Nelson, for instance, became a kind of **large** sister to Benjamin.

## *Why?*

- *big* has a sense that means being older, or grown up
- *large* lacks this sense

# Synonyms

## Shades of meaning

- What is the cheapest first class *fare*?
- \*What is the cheapest first class *price*?

# Synonyms

## Shades of meaning

- What is the cheapest first class *fare*?
- \*What is the cheapest first class *price*?

## Collocational constraints

- We frustate 'em and frustate 'em, and pretty soon they make a *big* mistake.
- \*We frustate 'em and frustate 'em, and pretty soon they make a *large* mistake.

# *Antonyms*

- Senses that are opposites with respect to one feature of their meaning
- Otherwise, they are similar!
  - ▶ dark / light
  - ▶ short / long
  - ▶ hot / cold
  - ▶ up / down
  - ▶ in / out

# *Antonyms*

- Senses that are opposites with respect to one feature of their meaning
- Otherwise, they are similar!
  - ▶ dark / light
  - ▶ short / long
  - ▶ hot / cold
  - ▶ up / down
  - ▶ in / out

*More formally: antonyms can*

- define a binary opposition or at opposite ends of a scale (*long/short, fast/slow*)
- Be **reversives**: *rise/fall*

# *Hyponymy and Hypernymy*

## *Hyponymy*

One sense is a hyponym of another if the first sense is more specific, denoting a subclass of the other

- *car* is a hyponym of *vehicle*
- *dog* is a hyponym of *animal*
- *mango* is a hyponym of *fruit*

# *Hyponymy and Hypernymy*

## *Hyponymy*

One sense is a hyponym of another if the first sense is more specific, denoting a subclass of the other

- *car* is a hyponym of *vehicle*
- *dog* is a hyponym of *animal*
- *mango* is a hyponym of *fruit*

## *Hypernymy*

Conversely

- *vehicle* is a hypernym/superordinate of *car*
- *animal* is a hypernym of *dog*
- *fruit* is a hypernym of *mango*

# *Hyponymy more formally*

## *Entailment*

Sense *A* is a hyponym of sense *B* if being an *A* entails being a *B*.

Ex: dog, animal

## *Transitivity*

*A* hypo *B* and *B* hypo *C* entails *A* hypo *C*

# *Meronyms and holonyms*

## *Definition*

**Meronymy:** an asymmetric, transitive relation between senses.

$X$  is a **meronym** of  $Y$  if it denotes a part of  $Y$ .

The inverse relation is **holonymy**.

| meronym | holonym |
|---------|---------|
| porch   | house   |
| wheel   | car     |
| leg     | chair   |
| nose    | face    |

# *Lexical Semantics - WordNet*

Pawan Goyal

CSE, IIT Kharagpur

Week 8, Lecture 2

# WordNet

<https://wordnet.princeton.edu/wordnet/>

- A hierarchically organized lexical database
- A machine-readable thesaurus, and aspects of a dictionary
- Versions for other languages are under development

| part of speech | no. synsets |
|----------------|-------------|
| noun           | 82,115      |
| verb           | 13,767      |
| adjective      | 18,156      |
| adverb         | 3,621       |

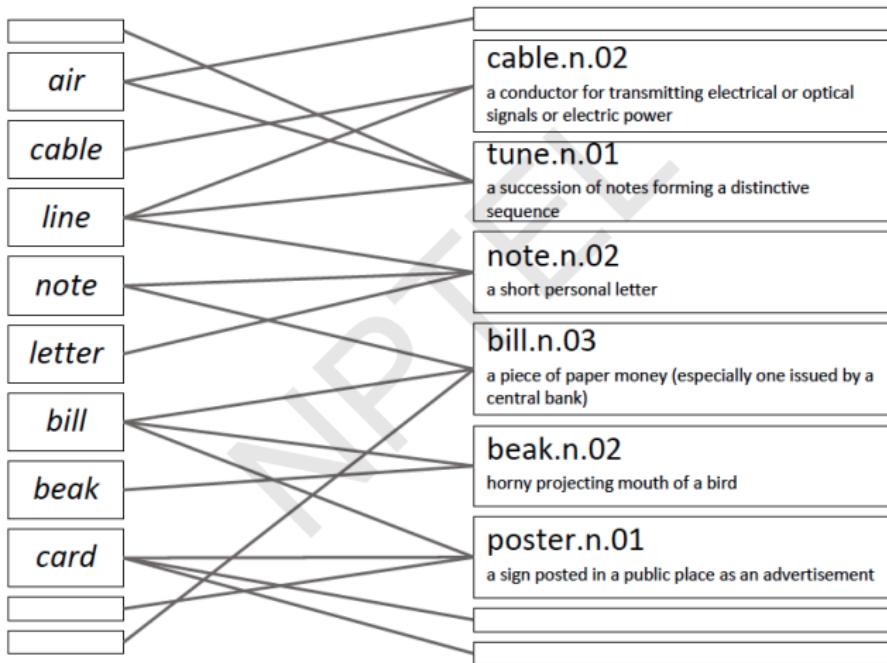
# *Synsets in WordNet*

- A **synset** is a set of synonyms representing a sense
- Example: chump as a noun to mean ‘a person who is gullible and easy to take advantage of’

{chump<sup>1</sup>, fool<sup>2</sup>, gull<sup>1</sup>, mark<sup>9</sup>, patsy<sup>1</sup>, fall guy<sup>1</sup>, sucker<sup>1</sup>, soft touch<sup>1</sup>, mug<sup>2</sup>}

- Each of these senses share this same gloss.
- For WordNet, the meaning of this sense of chump is this list.

# *lemma vs. synsets*



# All relations in WordNet

|                        |                                             |
|------------------------|---------------------------------------------|
| searchtype             | is at least one of the following:           |
| -ants{n v a r}         | Antonyms                                    |
| -hype{n v}             | Hypernyms                                   |
| -hypo{n v}, -tree{n v} | Hyponyms & Hyponym Tree                     |
| -entav                 | Verb Entailment                             |
| -syns{n v a r}         | Synonyms (ordered by estimated frequency)   |
| -smemn                 | Member of Holonyms                          |
| -ssubn                 | Substance of Holonyms                       |
| -sprtn                 | Part of Holonyms                            |
| -membn                 | Has Member Meronyms                         |
| -subsn                 | Has Substance Meronyms                      |
| -partn                 | Has Part Meronyms                           |
| -meron                 | All Meronyms                                |
| -holon                 | All Holonyms                                |
| -causv                 | Cause to                                    |
| -pert{a r}             | Pertainyms                                  |
| -attr{n a}             | Attributes                                  |
| -deri{n v}             | Derived Forms                               |
| -domn{n v a r}         | Domain                                      |
| -domt{n v a r}         | Domain Terms                                |
| -faml{n v a r}         | Familiarity & Polysemy Count                |
| -framv                 | Verb Frames                                 |
| -coor{n v}             | Coordinate Terms (sisters)                  |
| -simsv                 | Synonyms (grouped by similarity of meaning) |
| -hmern                 | Hierarchical Meronyms                       |
| -hholn                 | Hierarchical Holonyms                       |
| -grep{n v a r}         | List of Compound Words                      |
| -over                  | Overview of Senses                          |
|                        | -                                           |

# Wordnet noun and verb relations

| Relation       | Also called   | Definition                                | Example                                                     |
|----------------|---------------|-------------------------------------------|-------------------------------------------------------------|
| Hypernym       | Superordinate | From concepts to superordinates           | <i>breakfast</i> <sup>1</sup> → <i>meal</i> <sup>1</sup>    |
| Hyponym        | Subordinate   | From concepts to subtypes                 | <i>meal</i> <sup>1</sup> → <i>lunch</i> <sup>1</sup>        |
| Member Meronym | Has-Member    | From groups to their members              | <i>faculty</i> <sup>2</sup> → <i>professor</i> <sup>1</sup> |
| Has-Instance   |               | From concepts to instances of the concept | <i>composer</i> <sup>1</sup> → <i>Bach</i> <sup>1</sup>     |
| Instance       |               | From instances to their concepts          | <i>Austen</i> <sup>1</sup> → <i>author</i> <sup>1</sup>     |
| Member Holonym | Member-Of     | From members to their groups              | <i>copilot</i> <sup>1</sup> → <i>crew</i> <sup>1</sup>      |
| Part Meronym   | Has-Part      | From wholes to parts                      | <i>table</i> <sup>2</sup> → <i>leg</i> <sup>3</sup>         |
| Part Holonym   | Part-Of       | From parts to wholes                      | <i>course</i> <sup>7</sup> → <i>meal</i> <sup>1</sup>       |
| Antonym        |               | Opposites                                 | <i>leader</i> <sup>1</sup> → <i>follower</i> <sup>1</sup>   |

| Relation | Definition                                                        | Example                                                     |
|----------|-------------------------------------------------------------------|-------------------------------------------------------------|
| Hypernym | From events to superordinate events                               | <i>fly</i> <sup>9</sup> → <i>travel</i> <sup>5</sup>        |
| Troponym | From a verb (event) to a specific manner elaboration of that verb | <i>walk</i> <sup>1</sup> → <i>stroll</i> <sup>1</sup>       |
| Entails  | From verbs (events) to the verbs (events) they entail             | <i>snore</i> <sup>1</sup> → <i>sleep</i> <sup>1</sup>       |
| Antonym  | Opposites                                                         | <i>increase</i> <sup>1</sup> ⇔ <i>decrease</i> <sup>1</sup> |

# WordNet Hierarchies

Synonyms/Hypernyms (Ordered by Estimated Frequency) of noun mouse

4 senses of mouse

Sense 1

mouse

- => rodent, gnawer
- => placental, placental mammal, eutherian, eutherian mammal
- => mammal, mammalian
- => vertebrate, craniate
- => chordate
- => animal, animate being, beast, brute, creature, fauna
- => organism, being
  - => living thing, animate thing
- => whole, unit
- => object, physical object
- => physical entity
- => entity

Sense 4

mouse, computer mouse

- => electronic device
- => device
- => instrumentality, instrumentation
- => artifact, artefact
- => whole, unit
- => object, physical object
- => physical entity
- => entity

# *Word Similarity*

- Synonymy is a binary relation
  - ▶ Two words are either synonymous or not

# *Word Similarity*

- Synonymy is a binary relation
  - ▶ Two words are either synonymous or not
- We want a looser metric
  - ▶ Word similarity or
  - ▶ Word distance

# *Word Similarity*

- Synonymy is a binary relation
  - ▶ Two words are either synonymous or not
- We want a looser metric
  - ▶ Word similarity or
  - ▶ Word distance
- Two words are more similar
  - ▶ If they share more features of meaning

# Word Similarity

- Synonymy is a binary relation
  - ▶ Two words are either synonymous or not
- We want a looser metric
  - ▶ Word similarity or
  - ▶ Word distance
- Two words are more similar
  - ▶ If they share more features of meaning
- Actually these are really relations between **senses**:
  - ▶ Instead of saying “bank is like fund”
  - ▶ We say
    - ★ Bank<sup>1</sup> is similar to fund<sup>3</sup>
    - ★ Bank<sup>2</sup> is similar to slope<sup>5</sup>

# Word Similarity

- Synonymy is a binary relation
  - ▶ Two words are either synonymous or not
- We want a looser metric
  - ▶ Word similarity or
  - ▶ Word distance
- Two words are more similar
  - ▶ If they share more features of meaning
- Actually these are really relations between **senses**:
  - ▶ Instead of saying “bank is like fund”
  - ▶ We say
    - ★ Bank<sup>1</sup> is similar to fund<sup>3</sup>
    - ★ Bank<sup>2</sup> is similar to slope<sup>5</sup>
- We will compute similarity over both words and senses

# *Two classes of algorithms*

## *Distributional algorithms*

By comparing words based on their distributional context in the corpora

## *Thesaurus-based algorithms*

Based on whether words are “nearby” in WordNet

## *Thesaurus-based Word Similarity*

- We could use anything in the thesaurus:
    - ▶ Meronymy, hyponymy, troponymy
    - ▶ Glosses and example sentences

# *Thesaurus-based Word Similarity*

- We could use anything in the thesaurus:
  - ▶ Meronymy, hyponymy, troponymy
  - ▶ Glosses and example sentences
- In practice, “thesaurus-based” methods usually use:
  - ▶ the is-a/subsumption/hypernymy hierarchy
  - ▶ and sometimes the glosses too

# *Thesaurus-based Word Similarity*

- We could use anything in the thesaurus:
  - ▶ Meronymy, hyponymy, troponymy
  - ▶ Glosses and example sentences
- In practice, “thesaurus-based” methods usually use:
  - ▶ the is-a/subsumption/hypernymy hierarchy
  - ▶ and sometimes the glosses too
- Word similarity vs. word relatedness
  - ▶ Similar words are near-synonyms
  - ▶ Related words could be related any way
    - ★ car, gasoline : related, but not similar
    - ★ car, bicycle: similar

# *Path-based similarity*

## *Basic Idea*

- Two words are similar if they are nearby in the hypernym graph

# *Path-based similarity*

## *Basic Idea*

- Two words are similar if they are nearby in the hypernym graph
- $\text{pathlen}(c_1, c_2)$  = number of edges in shortest path (in hypernym graph) between senses  $c_1$  and  $c_2$

# *Path-based similarity*

## *Basic Idea*

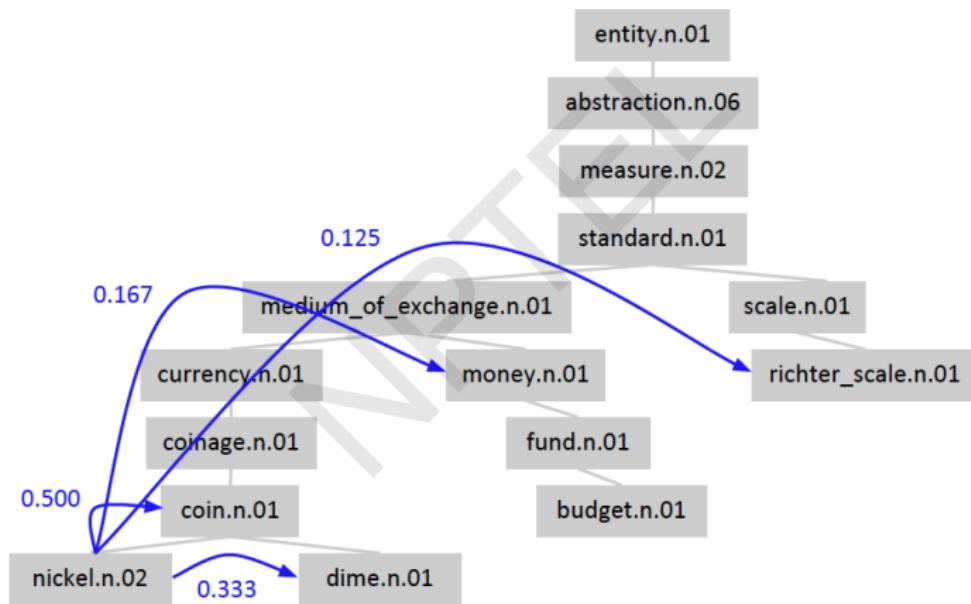
- Two words are similar if they are nearby in the hypernym graph
- $\text{pathlen}(c_1, c_2)$  = number of edges in shortest path (in hypernym graph) between senses  $c_1$  and  $c_2$
- $\text{sim}_{\text{path}}(c_1, c_2) = \frac{1}{1 + \text{pathlen}(c_1, c_2)}$

# *Path-based similarity*

## *Basic Idea*

- Two words are similar if they are nearby in the hypernym graph
- $\text{pathlen}(c_1, c_2)$  = number of edges in shortest path (in hypernym graph) between senses  $c_1$  and  $c_2$
- $\text{sim}_{\text{path}}(c_1, c_2) = \frac{1}{1 + \text{pathlen}(c_1, c_2)}$
- $\text{sim}(w_1, w_2) = \max_{c_1 \in \text{senses}(w_1), c_2 \in \text{senses}(w_2)} \text{sim}(c_1, c_2)$

## Shortest path in the hierarchy



# *Leacock-Chodorow (L-C) Similarity*

## *L-C similarity*

$$sim_{LC}(c_1, c_2) = -\log(pathlen(c_1, c_2)/2d)$$

*d*: maximum depth of the hierarchy

# *Leacock-Chodorow (L-C) Similarity*

## *L-C similarity*

$$sim_{LC}(c_1, c_2) = -\log(pathlen(c_1, c_2)/2d)$$

$d$ : maximum depth of the hierarchy

## *Problems with L-C similarity*

- Assumes each edge represents a uniform distance
- ‘nickel-money’ seems closer than ‘nickel-standard’

# *Leacock-Chodorow (L-C) Similarity*

## *L-C similarity*

$$sim_{LC}(c_1, c_2) = -\log(pathlen(c_1, c_2)/2d)$$

*d*: maximum depth of the hierarchy

## *Problems with L-C similarity*

- Assumes each edge represents a uniform distance
- ‘nickel-money’ seems closer than ‘nickel-standard’
- We want a metric which lets us assign different “lengths” to different edges - but how?

# *Concept probability models*

## *Concept probabilities*

- For each concept (synset)  $c$ , let  $P(c)$  be the probability that a randomly selected word in a corpus is an instance (hyponym) of  $c$

# *Concept probability models*

## *Concept probabilities*

- For each concept (synset)  $c$ , let  $P(c)$  be the probability that a randomly selected word in a corpus is an instance (hyponym) of  $c$
- $P(ROOT) = 1$
- The lower a node in the hierarchy, the lower its probability

# *Concept probability models*

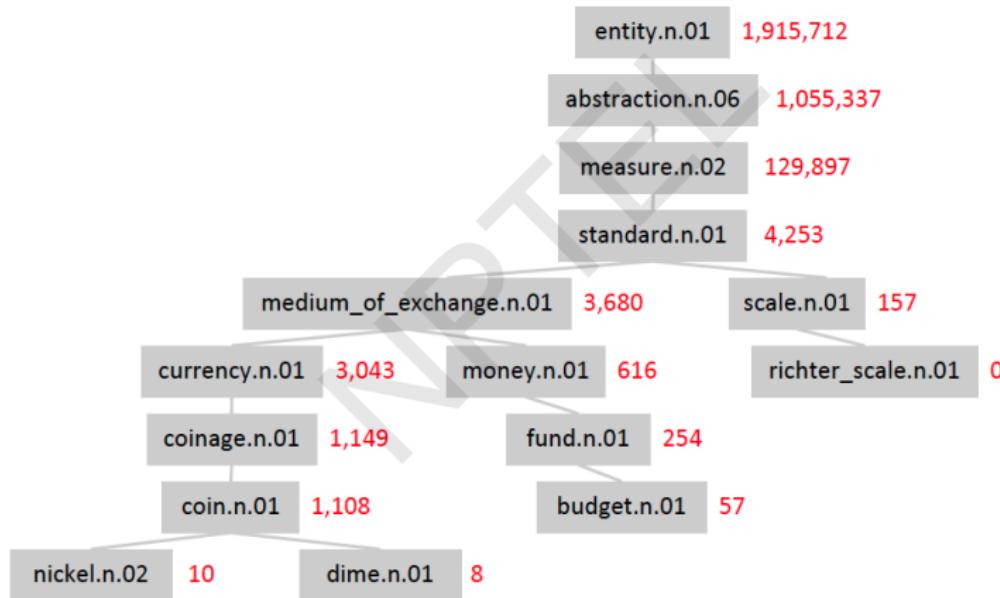
## *Cconcept probabilities*

- For each concept (synset)  $c$ , let  $P(c)$  be the probability that a randomly selected word in a corpus is an instance (hyponym) of  $c$
- $P(ROOT) = 1$
- The lower a node in the hierarchy, the lower its probability

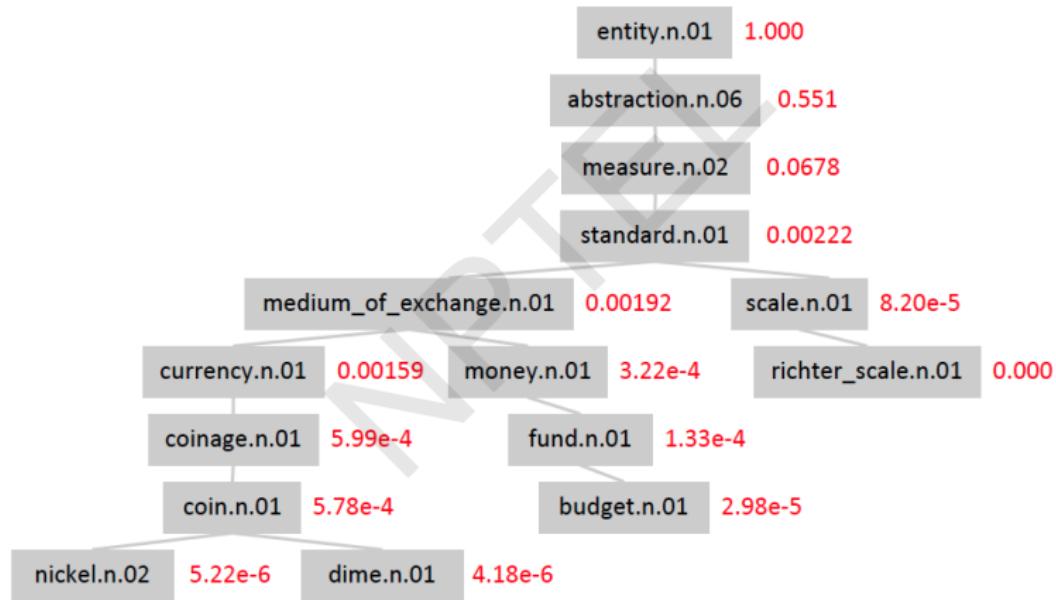
## *Estimating concept probabilities*

- Train by counting “concept activations” in a corpus
- Each occurrence of *dime* also increments counts for *coin*, *currency*, *standard*, etc.

## Example : concept count



## Example : concept probabilities

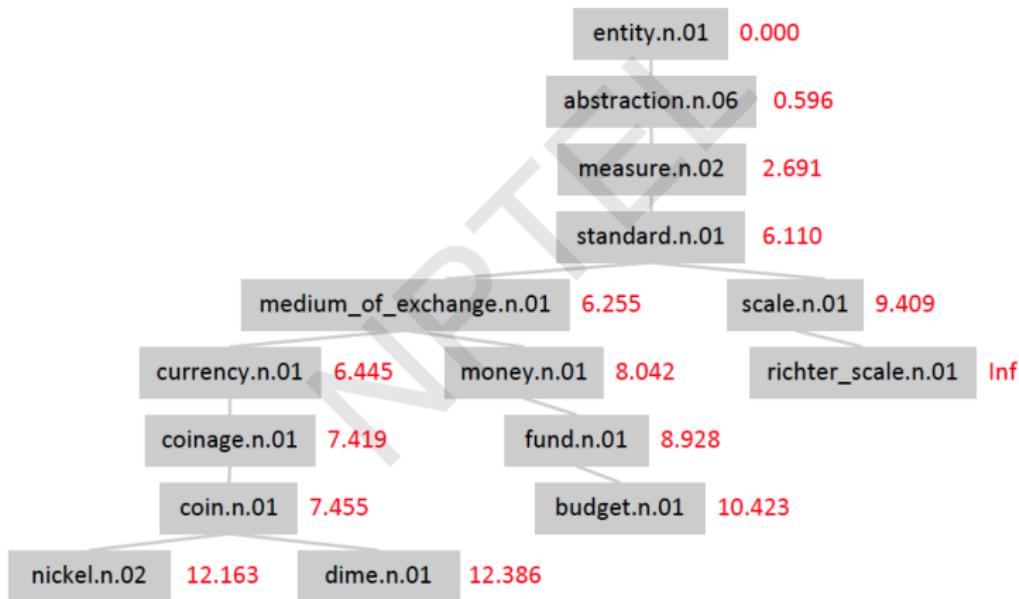


# *Information content*

## *Information content*

- Information content:  $IC(c) = -\log P(c)$
- Lowest common subsumer :  $LCS(c_1, c_2)$ : the lowest node in the hierarchy that subsumes (is a hypernym of) both  $c_1$  and  $c_2$
- We are now ready to see how to use information content ( $IC$ ) as a similarity metric.

## Example : Information content

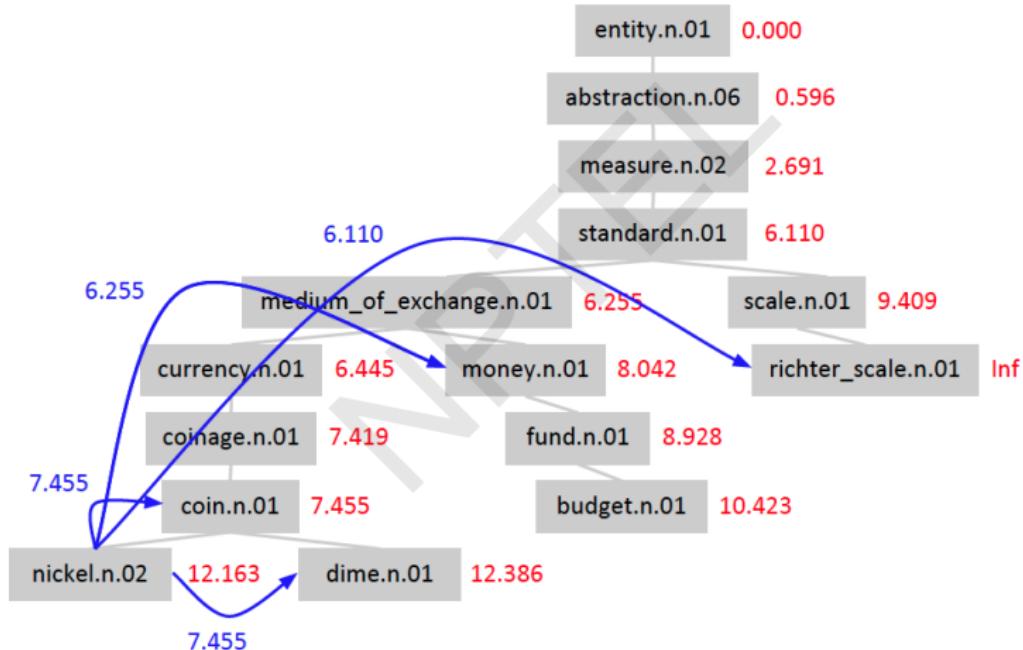


# Resnik Similarity

## Resnik Similarity

- Intuition: how similar two words are depends on how much they have in common
- It measures the commonality by the information content of the lowest common subsumer
- $sim_{resnik}(c_1, c_2) = IC(LCS(c_1, c_2)) = -\log P(LCS(c_1, c_2))$

## Example: Resnik similarity



# *Lin similarity*

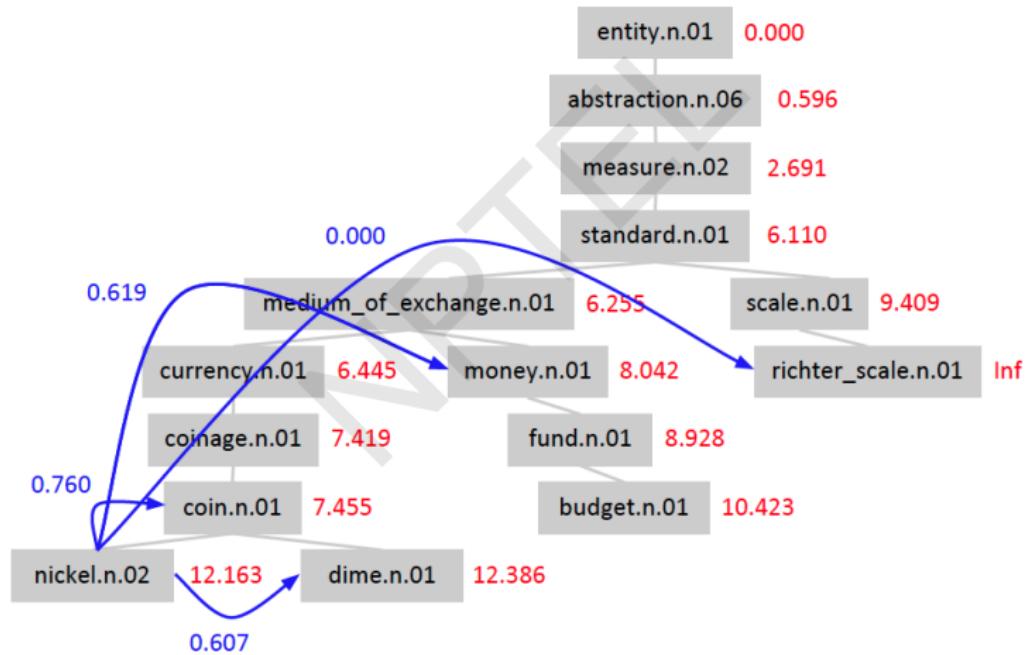
## *Proportion of shared information*

- It's not just about commonalities - it's also about differences!
- **Resnik:** The more information content they share, the more similar they are
- **Lin:** The more information content they don't share, the less similar they are
- Not the *absolute* quantity of shared information but the *proportion* of shared information

$$sim_{Lin}(c_1, c_2) = \frac{2\log P(LCS(c_1, c_2))}{\log P(c_1) + \log P(c_2)}$$

The information content common to  $c_1$  and  $c_2$ , normalized by their average information content.

## Example: Lin similarity



# Jiang-Conrath distance

## JC similarity

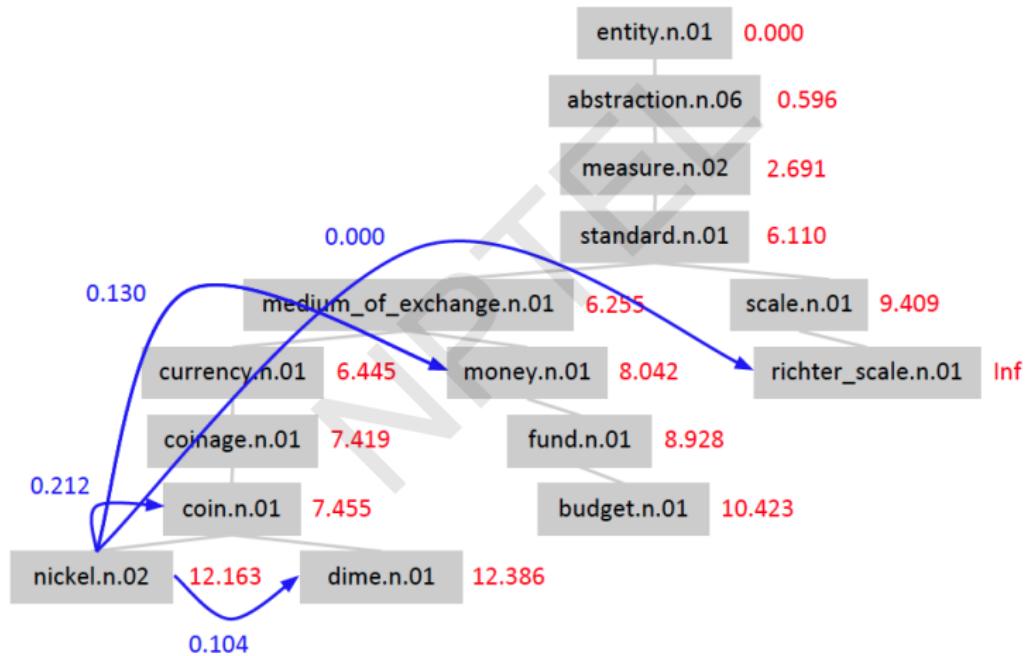
We can use IC to assign lengths to graph edges:

$$dist_{JC}(c, \text{hypernym}(c)) = IC(c) - IC(\text{hypernym}(c))$$

$$\begin{aligned} dist_{JC}(c_1, c_2) &= dist_{JC}(c_1, LCS(c_1, c_2)) + dist_{JC}(c_2, LCS(c_1, c_2)) \\ &= IC(c_1) - IC(LCS(c_1, c_2)) + IC(c_2) - IC(LCS(c_1, c_2)) \\ &= IC(c_1) + IC(c_2) - 2 \times IC(LCS(c_1, c_2)) \end{aligned}$$

$$sim_{JC}(c_1, c_2) = \frac{1}{IC(c_1) + IC(c_2) - 2 \times IC(LCS(c_1, c_2))}$$

## Example: Jiang-Conrath distance



# *The (extended) Lesk Algorithm*

- Two concepts are similar if their glosses contain similar words
  - ▶ *Drawing paper: paper that is **specially prepared** for use in drafting*
  - ▶ *Decal: the art of transferring designs from **specially prepared paper** to a wood or glass or metal surface*

# *The (extended) Lesk Algorithm*

- Two concepts are similar if their glosses contain similar words
  - ▶ *Drawing paper: paper that is **specially prepared** for use in drafting*
  - ▶ *Decal: the art of transferring designs from **specially prepared paper** to a wood or glass or metal surface*
- For each n-word phrase that occurs in both glosses, add a score of  $n^2$

# *The (extended) Lesk Algorithm*

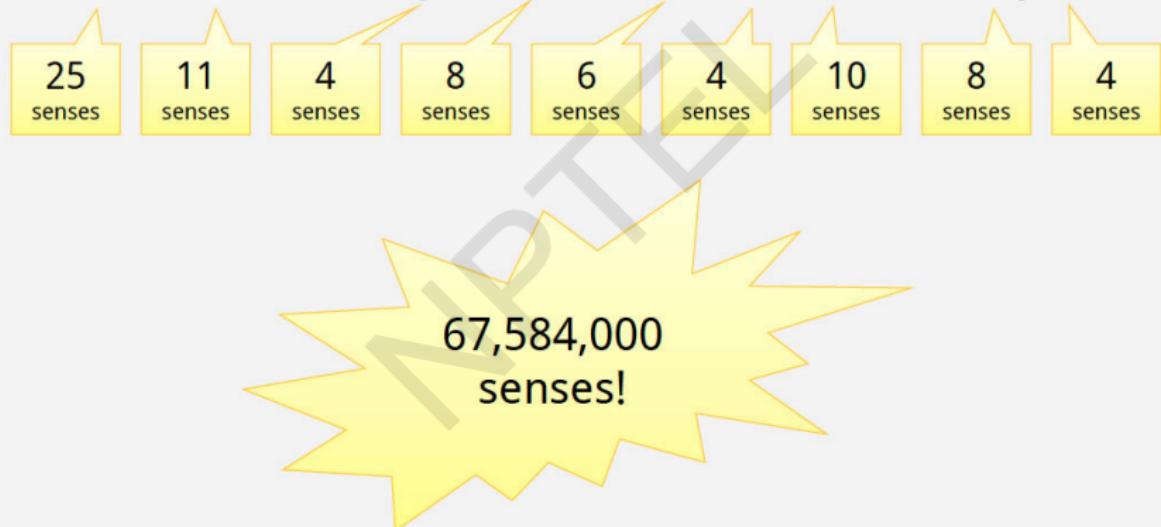
- Two concepts are similar if their glosses contain similar words
  - ▶ *Drawing paper: paper that is **specially prepared** for use in drafting*
  - ▶ *Decal: the art of transferring designs from **specially prepared paper** to a wood or glass or metal surface*
- For each n-word phrase that occurs in both glosses, add a score of  $n^2$
- **paper** and **specially prepared** →  $1 + 4 = 5$

## *Problem in mapping words to wordnet senses*

I saw a man who is 98 years old and can still walk and tell jokes

# *Ambiguity is rampant!*

*I saw a man who is 98 years old and can still walk and tell jokes*



# *Word Sense Disambiguation - I*

Pawan Goyal

CSE, IIT Kharagpur

Week 8, Lecture 3

# *Word Sense Disambiguation (WSD)*

## *Sense ambiguity*

- Many words have several meanings or senses
- The meaning of **bass** depends on the context
- Are we talking about music, or fish?
  - ▶ An electric guitar and **bass** player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.
  - ▶ And it all started when fishermen decided the striped **bass** in Lake Mead were too skinny.

# *Word Sense Disambiguation (WSD)*

## *Sense ambiguity*

- Many words have several meanings or senses
- The meaning of **bass** depends on the context
- Are we talking about music, or fish?
  - ▶ An electric guitar and **bass** player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.
  - ▶ And it all started when fishermen decided the striped **bass** in Lake Mead were too skinny.

## *Disambiguation*

- The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word.

# *Word Sense Disambiguation (WSD)*

## *Sense ambiguity*

- Many words have several meanings or senses
- The meaning of **bass** depends on the context
- Are we talking about music, or fish?
  - ▶ An electric guitar and **bass** player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.
  - ▶ And it all started when fishermen decided the striped **bass** in Lake Mead were too skinny.

## *Disambiguation*

- The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word.
- This is done by looking at the context of the word's use.

# Algorithms

- Knowledge Based Approaches
  - ▶ Overlap Based Approaches
- Machine Learning Based Approaches
  - ▶ Supervised Approaches
  - ▶ Semi-supervised Algorithms
  - ▶ Unsupervised Algorithms
- Hybrid Approaches

# *Knowledge Based Approaches*

## *Overlap Based Approaches*

- Require a **Machine Readable Dictionary** (MRD).
- Find the overlap between the features of different senses of an ambiguous word (**sense bag**) and the features of the words in its context (**context bag**).
- The features could be sense definitions, example sentences, hypernyms etc.
- The features could also be given weights.
- The sense which has the maximum overlap is selected as the contextually appropriate sense.

## *Lesk's Algorithm*

**Sense Bag:** contains the words in the definition of a candidate sense of the ambiguous word.

NPTEL

## *Lesk's Algorithm*

**Sense Bag:** contains the words in the definition of a candidate sense of the ambiguous word.

**Context Bag:** contains the words in the definition of each sense of each context word.

## *Lesk's Algorithm*

**Sense Bag:** contains the words in the definition of a candidate sense of the ambiguous word.

**Context Bag:** contains the words in the definition of each sense of each context word.

*On burning **coal** we get **ash**.*

# Lesk's Algorithm

**Sense Bag:** contains the words in the definition of a candidate sense of the ambiguous word.

**Context Bag:** contains the words in the definition of each sense of each context word.

*On burning coal we get ash.*

## Ash

- Sense 1  
Trees of the olive family with pinnate leaves, thin furrowed bark and gray branches.
- Sense 2  
The **solid** residue left when **combustible** material is thoroughly **burned** or oxidized.
- Sense 3  
To convert into ash

## Coal

- Sense 1  
A piece of glowing carbon or **burnt** wood.
- Sense 2  
charcoal.
- Sense 3  
A black **solid combustible** substance formed by the partial decomposition of vegetable matter without free access to air and under the influence of moisture and often increased pressure and temperature that is widely used as a fuel for **burning**

In this case Sense 2 of ash would be the winner sense.

# *Walker's Algorithm*

- A Thesaurus Based approach

NPTEL

# *Walker's Algorithm*

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs

## *Walker's Algorithm*

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs
- **Step 2:** Calculate the score for each sense by using the context words.

# *Walker's Algorithm*

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs
- **Step 2:** Calculate the score for each sense by using the context words.  
*A context word will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.*

# *Walker's Algorithm*

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs
- **Step 2:** Calculate the score for each sense by using the context words.  
*A context word will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.*
  - ▶ E.g. The money in this bank fetches an interest of 8% per annum

# *Walker's Algorithm*

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs
- **Step 2:** Calculate the score for each sense by using the context words.  
*A context word will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.*
  - ▶ E.g. The money in this bank fetches an interest of 8% per annum
  - ▶ Target word: *bank*
  - ▶ Clue words from the context: *money, interest, annum, fetch*

# Walker's Algorithm

- A Thesaurus Based approach
- **Step 1:** For each sense of the target word find the thesaurus category to which that sense belongs
- **Step 2:** Calculate the score for each sense by using the context words.  
*A context word will add 1 to the score of the sense if the thesaurus category of the word matches that of the sense.*
  - ▶ E.g. The money in this bank fetches an interest of 8% per annum
  - ▶ Target word: *bank*
  - ▶ Clue words from the context: *money, interest, annum, fetch*

|          | Sense1: Finance | Sense2: Location |
|----------|-----------------|------------------|
| Money    | +1              | 0                |
| Interest | +1              | 0                |
| Fetch    | 0               | 0                |
| Annum    | +1              | 0                |
| Total    | 3               | 0                |

Context words  
add 1 to the  
sense when  
the topic of the  
word matches that  
of the sense

# WSD Using Random Walk Algorithm

The church bells no longer rung on Sundays.

church

- 1: one of the groups of Christians who have their own beliefs and forms of worship
- 2: a place for public (especially Christian) worship
- 3: a service conducted in a church

bell

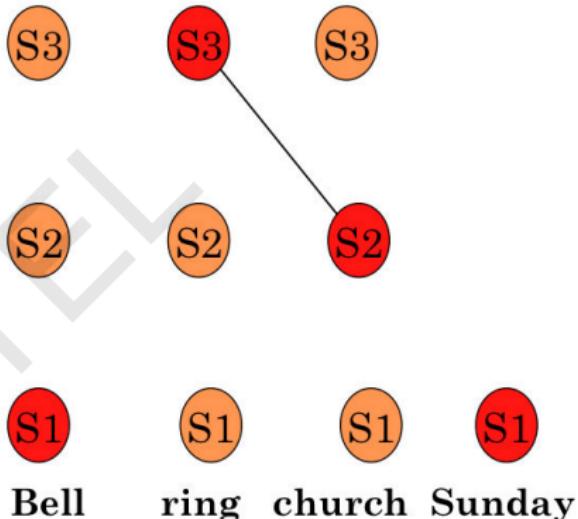
- 1: a hollow device made of metal that makes a ringing sound when struck
- 2: a push button at an outer door that gives a ringing or buzzing signal when pushed
- 3: the sound of a bell

ring

- 1: make a ringing sound
- 2: ring or echo with sound
- 3: make (bells) ring, often for the purposes of musical edification

Sunday

- 1: first day of the week; observed as a day of rest and worship by most Christians



**Step 1:** Add a vertex for each possible sense of each word in the text.

# WSD Using Random Walk Algorithm

The church bells no longer rung on Sundays.

church

- 1: one of the groups of Christians who have their own beliefs and forms of worship
- 2: a place for public (especially Christian) worship
- 3: a service conducted in a church

bell

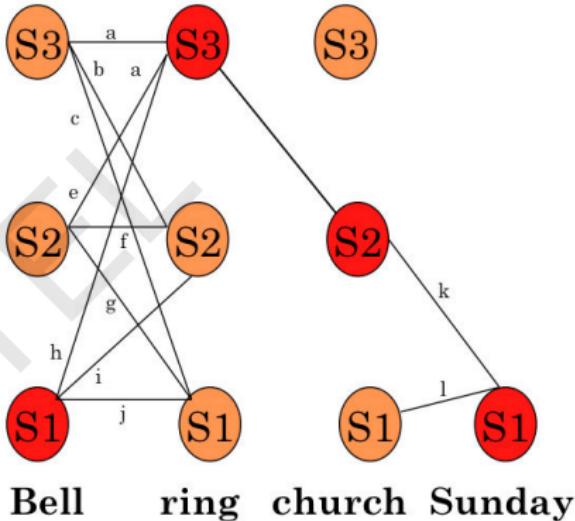
- 1: a hollow device made of metal that makes a ringing sound when struck
- 2: a push button at an outer door that gives a ringing or buzzing signal when pushed
- 3: the sound of a bell

ring

- 1: make a ringing sound
- 2: ring or echo with sound
- 3: make (bells) ring, often for the purposes of musical edification

Sunday

- 1: first day of the week; observed as a day of rest and worship by most Christians



**Step 2:** Add weighted edges using definition based semantic similarity (Lesk's method).

# WSD Using Random Walk Algorithm

The church bells no longer rung on Sundays.

church

- 1: one of the groups of Christians who have their own beliefs and forms of worship
- 2: a place for public (especially Christian) worship
- 3: a service conducted in a church

bell

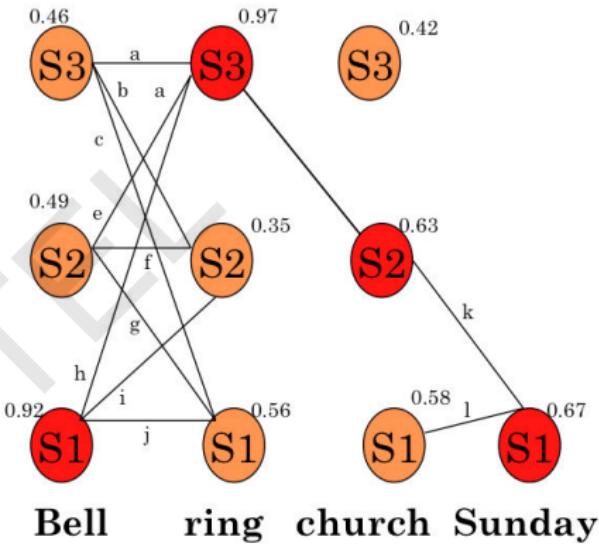
- 1: a hollow device made of metal that makes a ringing sound when struck
- 2: a push button at an outer door that gives a ringing or buzzing signal when pushed
- 3: the sound of a bell

ring

- 1: make a ringing sound
- 2: ring or echo with sound
- 3: make (bells) ring, often for the purposes of musical edification

Sunday

- 1: first day of the week; observed as a day of rest and worship by most Christians



**Step 3:** Apply graph based ranking algorithm to find score of each vertex (i.e. for each word sense).

# WSD Using Random Walk Algorithm

The church bells no longer rung on Sundays.

church

- 1: one of the groups of Christians who have their own beliefs and forms of worship
- 2: a place for public (especially Christian) worship
- 3: a service conducted in a church

bell

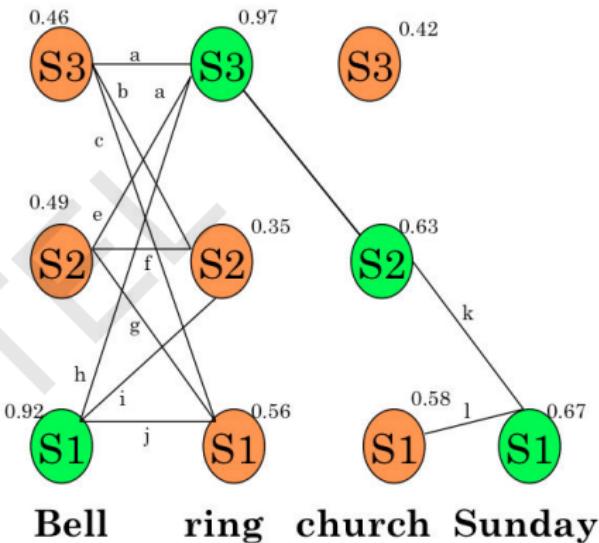
- 1: a hollow device made of metal that makes a ringing sound when struck
- 2: a push button at an outer door that gives a ringing or buzzing signal when pushed
- 3: the sound of a bell

ring

- 1: make a ringing sound
- 2: ring or echo with sound
- 3: make (bells) ring, often for the purposes of musical edification

Sunday

- 1: first day of the week; observed as a day of rest and worship by most Christians



**Step 4:** Select the vertex (sense)  
which has the highest score.

# *Naïve Bayes for WSD*

- A Naïve Bayes classifier chooses the most likely sense for a word given the features of the context:

$$\hat{s} = \arg \max_{s \in S} P(s|f)$$

# Naïve Bayes for WSD

- A Naïve Bayes classifier chooses the most likely sense for a word given the features of the context:

$$\hat{s} = \arg \max_{s \in S} P(s|f)$$

- Using Bayes' law, this can be expressed as:

$$\begin{aligned}\hat{s} &= \arg \max_{s \in S} \frac{P(s)P(f|s)}{P(f)} \\ &= \arg \max_{s \in S} P(s)P(f|s)\end{aligned}$$

# Naïve Bayes for WSD

- A Naïve Bayes classifier chooses the most likely sense for a word given the features of the context:

$$\hat{s} = \arg \max_{s \in S} P(s|f)$$

- Using Bayes' law, this can be expressed as:

$$\begin{aligned}\hat{s} &= \arg \max_{s \in S} \frac{P(s)P(f|s)}{P(f)} \\ &= \arg \max_{s \in S} P(s)P(f|s)\end{aligned}$$

- The 'Naïve' assumption: all the features are conditionally independent, given the sense':

$$\hat{s} = \arg \max_{s \in S} P(s) \prod_{j=1}^n P(f_j|s)$$

# *Training for Naïve Bayes*

- ‘ $f$ ’ is a feature vector consisting of:
  - ▶ POS of  $w$
  - ▶ Semantic and Syntactic features of  $w$
  - ▶ Collocation vector (set of words around it) → next word (+1), +2, -1, -2 and their POS’s
  - ▶ Co-occurrence vector

# Training for Naïve Bayes

- ‘ $f$ ’ is a feature vector consisting of:
  - ▶ POS of  $w$
  - ▶ Semantic and Syntactic features of  $w$
  - ▶ Collocation vector (set of words around it) → next word (+1), +2, -1, -2 and their POS’s
  - ▶ Co-occurrence vector
- Set parameters of Naïve Bayes using maximum likelihood estimation (MLE) from training data

$$P(s_i) = \frac{\text{count}(s_i, w_j)}{\text{count}(w_j)}$$

$$P(f_j|s_i) = \frac{\text{count}(f_j, s_i)}{\text{count}(s_i)}$$

# *Decision List Algorithm*

- Based on ‘One sense per collocation’ property
  - ▶ Nearby words provide strong and consistent clues as to the sense of a target word
- Collect a large set of collocations for the ambiguous word
- Calculate word-sense probability distributions for all such collocations

# Decision List Algorithm

- Based on ‘One sense per collocation’ property
  - ▶ Nearby words provide strong and consistent clues as to the sense of a target word
- Collect a large set of collocations for the ambiguous word
- Calculate word-sense probability distributions for all such collocations
- Calculate the log-likelihood ratio

$$\log\left(\frac{P(\text{Sense - A}|\text{Collocation}_i)}{P(\text{Sense - B}|\text{Collocation}_i)}\right)$$

- Higher log-likelihood  $\Rightarrow$  more predictive evidence

# Decision List Algorithm

- Based on ‘One sense per collocation’ property
  - ▶ Nearby words provide strong and consistent clues as to the sense of a target word
- Collect a large set of collocations for the ambiguous word
- Calculate word-sense probability distributions for all such collocations
- Calculate the log-likelihood ratio

$$\log\left(\frac{P(\text{Sense - A}|\text{Collocation}_i)}{P(\text{Sense - B}|\text{Collocation}_i)}\right)$$

- Higher log-likelihood  $\Rightarrow$  more predictive evidence
- Collocations are ordered in a decision list, with most predictive collocations ranked highest

# Decision List Algorithm

## Training Data

| Sense | Training Examples (Keyword in Context)                |
|-------|-------------------------------------------------------|
| A     | used to strain microscopic plant life from the ...    |
| A     | ... zonal distribution of plant life ...              |
| A     | close-up studies of plant life and natural ...        |
| A     | too rapid growth of aquatic plant life in water ...   |
| A     | ... the proliferation of plant and animal life ...    |
| A     | establishment phase of the plant virus life cycle ... |
| B     | ...<br>...<br>...                                     |
| B     | computer manufacturing plant and adjacent ...         |
| B     | discovered at a St. Louis plant manufacturing         |
| B     | ... copper manufacturing plant found that they        |
| B     | copper wire manufacturing plant, for example ...      |
| B     | 's cement manufacturing plant in Alpena ...           |
| B     | polystyrene manufacturing plant at its Dow ...        |
| B     | company manufacturing plant is in Orlando ...         |

## Resultant Decision List

| Final decision list for <i>plant</i> (abbreviated) |                                  |       |
|----------------------------------------------------|----------------------------------|-------|
| Log L                                              | Collocation                      | Sense |
| 10.12                                              | <i>plant growth</i>              | ⇒ A   |
| 9.68                                               | car (within $\pm k$ words)       | ⇒ B   |
| 9.64                                               | <i>plant height</i>              | ⇒ A   |
| 9.61                                               | union (within $\pm k$ words)     | ⇒ B   |
| 9.54                                               | equipment (within $\pm k$ words) | ⇒ B   |
| 9.51                                               | <i>assembly plant</i>            | ⇒ B   |
| 9.50                                               | <i>nuclear plant</i>             | ⇒ B   |
| 9.31                                               | flower (within $\pm k$ words)    | ⇒ A   |
| 9.24                                               | job (within $\pm k$ words)       | ⇒ B   |
| 9.03                                               | fruit (within $\pm k$ words)     | ⇒ A   |
| 9.02                                               | <i>plant species</i>             | ⇒ A   |
| ...                                                | ...                              |       |

# Decision List Algorithm

## Training Data

| Sense | Training Examples (Keyword in Context)                |
|-------|-------------------------------------------------------|
| A     | used to strain microscopic plant life from the ...    |
| A     | ... zonal distribution of plant life ...              |
| A     | close-up studies of plant life and natural ...        |
| A     | too rapid growth of aquatic plant life in water ...   |
| A     | ... the proliferation of plant and animal life ...    |
| A     | establishment phase of the plant virus life cycle ... |
| B     | ...<br>...                                            |
| B     | computer manufacturing plant and adjacent ...         |
| B     | discovered at a St. Louis plant manufacturing         |
| B     | ... copper manufacturing plant found that they        |
| B     | copper wire manufacturing plant, for example ...      |
| B     | 's cement manufacturing plant in Alpena ...           |
| B     | polystyrene manufacturing plant at its Dow ...        |
| B     | company manufacturing plant is in Orlando ...         |

## Resultant Decision List

| Final decision list for <i>plant</i> (abbreviated) |                                  |       |
|----------------------------------------------------|----------------------------------|-------|
| Log L                                              | Collocation                      | Sense |
| 10.12                                              | <i>plant growth</i>              | ⇒ A   |
| 9.68                                               | car (within $\pm k$ words)       | ⇒ B   |
| 9.64                                               | <i>plant height</i>              | ⇒ A   |
| 9.61                                               | union (within $\pm k$ words)     | ⇒ B   |
| 9.54                                               | equipment (within $\pm k$ words) | ⇒ B   |
| 9.51                                               | <i>assembly plant</i>            | ⇒ B   |
| 9.50                                               | <i>nuclear plant</i>             | ⇒ B   |
| 9.31                                               | flower (within $\pm k$ words)    | ⇒ A   |
| 9.24                                               | job (within $\pm k$ words)       | ⇒ B   |
| 9.03                                               | fruit (within $\pm k$ words)     | ⇒ A   |
| 9.02                                               | <i>plant species</i>             | ⇒ A   |
| ...                                                | ...                              |       |

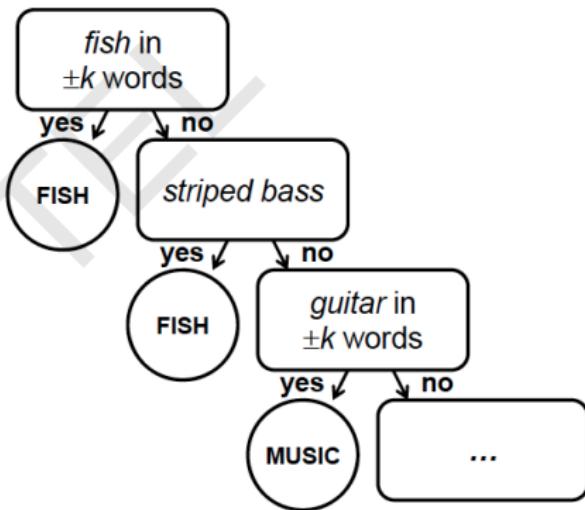
Classification of a test sentence is based on the highest ranking collocation, found in the test sentences.

*plucking flowers* affects *plant growth*.

## Decision List: Example

Example: discriminating between bass (fish) and bass (music):

| Context                                   | Sense |
|-------------------------------------------|-------|
| <i>fish in <math>\pm k</math> words</i>   | FISH  |
| <i>striped bass</i>                       | FISH  |
| <i>guitar in <math>\pm k</math> words</i> | MUSIC |
| <i>bass player</i>                        | MUSIC |
| <i>piano in <math>\pm k</math> words</i>  | MUSIC |
| <i>sea bass</i>                           | FISH  |
| <i>play bass</i>                          | MUSIC |
| <i>river in <math>\pm k</math> words</i>  | FISH  |
| <i>on bass</i>                            | MUSIC |
| <i>bass are</i>                           | FISH  |



# *Word Sense Disambiguation - II*

Pawan Goyal

CSE, IIT Kharagpur

Week 8, Lecture 4

# *Minimally Supervised WSD - Yarowsky*

- Annotations are expensive!
- “Bootstrapping” or co-training
  - ▶ Start with (small) seed, learn decision list
  - ▶ Use decision list to label rest of corpus
  - ▶ Retain ‘confident’ labels, treat as annotated data to learn new decision list
  - ▶ Repeat ...

# *Minimally Supervised WSD - Yarowsky*

- Annotations are expensive!
- “Bootstrapping” or co-training
  - ▶ Start with (small) seed, learn decision list
  - ▶ Use decision list to label rest of corpus
  - ▶ Retain ‘confident’ labels, treat as annotated data to learn new decision list
  - ▶ Repeat ...
- Heuristics (derived from observation):
  - ▶ One sense per discourse
  - ▶ One sense per collocation

# *More about heuristics*

## *One Sense per Discourse*

- A word tends to preserve its meaning across all its occurrences in a given discourse

# *More about heuristics*

## *One Sense per Discourse*

- A word tends to preserve its meaning across all its occurrences in a given discourse

## *One Sense per Collocation*

- A word tends to preserve its meaning when used in the same collocation
  - ▶ Strong for adjacent collocations
  - ▶ Weaker as the distance between the words increases

# *Yarowsky's Method*

## *Example*

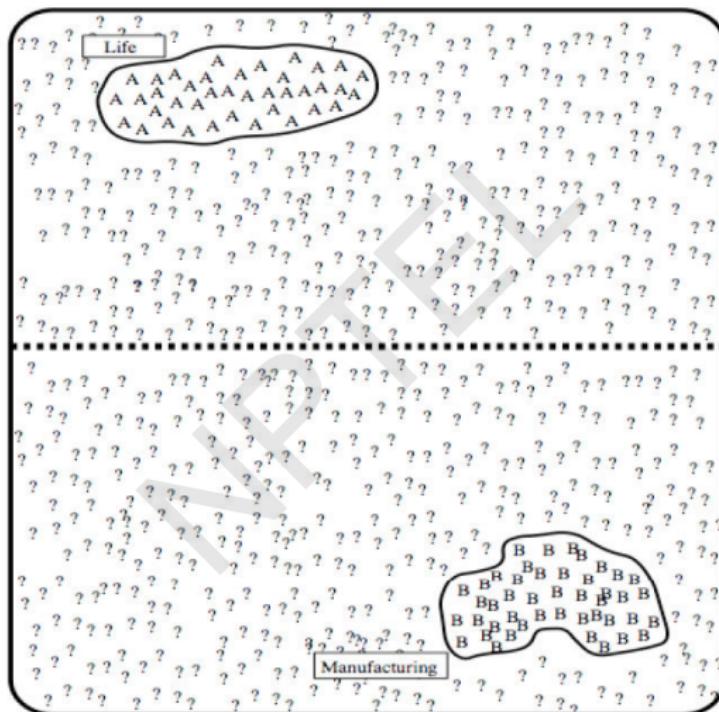
- Disambiguating plant (industrial sense) vs. plant (living thing sense)
- Think of seed features for each sense
  - Industrial sense: co-occurring with 'manufacturing'
  - Living thing sense: co-occurring with 'life'
- Use 'one sense per collocation' to build initial decision list classifier
- Treat results (having high probability) as annotated data, train new decision list classifier, iterate

# Yarowsky's Method: Example

used to strain microscopic plant life from the zonal distribution of plant life . close-up studies of plant life and natural too rapid growth of aquatic plant life in water the proliferation of plant and animal life establishment phase of the plant virus life cycle that divide life into plant and animal kingdom many dangers to plant and animal life mammals . Animal and plant life are delicately automated manufacturing plant in Fremont vast manufacturing plant and distribution chemical manufacturing plant , producing viscose keep a manufacturing plant profitable without computer manufacturing plant and adjacent discovered at a St. Louis plant manufacturing copper manufacturing plant found that they copper wire manufacturing plant , for example s cement manufacturing plant in Alpena

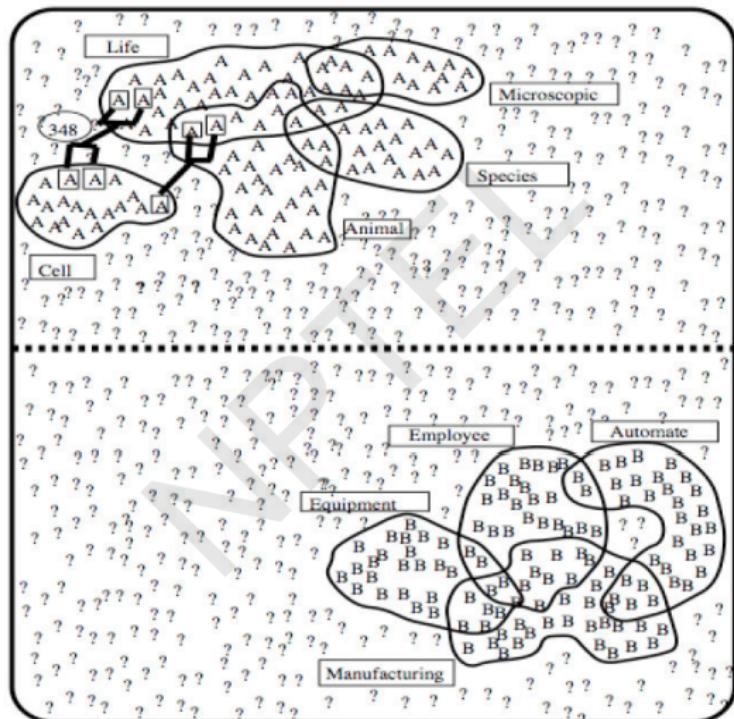
vinyl chloride monomer plant , which is molecules found in plant and animal tissue Nissan car and truck plant in Japan is and Golgi apparatus of plant and animal cells union responses to plant closures . cell types found in the plant kingdom are company said the plant is still operating Although thousands of plant and animal species animal rather than plant tissues can be

# Yarowsky's Method: Example



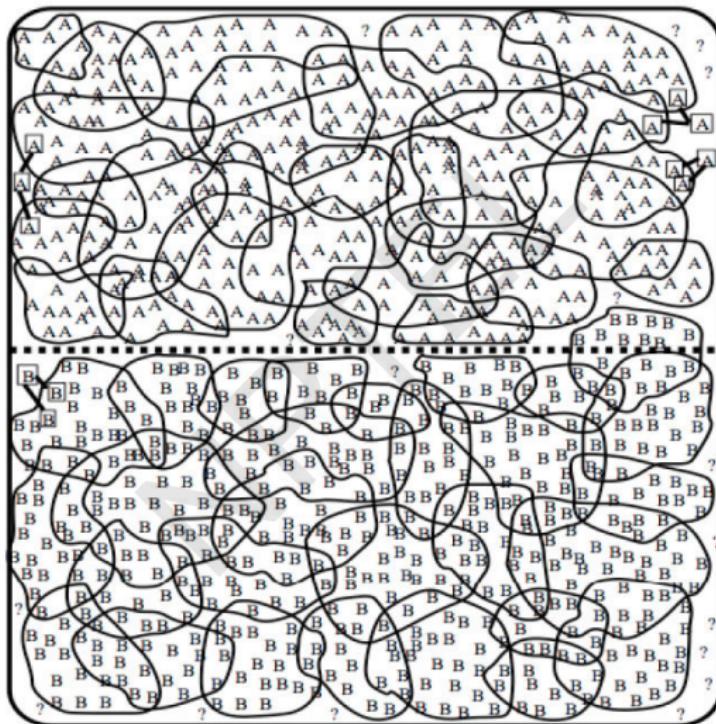
Initial state after use of seed rules

# Yarowsky's Method: Example



Intermediate state

# *Yarowsky's Method: Example*



Final state

# *Yarowsky's Method*

## *Termination*

- Stop when
  - ▶ Error on training data is less than a threshold
  - ▶ No more training data is covered
- Use final decision list for WSD

## *Advantages*

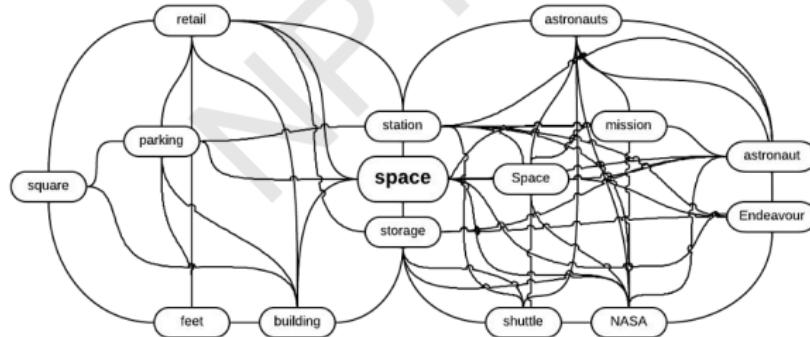
- Accuracy is about as good as a supervised algorithm
- Bootstrapping: far less manual effort

## *Key Idea: Word Sense Induction*

- Instead of using “dictionary defined senses”, extract the “senses from the corpus” itself
- These “corpus senses” or “uses” correspond to clusters of similar contexts for a word.

## Key Idea: Word Sense Induction

- Instead of using “dictionary defined senses”, extract the “senses from the corpus” itself
- These “corpus senses” or “uses” correspond to clusters of similar contexts for a word.



## *Detecting Root Hubs*

- Different uses of a target word form highly interconnected bundles (or high density components)
- In each high density component one of the nodes (hub) has a higher degree than the others.
- **Step 1:** Construct co-occurrence graph,  $G$ .

## *Detecting Root Hubs*

- Different uses of a target word form highly interconnected bundles (or high density components)
- In each high density component one of the nodes (hub) has a higher degree than the others.
- **Step 1:** Construct co-occurrence graph,  $G$ .
- **Step 2:** Arrange nodes in  $G$  in decreasing order of degree.

## Detecting Root Hubs

- Different uses of a target word form highly interconnected bundles (or high density components)
- In each high density component one of the nodes (hub) has a higher degree than the others.
- **Step 1:** Construct co-occurrence graph,  $G$ .
- **Step 2:** Arrange nodes in  $G$  in decreasing order of degree.
- **Step 3:** Select the node from  $G$  which has the highest degree. This node will be the hub of the first high density component.

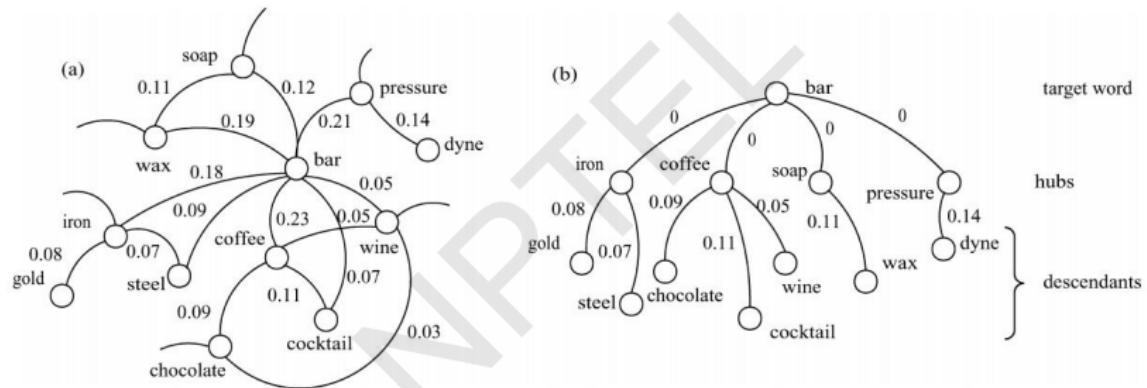
## Detecting Root Hubs

- Different uses of a target word form highly interconnected bundles (or high density components)
- In each high density component one of the nodes (hub) has a higher degree than the others.
- **Step 1:** Construct co-occurrence graph,  $G$ .
- **Step 2:** Arrange nodes in  $G$  in decreasing order of degree.
- **Step 3:** Select the node from  $G$  which has the highest degree. This node will be the hub of the first high density component.
- **Step 4:** Delete this hub and all its neighbors from  $G$ .

## Detecting Root Hubs

- Different uses of a target word form highly interconnected bundles (or high density components)
- In each high density component one of the nodes (hub) has a higher degree than the others.
- **Step 1:** Construct co-occurrence graph,  $G$ .
- **Step 2:** Arrange nodes in  $G$  in decreasing order of degree.
- **Step 3:** Select the node from  $G$  which has the highest degree. This node will be the hub of the first high density component.
- **Step 4:** Delete this hub and all its neighbors from  $G$ .
- **Step 5:** Repeat Step 3 and 4 to detect the hubs of other high density components

# HyperLex: Detecting Root Hubs



## *Delineating Components*

- Attach each node to the root hub closest to it.
- The distance between two nodes is measured as the smallest sum of weights of the edges on the paths linking them.

# Delineating Components

- Attach each node to the root hub closest to it.
- The distance between two nodes is measured as the smallest sum of weights of the edges on the paths linking them.

*Computing distance between two nodes  $w_i$  and  $w_j$*

$$w_{ij} = 1 - \max\{P(w_i|w_j), P(w_j|w_i)\}$$

where  $P(w_i|w_j) = \frac{\text{freq}_{ij}}{\text{freq}_j}$

# Disambiguation

- Let  $W = (w_1, w_2, \dots, w_i, \dots, w_n)$  be a context in which  $w_i$  is an instance of our target word.
- Let  $w_i$  has  $k$  hubs in its minimum spanning tree

# Disambiguation

- Let  $W = (w_1, w_2, \dots, w_i, \dots, w_n)$  be a context in which  $w_i$  is an instance of our target word.
- Let  $w_i$  has  $k$  hubs in its minimum spanning tree
- A score vector  $s$  is associated with each  $w_j \in W(j \neq i)$ , such that  $s_k$  represents the contribution of the  $k$ th hub as:

$$s_k = \frac{1}{1 + d(h_k, w_j)} \text{ if } h_k \text{ is an ancestor of } w_j$$
$$s_i = 0 \text{ otherwise.}$$

- All score vectors associated with all  $w_j \in W(j \neq i)$  are summed up
- The hub which receives the maximum score is chosen as the most appropriate sense

# *Novel Word Sense Detection*

Pawan Goyal

CSE, IIT Kharagpur

Week 8, Lecture 5

# *Tracking Sense Changes*

## *Classical sense*

sick



*adjective*

\sik\

: affected with a disease or illness

: of or relating to people who are ill

: very annoyed or bored by something because you have had too much of it

1

<sup>1</sup><http://www.merriam-webster.com/>

# Tracking Sense Changes

## Classical sense

sick  *adjective* \sik\

: affected with a disease or illness

: of or relating to people who are ill

: very annoyed or bored by something because you have had too much of it

1

## Novel sense



 Favorited 85,144 times

Niall Horan @NiallOfficial · Apr 24

Listening to Paulo nutini 's new record! It's sick !

[Collapse](#)

 Reply  Retweet  Favorite  More

RETWEETS

47,293

FAVORITES

85,145



11:50 PM - 24 Apr 2014 · Details

<sup>1</sup><http://www.merriam-webster.com/>

# *Comparing sense clusters*

- If a word undergoes sense change, this can be detected by comparing the sense clusters obtained from two different time periods

NPTEL

# Comparing sense clusters

- If a word undergoes sense change, this can be detected by comparing the sense clusters obtained from two different time periods

reporter/NN, auditor/NN, listener/NN, scribe/NN, translator/NN, writer/NN, reader/NN, editor/NN, author/NN, orator/NN, commentator/NN, composer/NN, biographer/NN, novelist/NN, ...

compiler/NN

scientist/NN, composer/NN, philosopher/NN, publisher/NN, preacher/NN, transcriber/NN, thinker/NN, teller/NN, statesman/NN, musician/NN, jurist/NN, essayist/NN, interpreter/NN, observer/NN, auditor/NN, experimenter/NN, artist/NN, dramatist/NN, ...

1909-1953

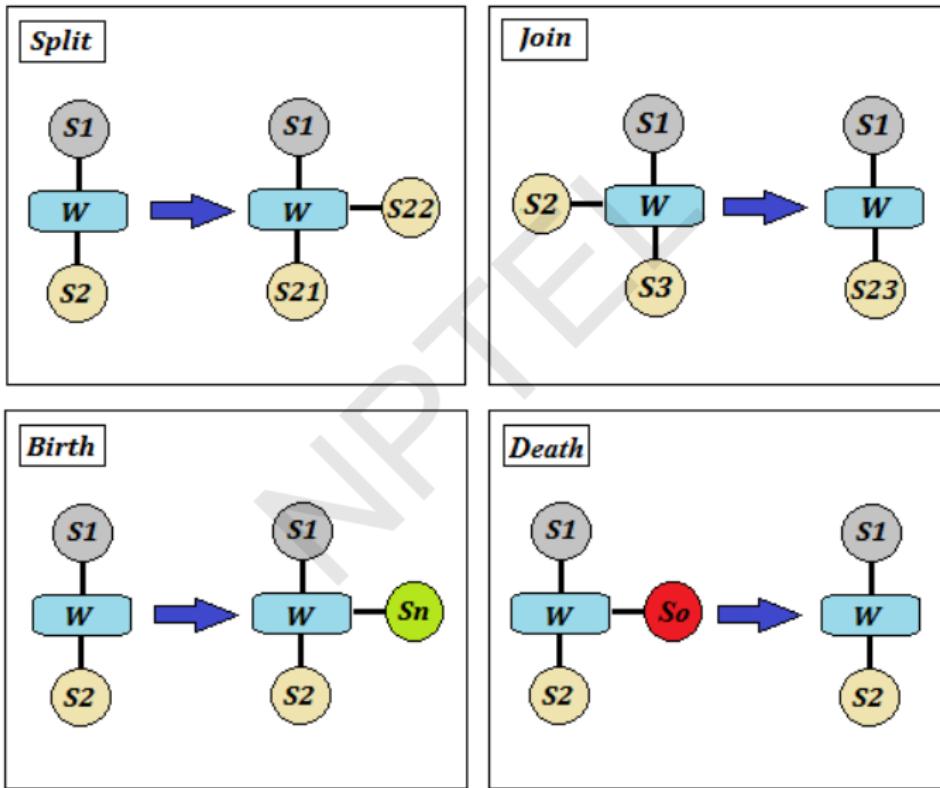
translator/NN, editor/NN, listener/NN, reader/NN, commentator/NN, author/NN, observer/NN, interpreter/NN, writer/NN, scribe/NN, redactor/NN, viewer/NN, ...

compiler/NN

preprocessor/NN, driver/NN, handler/NN, hardware/NN, software/NN, loader/NN, kernel/NN, dbms/NN, linker/NN, assembler/NN, scheduler/NN, debugger/NN, browsers/NNS, processors/NNS, parser/NN, subsystem/NN

2002-2005

# *Split, join, birth and death*



# *A real example of birth*

