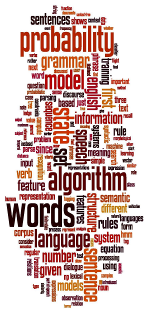
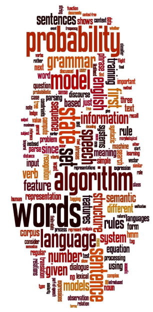


Module 2

Sentence Segmentation – Language Specific issues- Text Normalization – Stemming – Inflectional and Derivation Morphology – Morphological Analysis and Generation using Finite State Transducers – Introduction to poS Tagging – Hidden Markov Models for PoS Tagging – Viterbi Decoding for HMM



Basic Text Processing

Regular Expressions

Conversational Agents

Simple pattern-based methods assists in Answering questions, booking flights, or finds restaurants. Tool for describing pattern => **Regular Expression**

Dan Jurafsky

Text Normalization

- Normalizing text means converting it to a more convenient, standard form.
- Tasks in Normalization
  - Tokenization - Separating out or tokenizing words from running text
  - Lemmatization – Task of determining that two words have the same root
  - Stemming – Simpler version of lemmatization, striping suffixes from the end of the word.
  - Sentence Segmentation - Breaking up a text into individual sentences, using cues like sentence segmentation periods or exclamation points.
  - Edit Distance – A metric that measures how similar two strings are based on the number of edits (insertions, deletions, substitutions) it takes to change one string into the other

Dan Jurafsky

Regular expressions

- A formal language for specifying text strings
  - Concatenation – Characters in sequence
  - Regular Expressions – Case Sensitive
    - /s/ not same as /S/
      - -> [sS]
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Dan Jurafsky

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

- Ranges [A-Z]

Pattern	Matches	
[A-Z]	An upper case letter	Drenched Blossoms
[a-z]	A lower case letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole
[abc]	'a', 'b', or 'c'	"In uomini, in soldati"

Dan Jurafsky

Regular Expressions: Negation in Disjunction

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

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither ‘S’ nor ‘s’	`I have no exquisite reason”`
`[^e^]`	Neither e nor ^	`Look here`
`a^b`	The pattern a carat b	`Look up a^b now`

Dan Jurafsky

Regular Expressions: More Disjunction

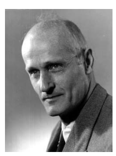
- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
`groundhog|woodchuck`	groundhog woodchuck
`yours|mine`	yours mine
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	Groundhog groundhog Woodchuck woodchuck

Dan Jurafsky

Regular Expressions: ? * + .

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



Stephen C Kleene

Kleene *, Kleene +

Dan Jurafsky

Regular Expressions: Anchors ^ $

Pattern	Matches
`^[A-Z]`	`Palo Alto`
`^[^A-Za-z]`	`1 “Hello”`
`\.$`	`The end.`
`.$`	`The end? The end!`

Dan Jurafsky

Example

- Find me all instances of the word “the” in a text.

`the` Misses capitalized examples

`[tT]he` Incorrectly returns `other` or `theology`

Dan Jurafsky

Errors

- The process we just went through was based on fixing two 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)

Dan Jurafsky

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves:
  - Increasing accuracy or precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

Dan Jurafsky

More Operators

Regex	Expansion	Match	First Matches
\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)	in Concord
\S	[^\s]	Non-whitespace	in␣Concord

Figure 2.8 Aliases for common sets of characters.

14

Dan Jurafsky

More Operators

/{3}/ means "exactly 3 occurrences of the previous character or expression"

/a\.{24}z/ will match a followed by 24 dots followed by z

/{n,m}/ specifies from n to m occurrences of the previous char or expression

/{n,}/ means at least n occurrences of the previous expression.

15

Dan Jurafsky

More Operators

Regex	Match
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	zero or one occurrence of the previous char or expression
{n}	exactly *n* occurrences of the previous char or expression
{n,m}	from *n* to *m* occurrences of the previous char or expression
{n,}	at least *n* occurrences of the previous char or expression
{,m}	up to *m* occurrences of the previous char or expression

Figure 2.9 Regular expression operators for counting.

16

Dan Jurafsky

More Operators

Regex	Match	First Patterns Matched
*	an asterisk "*"	"K*A*P*L*A*N"
\.	a period "."	"Dr. Livingston, I presume"
\?	a question mark	"Why don't they come and lend a hand?"
\n	a newline	
\t	a tab	

Figure 2.10 Some characters that need to be backslashed.

17

Dan Jurafsky

Capture Group

- The use of parentheses to store a pattern in memory is called a capture group.
- Every time a capture group is used (i.e., parentheses surround a pattern), the resulting match is stored in a numbered register.
- If you match two different sets of parentheses, \2 means whatever matched the second capture group.

Example

/the (.*)er they (.*), the \1er we \2/

will match the faster they ran, the faster we ran but not the faster they ran, the faster we ate.

18

Non Capturing group

- Use parentheses for grouping, but don’t want to capture the resulting pattern in a register.
- In that case use a non-capturing group, which is specified by putting the special commands ?: after the open parenthesis, in the form (?: pattern).

/(?:some|a few) (people|cats) like some \1/

=>Matches some cats like some cats

but does not match some cats like some some

Substitutions

- Substitution Operator - s/regexp1/pattern/

s/([0-9]+)/<\1>/

For example, suppose we wanted to put angle brackets around all integers in a text.
For example, changing the 35 boxes to the <35> boxes.

Sample Application - ELIZA

How ELIZA works

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?/

Summary

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

Digital Assignment 1

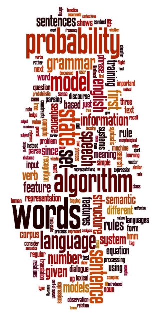
- Create a simple chatbot using regular expressions

Write regular expressions for the following languages.

1. the set of all alphabetic strings
2. the set of all lower case alphabetic strings ending in a b;
3. the set of all strings from the alphabet a,b such that each a is immediately preceded by and immediately followed by a b;

Dan Jurafsky

- Write regular expressions for the following languages. By “word”, we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line breaks, and so forth.
- 1. The set of all strings with two consecutive repeated words (e.g., “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);
- 2. All strings that start at the beginning of the line with an integer and that end at the end of the line with a word;
- 3. All strings that have both the word grotto and the word raven in them (but not, e.g., words like grottos that merely contain the word grotto);



Basic Text Processing

Word tokenization

Dan Jurafsky

Text Normalization

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

Dan Jurafsky

How many words?

- Two kinds of disfluencies in running text
  - Fragments(incomplete words), filled pauses(uh,um…)
  - I do uh main- mainly business data processing
- Word types and word instances
  - Word types – No.of distinct words in the corpus
  - Word instances – No.of. Running words

Dan Jurafsky

Lemma , Word Form and Lemmatization

- A **lemma/ citation form**
  - Grammatical for used to represent a lexeme.
  - The lemma or citation form for sing, sang, sung is sing
- The **wordform** is the full inflected or derived form of the word.
  - Sing, sang, sung
- The process of mapping from a wordform to a lemma is called **lemmatization**.

Dan Jurafsky

How many words?

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

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

Dan Jurafsky

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{½})$

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

Dan Jurafsky

Herdan's Law

- Relationship between the number of types(Vocabulary) |V| and number of instances(Tokens) N is called Herdan's Law (Herdan, 1960) or Heaps' Law (Heaps, 1978) after its discoverers Heaps' Law (in linguistics and information retrieval respectively).

$$|V| = kN^{\beta}$$

- K(30<=k<=100) and β are positive constants,

  and 0 < β < 1(Probably 0.5)

Note :

- Vocabulary growth depends a lot on the nature of the collection and how it is processed.
- Case-folding and stemming reduce the growth rate of the vocabulary.
- Including numbers and spelling errors increase it.

Dan Jurafsky

Word Tokenization

- Three Algorithms
  - Byte Pair Encoding(BPE)
  - Unigram Language Modeling Tokenization
  - Wordpiece
- All have 2 parts
  - A token Learner - > Takes a raw training corpus and induces Vocabulary(A set of tokens)
  - A toke Segmenter that takes a raw sentence and tokenizes according to the vocabulary.

Dan Jurafsky

Byte Pair Encoding

- Consider the Vocabulary { A, B, C, …….a,b,c…..}
- Repeat :
  - Choose the two symbols that are more frequently adjacent in the training corpus(say 'A' and 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every occurrence of 'A' 'B' with 'AB'
- Until "k" merges are done
- The Byte Pair Encoding algorithm is a commonly-used tokenizer that is found in many transformer models such as GPT and GPT-2 models

Dan Jurafsky

BPE Learner

Original (very fascinating😏) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

vocabulary
_, d, e, i, l, n, o, r, s, t, w

corpus representation
5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

Dan Jurafsky

BPE Token Learner

corpus
5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

vocabulary
_, d, e, i, l, n, o, r, s, t, w

Merge e r to er

corpus
5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary
_, d, e, i, l, n, o, r, s, t, w, er

Dan Jurafsky

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Merge er _ to er_

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

37

Dan Jurafsky

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Merge n e to ne

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

38

Dan Jurafsky

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

39

Dan Jurafsky

BPE Token Segmenter

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

Result:

- Test set "n e w e r _" would be tokenized as a full word

- But "lower_" would be tokenized as 2 new tokens : low and er_

40

Dan Jurafsky

- Construct the vocabulary for the given corpus :

cat cat cat cat cat cats cats eat eat eat eat eat eat eat eat eat eat eating eating eating running running jumping food food food food food food

41

Dan Jurafsky

- INITIAL CORPUS:
  [(['c', 'a', 't'], 5), (['c', 'a', 't', 's'], 2), (['e', 'a', 't'], 10), (['e', 'a', 't', 'i', 'n', 'g'], 3), (['r', 'u', 'n', 'n', 'i', 'n', 'g'], 2), (['j', 'u', 'm', 'p', 'i', 'n', 'g'], 1), (['f', 'o', 'o', 'd'], 6)]

  NEW MERGE RULE: Combine "a" and "t"
  [(['c', 'at'], 5), (['c', 'at', 's'], 2), (['e', 'at'], 10), (['e', 'at', 'i', 'n', 'g'], 3), (['r', 'u', 'n', 'n', 'i', 'n', 'g'], 2), (['j', 'u', 'm', 'p', 'i', 'n', 'g'], 1), (['f', 'o', 'o', 'd'], 6)]

42

Dan Jurafsky

- NEW MERGE RULE: Combine "e" and "at"
  [(['c', 'at'], 5), (['c', 'at', 's'], 2), (['eat'], 10),
  (['eat', 'i', 'n', 'g'], 3), (['r', 'u', 'n', 'n', 'i', 'n', 'g'], 2),
  (['j', 'u', 'm', 'p', 'i', 'n', 'g'], 1), (['f', 'o', 'o', 'd'], 6)]

  NEW MERGE RULE: Combine "c" and "at"
  [(['cat'], 5), (['cat', 's'], 2), (['eat'], 10), (['eat', 'i', 'n', 'g'], 3),
  (['r', 'u', 'n', 'n', 'i', 'n', 'g'], 2),
  (['j', 'u', 'm', 'p', 'i', 'n', 'g'], 1), (['f', 'o', 'o', 'd'], 6)]

43

Dan Jurafsky

- NEW MERGE RULE: Combine "i" and "n"
  [(['cat'], 5), (['cat', 's'], 2), (['eat'], 10), (['eat', 'in', 'g'], 3),
  (['r', 'u', 'n', 'n', 'in', 'g'], 2), (['j', 'u', 'm', 'p', 'in', 'g'], 1),
  (['f', 'o', 'o', 'd'], 6)]

  NEW MERGE RULE: Combine "in" and "g"
  [(['cat'], 5), (['cat', 's'], 2), (['eat'], 10), (['eat', 'ing'], 3),
  (['r', 'u', 'n', 'n', 'ing'], 2), (['j', 'u', 'm', 'p', 'ing'], 1),
  (['f', 'o', 'o', 'd'], 6)]

44

Dan Jurafsky

Language Issues

45

Dan Jurafsky

Issues in Tokenization

Clitic – A clitic is a part of a word that can't stand on its own, and can only occur when it is attached to another word.

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

Dan Jurafsky

Tokenization: language issues

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

Dan Jurafsky

Tokenization: language issues

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

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romaji

End-user can express query entirely in hiragana!

Collocation

- Selection of Collocation by frequency
- Selection of Collocation by mean
- Selection of Collocation by variance of the distance between the focal word and collocating word
- Hypothesis testing
- Mutual Information

49

Basic Text Processing

Word Normalization and Stemming

Dan Jurafsky

Normalization

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

Dan Jurafsky

Case folding

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

Dan Jurafsky

Lemmatization and Stemming

53

Dan Jurafsky

Lemmatization

How is lemmatization done?

- The most sophisticated methods for lemmatization involve complete morphological parsing of the word.
- **Morphemes**:
  - The small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Bits and pieces that adhere to stems
    - Often with grammatical functions
- Reduce inflections or variant forms to base form
  - *1. am, are, is → be* *2. car, cars, car's, cars' → car*

    the boy's cars are different colors → the boy car be different color
- Lemmatization: have to find correct dictionary head wordform

Dan Jurafsky

Stemming

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

for example compressed and compression are both accepted as equivalent to compress. → for exampl compress and compress ar both accept as equival to compress

---

Dan Jurafsky

Stemmer Algorithms

- Porter Stemmer -
- Encodings of the Basic Algorithm - [Porter Stemming Algorithm (tartarus.org)]
- Snowball Stemmer
- Lancaster Stemmer

56

Dan Jurafsky

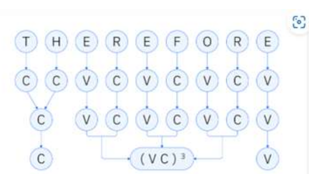
Porter Stemmer - Definitions

- A \consonant\ in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant.
- If a letter is not a consonant it is a \vowel\.
- A consonant will be denoted by c, a vowel by v.
- A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:
  - CVCV ... C
  - CVCV ... V
  - VCVC ... C
  - These may all be represented by the single form **[C]VCVC ... [V]** where the square brackets denote arbitrary presence of their contents. Using (VC){m} to denote VC repeated m times, this may again be written as,

    [C](VC){m}[V].
- m will be called the \measure\ of any word or word part when represented in this form.

57

Dan Jurafsky



58

Dan Jurafsky

- Examples
  - m=0 TR, EE, TREE, Y, BY.
  - m=1 TROUBLE, OATS, TREES, IVY.
  - m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

59

Dan Jurafsky

Rules for removing Suffix

- The \rules\ for removing a suffix will be given in the form (condition) S1 -> S2
- The condition is usually given in terms of m,
  - e.g. (m > 1) EMENT ->
- The `condition' part may also contain the following:
  - *S - the stem ends with S (and similarly for the other letters).
  - *v* - the stem contains a vowel.
  - *d - the stem ends with a double consonant (e.g. -TT, -SS).
  - *o - the stem ends cvc, where the second c is not W, X or Y (e.g.-WIL, -HOP).

60

- the condition part may also contain expressions with \and\, \or\ and \not\

Dan Jurafsky

Rules - Examples

Rule	Description
(m>1 and (*S or *T))	Tests for a stem with m>1 ending in S or T
(*d and not (*L or *S or *Z))	Tests for a stem ending with a double consonant other than L, S or Z

61

Dan Jurafsky

Steps in Porter Stemmer

- **Step 1**
  - **Sub-step 1a**: Remove plurals and past participles.
  - **Sub-step 1b**: Handle common suffixes like 'ed' and 'ing'.
  - **Sub-step 1c**: Transform 'y' to 'i' if it follows a consonant.
- **Step 2 - 4**
  - **Derivational Morphology** (Ational - > Ate , Ization -> ize,……)
- **Step 5**
  - Cleanup

62

Dan Jurafsky

Step 1

Step 1a

1. SSES → SS
2. IES → I
3. SS → SS
4. S →

Step 1b

1. (m>0) EED → EE
2. (*v*) ED →
3. (*v*) ING →

If the second or third of the rules in Step 1b is successful, the following is performed.

1. AT → ATE
2. BL → BLE
3. IZ → IZE
4. (*d and not (*L or *S or *Z)) → single letter
5. (m=1 and *o) → E

63

Dan Jurafsky

Step 2

1. (m>0) ATIONAL → ATE
2. (m>0) TIONAL → TION
3. (m>0) ENCI → ENCE
4. (m>0) ANCI → ANCE
5. (m>0) IZER → IZE
6. (m>0) ABLI → ABLE
7. (m>0) ALLI → AL
8. (m>0) ENTLI → ENT
9. (m>0) ELI → E
10. (m>0) OUSLI → OUS
11. (m>0) IZATION → IZE
12. (m>0) ATION → ATE
13. (m>0) ATOR → ATE
14. (m>0) ALISM → AL
15. (m>0) IVENESS → IVE
16. (m>0) FULNESS → FUL
17. (m>0) OUSNESS → OUS
18. (m>0) ALITI → AL
19. (m>0) IVITI → IVE
20. (m>0) BILITI → BLE

Step 3

1. (m>0) ICATE → IC
2. (m>0) ATIVE →
3. (m>0) ALIZE → AL
4. (m>0) ICITI → IC
5. (m>0) ICAL → IC
6. (m>0) FUL →
7. (m>0) NESS →

64

Dan Jurafsky

Step 4

1. (m>1) AL →
2. (m>1) ANCE →
3. (m>1) ENCE →
4. (m>1) ER →
5. (m>1) IC →
6. (m>1) ABLE →
7. (m>1) IBLE →
8. (m>1) ANT →
9. (m>1) EMENT →
10. (m>1) MENT →
11. (m>1) ENT →
12. (m>1 and (*S or *T)) ION →
13. (m>1) OU →
14. (m>1) ISM →
15. (m>1) ATE →
16. (m>1) ITI →
17. (m>1) OUS →
18. (m>1) IVE →
19. (m>1) IZE →

Step 5

1. (m>1) E →
2. (m=1 and not *o) E →

1. (m > 1 and *d and *L) → single letter

65

Dan Jurafsky

Porter's algorithm
The most common English stemmer

Step 1a

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

Step 1b

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

Step 2 (for long stems)

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

Step 3 (for longer stems)

al → ø revival → reviv
able → ø adjustable → adjust
ate → ø activate → activ
…

Dan Jurafsky

Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

```
(*v*)ing → ø  walking  → walk
              singing  → sing
```

67

Dan Jurafsky

Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

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

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

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

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

68

Dan Jurafsky

Dealing with complex morphology is sometimes necessary

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

Dan Jurafsky

Edit Distance

- Dynamic Programming

```
INTE*NTION
||||||||||
*EXECUTION
d s s  i s
```

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

70

Dan Jurafsky

Edit Distance

- Dynamic Programming

```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix D[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
  D[0,0] = 0
  for each row i from 1 to n do
     D[i,0] ← D[i-1,0] + del-cost(source[i])
  for each column j from 1 to m do
     D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
       for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i],target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```

71

Dan Jurafsky

Edit Distance

- Dynamic Programming

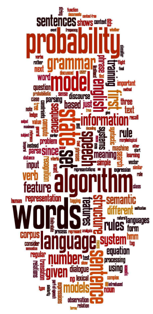
Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

72

Dan Jurafsky

Backtracing

	#	e	x	e	c	u	t	i	o	n
#	0	← 1	← 2	← 3	← 4	← 5	← 6	← 7	← 8	← 9
i	↑ **1**	↖←↑ 2	↖←↑ 3	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖ 6	← 7	← 8
n	↑ 2	↖←↑ **3**	↖←↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↑ 7	↖←↑ 8	↖ 7
t	↑ 3	↖←↑ 4	↖←↑ **5**	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖ 7	←↑ 8	↖←↑ 9	↑ 8
e	↑ 4	↖ 3	← 4	↖← **5**	← **6**	← 7	←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9
n	↑ 5	↑ 4	↖←↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ **8**	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖↑ 10
t	↑ 6	↑ 5	↖←↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖ **8**	← 9	← 10	←↑ 11
i	↑ 7	↑ 6	↖←↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↑ 9	↖ **8**	← 9	← 10
o	↑ 8	↑ 7	↖←↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↑ 10	↑ 9	↖ **8**	← 9
n	↑ 9	↑ 8	↖←↑ 9	↖←↑ 10	↖←↑ 11	↖←↑ 12	↑ 11	↑ 10	↑ 9	↖ **8**

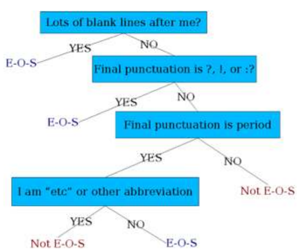


Basic Text Processing

Sentence Segmentation and Decision Trees

Dan Jurafsky

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



Dan Jurafsky

More sophisticated decision tree features

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

Dan Jurafsky

Implementing Decision Trees

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

Dan Jurafsky

Decision Trees and other classifiers

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