Lecture 6: Statistical Methods & Pattern Matching

NLP

Model Basics: Definitions

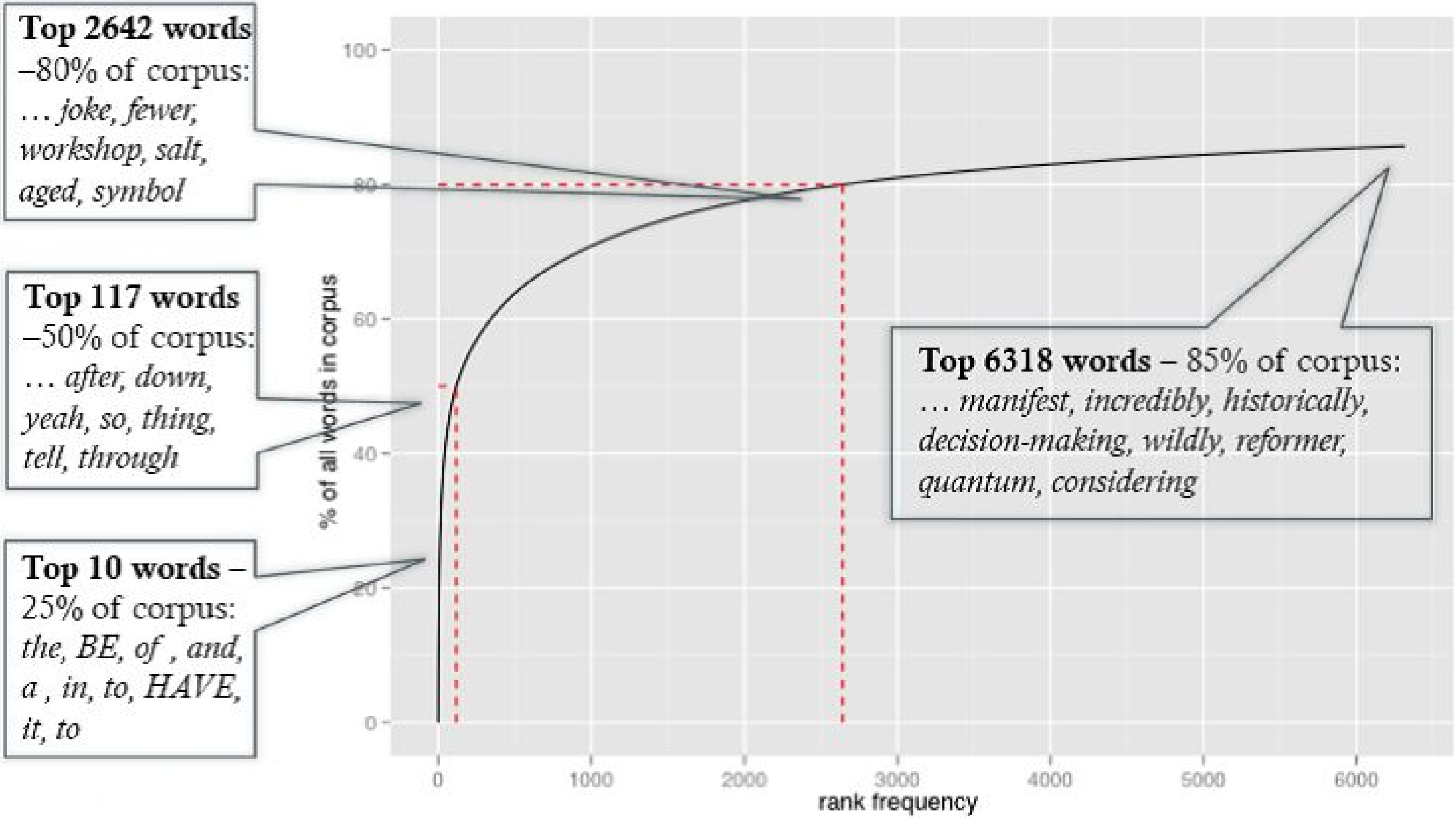
- A *corpus* is a set of *documents*. Each *document* is a collection of *terms*.
  - Example: The set of all tweets in assignment 2 is a corpus **D**. Each tweet is a document **d** in **D**, and each word (or hashtag, etc) is a term **t** in **d**.

- For each term and document, define the *term frequency*
  - **tf(*t,d*) = # of occurrences of t in d / # of terms in d**

Bag of Words

- Model each document using term frequency as a weight, or probability of each term’s occurrence.
  - “Mary is quicker than John”
  - “John is quicker than Mary”

- Problem: Some words are more common than others
  - Solution: Some words are more important than others



Top 2642 words –80% of corpus: … *joke, fewer, workshop, salt, aged, symbol*
Top 117 words –50% of corpus: … *after, down, yeah, so, thing, tell, through*
Top 10 words – 25% of corpus: *the, BE, of , and, a , in, to, HAVE, it, to*
Top 6318 words – 85% of corpus: … *manifest, incredibly, historically, decision-making, wildly, reformer, quantum, considering*
% of all words in corpus
rank frequency
0
20
40
60
80
100
0
1000
2000
3000
4000
5000
6000

Stop words

Just ignore them

TFIDF

- Define the *document frequency* as the proportion of documents in a corpus **D** for which a specific term ***t*** occurs
  - **df(*t*,*D*) = # of documents containing *t* / # of documents in *D***

- **tfidf(t,d,D) = tf(t,d) * log(1 / df(t,D))**

TFIDF properties

Straight from Wikipedia:

The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

TFIDF example

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

Probabilistic Models

- Simple approach: Let the probability of a term occurring be the term frequency
  - **P(t) = tf(t,d)**

- Problem: Some words don't occur at all in the corpus, or only in some documents, which gives **P(t) = 0** for unseen events

Add-One Smoothing

- Assume every (seen or unseen) event occurred once more than it did in the training data.

- **P(t,d) = tf(t,d) + 1 / |d| + |V|**

- Where **V** is the *vocabulary*, or unique set of all terms being considered

Expanding to N-Grams

- Unigrams: Only terms
- P("You like green cream")≈ P("You")P("like")P("green")P("cream")
  - Overestimates probability for this rare sentence since all words in it are fairly common.

- Bigram Model: Prob of next word depends only on last word.
  - $P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-1})$
  - P("You like green cream")≈ P("You")P("like"|"You")P("green"|"like")P("cream"|"green")

Regular Expressions

- A regex is a pattern enclosed within delimiters.
- Most characters match themselves.
- r'REGex' is a regular expression that matches “REGex”.
    - ' is the delimiter enclosing the expression.
    - “REGex” is the pattern.

These slides are based on Ben Brumfield’s slides as presented at THATCamp Texas 2011.

at

- Matches strings with “a” followed by “t”.

at	hat
that	atlas
aft	Athens

at

- Matches strings with “a” followed by “t”.

at	h***at***
th***at***	***at***las
aft	Athens

Characters

- Matching is case sensitive, but you can use the re.IGNORECASE (re.I works too) to tell Python to ignore case if you want
- Special characters: ( ) ^ $ { } [ ] \ | . + ? *
- To match a special character in your text, precede it with \ in your pattern:
  - ironic [sic] does not match “ironic [sic]”
  - ironic \[sic\] matches “ironic [sic]”

These slides are based on Ben Brumfield’s slides as presented at THATCamp Texas 2011.

Character Classes

- Characters within [ ] are choices for a single-character match.
- Think of a set operation, or a type of *or*.
- Order within the set is unimportant.
- x[01] matches “x0” and “x1”.
- [10][23] matches “02”, “03”, “12” and “13”.
- Initial^ negates the class:
  - [^45] matches all characters except 4 or 5

[ch]at

- Matches strings with “c” or “h”, followed by “a”, followed by “t”.

that	at
chat	cat
fat	phat

[ch]at

- Matches strings with “c” or “h”, followed by “a”, followed by “t”.

t***hat***	at
c***hat***	***cat***
fat	p***hat***

Ranges

- Ranges define sets of characters within a class.
  - [1-9] matches any non-zero digit.
  - [a-zA-Z] matches any letter.
  - [12][0-9] matches numbers between 10 and 29.

Shortcuts

Shortcut	Name	Equivalent Class
\d	digit	[0-9]
\D	not digit	[^0-9]
\w	word	[a-zA-Z0-9_]
\W	not word	[^a-zA-Z0-9_]
\s	space	[\t\n\r\f\v]
\S	not space	[^\t\n\r\f\v]
.	everything	[^\n] (depends on mode*)

These slides are based on Ben Brumfield’s slides as presented at THATCamp Texas 2011.

\d\d\d[-]\d\d\d\d

- Matches strings with:
  - Three digits
  - **Space** or dash
  - Four digits

501-1234	234 1252
652.2648	713-342-7452
PE6-5000	653-6464x256

\d\d\d[-]\d\d\d\d

- Matches strings with:
  - Three digits
  - Space or dash
  - Four digits

501-1234	***234 1252***
652.2648	713-***342-7452***
PE6-5000	***653-6464***x256

Repeaters

- Symbols indicating that the preceding element of the pattern can repeat.
- Runs? matches runs or run
- 1\d* matches any number beginning with "1".

Repeater	Count
?	zero or one
+	one or more
*	zero or more
{*n*}	exactly *n*
{*n*,*m*}	between *n* and *m* times
{,*m*}	no more than *m* times
{*n*,}	at least *n* times

These slides are based on Ben Brumfield's slides as presented at THATCamp Texas 2011.

Repeaters

Strings:

1: “at” 2: “art”

3: “arrrrt” 4: “aft”

Patterns:

A: ar?t B: a[fr]?t

C: ar*t D: ar+t

E: a.*t F: a.+t

Repeater	Count
?	zero or one
+	one or more
*	zero or more
{*n*}	exactly *n*
{*n*,*m*}	between *n* and *m* times
{,*m*}	no more than *m* times
{*n*,}	at least *n* times

These slides are based on Ben Brumfield’s slides as presented at THATCamp Texas 2011.

Repeaters

Strings:	Matches:
1: “at”	**ar?t**, **a[fr]?t**, **ar*t**, **a.*t**
2: “art”	**ar?t**, **a[fr]?t**, **ar*t**, **ar+t**, **a.*t**, **a.+t**
3: “arrrrt”	**ar*t**, **ar+t**, **a.*t**, **a.+t**
4: “aft”	**a[fr]?t**, **a.*t**, **a.+t**

Anchors

- Anchors match between characters.
- Used to assert that the characters you're matching must appear in a certain place.
- \bat\b matches "at work" but not "batch".

Anchor	Matches
^	start of line
$	end of line
\b	word boundary
\B	not boundary
\A	start of string
\Z	end of string
\z	raw end of string (rare)

Logical Or

- In Regex, | means “or”.
- You can put a full expression on the left and another full expression on the right.
- Either can match.
- r"s?he can’t|s?he cannot" matches “she can’t”, “she cannot”, “he can’t”, and “he cannot”

Grouping

- Everything within (…) is grouped into a single element for the purposes of repetition and alternation.
- The expression (la)+ matches “la”, “lala”, “lalalala” but not “all”.
- \bschema(ta)?\b matches “schema” and “schemata” but not “schematic”.

Grouping Example

- What regular expression matches “eat”, “eats”, “ate” and “eaten”?

Grouping Example

- What regular expression matches “eat”, “eats”, “ate” and “eaten”?
- eat(s|en)?|ate

- Add word boundary anchors to exclude “sate” and “eating”: \b(eat(s|en)?|ate)\b

Capture

- During searches, (…) groups capture patterns for use in replacement.
- Special variables \1, \2, \3 etc. contain the capture.
- (\d\d\d)-(\d\d\d\d)   “123-4567”
  - \1 contains “123”
  - \2 contains “4567”

Capture

- How do you convert
  - “Smith, James” and “Jones, Sally” to
  - “James Smith” and “Sally Jones”?

Capture

- How do you convert
  - “Smith, James” and “Jones, Sally” to
  - “James Smith” and “Sally Jones”?

- `import re`
- `names = ['Smith, James', 'Jones, Sally']`
- `pat = re.compile(r'(\w+), (\w+)’)`
- `matches = [re.search(pat, name) for name in names]`
- `newnames = [match.group(2) + " " + match.group(1) for match in matches]`
- `newnames`

These slides are based on Ben Brumfield’s slides as presented at THATCamp Texas 2011.

Backreference

- How do you identify variations on generative memes such as
  - “The blind leading the blind” becoming “The foolish leading the foolish”, for example?

Backreference

```
> import re
> pat = re.compile(r'the (\w+) leading the \1', re.I)
> tocheck = {'original': 'The blind leading the blind', 'variation': 'the foolish leading the foolish', 'redherring': 'The kid leading the dog'}
> for key in tocheck:
…  if re.search(pat, tocheck[key]):
…      print key
…
original
```

Capture with backreference

- How do you convert
  - “Smith, James” and “Jones, Sally” to
  - “James Smith” and “Sally Jones”?

- `import re`
- `names = ['Smith, James', 'Jones, Sally']`
- `pat = re.compile(r'(\w+), (\w+)’)`
- `newnames = [re.sub(pat, r'\2 \1', name) for name in names]`
- `newnames`

```
['James Smith', 'Sally Jones']
```