Lecture 2:
Words and Basic Text Processing

CS 585, Fall 2017
Introduction to Natural Language Processing
http://people.cs.umass.edu/~brenocon/inlp2017

Brendan O’Connor
College of Information and Computer Sciences
University of Massachusetts Amherst

[Includes slides from SLP3 site]
Announcements

- Currently
  - HW0 due tomorrow
  - HW1 released next week

- HW0: Gradescope submission
• Collaboration policy
  • All of the content you submit, both code and text, needs to be produced independently.
  • You may discuss problems. List your collaborators you worked with.
  • Do not share code or written materials.
  • Cite sources.
• Course website will have more complete version.
• NLP is an active research area!
Today

- Python demo
- Basic text processing
- Word counts
• This weekend: make sure you can run Python
  • Recommended: Anaconda Python
    https://www.continuum.io/downloads
  • Python 2.7
  • IPython Notebook  http://ipython.org/notebook.html
• Python interactive interpreter
• Python scripts
Text normalization

• Every NLP task needs text normalization
  • 1. Segment/tokenize words in running text
  • 2. Normalizing word formats
  • 3. Sentence segmentation (typically)
Regular expressions

• A formal language for specifying text strings
• How can we search for any of these?
  • woodchuck
  • woodchucks
  • Woodchuck
  • Woodchucks
Regular Expressions: Disjunctions

• Letters inside square brackets []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[wW]oodchuck</td>
<td>Woodchuck, woodchuck</td>
</tr>
<tr>
<td>[1234567890]</td>
<td>Any digit</td>
</tr>
</tbody>
</table>

• Ranges [A–Z]

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A–Z]</td>
<td>An upper case letter</td>
</tr>
<tr>
<td>[a–z]</td>
<td>A lower case letter</td>
</tr>
<tr>
<td>[0–9]</td>
<td>A single digit</td>
</tr>
<tr>
<td></td>
<td>Drenched Blossoms</td>
</tr>
<tr>
<td></td>
<td>my beans were impatient</td>
</tr>
<tr>
<td></td>
<td>Chapter 1: Down the Rabbit Hole</td>
</tr>
</tbody>
</table>
Regular Expressions: Negation in Disjunction

• Negations \[^Ss\]
  • Carat means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A–Z]</td>
<td>Not an upper case letter</td>
<td>Oyfn pripetchik</td>
</tr>
<tr>
<td>[^Ss]</td>
<td>Neither ‘S’ nor ‘s’</td>
<td>I have no exquisite reason”</td>
</tr>
<tr>
<td>[^e]</td>
<td>Neither e nor ^</td>
<td>Look here</td>
</tr>
<tr>
<td>a^b</td>
<td>The pattern a carat b</td>
<td>Look up a^b_now</td>
</tr>
</tbody>
</table>
Regular Expressions: More Disjunction

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

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>groundhog</td>
<td>woodchuck</td>
</tr>
<tr>
<td>yours</td>
<td>mine</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>[gG]roundhog</td>
<td>[Ww]oodchuck</td>
</tr>
</tbody>
</table>
## Regular Expressions: \? \* \+ \.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>colou?r</td>
<td>Optional previous char</td>
</tr>
<tr>
<td>oo*h!</td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td>o+h!</td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td>baa+</td>
<td></td>
</tr>
<tr>
<td>beg.n</td>
<td></td>
</tr>
</tbody>
</table>

Stephen C Kleene
Kleene *,  Kleene +
3 #ALLLIVESMATTER
1 #AllJonasLivesMatter
81 #AllLivesMatter
1 #Alllivesmatter
2 #AmericanLivesMatter
2 #ArmenianLivesMatter
20 #BLACKLIVESMATTER
1 #BLACKLivesMatter
4 #BLackLivesMatter
1 #BearLivesMatter
1 #BeerLivesMatter
1 #BeigeLivesMatter
948 #BlackLivesMatter
1 #BlackLlivesMatter
1 #BlackMuslimLivesMatter
1 #BlackTransLivesMatter
1 #BlacklLivesMatter
2 #BlacklivesMatter
26 #Blacklivesmatter
1 #Blackslivesmatter
2 #BlueLlivesMatter
90 #BlueLivesMatter
1 #Bluelivesmatter
1 #BookoutLivesMatter
2 #BrownLivesMatter
1 #BugLivesMatter
1 #CatsLivesMatter
1 #Chickenlivesmatter
grep -Poi '#\^[ ]*livesmatter'

grep -Poi '#[a-z0-9]*livesmatter'
Example

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

  the

  Misses capitalized examples

  [tT]he

  Incorrectly returns other or theology

  [^a-zA-Z][tT]he[^a-zA-Z]
Example

• Find me all instances of the word “the” in a text.
  the
  Misses capitalized examples
  [tT]he
  Incorrectly returns other or theology
  [^a-zA-Z][tT]he[^a-zA-Z]

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)
Errors cont.

- In NLP we are always dealing with these kinds of errors.
- 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).
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
Simple Tokenization in UNIX

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

```
tr -sc 'A-Za-z' '
' < shakes.txt
  | sort        Sort in alphabetical order
  | uniq -c     Merge and count each type
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1945</td>
<td>A</td>
</tr>
<tr>
<td>72</td>
<td>AARON</td>
</tr>
<tr>
<td>19</td>
<td>ABBESS</td>
</tr>
<tr>
<td>5</td>
<td>ABBOT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Aaron</td>
</tr>
<tr>
<td>6</td>
<td>Abate</td>
</tr>
<tr>
<td>1</td>
<td>Abates</td>
</tr>
<tr>
<td>5</td>
<td>Abbess</td>
</tr>
<tr>
<td>6</td>
<td>Abbey</td>
</tr>
<tr>
<td>3</td>
<td>Abbot</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Issues in Tokenization

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

• French
  • *L'ensemble* → one token or two?
    • *L* ? *L’* ? *Le* ?
    • Want *l’ensemble* to match with *un ensemble*

• German noun compounds are not segmented
  • *Lebensversicherungsgesellschaftsangestellter*
  • ‘life insurance company employee’
  • German information retrieval needs *compound splitter*
How many words?

\[ N = \text{number of tokens} \]
\[ V = \text{vocabulary} = \text{set of types} \]
\[ |V| \text{ is the size of the vocabulary} \]

Church and Gale (1990): \[ |V| > \mathcal{O}(N^{\frac{1}{2}}) \]

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

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>1629</td>
</tr>
<tr>
<td>and</td>
<td>844</td>
</tr>
<tr>
<td>to</td>
<td>721</td>
</tr>
<tr>
<td>a</td>
<td>627</td>
</tr>
<tr>
<td>she</td>
<td>537</td>
</tr>
<tr>
<td>it</td>
<td>526</td>
</tr>
<tr>
<td>of</td>
<td>508</td>
</tr>
<tr>
<td>said</td>
<td>462</td>
</tr>
<tr>
<td>i</td>
<td>400</td>
</tr>
<tr>
<td>alice</td>
<td>385</td>
</tr>
</tbody>
</table>

*Alice’s Adventures in Wonderland*, by Lewis Carroll
Zipf’s Law

- When word types are ranked by frequency, then frequency \( f \) * rank \( r \) is roughly equal to some constant \( k \)

\[ f \times r = k \]
<table>
<thead>
<tr>
<th>Rank (r)</th>
<th>Word</th>
<th>Frequency (f)</th>
<th>r \cdot f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>the</td>
<td>1629</td>
<td>1629</td>
</tr>
<tr>
<td>2</td>
<td>and</td>
<td>844</td>
<td>1688</td>
</tr>
<tr>
<td>3</td>
<td>to</td>
<td>721</td>
<td>2163</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>627</td>
<td>2508</td>
</tr>
<tr>
<td>5</td>
<td>she</td>
<td>537</td>
<td>2685</td>
</tr>
<tr>
<td>6</td>
<td>it</td>
<td>526</td>
<td>3156</td>
</tr>
<tr>
<td>7</td>
<td>of</td>
<td>508</td>
<td>3556</td>
</tr>
<tr>
<td>8</td>
<td>said</td>
<td>462</td>
<td>3696</td>
</tr>
<tr>
<td>9</td>
<td>i</td>
<td>400</td>
<td>3600</td>
</tr>
<tr>
<td>10</td>
<td>alice</td>
<td>385</td>
<td>3850</td>
</tr>
<tr>
<td>20</td>
<td>all</td>
<td>179</td>
<td>3580</td>
</tr>
<tr>
<td>30</td>
<td>little</td>
<td>128</td>
<td>3840</td>
</tr>
<tr>
<td>40</td>
<td>about</td>
<td>94</td>
<td>3760</td>
</tr>
<tr>
<td>50</td>
<td>again</td>
<td>82</td>
<td>4100</td>
</tr>
<tr>
<td>60</td>
<td>queen</td>
<td>68</td>
<td>4080</td>
</tr>
<tr>
<td>70</td>
<td>don’t</td>
<td>60</td>
<td>4200</td>
</tr>
<tr>
<td>80</td>
<td>quite</td>
<td>55</td>
<td>4400</td>
</tr>
<tr>
<td>90</td>
<td>just</td>
<td>51</td>
<td>4590</td>
</tr>
<tr>
<td>100</td>
<td>voice</td>
<td>47</td>
<td>4700</td>
</tr>
<tr>
<td>200</td>
<td>hand</td>
<td>20</td>
<td>4000</td>
</tr>
<tr>
<td>300</td>
<td>turning</td>
<td>12</td>
<td>3600</td>
</tr>
<tr>
<td>400</td>
<td>hall</td>
<td>9</td>
<td>3600</td>
</tr>
<tr>
<td>500</td>
<td>kind</td>
<td>7</td>
<td>3500</td>
</tr>
</tbody>
</table>
Plot: log frequencies

Recall:

\[ f^* r = k \]

\[ \log f + \log r = \log k \]
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
Case folding

- Applications like IR: reduce all letters to lower case
  - Since users tend to use lower case
  - Possible exception: upper case in mid-sentence?
    - e.g., General Motors
    - Fed vs. fed
    - SAIL vs. sail

- For sentiment analysis, MT, Information extraction
  - Case is helpful (US versus us is important)
Lemmatization

• Reduce inflections or variant forms to base form
  • *am, are, is* → *be*
  • *car, cars, car's, cars'* → *car*
  • *the boy's cars are different colors* → *the boy car be different color*

• Lemmatization: have to find correct dictionary headword form

• Machine translation
  • Spanish *quiero* (‘I want’), *quieres* (‘you want’) same lemma as *querer* ‘want’
Morphology

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

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

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

*for example compress and compress ar both accept as equival to compress*
Porter’s algorithm
The most common English stemmer

Step 1a
sses $\rightarrow$ ss caresses $\rightarrow$ caress
ies $\rightarrow$ i ponies $\rightarrow$ poni
ss $\rightarrow$ ss caress $\rightarrow$ caress
s $\rightarrow$ $\emptyset$ cats $\rightarrow$ cat

Step 1b
(*v*)ing $\rightarrow$ $\emptyset$ walking $\rightarrow$ walk
sing $\rightarrow$ sing
(*v*)ed $\rightarrow$ $\emptyset$ plastered $\rightarrow$ plaster
...

Step 2 (for long stems)
ational $\rightarrow$ ate relational $\rightarrow$ relate
izer $\rightarrow$ ize digitizer $\rightarrow$ digitize
ator $\rightarrow$ ate operator $\rightarrow$ operate
...

Step 3 (for longer stems)
al $\rightarrow$ $\emptyset$ revival $\rightarrow$ reviv
able $\rightarrow$ $\emptyset$ adjustable $\rightarrow$ adjust
ate $\rightarrow$ $\emptyset$ activate $\rightarrow$ activ
...

Consider the IR query matching problem. What are the precision/recall tradeoffs of the Porter stemmer?