Basic Text Processing

Regular Expressions

Word Tokenization

Word Normalization

Sentence Segmentation

Many slides adapted from slides by Dan Jurafsky
Basic Text Processing

Regular Expressions
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] \[a–z] \[0–9]**

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

- Negations \[^{\text{Ss}}\]
  - Carat means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^{\text{A-Z}}]</td>
<td>Not an upper case</td>
<td>0yfn pripetchik</td>
</tr>
<tr>
<td>[^{\text{Ss}}]</td>
<td>Neither ‘S’ nor ‘s’</td>
<td>I have no exquisite reason”</td>
</tr>
<tr>
<td>[^{\text{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></td>
<td></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><code>colo?r</code></td>
<td>0 or 1 of previous char</td>
</tr>
<tr>
<td></td>
<td>color, colour</td>
</tr>
<tr>
<td><code>oo*h!</code></td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td></td>
<td>oh!, ooh!, oooh!, ooooh!</td>
</tr>
<tr>
<td><code>o+h!</code></td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td></td>
<td>oh!, ooh!, oooh!, ooooh!</td>
</tr>
<tr>
<td><code>baa+</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td>baa, baaa, baaaa, baaaaa</td>
</tr>
<tr>
<td><code>beg.n</code></td>
<td>any char</td>
</tr>
<tr>
<td></td>
<td>begin, begun, begun, begun, beg3n</td>
</tr>
</tbody>
</table>
# Regular Expressions: Anchors

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^[A-Z]</code></td>
<td>Palo Alto</td>
</tr>
<tr>
<td><code>^[^A-Za-z]</code></td>
<td>1 “Hello”</td>
</tr>
<tr>
<td><code>\.$</code></td>
<td>The end.</td>
</tr>
<tr>
<td><code>\.$</code></td>
<td>The end? The end!</td>
</tr>
</tbody>
</table>
Example

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

  Misses capitalized examples

  [tT]he

  theology

  Incorrectly returns other or

  [^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 task

• 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
Basic Text Processing

Regular Expressions
Basic Text Processing

Word tokenization
Text Normalization

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

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

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

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

\( N = \text{number of tokens} \)

\( V = \text{vocabulary} = \text{set of types} \)

\( |V| \) is the size of the vocabulary

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

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

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

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

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aaron</td>
<td>25</td>
</tr>
<tr>
<td>Abate</td>
<td>6</td>
</tr>
<tr>
<td>Abates</td>
<td>1</td>
</tr>
<tr>
<td>Abbess</td>
<td>5</td>
</tr>
<tr>
<td>Abbey</td>
<td>6</td>
</tr>
<tr>
<td>Abbot</td>
<td>3</td>
</tr>
</tbody>
</table>

```
Change all non-alpha to newlines
```
tr: translate, -s: squeeze, -c: complement

```
Sort in alphabetical order
```

```
Merge and count each type
```

```
Will likes to eat.
Will likes to babble.
```

1 babble
1 eat
2 likes
2 to
2 Will
The first step: tokenizing

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

(head: will print the first lines (10 by default) of its input. head -n NUM input)

THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
*Assignment for you*

The second step: sorting

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

A
A
A
A
A
A
A
A
...

*Assignment for you*

More counting

- Merging upper and lower case
  \[
  \text{tr ‘A-Z’ ‘a-z’ < shakes.txt | tr –sc ‘A-Za-z’ ‘\n’ | sort | uniq –c}
  \]

- Sorting the counts (-n: numerical value, -k: column, -r: reverse)
  \[
  \text{tr ‘A-Z’ ‘a-z’ < shakes.txt | tr –sc ‘A-Za-z’ ‘\n’ | sort | uniq –c | sort –n –r}
  \]

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

What happened here?
Issues in Tokenization

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

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

• German noun compounds are not segmented
  • *Lebensversicherungsgesellschaftsangestellter*
  • ‘life insurance company employee’
  • German information retrieval needs *compound splitter*
Tokenization: language issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
Basic Text Processing

Word tokenization
Basic Text Processing

Word Normalization and Stemming
Normalization

- Need to “normalize” terms
  - Information Retrieval: indexed text & query terms must have same form.
    - We want to match **U.S.A.** and **USA**
- We implicitly define equivalence classes of terms
  - e.g., deleting periods in a term
- Alternative: asymmetric expansion:
  - Enter: *window*  Search: *window, windows*
  - Enter: *windows*  Search: *Windows, windows, window*
  - Enter: *Windows*  Search: *Windows*
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
  • \textit{am, are, is} \rightarrow \textit{be}
  • \textit{car, cars, car's, cars'} \rightarrow \textit{car}
  • \textit{the boy's cars are different colors} \rightarrow \textit{the boy car be different color}

• Lemmatization: have to find correct dictionary headword form

\textbf{Context dependent. for instance:}
in our last meeting (noun, meeting).
We’re meeting (verb, meet) tomorrow.
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.*
Porter’s algorithm
The most common English stemmer

**fixed rules put in groups, applied in order.**

https://tartarus.org/martin/PorterStemmer/

**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

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

(*v*)ing → Ø  walking → walk
sing → sing
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' '
' < shakes.txt | grep 'ing$' | sort | uniq -c | sort –nr
```

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

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

Word Normalization and Stemming
Basic Text Processing

Sentence Segmentation and Decision Trees
Sentence Segmentation

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

Lots of blank lines after me?

- YES: E-O-S
- NO: Final punctuation is ?, !, or :?
  - YES: Final punctuation is period
  - NO: I am “etc” or other abbreviation
    - YES: Not E-C-S
    - NO: Not E-C-S

E-O-S
More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number

- Numeric features
  - Length of word with “.”
  - Probability(word with “.” occurs at end-of-s)
  - Probability(word after “.” occurs at beginning-of-s)
Implementing Decision Trees

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

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

- Stanford coreNLP: (deterministic)
  - [http://stanfordnlp.github.io/CoreNLP/](http://stanfordnlp.github.io/CoreNLP/)

- UIUC sentence splitter: (deterministic)
  - [https://cogcomp.cs.illinois.edu/page/tools_view/2](https://cogcomp.cs.illinois.edu/page/tools_view/2)
Basic Text Processing

Sentence Segmentation and Decision Trees