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]

<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>
</tbody>
</table>

Chapter 1: Down the Rabbit Hole
Regular Expressions: Negation in Disjunction

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

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>Not an upper case letter</td>
<td>Oyfn pripeatchik</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>mine</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>

Photo D. Fletcher
### 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>baa  baaa  baaaa  baaaaa</td>
</tr>
<tr>
<td>beg.n</td>
<td>begin begun begun begun beg3n</td>
</tr>
</tbody>
</table>

Stephen C Kleene

Kleene *, Kleene +
## Regular Expressions: Anchors $^\$$

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

- \( N \) = number of tokens
- \( V \) = vocabulary = 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 conversations | 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
```

Change all non-alpha to newlines
Sort in alphabetical order
Merge and count each type

```
1945 A 25 Aaron
  72 AARON 6 Abate
  19 ABBESS 1 Abates
  5 ABBOT 5 Abbess
  6 Abbey
... ...
```
The first step: tokenizing

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

THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...

The second step: sorting

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

A
A
A
A
A
A
A
A
A
A
...

More counting

- Merging upper and lower case
  tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '
' | sort | uniq -c

- Sorting the counts
  tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '
' | 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
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?
    - 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

- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

End-user can express query entirely in hiragana!
Word Tokenization in Chinese

• Also called **Word Segmentation**
• Chinese words are composed of characters
  • Characters are generally 1 syllable and 1 morpheme.
  • Average word is 2.4 characters long.
• Standard baseline segmentation algorithm:
  • Maximum Matching (also called Greedy)
Maximum Matching Word Segmentation Algorithm

• Given a wordlist of Chinese, and a string.
  1) Start a pointer at the beginning of the string
  2) Find the longest word in dictionary that matches the string starting at pointer
  3) Move the pointer over the word in string
  4) Go to 2
Max-match segmentation illustration

- The cat in the hat
- The table down there
- Doesn’t generally work in English!
- But works astonishingly well in Chinese
  - 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
- Modern probabilistic segmentation algorithms even better
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*

• Potentially more powerful, but less efficient
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* are both accepted as equivalent to *compress*. 

Porter’s algorithm
The most common English stemmer

Step 1a

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Stemmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>sses → ss</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>ies → i</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>ss → ss</td>
<td>caress → caress</td>
</tr>
<tr>
<td>s → ø</td>
<td>cats → cat</td>
</tr>
</tbody>
</table>

Step 1b

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Stemmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(<em>v</em>)ing → ø</td>
<td>walking → walk</td>
</tr>
<tr>
<td></td>
<td>sing → sing</td>
</tr>
<tr>
<td>(<em>v</em>)ed → ø</td>
<td>plastered → plaster</td>
</tr>
</tbody>
</table>

Step 2 (for long stems)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Stemmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ational → ate</td>
<td>relational → relate</td>
</tr>
<tr>
<td>izer → ize</td>
<td>digitizer → digitize</td>
</tr>
<tr>
<td>ator → ate</td>
<td>operator → operate</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Step 3 (for longer stems)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Stemmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>al → ø</td>
<td>revival → reviv</td>
</tr>
<tr>
<td>able → ø</td>
<td>adjustable → adjust</td>
</tr>
<tr>
<td>ate → ø</td>
<td>activate → activ</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
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' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort -nr
```

```
1312 King
   548 being
548 being
541 nothing
388 king
375 bring
358 thing
307 ring
152 something
145 coming
130 morning
```

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

37
Dealing with complex morphology is sometimes necessary

- Some languages require 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’
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

NO

E-O-S

Final punctuation is ?, !, or :?

YES

NO

E-O-S

Final punctuation is period

YES

NO

I am “etc” or other abbreviation

YES

NO

Not E-O-S

Not E-O-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.
Basic Text Processing
Sentence Segmentation and Decision Trees