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
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>
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>
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>
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>
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>
<th>the First Match in an example</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A–Z]</td>
<td>An upper case letter</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 [^Ss]
  - Carat means negation only when first in []

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>[^A-Z]</td>
<td>Not an upper case letter</td>
<td>^[^e^]</td>
</tr>
<tr>
<td>[^Ss]</td>
<td>Neither ‘S’ nor ‘s’</td>
<td>!</td>
</tr>
<tr>
<td>[^e^]</td>
<td>Neither e nor ^</td>
<td>^a^b</td>
</tr>
<tr>
<td>a\bandb</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>

Photo D. Fletcher
### Regular Expressions: `? * + .`

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>colou?r</code></td>
<td>0 or 1 of previous char</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>color</code> <code>colour</code></td>
</tr>
<tr>
<td><code>oo*h!</code></td>
<td>0 or more of previous char</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>oh! ooh! oooh! ooooh!</code></td>
</tr>
<tr>
<td><code>o+h!</code></td>
<td>1 or more of previous char</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>oh! ooh! oooh! ooooh!</code></td>
</tr>
<tr>
<td><code>baa+</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>baa baaa baaaa baaaaa</code></td>
</tr>
<tr>
<td><code>beg.n</code></td>
<td>any char</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>begin begun begun begun beg3n</code></td>
</tr>
</tbody>
</table>

![Stephen C Kleene](image6.png)

Kleene *, Kleene +

Thursday, September 1, 16
### Regular Expressions: Anchors

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>^[A–Z]</td>
<td><strong>Palo Alto</strong></td>
</tr>
<tr>
<td>^[^A–Za–z]</td>
<td>1 &quot;Hello&quot;</td>
</tr>
<tr>
<td>.$</td>
<td>The end.</td>
</tr>
<tr>
<td>$</td>
<td>The end? The end!</td>
</tr>
</tbody>
</table>
Example
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 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
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
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
How many words?

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

• **Type**: an element of the vocabulary.
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 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?
How many words?

\[ N = \text{number of tokens} \]
How many words?

$N =$ number of tokens

$V = $ vocabulary = set of types

$|V|$ is the size of the vocabulary
## How many words?

\[ N = \text{number of tokens} \]

\[ V = \text{vocabulary} = \text{set of types} \]

\[ |V| \text{ is the size of the vocabulary} \]

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

\[ N = \text{number of tokens} \]

\[ V = \text{vocabulary = set of types} \]

\[ |V| \text{ is the size of the vocabulary} \]

Switchboard phone conversations: 2.4 million tokens, 20 thousand types

Shakespeare: 884,000 tokens, 31 thousand types

Google N-grams: 1 trillion tokens, 13 million types

Church and Gale (1990): \[ |V| > O(N^{1/2}) \]
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' '\n' < shakes.txt
  | sort
  | uniq -c
```

```
1945 A
  72 AARON
  19 ABBESS
  5 ABBOT
...
```

Will likes to eat.
Will likes to laugh.
Simple Tokenization in UNIX

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

\[
\text{tr} \ -\text{sc} \ 'A-Za-z' \ '\n' \ < \ \text{shakes.txt} \quad \text{Change all non-alpha to newlines}
\]
  \[
  | \ \text{sort}
  | \ \text{uniq} \ -\text{c}
\]

1945 A
72 AARON
19 ABBESS
5 ABBOT
...

Will likes to eat.
Will likes to laugh.
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' '\n' < shakes.txt
  | sort
  | uniq -c
```

1945 A
72 AARON
19 ABBESS
5 ABBOT
...

Will likes to eat.
Will likes to laugh.
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
```

- Change all non-alpha to newlines
- Sort in alphabetical order

```
tr: translate, -s: squeeze, -c: complement
```

Will likes to eat.
Will likes to laugh.
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
```
  \textbf{Change all non-alpha to newlines}

```
Sort in alphabetical order
```
  \textbf{tr: translate, -s: squeeze, -c: complement}

```
Merge and count each type
```

Will likes to eat.
Will likes to laugh.
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
```

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

Will likes to eat.
Will likes to laugh.
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  # Change all non-alpha to newlines
  | sort            # Sort in alphabetical order
  | uniq -c         # Merge and count each type
```

Will likes to eat.
Will likes to laugh.
Simple Tokenization in UNIX

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

\[
\text{tr -sc 'A-Za-z' 'n' < shakes.txt} \\
| \text{sort} \quad \text{Sort in alphabetical order} \\
| \text{uniq -c} \quad \text{Merge and count each type}
\]

Will likes to eat.
Will likes to laugh.
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
tr: translate, -s: squeeze, -c: complement
```

Merge and count each type

```
1945  A
```

Will likes to eat.
Will likes to laugh.
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' '\n' < shakes.txt  # Change all non-alpha to newlines
  | sort  # Sort in alphabetical order
  | uniq  # Merge and count each type
```

```
1945 A
72 AARON
```

Will likes to eat.
Will likes to laugh.
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  # Change all non-alpha to newlines
| sort           # Sort in alphabetical order
| uniq -c        # Merge and count each type
```

```
1945 A
72 AARON  # Will likes to eat.
19 ABBESS  # Will likes to laugh.
```

tr: translate, -s: squeeze, -c: complement
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>A</td>
<td>1945</td>
</tr>
<tr>
<td>AARON</td>
<td>72</td>
</tr>
<tr>
<td>ABBESS</td>
<td>19</td>
</tr>
<tr>
<td>ABBOT</td>
<td>5</td>
</tr>
</tbody>
</table>

Will likes to eat.
Will likes to laugh.
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
```

1945 A
72 AARON
19 ABBESS
5 ABBOT
...

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

Will likes to eat.
Will likes to laugh.
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  # Change all non-alpha to newlines
  | sort  # Sort in alphabetical order
  | uniq -c  # Merge and count each type
```

```
1945 A
  25 Aaron
  72 AARON
  19 ABBESS
  5 ABBOT
...
25 Aaron
  6 Abate
  1 Abates
  5 Abbess
  6 Abbey
  3 Abbot
...
```

Will likes to eat.
Will likes to laugh.
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' '\n' < shakes.txt
  | sort
  | uniq -c
```

```
1945 A
72 AARON
19 ABBESS
  5 ABBOT
... ...
```

Will likes to eat.
Will likes to babble.

```
1 babble
1 eat
2 likes
2 to
2 Will
```

Thursday, September 1, 16
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
```

```
1945 A
72 AARON
19 ABBESS
5 ABBOT
...
```

```
1 babble
1 eat
2 likes
2 to
2 Will
```

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

```
1945 A
72 AARON
19 ABBESS
5 ABBOT
...
```

```
tr: translate, -s: squeeze, -c: complement
```

```
1945 A
72 AARON
19 ABBESS
5 ABBOT
...
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
Simple Tokenization in UNIX

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

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

Change all non-alpha to newlines

Sort in alphabetical order

tr: translate, -s: squeeze, -c: complement

Will likes to eat.
Will likes to babble.

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>babble</td>
<td>1</td>
</tr>
<tr>
<td>eat</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>2</td>
</tr>
<tr>
<td>to</td>
<td>2</td>
</tr>
<tr>
<td>Will</td>
<td>2</td>
</tr>
</tbody>
</table>
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
```

Change all non-alpha to newlines

```
Sort in alphabetical order
```

tr: translate, -s: squeeze, -c: complement

```
Merge and count each type
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
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
```

Change all non-alpha to newlines

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
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  # Change all non-alpha to newlines
  | sort          # Sort in alphabetical order
  | uniq -c       # Merge and count each type
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
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  # Change all non-alpha to newlines
  | sort              # Sort in alphabetical order
  | uniq -c           # Merge and count each type
```

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

1 babble
1 eat
2 likes
2 to
2 Will
```
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' '\n' < shakes.txt
  | sort
  | uniq -c
```

1945 A

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
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
```

1945 A
72 AARON

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
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' '\n' < shakes.txt
  | sort
  | uniq -c
```

```
1945 A
72 AARON
19 ABBESS
```

1 babble
1 eat
2 likes
2 to
2 Will

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

1945 A
72 AARON
19 ABBESS
5 ABBOT

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will
Simple Tokenization in UNIX

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

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

1945 A
72 AARON
19 ABBESS
5 ABBOT
...

Change all non-alpha to newlines
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
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' '\n' < shakes.txt
| sort
| uniq -c
```

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1945</td>
</tr>
<tr>
<td>AARON</td>
<td>72</td>
</tr>
<tr>
<td>ABBESS</td>
<td>19</td>
</tr>
<tr>
<td>ABBOT</td>
<td>5</td>
</tr>
<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
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' '
' < 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

Thursday, September 1, 16
The second step: sorting

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

```
A
A
A
A
A
A
A
A
A
A
A
...
```
More counting
More counting

- Merging upper and lower case
More counting

- Merging upper and lower case

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

- Merging upper and lower case
  \texttt{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)
More counting

- Merging upper and lower case
  \texttt{tr 'A-Z' 'a-z' < shakes.txt | tr \textasciitilde{sc} 'A-Za-z' '\n' | sort | uniq \textasciitilde{c}}

- Sorting the counts (-n: numerical value, -k: column, -r: reverse)
  \texttt{tr 'A-Z' 'a-z' < shakes.txt | tr \textasciitilde{sc} 'A-Za-z' '\n' | sort | uniq \textasciitilde{c} | sort \textasciitilde{n \textasciitilde{r}}}

Thursday, September 1, 16
More counting

• Merging upper and lower case
  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)
  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

Thursday, September 1, 16
More counting

- Merging upper and lower case
  \texttt{tr 'A-Z' 'a-z' < shakes.txt | tr –sc 'A-Za-z' '\n' | sort | uniq –c}

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

\begin{verbatim}
  23243 the
  22225 i
  18618 and
  16339 to
  15687 of
  12780 a
  12163 you
  10839 my
  10005 in
  8954 d
\end{verbatim}

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?
    - 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
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
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り$500K(約6,000万円)

Katakana  Hiragana  Kanji  Romaji

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
Max-match segmentation illustration

- Thecatinthehat
Max-match segmentation illustration

- Thecatinthehat  the cat in the hat
Max-match segmentation illustration

- Thecatinthehat
- Thetabledowndthere
Max-match segmentation illustration

- The cat in the hat
- The table down there
Max-match segmentation illustration

- The cat in the hat
- The table down there
- Theta bled own there
Max-match segmentation illustration

- The cat in the hat
- The table down there
- Theta bled own there
Max-match segmentation illustration

- Thecatinthehat  the cat in the hat
- Thetabledownthere  the table down there
- theta bled own there
- Doesn’t generally work in English!
Max-match segmentation illustration

- Thecatinthehat  the cat in the hat
- Thetabledownthere  the table down there
- Doesn’t generally work in English!
Max-match segmentation illustration

• Thecatinthehat  the cat in the hat
• Thetabledownthere  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
Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere
- 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**
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
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*
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*
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*

  **Context dependent.** for instance:
  in our last meeting (noun, meeting).
  We’re meeting (verb, meet) tomorrow.
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*

*Context dependent.* for instance:
- in our last meeting (noun, meeting). We’re meeting (verb, meet) tomorrow.
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.
Lemmatization

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

  **Context dependent.** for instance:
  in our last meeting (noun, meeting).
  We’re meeting (verb, meet) tomorrow.

• *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
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

context independent

• Reduce terms to their stems in information retrieval
Stemming

context independent

• 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.
Stemming

context independent

• 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.
Stemming

context independent

- 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 equal to compress
Porter’s algorithm
The most common English stemmer

fixed rules put in groups, applied in order.  https://tartarus.org/martin/PorterStemmer/

Text
Porter’s algorithm
The most common English stemmer

fixed rules put in groups, applied in order.  https://tartarus.org/mar5n/PorterStemmer/

Step 1a

- sses → ss  caresses → caress
- ies → i   ponies → poni
- ss → ss  caress → caress
- s → ø   cats → cat
Porter’s algorithm

The most common English stemmer

Fixed rules put in groups, applied in order.  

https://tartarus.org/mar5n/PorterStemmer/

Step 1a

\[
\begin{align*}
\text{sses} & \rightarrow \text{ss} & \text{caresses} & \rightarrow \text{caress} \\
\text{ies} & \rightarrow \text{i} & \text{ponies} & \rightarrow \text{poni} \\
\text{ss} & \rightarrow \text{ss} & \text{caress} & \rightarrow \text{caress} \\
\text{s} & \rightarrow \emptyset & \text{cats} & \rightarrow \text{cat}
\end{align*}
\]

Step 1b

\[
\begin{align*}
(*v*)\text{ing} & \rightarrow \emptyset & \text{walking} & \rightarrow \text{walk} \\
& & \text{sing} & \rightarrow \text{sing} \\
(*v*)\text{ed} & \rightarrow \emptyset & \text{plastered} & \rightarrow \text{plaster}
\end{align*}
\]

...
Porter’s algorithm
The most common English stemmer

**fixed rules put in groups, applied in order.**  [https://tartarus.org/mar5n/PorterStemmer/](https://tartarus.org/mar5n/PorterStemmer/)

### Step 1a

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>sses -&gt; ss</td>
<td>caresses -&gt; caress</td>
</tr>
<tr>
<td>ies -&gt; i</td>
<td>ponies -&gt; poni</td>
</tr>
<tr>
<td>ss -&gt; ss</td>
<td>caress -&gt; caress</td>
</tr>
<tr>
<td>s -&gt; ø</td>
<td>cats -&gt; cat</td>
</tr>
</tbody>
</table>

### Step 1b

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>(<em>v</em>)ing -&gt; ø</td>
<td>walking -&gt; walk</td>
</tr>
<tr>
<td></td>
<td>sing -&gt; sing</td>
</tr>
<tr>
<td>(<em>v</em>)ed -&gt; ø</td>
<td>plastered -&gt; plaster</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

### Step 2 (for long stems)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ational -&gt; ate</td>
<td>relational -&gt; relate</td>
</tr>
<tr>
<td>izer -&gt; ize</td>
<td>digitizer -&gt; digitize</td>
</tr>
<tr>
<td>Textator -&gt; ate</td>
<td>operator -&gt; operate</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Thursday, September 1, 16
Porter’s algorithm
The most common English stemmer

fixed rules put in groups, applied in order.  https://tartarus.org/mar5n/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
Textator → 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?
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?
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
Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

(*v*)ing $\rightarrow \emptyset$  walking $\rightarrow$ walk
  sing $\rightarrow$ sing

tr -sc 'A-Za-z' '\n' < 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
Viewing morphology in a corpus
Why only strip –ing if there is a vowel?

\[(\ast v\ast)ing \rightarrow \emptyset \quad \text{walking} \rightarrow \text{walk} \]
\[\quad \text{sing} \rightarrow \text{sing} \]

```
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
```

```
tr -sc 'A-Za-z' '
' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort –nr
```
```
39
```

Thursday, September 1, 16
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

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>being</td>
<td>548</td>
</tr>
<tr>
<td>nothing</td>
<td>541</td>
</tr>
<tr>
<td>king</td>
<td>388</td>
</tr>
<tr>
<td>bring</td>
<td>375</td>
</tr>
<tr>
<td>thing</td>
<td>358</td>
</tr>
<tr>
<td>ring</td>
<td>307</td>
</tr>
<tr>
<td>something</td>
<td>152</td>
</tr>
<tr>
<td>coming</td>
<td>145</td>
</tr>
<tr>
<td>morning</td>
<td>130</td>
</tr>
<tr>
<td>having</td>
<td>122</td>
</tr>
<tr>
<td>living</td>
<td>120</td>
</tr>
<tr>
<td>loving</td>
<td>117</td>
</tr>
<tr>
<td>Being</td>
<td>116</td>
</tr>
<tr>
<td>going</td>
<td>102</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>39</td>
</tr>
<tr>
<td>bring</td>
<td>375</td>
</tr>
<tr>
<td>ring</td>
<td>307</td>
</tr>
<tr>
<td>something</td>
<td>152</td>
</tr>
<tr>
<td>coming</td>
<td>145</td>
</tr>
<tr>
<td>morning</td>
<td>130</td>
</tr>
</tbody>
</table>
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’
Basic Text Processing

Word Normalization and Stemming
Basic Text Processing

Sentence Segmentation and Decision Trees
Sentence Segmentation
Sentence Segmentation

- !, ? are relatively unambiguous
Sentence Segmentation

- !, ? are relatively unambiguous
- Period “.” is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
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: E-O-S
    - NO: Final punctuation is period
      - YES: I am “etc” or other abbreviation
        - YES: Not E-O-S
        - NO: Not E-O-S
      - NO: Not E-O-S
  - E-O-S

Thursday, September 1, 16
More sophisticated decision tree features
More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number
More sophisticated decision tree features

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

- A decision tree is just an if-then-else statement
- The interesting research is choosing the features
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
Decision Trees and other classifiers

- We can think of the questions in a decision tree
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/

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

Sentence Segmentation and Decision Trees