Social Media & Text Analysis
lecture 4 - natural language processing (part 2):
tokenization and normalization

Instructor: Wei Xu
Website: socialmedia-class.org
BAD LANGUAGE!
...on the INTERNET!!

Boom! Ya ur website suxx bro

...dats why pluto is pluto it can neva be a star

michelle obama great. job. and. whit all my respect she. look. great. congrats. to. her.

What can we do about it?
Why don't they just write NORMALLY??
Can our software ever ADAPT???
How does language go bad?

**Illiteracy? No.**
(Tagliamonte and Denis 2008; Drouin and Davis 2009)

**Length limits? (probably not)**

**Hardware input constraints?**
(Gouws et al 2011)

**Social variables**
- Non-standard language does *identity work*, signaling authenticity, solidarity, etc.
- Social variation is usually inhibited in written language, but social media is less regulated than other written genres.

Source: Jacob Eisenstein
Why is Social Media Text “Bad”?

- Lack of literacy? no [Drouin and Davis, 2009]
- Length restrictions? not primarily [Eisenstein, 2013]
- Text input method? to some degree, yes [Gouws et al., 2011]
- Pragmatics (mimicking prosodic effects etc. in speech)? yeeees [Eisenstein, 2013]
- Social variables/markers of social identity? blood oath! [Eisenstein, 2013]
NLP Pipeline

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

Stemming
Normalization
Tokenization

• breaks up the string into words and punctuation
• need to handle:
  - abbreviations (“jr.”), number (“5,000”) …

```
seas479:training weixu$ ./penn-treebank-tokenizer.perl
Tokenizer v3
Language: en

Ms. Hilton last year called Mr. Rothschild “the love of my life.”
Ms. Hilton last year called Mr. Rothschild “the love of my life.”
```
Tokenization

• for Twitter, additionally need to handle:
  - emoticons, urls, #hashtags, @mentions ...
Tool: twokenize.py

```
# -*- coding: utf-8 -*-

""
Twokenize -- a tokenizer designed for Twitter text in English and some other European languages.
```
Tool: twokenize.py

Twokenize -- a tokenizer designed for Twitter text in English and some other European languages. This tokenizer code has gone through a long history:

(1) Brendan O'Connor wrote original version in Python, http://github.com/brendano/tweetmotif
    TweetMotif: Exploratory Search and Topic Summarization for Twitter.
    Brendan O'Connor, Michel Krieger, and David Ahn.
(2a) Kevin Gimpel and Daniel Mills modified it for POS tagging for the CMU ARK Twitter POS Tagger
(2b) Jason Baldridge and David Snyder ported it to Scala
(3) Brendan bugfixed the Scala port and merged with POS-specific changes
    for the CMU ARK Twitter POS Tagger
(4) Tobi Owoputi ported it back to Java and added many improvements (2012-06)

Current home is http://github.com/brendano/ark-tweet-nlp and http://www.ark.cs.cmu.edu/TweetNLP
Tokenization

• main techniques:
  - hand-crafted rules as regular expressions
Regular Expression

- a pattern matching language
- invented by American Mathematician Stephen Kleene in the 1950s
- used for search, find, replace, validation … (very frequently used when dealing with strings)
- supported by most programming languages
- easy to learn, but hard to master
Regular Expression

• [] indicates a set of characters:
  - [amk] will match ‘a’, ‘m’, or ‘k’
  - [a-z] will match any lowercase letter ('abcdefghijklmnopqrstuvwxyz')
  - [a-zA-Z0-9_] will match any letter or digit or ‘_’

• + matches 1 or more repetitions of preceding RE

|   | Hashtag = "#^[a-zA-Z0-9_]+$" |
Regular Expression

Hashtag = "#^[a-zA-Z0-9_]++"

- will match strings that:
  - start with a ‘#’
  - follow with one or more letters/digits/‘_’
Regular Expression

>>> import re
>>> Hashtag = "#[a-zA-Z0-9_]+"
>>> hashtagpattern = re.compile(Hashtag)
>>> hashtagpattern.findall("So that's what #StarWars")
['#StarWars']
Regular Expression

\[
\text{Hearts} = "(?::<+/?3+)+"
\]

- ‘+’ matches 1 or more repetitions of the preceding RE
  - ‘<+’ matches ‘<’, ‘<<’, ‘<<<’ …
  - ‘3+’ matches ‘3’, ‘33’, ‘333’ …
- ‘?’ matches 0 or 1 repetitions of the preceding RE
  - ‘/?’ matches ‘/’ or nothing (so handles ‘</3’)
- (?: …) is a non-capturing version of ( … )
- ( … ) matches whatever RE is inside the parentheses
Regular Expression

\[ \text{Hearts} = "(?:<+/?3+)+" \]

- will match strings that:
  - start with one or more ‘<‘
  - then maybe a ‘/’
  - then one or more ‘3’
  - and maybe repetitions of the above
Regular Expression

\[
\text{Hearts} = "(?::<+/?3+)+"
\]

```python
>>> import re
>>> Hearts = "(?::<+/?3+)+"
>>> heartspattern = re.compile(Hearts)
>>> heartspattern.findall("I <3 u <3<333333")
['<3', '<3<333333']
>>> heartspattern.findall("sooo sad */3")
['*/3']
```
Regular Expression

- learn more (https://docs.python.org/2/library/re.html)
Tokenization

- for Twitter, additionally need to handle:
  - emoticons, urls, hashtags, @mentions ...
Tokenization

- language dependent

<table>
<thead>
<tr>
<th>下雨天留客天留我不留</th>
<th>Unpunctuated Chinese sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>下雨、天留客。天留、我不留！</td>
<td>It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!</td>
</tr>
<tr>
<td>下雨天、留客天。留我不？留！</td>
<td>The rainy day, the staying day. Would you like me to stay? Sure!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>我喜欢新西兰兰花</th>
<th>Unsegmented Chinese sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>我 喜欢 新西兰 花</td>
<td>I like New Zealand flowers</td>
</tr>
<tr>
<td>我 喜欢 新 西兰花</td>
<td>I like fresh broccoli</td>
</tr>
</tbody>
</table>

Source: http://what-when-how.com
NLP Pipeline

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

- Stemming
- Normalization
NLP Pipeline

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

Stemming
Normalization
NLP Pipeline

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

Stemming → Normalization
Stemming

• reduce inflected words to their word stem, base or root form (not necessarily the morphological root)

• studied since the 1960s

```python
>>> from nltk.stem.porter import PorterStemmer
>>> porter_stemmer = PorterStemmer()
>>> porter_stemmer.stem('maximum')
'maximum'
>>> porter_stemmer.stem('presumably')
'presum'
>>> porter_stemmer.stem('multiply')
'multipli'
```
## Stemming

- different steamers: Porter, Snowball, Lancaster ...

- WordNet’s built-in lemmatized (dictionary-based)

```python
>>> from nltk.stem import WordNetLemmatizer
>>> wordnet_lemmatizer = WordNetLemmatizer()
>>> wordnet_lemmatizer.lemmatize('leaves', pos='n')
'leaf'
>>> wordnet_lemmatizer.lemmatize('leaves', pos='v')
'leave'
```
Stemming

- language dependent

Source: All Things Linguistic
Text Normalization

- convert non-standard words to standard

Original tweet
@USER, r u cuming 2 MidCorner dis Sunday?
Normalized tweet
@USER, are you coming to MidCorner this Sunday?

Original tweet
Still have to get up early 2mr thou 😞 so Gn 😞
Normalized tweet
Still have to get up early tomorrow though 😞 so Good night 😞

Source: Baldwin, de Marneffe, Han, Kim, Ritter, Xu
Shared Tasks of the 2015 Workshop on Noisy User-generated Text:
Twitter Lexical Normalization and Named Entity Recognition
Text Normalization

- types of non-standard words in 449 English tweets:

<table>
<thead>
<tr>
<th>Category</th>
<th>Ratio</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>letter&amp;numeral</td>
<td>2.36%</td>
<td>b4 → before</td>
</tr>
<tr>
<td>letter</td>
<td>72.44%</td>
<td>shuld → should</td>
</tr>
<tr>
<td>number substitution</td>
<td>2.76%</td>
<td>4 → for</td>
</tr>
<tr>
<td>slang</td>
<td>12.20%</td>
<td>lol → laugh out loud</td>
</tr>
<tr>
<td>other</td>
<td>10.24%</td>
<td>sucha → such a</td>
</tr>
</tbody>
</table>

most non-standard words are morphophonemic “errors”

Source: Bo Han and Timothy Baldwin
“Lexical normalisation of short text messages: Makn sens a #twitter” ACL 2011
A Normalization Lexicon

- automatically derived from Twitter data + dictionary

Performance
Precision = 0.847
Recall = 0.630
F1-Score = 0.723

Source: Bo Han, Paul Cook and Timothy Baldwin
“Automatically Constructing a Normalisation Dictionary for Microblogs” EMNLP-CoNLL 2012
F-measure

\[ F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \]

Precision-Recall Curve

showing the trade-off, no threshold picking
Phrase-level Normalization

- word-level normalization is insufficient for many cases:

<table>
<thead>
<tr>
<th>Category</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-to-many</td>
<td>everytime → every time</td>
</tr>
<tr>
<td>incorrect IVs</td>
<td>can’t want for → can’t wait for</td>
</tr>
<tr>
<td>grammar</td>
<td>I’m going a movie → I’m going to a movie</td>
</tr>
<tr>
<td>ambiguities</td>
<td>4 → 4 / 4th / for / four</td>
</tr>
</tbody>
</table>

Source: Wei Xu, Alan Ritter, Ralph Grishman

“Gathering and Generating Paraphrases from Twitter with Application to Normalization” BUCC 2013
Summary

classification (Naïve Bayes)

Language Identification

Tokenization

Part-of-Speech (POS) Tagging

Shallow Parsing (Chunking)

Named Entity Recognition (NER)

Regular Expression

Stemming

Normalization
Next Lecture

Language Identification → Tokenization → Part-of-Speech (POS) Tagging → Shallow Parsing (Chunking) → Named Entity Recognition (NER)

- Stemming
- Normalization

Sequential Tagging
Thank You!

Instructor: Wei Xu

www.cis.upenn.edu/~xwe/

Course Website: socialmedia-class.org