Social Media & Text Analysis
lecture 7 - Twitter NLP Pipeline
Tokenization, Normalization, POS/NE Tagging

CSE 5539-0010 Ohio State University
Instructor: Wei Xu
Website: socialmedia-class.org
Tokenization

• breaks up the string into words and punctuation

• need to handle:
  - abbreviations ("jr."), number ("5,000") …

```bash
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 …

```python
>>> import twokenize
>>> input = "Clowns are pretty gross tho O.o (I'm afraid of clowns :p) ask.fm/a/cc301167"
>>> twokenize.tokenizeRawTweetText(input)
['Clowns', 'are', 'pretty', 'gross', 'tho', 'O.o', '(', 'I', 'm', 'afraid', 'of', 'clowns', ':p', ')', 'ask.fm/a/cc301167']
```
Tool: twokenize.py

```python
# -*- 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

147 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

```
147 | Hashtag = "#[a-zA-Z0-9_]++"
```

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

```
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

```
>>> 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

```
133 Hearts = "(?:<+/?3+)+"
```

Python 2.7.10 (default, Feb 7 2017, 00:08:15)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.34)] on darwin
Type "help", "copyright", "credits" or "license" for more information.

```python
>>> import re
>>> heart1 = "(<+/?3+)+"
>>> heartpattern1 = re.compile(heart1)
>>> heartpattern1.findall("I <3 u <3<333")
['<3', '<3<333']
>>>心2 = "(?:<+/>?3+)+"
>>> heartpattern2 = re.compile(heart2)
>>> heartpattern2.findall("I <3 u <3<333")
['<3', '<3<333']
```
Regular Expression

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

This module provides regular expression matching operations similar to those found in Perl. Both patterns and strings to be searched can be Unicode strings as well as 8-bit strings.

Regular expressions use the backslash character (``\``) to indicate special forms or to allow special characters to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals; for example, to match a literal backslash, one might have to write `\\` as the pattern string, because the regular expression must be `\`, and each backslash must be expressed as `\` inside a regular Python string literal.

The solution is to use Python's raw string notation for regular expression patterns; backslashes are not handled in any special way in a string literal prefixed with `r`. So `r"\"` is a two-character string containing `\` and `\`, while `\n` is a one-character string containing a newline. Usually patterns will be expressed in Python code using this raw string notation.

It is important to note that most regular expression operations are available as module-level functions and `RegexObject` methods. The functions are shortcuts that don't require you to compile a regex object first, but miss some fine-tuning parameters.

7.2.1. Regular Expression Syntax
Tokenization

• for Twitter, additionally need to handle:
  - emoticons, urls, #hashtags, @mentions …

```python
>>> import twokenize
>>> input = "Clowns are pretty gross tho O.o (I'm afraid of clowns :p) ask.fm/a/cc301167"
>>> twokenize.tokenizeRawTweetText(input)
['Clowns', 'are', 'pretty', 'gross', 'tho', '0.o', '(', 'I'm', 'afraid', 'of', 'clowns', ':p', ')', 'ask.fm/a/cc301167']
```
Emoticons

Figure 3: Usage of emoticons with and without nose by age group, aggregated over all countries

Dirk Hovy, Anders Johannsen, and Anders Søgaard.
User review sites as a resource for large-scale sociolinguistic studies. WWW, 2015
Emoticons

With respect to gender, we find that women tend to use the noseless variant significantly more than men, except for France, where the difference between genders is not statistically significant at the chosen level.

<table>
<thead>
<tr>
<th>country</th>
<th>Spearman $\rho$</th>
<th>AGE significant</th>
<th>GENDER significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denmark</td>
<td>0.89</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>France</td>
<td>0.63</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Germany</td>
<td>0.83</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>UK</td>
<td>0.83</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>US</td>
<td>0.82</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Dirk Hovy, Anders Johannsen, and Anders Søgaard. User review sites as a resource for large-scale sociolinguistic studies. WWW, 2015
Tokenization

- language dependent

<table>
<thead>
<tr>
<th>Unpunctuated Chinese sentence</th>
<th>It is raining, the god would like the guest to stay. Although the god wants you to stay, I do not!</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>下雨天，留客天。留我不？留！</td>
<td></td>
</tr>
<tr>
<td>我喜欢新西兰兰花</td>
<td>Unsegmented Chinese sentence</td>
</tr>
<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
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
>>> stemmer = PorterStemmer()
>>> stemmer.stem('automate')
'autom'
>>> stemmer.stem('automates')
'autom'
>>> stemmer.stem('automation')
'autom'
```
Stemming

- different stemmers: 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

Agglutinative

Turkish

Avrupa- li- (laş- )- tir- - ama- - dik- - lar- - imiz- - dan

European become- ize NEG whom those we one. of

"Are you one of those whom we could not Europeanize?"
Text Normalization

• convert non-standard words to standard

Source: Tim Baldwin, Marie de Marneffe, Han Bo, Young-Bum Kim, Alan Ritter, Wei 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;numer</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

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

Performance
Precision = 0.847
Recall = 0.630
F1-Score = 0.723
## 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
NLP Pipeline (summary so far)

1. Language Identification
2. Tokenization
3. Part-of-Speech (POS) Tagging
4. Shallow Parsing (Chunking)
5. Named Entity Recognition (NER)

- Classification (Naïve Bayes)
- Regular Expression

- Stemming
- Normalization
NLP Pipeline  (next)

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

Sequential Tagging

- Stemming
- Normalization
Part-of-Speech (POS) Tagging

<table>
<thead>
<tr>
<th></th>
<th>POS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cant</td>
<td>MD</td>
</tr>
<tr>
<td>wait</td>
<td>VB</td>
</tr>
<tr>
<td>for</td>
<td>IN</td>
</tr>
<tr>
<td>the</td>
<td>DT</td>
</tr>
<tr>
<td>ravens</td>
<td>NNP</td>
</tr>
<tr>
<td>game</td>
<td>NN</td>
</tr>
<tr>
<td>tomorrow</td>
<td>NN</td>
</tr>
<tr>
<td>...</td>
<td>:</td>
</tr>
<tr>
<td>go</td>
<td>VB</td>
</tr>
<tr>
<td>ray</td>
<td>NNP</td>
</tr>
<tr>
<td>rice</td>
<td>NNP</td>
</tr>
<tr>
<td>!!!!!!!</td>
<td></td>
</tr>
</tbody>
</table>
### Penn Treebank POS Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
<tr>
<td>CD</td>
<td>Cardinal number</td>
</tr>
<tr>
<td>DT</td>
<td>Determiner</td>
</tr>
<tr>
<td>EX</td>
<td>Existential <em>there</em></td>
</tr>
<tr>
<td>FW</td>
<td>Foreign word</td>
</tr>
<tr>
<td>IN</td>
<td>Preposition/subordinating conjunction</td>
</tr>
<tr>
<td>JJ</td>
<td>Adjective</td>
</tr>
<tr>
<td>JJR</td>
<td>Adjective, comparative</td>
</tr>
<tr>
<td>JJS</td>
<td>Adjective, superlative</td>
</tr>
<tr>
<td>LS</td>
<td>List item marker</td>
</tr>
<tr>
<td>MD</td>
<td>Modal</td>
</tr>
<tr>
<td>NN</td>
<td>Noun, singular or mass</td>
</tr>
<tr>
<td>NNS</td>
<td>Noun, plural</td>
</tr>
<tr>
<td>NNP</td>
<td>Proper noun, singular</td>
</tr>
<tr>
<td>NNPS</td>
<td>Proper noun, plural</td>
</tr>
<tr>
<td>PDT</td>
<td>Predeterminer</td>
</tr>
<tr>
<td>POS</td>
<td>Possessive ending</td>
</tr>
<tr>
<td>PRP</td>
<td>Personal pronoun</td>
</tr>
<tr>
<td>PP$</td>
<td>Possessive pronoun</td>
</tr>
<tr>
<td>RB</td>
<td>Adverb</td>
</tr>
<tr>
<td>RBR</td>
<td>Adverb, comparative</td>
</tr>
<tr>
<td>RBS</td>
<td>Adverb, superlative</td>
</tr>
<tr>
<td>RP</td>
<td>Particle</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbol (mathematical or scientific)</td>
</tr>
<tr>
<td>TO</td>
<td><em>to</em></td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
</tr>
<tr>
<td>VB</td>
<td>Verb, base form</td>
</tr>
<tr>
<td>VBD</td>
<td>Verb, past tense</td>
</tr>
<tr>
<td>VBG</td>
<td>Verb, gerund/present participle</td>
</tr>
<tr>
<td>VBN</td>
<td>Verb, past participle</td>
</tr>
<tr>
<td>VBP</td>
<td>Verb, non-3rd ps. sing. present</td>
</tr>
<tr>
<td>VBZ</td>
<td>Verb, 3rd ps. sing. present</td>
</tr>
<tr>
<td>WDT</td>
<td><em>wh</em>-determiner</td>
</tr>
<tr>
<td>WP</td>
<td><em>wh</em>-pronoun</td>
</tr>
<tr>
<td>WP$</td>
<td>Possessive <em>wh</em>-pronoun</td>
</tr>
<tr>
<td>WRB</td>
<td><em>wh</em>-adverb</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
</tr>
<tr>
<td>$</td>
<td>Dollar sign</td>
</tr>
<tr>
<td>.</td>
<td>Sentence-final punctuation</td>
</tr>
<tr>
<td>,</td>
<td>Comma</td>
</tr>
<tr>
<td>:</td>
<td>Colon, semi-colon</td>
</tr>
<tr>
<td>(</td>
<td>Left bracket character</td>
</tr>
<tr>
<td>)</td>
<td>Right bracket character</td>
</tr>
<tr>
<td>&quot;</td>
<td>Straight double quote</td>
</tr>
<tr>
<td>′</td>
<td>Left open single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>Left open double quote</td>
</tr>
<tr>
<td>′</td>
<td>Right close single quote</td>
</tr>
<tr>
<td>&quot;</td>
<td>Right close double quote</td>
</tr>
</tbody>
</table>
Part-of-Speech (POS) Tagging

• Words often have more than one POS:
  - The back door = JJ
  - On my back = NN
  - Win the voters back = RB
  - Promised to back the bill = VB

• POS tagging problem is to determine the POS tag for a particular instance of a word.
Twitter-specific Tags

- #hashtag
- @mention
- url
- email address
- emoticon
- discourse marker
- symbols
- ...

Retweet construction:

```
RT @user1: I never bought candy bars from those kids on my doorstep so I guess they're all in gangs now.
```

```
RT @user2: LMBO! This man filed an EMERGENCY Motion for Continuance on account of the Rangers game tonight. 😮 Wow lmao
```

Notable Twitter POS Taggers

- Gimpel et al., 2011
- Ritter et al., 2011
- Derczynski et al, 2013
- Owoputi et al. 2013

State-of-the-art:
Token Accuracy: ~ 88%  Sentence Accuracy ~20%

(97% on news text)

Source: Derczynski, Ritter, Clark, Bontcheva
"Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data" RANLP 2013
Chunking

| Cant wait for the ravens game tomorrow....go ray rice!!!!!! |

- Cant: VP
- wait: VP
- for: PP
- the: PP
- ravens: NP
- game: NP
- tomorrow: NP
- ...: NP
- go: VP
- ray: VP
- rice: NP

Wei Xu • socialmedia-class.org
Chunking

• recovering phrases constructed by the part-of-speech tags

• a.k.a shallow (partial) parsing:
  - full parsing is expensive, and is not very robust
  - partial parsing can be much faster, more robust, yet sufficient for many applications
  - useful as input (features) for named entity recognition or full parser
Named Entity Recognition (NER)

<table>
<thead>
<tr>
<th>Cant</th>
<th>wait</th>
<th>for</th>
<th>the</th>
<th>ravens</th>
<th>game</th>
<th>tomorrow</th>
<th>...</th>
<th>go</th>
<th>ray</th>
<th>rice</th>
<th>!!!!!!!</th>
</tr>
</thead>
</table>

ORG: organization

PER: person

LOC: location
NER: Basic Classes

Cant wait for the ravens game tomorrow....go ray rice!!!!!!!
NER: Rich Classes

Source: Strauss, Toma, Ritter, de Marneffe, Xu
Results of the WNUT16 Named Entity Recognition Shared Task (WNUT@COLING 2016)
# NER: Genre Differences

<table>
<thead>
<tr>
<th></th>
<th><strong>News</strong></th>
<th><strong>Tweets</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PER</strong></td>
<td>Politicians, business leaders, journalists, celebrities</td>
<td>Sportsmen, actors, TV personalities, celebrities, names of friends</td>
</tr>
<tr>
<td><strong>LOC</strong></td>
<td>Countries, cities, rivers, and other places related to current affairs</td>
<td>Restaurants, bars, local landmarks/areas, cities, rarely countries</td>
</tr>
<tr>
<td><strong>ORG</strong></td>
<td>Public and private companies, government organisations</td>
<td>Bands, internet companies, sports clubs</td>
</tr>
</tbody>
</table>

Source: Kalina Bontcheva and Leon Derczynski

“Tutorial on Natural Language Processing for Social Media” EACL 2014
Notable Twitter NE Research

- Liu et al., 2011
- Ritter et al., 2011
- Owoputi et al. 2013
- Plank et al, 2014
- Cherry & Guo, 2015

<table>
<thead>
<tr>
<th>System</th>
<th>P</th>
<th>R</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>COTRAIN-NER (10 types)</td>
<td>0.55</td>
<td>0.33</td>
<td>0.41</td>
</tr>
<tr>
<td>T-NER (10 types)</td>
<td>0.65</td>
<td>0.42</td>
<td><strong>0.51</strong></td>
</tr>
<tr>
<td>COTRAIN-NER (PLO)</td>
<td>0.57</td>
<td>0.42</td>
<td>0.49</td>
</tr>
<tr>
<td>T-NER (PLO)</td>
<td>0.73</td>
<td>0.49</td>
<td><strong>0.59</strong></td>
</tr>
<tr>
<td>Stanford NER (PLO)</td>
<td>0.30</td>
<td>0.27</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Table 12: Performance at predicting both segmentation and classification. Systems labeled with PLO are evaluated on the 3 MUC types PERSON, LOCATION, ORGANIZATION.
Tool: twitter_nlp

Twitter NLP Tools

55 commits
2 branches
0 releases
1 contributor

a few corrections to the NER annotation from Brendan

- data
- hbc
- lib
- mallet-2.0.6
- models
Tool: twitter_nlp

Had a great time in New York w my love :)!

```
xuwei@proteus100[twitter_nlp]$: export TWITTER_NLP=./
xuwei@proteus100[twitter_nlp]$
xuwei@proteus100[twitter_nlp]$: echo "Had a great time in New York w my love :) ! " | python python/ner/extractEntities2.py

Had/O a/O great/O time/O in/O New/B-ENTITY York/I-ENTITY w/O my/O love/O :)/O !/O
Average time per tweet = 3.04769945145s
xuwei@proteus100[twitter_nlp]$
xuwei@proteus100[twitter_nlp]$: echo "Had a great time in New York w my love :) ! " | python python/ner/extractEntities2.py --pos --chunk

Average time per tweet = 5.49846148491s
xuwei@proteus100[twitter_nlp]$
```
Summary

classification (Naïve Bayes)

Language Identification

Regular Expression

Tokenization

Part-of-Speech (POS) Tagging

Shallow Parsing (Chunking)

Named Entity Recognition (NER)

Sequential Tagging

Stemming

Normalization
Instructor: Wei Xu
http://web.cse.ohio-state.edu/~weixu/

Course Website: socialmedia-class.org