Social Media Mining, Part 1: Natural Language Processing (NLP)

Muhammad Abdul-Mageed & Markus Dickinson
Dept. of Linguistics, Indiana University
November 6, 2015
Social Media Mining

What we’re doing

▶ Delving into ways to **add value** to text

Where we’re going:

1. Natural Language Processing (NLP): add linguistic structure to (any) text

2. Social Media Mining (SMM): add information about meaningful patterns to social media text

In both cases, we are doing **text processing**, for the purposes of **extracting meaning** of one sort or another
Part 1: Well-formed data

Goals of this part:

▶ Make you familiar with the general ideas of NLP
▶ Give you pointers to various packages available
▶ Make you aware of the difficulties in deploying ready-made NLP tools into social media data

NLP is delving into messier & messier data, but the core tools work best with well-formed data
Natural Language Processing

Natural Language Processing (NLP): “The goal of this field is to get computers to perform useful tasks involving human language” (Jurafsky & Martin 2009, p. 1)

Applications include:

▶ conversational agents / dialogue systems
▶ machine translation
▶ question answering
▶ language teaching
▶ ...

We will focus on natural language understanding (NLU): obtaining linguistic information (meaning) from input (text)
What do we need NLP for?

- One hand: SMM can be NLP, i.e., automatically analyze natural language for the purposes of providing meaning (of a sort) from a text

- Other hand: use NLP tools to pre-process data for SMM, i.e., provide sentence-level grammatical information:
  - Segment sentences
  - Tokenize words
  - Part-of-speech tag words
  - Syntactically (and semantically?) parse sentences
  - Provide semantic word senses
  - Provide named entities
  - Provide language models

This kind of (pre-)processing is the focus for part 1
NLP Overview

We are going to focus on:

▶ what the general tasks are & what the uses are
▶ what kinds of information they generally rely on
▶ what tools are available

We’ll look at POS tagging, parsing, word sense assignment, named entity recognition, & semantic role labeling

▶ We’ll focus on English, but try to note general applicability

Many taggers/parsers have *pre-built* models; others can be *trained* on annotated data

▶ For now, we’ll focus on pre-built models
Wikis with useful technology information

Places you can get your own information:

- Our very own IU CL wiki, which includes some people’s experiences with various tools
  - http://cl.indiana.edu/wiki
- ACL wiki & resources
- ACL software registry: http://registry.dfki.de/
General NLP packages

- Stanford NLP: http://nlp.stanford.edu/software/ (see esp. the CoreNLP package)
- ClearNLP: http://www.clearnlp.com
- FreeLing: http://nlp.lsi.upc.edu/freeling/
- LingPipe: http://alias-i.com/lingpipe/
- OpenNLP: http://opennlp.apache.org/index.html
- Natural Language Toolkit (NLTK): http://www.nltk.org/
- Illinois tools:
  http://cogcomp.cs.illinois.edu/page/software
- DKPro: https://www.ukp.tu-darmstadt.de/research/current-projects/dkpro/
  ▸ Also includes a text classification tool built on top of weka
Topic #1: POS Tagging

**Idea:** assign a part-of-speech to every word in a text

- Taggers work by:
  - looking up a set of appropriate tags for a word in a dictionary
  - using local context to disambiguate from among the set
- Sequence modeling (HMMs, CRFs) is thus popular

Some examples illustrating the utility of local context:

- for the *man*: noun or verb?
- we will *man*: noun or verb?
- I can *put*: verb base form or past?
- re-cap *real* *quick*: adjective or adverb?

Bigram or trigram tagging is quite popular

- Take L545/L645 if you want to know more
Motivation for POS tags

What are POS tags good for in downstream applications?

- First step towards knowing the meaning, e.g., for word senses (e.g., leaves)
- Help identify function words vs. content words (e.g., for author stylometry)
- POS sequences (n-grams) may be indicative of properties such as style or even opinion patterns
  - POS n-grams approximate syntax

Note that POS tags are generally very fast to obtain & are generally accurate (for English, on well-formed data)
Challenges for POS tagging

General challenges:

▶ Ambiguity
  ▶ e.g., *still* as noun, verb, adverb, adjective, ...
▶ Unknown words
  ▶ Programs use things like suffix tries to guess at the possible POS tags for unknown words

These challenges are exacerbated in:

▶ Morphologically-rich languages
▶ Data which is not well-edited (e.g., web data)
POS taggers

- TnT: http://www.coli.uni-saarland.de/~thorsten/tnt/
  - Trainable; models for German & English
- TreeTagger: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/
  - Trainable; models for English, German, Italian, Dutch, Spanish, Bulgarian, Russian, & French; unix, mac, PC
- Qtag: http://www.english.bham.ac.uk/staff/omason/software/qtag.html
  - Trainable; models for German & English
  - Has a variety of NLP modules
- OpenNLP: http://opennlp.sourceforge.net/
  - Models for English, German, Spanish, & Thai; Has a variety of NLP modules
POS taggers (2)

- ACOPOST: http://acopost.sourceforge.net/
  - Trainable; integrates different technologies
  - Trainable; models for English, Arabic, Chinese, & German
- CRFTagger: http://crftagger.sourceforge.net/
  - English
- Can also use SVMTool (http://www.lsi.upc.edu/~nlp/SVMTool/) or CRF++ (http://crfpp.sourceforge.net/) for tagging sequential data, or fnTBL for classification tasks (http://www.cs.jhu.edu/~rflorian/fntbl/index.html)
Specialized POS taggers

Twitter tagger:
- CMU Ark: http://www.ark.cs.cmu.edu/TweetNLP/
- GATE: https://gate.ac.uk/wiki/twitter-postagger.html (also available to plug into Stanford tagger)

Biomedical tagger:
- GENIA tagger: http://www.nactem.ac.uk/tsujii/GENIA/tagger/
Topic #2: Parsing

Parsers attempt to build a tree (=linguistic analysis of groupings/phrases), based on some grammar

- Efficiency based on many things, including the manner in which the tree is built
- They often disambiguate by using probabilities of rules

Again, take L545/L645 for more details
Constituencies & Dependencies

Rough idea of the difference:

Constituency:

```
S
   / \  
  NP  VP
     /   /
    DT  NN  VBD  NN
   the  dragon breathed fire
```

Dependency:

```
vroot
   / \  
  the DET dragon SUBJ breathed OBJ
     /   /
    DT  NN VBD  NN
```

Det, Subj, Obj, vroot
Constituency parsing

Goal is to obtain phrases
- Structured prediction: dealing with embedded / recursive structures
- Parsing can be slow, but tends to be fairly accurate
  - POS tags obtained while parsing more accurate than with a standalone POS tagger

Usefulness for downstream applications:
- Identifying sequences, e.g., named entities
- Identifying complexity, e.g., depth of embedding
- Identifying particular types of constructions, e.g., relative clauses
Challenges in parsing

In addition to things like lexical ambiguity & unknown words, additional challenges include:

▶ Structural ambiguity: e.g., *They saw the man in the park with a telescope*

▶ Garden paths: e.g., *The horse raced past the barn fell*

Again, out-of-domain data poses a challenge

▶ Note that for morphologically-rich languages, parsing is underdeveloped and that more of the work is in the morphology
Dependency parsing

Dependency parsing is the task of assigning dependency (grammatical) relations to a sentence

- Provides quick access to semantic relations ("who did what to whom")
- Can be done on top of constituency parsing or on its own
  - Formally, dependency parsing is simpler: assign a single head & relation for every word

Useful applications:

- Pretty close to the same set as with constituencies ...
Constituency Parsers

- LoPar: http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/LoPar.html
  - Trainable; models for English & German
- BitPar: http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/BitPar.html
  - Trainable; models for English & German
- Charniak & Johnson parser: http://www.cs.brown.edu/people/ec/#software
  - Trainable; mainly used for English
Constituency Parsers (2)

- Collins/Bikel parser:
  http://people.csail.mit.edu/mcollins/code.html
  http://www.cis.upenn.edu/~dbikel/software.html
  - Trainable on English, Chinese, and Arabic; designed for Penn Treebank-style annotation

- Stanford parser:
  - Trainable; models for English, German, Chinese, & Arabic; dependencies also available

- Berkeley parser:
  http://code.google.com/p/berkeleyparser/
  - Trainable; models for English, German, and Chinese
Dependency parsers

Recent parsers, which generally include other NLP tools:

- Mate Parser: https://code.google.com/p/mate-tools/
- TurboParser: http://www.ark.cs.cmu.edu/TurboParser/
- ZPar: http://sourceforge.net/projects/zpar/

Classic dependency parsers:

- MaltParser:
  http://w3.msi.vxu.se/~nivre/research/MaltParser.html
  - Trainable; models for Swedish, English, & Chinese
- MSTParser: http://sourceforge.net/projects/mstparser
  - Trainable; has some models for English & Portuguese
- Link Grammar parser:
  http://www.abisource.com/projects/link-grammar/
  - English only

CCG parsers: http://groups.inf.ed.ac.uk/ccg/software.html

- Primarily for English, although can be trained on German CCGbank
Topic #3: Semantics

**Semantics** is the study of meaning in language

We’ll break it down into:

- Lexical semantics: word meaning
- Compositional semantics: sentence meaning
Semantic class assignment
Word sense disambiguation

**Word sense disambiguation (WSD):** for a given word, determine its semantic class

- bank.01: They robbed a **bank** and took the cash.
- bank.02: They swam awhile and then rested on the **bank**.

Lexical resources define the senses, e.g.

- WordNet: [http://wordnet.princeton.edu](http://wordnet.princeton.edu)
- BabelNet: [http://babelnet.org](http://babelnet.org)
WSD software

- GWSD: Unsupervised Graph-based Word Sense Disambiguation
  http://web.eecs.umich.edu/~mihalcea/downloads.html
- SenseLearner: All-Words Word Sense Disambiguation Tool:
  http://web.eecs.umich.edu/~mihalcea/downloads.html
- KYOTO UKB graph-based WSD:
  http://ixa2.si.ehu.es/ukb/
- pyWSD: Python Implementation of Simple WSD algorithms: https://github.com/alvations/pywsd
- Various packages from Ted Pedersen, including Senseval systems:
  http://www.d.umn.edu/~tpederse/code.html
Semantic class assignment

Named entity recognition

**Named entity recognition (NER):** classify elements (words, phrases) into pre-defined entity classes

- Common categories include: PER(son), ORG(anization), LOC(ation), etc.
- May have hierarchical categories

Techniques often rely on phrase chunking & may involve using a gazetteer (external list of entities)

- From the list of general NLP tools above, Stanford, UIUC, & OpenNLP have NER modules
Semantic role labeling

Idea: The words of a sentence combine to form a meaning
  ▶ Hypothesis: the syntax and semantics can be built up in a corresponding fashion

Semantic role labeling is the task of assigning semantic roles to arguments in a sentence

e.g., for John loves Mary:
  ▶ (to) love is the predicate
  ▶ John is the agent (ARG0)
  ▶ Mary is the patient (ARG1)
Semantic role labelers

- Clear: http://www.clearnlp.com
- SENNA: http://ml.nec-labs.com/senna/
- UIUC: http://cogcomp.cs.illinois.edu/page/software_view/SRL
- SEMAFOR: https://code.google.com/p/semafor-semantic-parser/
- SwiRL: http://www.surdeanu.info/mihai/swirl/
- Shalmaneser: http://www.coli.uni-saarland.de/projects/salsa/shal/
- MATE: https://code.google.com/p/mate-tools/
- Turbo: http://www.ark.cs.cmu.edu/TurboParser/
Topic #4: Language modeling

Language models store lots of text in $n$-gram form, using it to assign probabilities to new sequences of text

▶ Tend to be fast & surprisingly accurate

Useful applications (for LMs and $n$-grams more generally):

▶ Assist in spelling / grammar correction
▶ Define useful features for various classification tasks
Language modeling toolkits

Some packages:

▶ KenLM Language Model Toolkit:
  https://kheafield.com/code/kenlm/

▶ MIT Language Modeling Toolkit:
  https://code.google.com/p/mitlm/

▶ SRI Language Modeling Toolkit:
  http://www.speech.sri.com/projects/srilm/

▶ CMU-Cambridge Statistical Language Modeling Toolkit v2:
  http://www.speech.cs.cmu.edu/SLM/toolkit.html
More detailed how-to

Stanford CoreNLP

Let’s look at how to actually obtain some linguistic information.

The Stanford CoreNLP tools are a good place to start to see how NLP tools can work for you:


Reasons to use Stanford (true of some other tools, too):

▶ Fast & reliable
▶ A variety of linguistic “annotators” available
▶ Well documented & easy to use
▶ Support for English, Spanish, Chinese, German, & Arabic
CoreNLP tools

The tools include:

- Tokenizer (tokenize)
- Sentence splitter (ssplit)
- Part-of-speech (POS) tagger (pos)
- Named entity recognizer (NER) (ner)
- Parser (parse)
- Coreference resolution system (dcoref)
- Sentiment analysis system (sentiment)
- Bootstrapped pattern learning tools
Installation tips

Gleaned from a class using CoreNLP this semester ...

▶ Download the current version from the main site (not the GitHub or Maven sites)
▶ Make sure you have the specified/latest Java version working on your machine.
▶ First test is to type:
  ./corenlp.sh -file input.txt
    ▶ Creates output file input.txt.xml
Out-of-the-box

./corenlp.sh
java -mx3g -cp "./*" edu.stanford.nlp.pipeline.StanfordCoreNLP
Searching for resource: StanfordCoreNLP.properties
Searching for resource: edu/stanford/nlp/pipeline/StanfordCoreNLP.properties
Adding annotator tokenize
Adding annotator ssplit

Adding annotator pos
Reading POS tagger model ...
Adding annotator lemma
Adding annotator ner
Loading classifier ...
Initializing JollyDayHoliday for sutime with ...
Reading TokensRegex rules ...
Ignoring inactive rule: null
Ignoring inactive rule: temporal-composite-8:ranges
Reading TokensRegex rules from ...
Adding annotator parse
Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz ...done [0.6 sec].
Adding annotator dcoref
Interactive shell

Entering interactive shell. Type q RETURN or EOF to quit.

NLP> Analyze this sentence!
Sentence #1 (4 tokens):
Analyze this sentence!

[Text=Analyze CharacterOffsetBegin=0 CharacterOffsetEnd=7
  PartOfSpeech=VB Lemma=analyze NamedEntityTag=O]
[Text=this CharacterOffsetBegin=8 CharacterOffsetEnd=12
  PartOfSpeech=DT Lemma=this NamedEntityTag=O]
[Text=sentence CharacterOffsetBegin=13 CharacterOffsetEnd=21
  PartOfSpeech=NN Lemma=sentence NamedEntityTag=O]
[Text=! CharacterOffsetBegin=21 CharacterOffsetEnd=22
  PartOfSpeech=. Lemma=! NamedEntityTag=O]

(Root
  (S
    (VP (VB Analyze)
      (NP (DT this) (NN sentence)))
    (.) (!)))

root(ROOT-0, Analyze-1)
det(sentence-3, this-2)
dobj(Analyze-1, sentence-3)
Specifying annotators

1. Java properties file
   ▶ Content of config.properties:
     annotators = tokenize, ssplit, pos, lemma, ner, parse, dcoref
   ▶ Command: java -cp "*" -Xmx2g
     edu.stanford.nlp.pipeline.StanfordCoreNLP
     -props config.properties -file input.txt

2. Command line: java -cp "*" -Xmx2g
   edu.stanford.nlp.pipeline.StanfordCoreNLP
   -annotators
   tokenize,ssplit,pos,lemma,ner,parse,dcoref
   -file input.txt
   ▶ These are the default annotators
Working from another directory

To call the files, make sure the classpath (cp) is set properly, e.g.,

- Command line: java -cp 
  "/path/to/stanfordcorenlp/*" -Xmx2g 
  edu.stanford.nlp.pipeline.StanfordCoreNLP 
  -file input.txt

  - Note the asterisk
  - Make sure you have input.txt present
Output format

One token

<token id="1">
  <word>Stanford</word>
  <lemma>Stanford</lemma>
  <CharacterOffsetBegin>0</CharacterOffsetBegin>
  <CharacterOffsetEnd>8</CharacterOffsetEnd>
  <POS>NNP</POS>
  <NER>ORGANIZATION</NER>
  <Speaker>PER0</Speaker>
</token>
Output format
Constituency parse

<parse>(ROOT (S (NP (NNP Stanford) (NNP University)) (VP (VBZ is) (ADJP (JJ located) (PP (IN in) (NP (NNP California)))))) (. .)))</parse>
Output format
Dependency parse

<dependencies type="basic-dependencies">
  <dep type="root">
    <governor idx="0">ROOT</governor>
    <dependent idx="4">located</dependent>
  </dep>
  <dep type="compound">
    <governor idx="2">University</governor>
    <dependent idx="1">Stanford</dependent>
  </dep>
  <dep type="nsubj">
    <governor idx="4">located</governor>
    <dependent idx="2">University</dependent>
  </dep>
  ...
</dependencies>
Output format

Coreference

```xml
<coreference>
  <coreference>
    <mention representative="true">
      <sentence>1</sentence>
      <start>1</start>
      <end>3</end>
      <head>2</head>
      <text>Stanford University</text>
    </mention>
    <mention>
      <sentence>2</sentence>
      <start>1</start>
      <end>2</end>
      <head>1</head>
      <text>It</text>
    </mention>
  </coreference>
</coreference>
```

Outputting as text

```java
java -cp "*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP -props config.properties -file input.txt -outputFormat text

[Text=Stanford CharacterOffsetBegin=0 CharacterOffsetEnd=8 PartOfSpeech=NNP Lemma=Stanford NamedEntityTag=ORGANIZATION] ...

(ROOT
  (S
   (NP (NNP Stanford) (NNP University))
   (VP (VBZ is)
    (ADJP (JJ located)
     (PP (IN in)
      (NP (NNP California))))))
  (. .)))

root(ROOT-0, located-4)
compound(University-2, Stanford-1)
```
Memory issues

Note, by the way, the following warning:
http://nlp.stanford.edu/software/corenlp-faq.shtml

Either give CoreNLP more memory, use fewer annotators, or give CoreNLP smaller documents. Nearly all our annotators load large model files which use lots of memory. Running the full CoreNLP pipeline requires the sum of all these memory requirements. Typically, this means that CoreNLP needs about 2GB to run the entire pipeline. Additionally, the coreference module operates over an entire document. Unless things are size-limited, as either sentence length or document size increases, processing time and space increase without bound.
There are some extensions to the CoreNLP package, including many wrappers in various languages

- Java, .NET, Python, Ruby, Perl, Scala, Clojure, JavaScript
Social Media Mining

So, what happens if you use these tools on social media data?

▶ Will the tools be more help than harm?

Rule of thumb:

▶ The farther the abstraction from the literal word forms, the greater the loss in accuracy, e.g., by my reckoning:
  ▶ Tokenization, POS tagging: probably workable
  ▶ Parsing: borderline
  ▶ Coreference: probably not good enough

Note: text normalization is a step one often wants to take before running any other process
Spot the issues for NLP!

Carrie Fisher’s Twitter account, November 4, 2015:

Everyone tried 2outdo eachother costume wise this Halloween! Look at the new JustSay"I know"bot robot..do u love it?