Text Mining

Emanuele Guidotti

2019/2020

Natural Language Processing (NLP)

Natural Language Processing is the technology used to aid computers to understand the human's natural language. Usually shortened as NLP, is a branch of artificial intelligence that deals with the interaction between computers and humans using the natural language.

The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable.

A typical interaction between humans and machines using Natural Language Processing could go as follows:

1. A human talks to the machine
2. The machine captures the audio
3. Audio to text conversion takes place
4. Processing of the text's data
5. Data to audio conversion takes place
6. The machine responds to the human by playing the audio file

Think of Google Translate, spellcheckers, or personal assistant applications such as OK Google, Siri, Cortana, and Alexa.

Why is NLP difficult?

It's the nature of the human language that makes NLP difficult. The rules that dictate the passing of information using natural languages are not easy for computers to understand.

Some of these rules can be high-leveled and abstract; for example, when someone uses a sarcastic remark to pass information.

On the other hand, some of these rules can be low-levelled; for example, using the character "s" to signify the plurality of items.

The ambiguity and imprecise characteristics of the natural languages are what make NLP difficult for machines to implement.

What are the techniques used in NLP?

Syntactic analysis and **semantic** analysis are the main techniques used to *convert the unstructured language data into a form that computers can understand*.

Syntactic analysis

- Apply grammatical rules

Semantic analysis

- Understand the meaning (much harder)

NLP Tasks | Regular Expressions

Regex

A regular expression (regex or regexp for short) is a special text string for describing a search pattern.

Typically, these patterns are used for four main tasks:

- Find text within a larger body of text
- Validate that a string conforms to a desired format
- Replace or insert text
- Split strings

Example

Extract hashtags from the following tweet:

"It's our job to #GoThere & tell the most difficult stories. Join us! For more breaking news updates follow @CNNBRK & Download our app. Email fake@cnn.com to get involved in the new year."

Hashtags are identified by the "#" symbol followed by one or more alphanumeric characters.

```
import re
re.findall(r'#\w+', text)
```

```
## ['#GoThere']
```

Extract callouts: strings identified by the “@” symbol followed by one or more alphanumeric characters.

```
re.findall(r'@\w+', text)
```

```
## ['@CNNBRK', '@cnn']
```

Oops. . . also part of the email address was extracted. We need to check for word boundaries.

```
re.findall(r'\B@\w+', text)
```

```
## ['@CNNBRK']
```

Online tool: https://regex101.com

Meta-characters: Character matches

Metacharacter	Description
.	wildcard, matches a single character
ˆ	start of a string
$	end of a string
[]	matches one of the set of characters within []
[a-z]	matches one of the range of characters a, b, . . . , z
[ˆabc]	matches a character that is not a, b, or, c
a|b	matches either a or b, where a and b are strings
()	scoping for operators
\	escape character for special characters (\t, \n, \b)

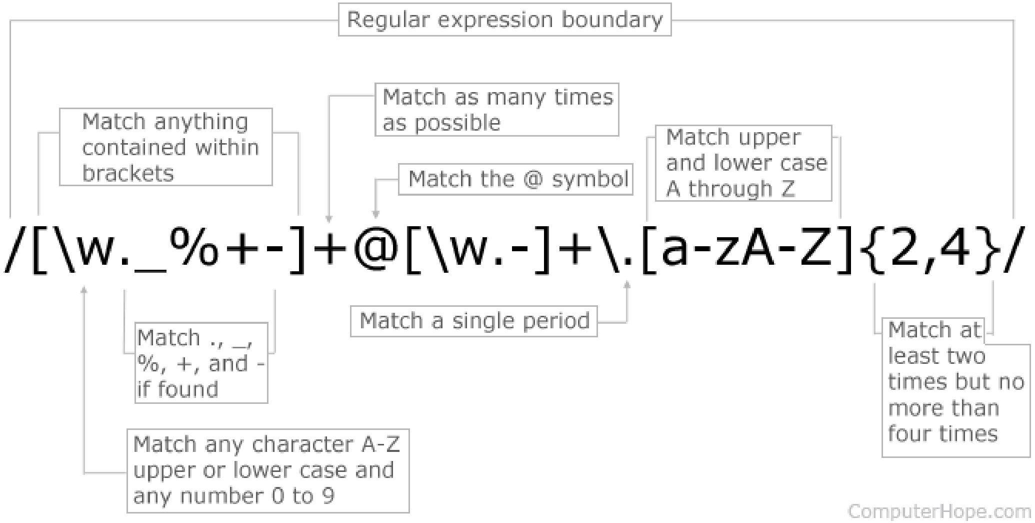
Meta-characters: Character symbols

Metacharacter	Description
\b	matches word boundary
\B	matches where \b does not match
\d	any digit, equivalent to [0-9]
\D	any non-digit, equivalent to [^0-9]
\s	any whitespace, equivalent to [\t\n\r\f\v]
\S	any non-whitespace, equivalent to [^ \t\n\r\f\v]
\w	alphanumeric character, equivalent to [a-zA-Z0-9_]
\W	non-alphanumeric, equivalent to [^a-zA-Z0-9_]

Meta-characters: Quantifiers

Metacharacter	Description
*	matches zero or more occurrences
+	matches one or more occurrences
?	matches zero or one occurrences
{n}	exactly n repetitions
{n,}	at least n repetitions
{,n}	at most n repetitions
{m,n}	at least m and at most n repetitions

Regular Expression E-mail Matching Example


Regular expression boundary
Match anything contained within brackets
Match as many times as possible
Match the @ symbol
Match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
Match ., _, %, +, and - if found
Match a single period
Match at least two times but no more than four times
Match any character A-Z upper or lower case and any number 0 to 9
ComputerHope.com

NLP Tasks | Tokenization

Tokenization

Tokenization is the act of breaking up a sequence of strings into units such as words, keywords, phrases, symbols and other elements called *tokens*.

One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph.

Tokenization is non trivial.

How would you split this sentence into words?

"Children shouldn't drink a sugary drink before bed."

```
text.split(' ') # Naive
```

```
## ['Children', "shouldn't", 'drink', 'a', 'sugary', 'drink',
'before', 'bed.']
```

```
import nltk  # NLTK has an in-built tokenizer!
nltk.word_tokenize(text)
```

```
## ['Children', 'should', "n't", 'drink', 'a', 'sugary', 'drink',
'before', 'bed', '.']
```

How would you split sentences from a long text string?

"This is the first sentence. A gallon of milk in the U.S. costs $2.99. Is this the third sentence? Yes, it is!"

```
# NLTK has an in-built sentence splitter too!
nltk.sent_tokenize(text)
```

```
## ['This is the first sentence.', 'A gallon of milk in the U.S.
costs $2.99.', 'Is this the third sentence?', 'Yes, it is!']
```

N-grams

An **n-gram** is a contiguous sequence of *n* items from a given sample of text. The items can be phonemes, syllables, letters, words or base pairs according to the application.

This is not bad

```
# 2-grams (bigrams)
bigrams = nltk.ngrams(nltk.word_tokenize(text), n = 2)

for grams in bigrams:
  print (grams)
```

```
## ('This', 'is')
## ('is', 'not')
## ('not', 'bad')
```

NLP Tasks | Normalization

Case-folding

A common pre-processing step is to do case-folding by reducing all letters to lower case.

"To be or not to be"

```
nltk.FreqDist(nltk.word_tokenize(text))
```

```
## FreqDist({'be': 2, 'To': 1, 'or': 1, 'not': 1, 'to': 1})
```

```
nltk.FreqDist(nltk.word_tokenize(text.lower()))
```

```
## FreqDist({'to': 2, 'be': 2, 'or': 1, 'not': 1})
```

Although it is often a good idea, such case folding can equate words that might better be kept apart. Many proper nouns are derived from common nouns and so are distinguished only by case, including companies (General Motors, The Associated Press), government organizations (the Fed vs. fed) and person names (Bush, Black).

Depending on the language, an alternative to making every token lowercase is to just make some tokens lowercase. The simplest heuristic is to convert to lowercase words at the beginning of a sentence. These words are usually ordinary words that have been capitalized. Mid-sentence capitalized words are left as capitalized, which is usually correct.

Stop words and Punctuation

Stop words are tokens considered as noise in the text. These tokens should be removed only if they don't add any relevant information for the problem.

NLTK implements pre-defined stop words for several languages, but keep in mind that there is no universal stop words list because a word can be empty of meaning depending on the corpus we are using or on the problem we are analysing.

```python
from nltk.corpus import stopwords
stop_words_en = stopwords.words("english")
stop_words_it = stopwords.words("italian")
```

If punctuation is not relevant for the problem being analysed, we might want to add it to our stop words list. We may add custom stop words as well.

```python
import string
stop_words = stop_words_en + list(string.punctuation)
```

"Remove stop words from this sample sentence, showing off the stop words filtration!"

```python
[w for w in nltk.word_tokenize(text.lower())
    if w not in stop_words]
```

```
## ['remove', 'stop', 'words', 'sample', 'sentence', 'showing',
'stop', 'words', 'filtration']
```

Stemming

Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem **even if the stem itself is not a valid word in the language**.

Stem (root) is the part of the word to which you add inflectional affixes such as -ed,-ize, -s,-de, mis... So stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.

NLTK implemets several kinds of stemmers: PorterStemmer, LancasterStemmer, SnowballStemmers.

PorterStemmer

- simple and fast.
- uses Suffix Stripping to produce stems.
- uses the rules to decide whether it is wise to strip a suffix.

```
porter = nltk.PorterStemmer()
porter.stem('having')
```

```
## 'have'
```

LancasterStemmer

- iterative algorithm.
- on each iteration, it tries to find an applicable rule by the last character of the word.
- each rule specifies either a deletion or replacement of an ending. If there is no such rule, it terminates.
- heavy stemming due to iterations.

```
porter = nltk.LancasterStemmer()
porter.stem('having')
```

```
## 'hav'
```

SnowballStemmers

- One can generate its own set of rules for any language that is why NLTK introduced SnowballStemmers that are used to create non-English Stemmers.

```
porter = nltk.SnowballStemmer(language = 'italian')
porter.stem('avere')
```

```
## 'aver'
```

Lemmatization

Lemmatization, unlike Stemming, reduces the inflected words properly **ensuring that the root word belongs to the language**.

In Lemmatization root word is called Lemma. A *lemma* is the canonical form, dictionary form, or citation form of a set of words.

Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to lookup lemmas of words. English only.

```
lemmatizer = nltk.WordNetLemmatizer()
```

```python
# lemmatize as noun (default)
lemmatizer.lemmatize('better')
```

```
## 'better'
```

```python
# lemmatize as adverb
lemmatizer.lemmatize('better', pos = nltk.corpus.wordnet.ADV)
```

```
## 'well'
```

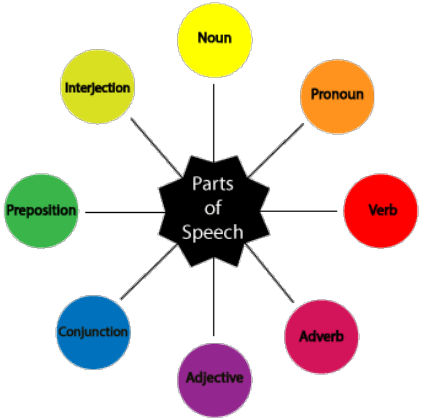
```python
# lemmatize as adjective
lemmatizer.lemmatize('better', pos = nltk.corpus.wordnet.ADJ)
```

```
## 'good'
```

NLP Tasks | Part of Speech Tagging (POS)

Part of Speech

The part of speech explains **how a word is used in a sentence**. There are eight main parts of speech - nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions and interjections.



Part of Speech Tagging (POS)

The process of automatically assigning parts of speech to words in text is called **part-of-speech tagging**, or **POS tagging**.

```
text = "This automatic tagging is awesome!"
nltk.pos_tag(nltk.word_tokenize(text))
```

```
## [('This', 'DT'), ('automatic', 'JJ'), ('tagging', 'NN'), ('is',
'VBZ'), ('awesome', 'JJ'), ('!', '.')]
```

```
nltk.help.upenn_tagset("PRP$")
```

```
## PRP$: pronoun, possessive
##     her his mine my our ours their thy your
```

Tagging Methods

- default tagger: assigns the same tag to each token
- regular expression tagger: assigns tags to tokens on the basis of matching patterns
- supervised learning
  - unigram tagger: for each token, assign the tag that is most likely for that particular token
  - n-gram taggers: generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the n-1 preceding tokens
  - machine learning

These methods can be combined using a technique known as backoff. Backoff is a method for combining models: when a more specialized model (such as a bigram tagger) cannot assign a tag in a given context, we backoff to a more general model (such as a unigram tagger).

NLP Tasks | Chunking

Chunking

Chunking is a process of **extracting phrases from unstructured text**.

- Chunking works on top of POS tagging, it uses pos-tags as input and provides chunks as output.
- Similar to POS tags, there are a standard set of Chunk tags like Noun Phrase (NP), Verb Phrase (VP), etc.
- There are libraries which gives phrases out-of-box such as **Spacy**. NLTK provides a mechanism using regular expressions to generate chunks.

Chunking with NLTK

In order to create NP chunk, we define the chunk grammar using POS tags. We will define this using a regular expression.

```
grammar = ('''
    NP: {<DT>?<JJ>*<NN>}
    V: {<VB[\w]?>}
    ''')
```

The rule states that whenever the chunk finds:

- an optional determiner (DT) followed by any number of adjectives (JJ) and then a noun (NN) then the chunk *NP* should be formed
- a verb (VB, VBD, VBG, VBN, VBP, VBZ) then the chunk *V* should be formed

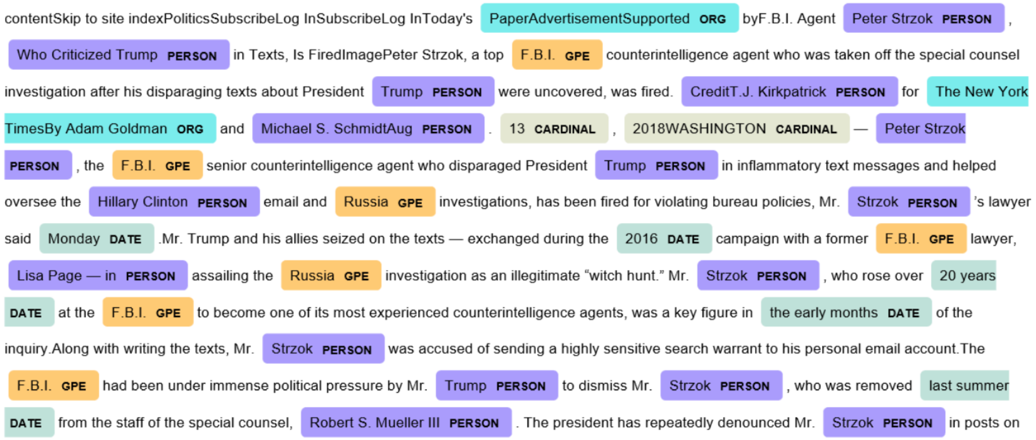
```python
text   = "This is a simple example of chuncking a sentence"
tagged = nltk.pos_tag(nltk.word_tokenize(text))
tree   = nltk.RegexpParser(grammar).parse(tagged)
for subtree in tree.subtrees():
    print(subtree)
```

```
## (S
##   This/DT
##   (V is/VBZ)
##   (NP a/DT simple/JJ example/NN)
##   of/IN
##   (V chuncking/VBG)
##   (NP a/DT sentence/NN))
## (V is/VBZ)
## (NP a/DT simple/JJ example/NN)
## (V chuncking/VBG)
## (NP a/DT sentence/NN)
```

NLP Tasks | Named Entity Recognition (NER)

Named Entity

Named Entities are definite chuncks that refer to specific types of real-world objects, such as organizations, persons, dates, and so on.



contentSkip to site indexPoliticsSubscribeLog InSubscribeLog InToday's PaperAdvertisementSupported ORG byF.B.I. Agent Peter Strzok PERSON , Who Criticized Trump PERSON in Texts, Is FiredImagePeter Strzok, a top F.B.I. GPE counterintelligence agent who was taken off the special counsel investigation after his disparaging texts about President Trump PERSON were uncovered, was fired. CreditT.J. Kirkpatrick PERSON for The New York TimesBy Adam Goldman ORG and Michael S. SchmidtAug PERSON . 13 CARDINAL , 2018WASHINGTON CARDINAL — Peter Strzok PERSON , the F.B.I. GPE senior counterintelligence agent who disparaged President Trump PERSON in inflammatory text messages and helped oversee the Hillary Clinton PERSON email and Russia GPE investigations, has been fired for violating bureau policies, Mr. Strzok PERSON 's lawyer said Monday DATE .Mr. Trump and his allies seized on the texts — exchanged during the 2016 DATE campaign with a former F.B.I. GPE lawyer, Lisa Page — in PERSON assailing the Russia GPE investigation as an illegitimate “witch hunt.” Mr. Strzok PERSON , who rose over 20 years DATE at the F.B.I. GPE to become one of its most experienced counterintelligence agents, was a key figure in the early months DATE of the inquiry.Along with writing the texts, Mr. Strzok PERSON was accused of sending a highly sensitive search warrant to his personal email account.The F.B.I. GPE had been under immense political pressure by Mr. Trump PERSON to dismiss Mr. Strzok PERSON , who was removed last summer DATE from the staff of the special counsel, Robert S. Mueller III PERSON . The president has repeatedly denounced Mr. Strzok PERSON in posts on

Named Entity Recognition (NER)

The goal of Named Entity Recognition (NER) is to identify all textual mentions of the Named Entities. This can be broken down into two sub-tasks:

- identifying the boundaries of the Named Entity
- identifying the type of the Named Entity

The task is well-suited to the type of classifier-based approach that we saw for POS tagging and noun phrase chunking. In particular, we can:

- extract chuncks corresponding to noun phrases
- build a tagger that labels each chunck using the appropriate type based on the trainig data (unigram/n-gram tagger, . . .).

Named Entity Recognition with NLTK

NLTK provides a classifier that has already been trained to recognize Named Entities. Example:

"European authorities fined Google a record $5.1 billion on Wednesday for abusing its power. . . "

```
nltk.ne_chunk(nltk.pos_tag(nltk.word_tokenize(text)))
```

```
## Tree('S', [Tree('GPE', [('European', 'JJ')]), ('authorities',
'NNS'), ('fined', 'VBD'), Tree('PERSON', [('Google', 'NNP')]), ('a',
'DT'), ('record', 'NN'), ('$', '$'), ('5.1', 'CD'), ('billion',
'CD'), ('on', 'IN'), ('Wednesday', 'NNP'), ('for', 'IN'), ('abusing',
'VBG'), ('its', 'PRP$'), ('power', 'NN'), ('...', ':')])
```

Named Entity Recognition with SpaCy

SpaCy features an extremely fast statistical entity recognition system, that assigns labels to contiguous spans of tokens.

```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp(text)

for ent in doc.ents:
    print(ent.text, ent.label_)
```

```
## European NORP
## Google ORG
## $5.1 billion MONEY
## Wednesday DATE
```

Take Home Concepts

Take Home Concepts

- Regex are useful text strings to match patterns
- Case-folding, stop words and punctuation removal, are usually but not always a good idea
- Tokenization, sentence splitting, stemming and lemmatization are non-trivial tasks
- Part of Speech Tagging helps the machine understand how a word is used in a sentence
- Chunking is the process of extracting phrases from unstructured text
- Named Entity Recognition allows us to identify real-world objects in a text

References

References

- https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32
  - read (5 mins)
- https://www.nltk.org/book/ch03.html
  - 3.4, 3.5, 3.6, 3.8
- https://www.nltk.org/book/ch05.html
  - 1, 4.1, 4.2, 5.1, 5.2, 5.3, 5.4, 5.5
- https://www.nltk.org/book/ch07.html
  - 1, 2, 5