Natural Language Processing

Text Preprocessing Techniques

NLP Toolkits and Preprocessing Techniques

- NLP Toolkits
  - Python libraries for natural language processing
- Text Preprocessing Techniques
  - Converting text to a meaningful format for analysis
  - Preprocessing and cleaning text

NLP Toolkits

- NLTK (Natural Language Toolkit)
  - The most popular NLP library
- TextBlob
  - Wraps around NLTK and makes it easier to use
- spaCy
  - Built on Cython, so it’s fast and powerful
- gensim
  - Great for topic modeling and document similarity

Code: How to Install NLTK

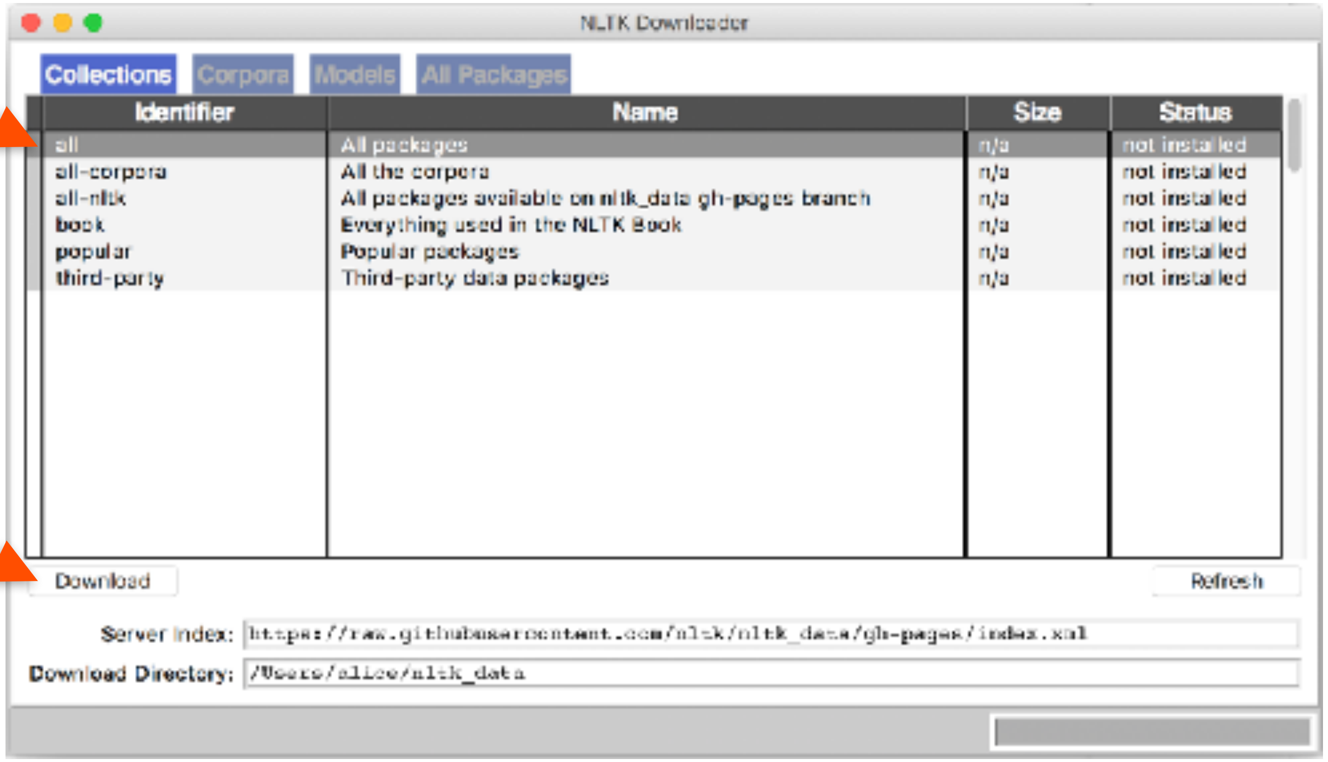
Command Line

pip install nltk

Jupyter Notebook

```
import nltk
nltk.download()
```

```
# downloads all data & models
# this will take a while
```



Sample Text Data

Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?

Text data is messy.

To analyze this data, we need to preprocess and normalize the text.

Preprocessing Techniques

1. Turn text into a meaningful format for analysis
   - Tokenization

2. Clean the data
   - Remove: capital letters, punctuation, numbers, stop words
   - Stemming, parts of speech tagging
   - Chunking: named entity recognition, compound term extraction

Tokenization

Tokenization = splitting raw text into small, indivisible units for processing

These units can be:

- Words
- Sentences
- N-grams
- Other characters defined by regular expressions

Code: Tokenization (Words)

Input:

```
from nltk.tokenize import word_tokenize

my_text = "Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from
the store. Should I pick up some black-eyed peas as well?"

print(word_tokenize(my_text))
```

Output:

```
['Hi', 'Mr.', 'Smith', '!', 'I', '’', 'm', ‘going', ‘to', 'buy', 'some',
'vegetables', '(', 'tomatoes', 'and', 'cucumbers', ')', 'from', 'the', 'store',
'.', 'Should', 'I', 'pick', 'up', ‘some', 'black-eyed', 'peas', 'as', 'well',
'?']
```

Tokenization: Sentences

Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?

Tokens can be sentences. How would you split this into sentences? What rules would you put in place?

It’s a difficult task. This is where tokenizers in Python can help.

Code: Tokenization (Sentences)

Input:

```
from nltk.tokenize import sent_tokenize


my_text = "Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?"


print(sent_tokenize(my_text))
```

Output:

```
['Hi Mr. Smith!',
 'I’m going to buy some vegetables (tomatoes and cucumbers) from the store.',
 'Should I pick up some black-eyed peas as well?']
```

Code: Tokenization (N-Grams)

Input:

```
from nltk.util import ngrams

my_words = word_tokenize(my_text) # This is the list of all words
twograms = list(ngrams(my_words,2)) # This is for two-word combos, but can pick any n
print(twograms)
```

Output:

```
[('Hi', 'Mr.'), ('Mr.', 'Smith'), ('Smith', '!'), ('!', 'I'), ('I', '’'), ('’', 'm'), ('m',
'going'), ('going', 'to'), ('to', ‘buy'), ('buy', 'some'), ('some', 'vegetables'),
('vegetables', '('), ('(', 'tomatoes'), ('tomatoes', 'and'), ('and', 'cucumbers'),
('cucumbers', ')'), (')', 'from'), ('from', 'the'), ('the', 'store'), ('store', '.'), ('.',
'Should'), ('Should', 'I'), ('I', 'pick'), ('pick', 'up'), ('up', '1/2'), ('1/2', 'lb'),
('lb', 'of'), ('of', 'black-eyed'), ('black-eyed', 'peas'), ('peas', 'as'), ('as', 'well'),
('well', '?')]
```

Tokenization: Regular Expressions

Let's say you want to tokenize by some other type of grouping or pattern.

Regular expressions (regex) allows you to do so.

Some examples of regular expressions:

- Find white spaces: \s+
- Find words starting with capital letters: [A-Z]['\w]+

Code: Tokenization (Regular Expressions)

Input:

```
from nltk.tokenize import RegexpTokenizer

# RegexpTokenizer with whitespace delimiter
whitespace_tokenizer = RegexpTokenizer("\s+", gaps=True)
print(whitespace_tokenizer.tokenize(my_text))
```

Output:

```
['Hi', 'Mr.', 'Smith!', 'I’m', 'going', ‘to', 'buy', 'some', 'vegetables',
'(tomatoes', 'and', 'cucumbers)', 'from', 'the', 'store.', 'Should', 'I', 'pick',
'up', ‘some', 'black-eyed', 'peas', 'as', 'well?']
```

```
['Hi', 'Mr.', 'Smith', '!', 'I', '’', 'm', 'going', ‘to’, 'buy', 'some',
'vegetables', '(', 'tomatoes', 'and', 'cucumbers', ')', 'from', 'the', 'store',
'.', 'Should', 'I', 'pick', 'up', ‘some', 'black-eyed', 'peas', 'as', 'well',
'?']
```

Token = Word

Code: Tokenization (Regular Expressions)

Input:

```
from nltk.tokenize import RegexpTokenizer

# RegexpTokenizer to match only capitalized words
cap_tokenizer = RegexpTokenizer("[A-Z]['\w]+")
print(cap_tokenizer.tokenize(my_text))
```

Output:

```
['Hi', 'Mr', 'Smith', 'Should']
```

Tokenization Summary

Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?

With tokenization, we were able to break this messy text data down into small units for us to do analysis

- By sentence, word, n-grams
- By characters and patterns using regular expressions

Preprocessing Checkpoint

What have we done so far?

- Tokenized text by sentence, word, n-grams and using regex

This is only one step. There is a lot more preprocessing that we can do.

Other Preprocessing Considerations

- Remove: punctuation, capital letters, numbers, stop words
- Stemming, parts of speech tagging
- Chunking: named entity recognition, compound term extraction
- Misspellings
- Different languages

Preprocessing: Remove Characters

Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up 2lbs of black-eyed peas as well?

How can we normalize this text?

- Remove punctuation
- Remove capital letters and make all letters lowercase
- Remove numbers

Code: Remove Punctuation

Input:

```
import re # Regular expression library
import string

# Replace punctuations with a white space
clean_text = re.sub('[%s]' % re.escape(string.punctuation), ' ', my_text)
clean_text
```

Output:

```
'Hi Mr  Smith  I m going to buy some vegetables  tomatoes and cucumbers  from the store  Should I pick up 2lbs of black eyed peas as well  '
```

```
'Hi Mr Smith Im going to buy some vegetables tomatoes and cucumbers from the store Should I pick up 2lbs of blackeyed peas as well'
```

Replace with '' instead of ' '

Code: Make All Text Lowercase

Input:

```
clean_text = clean_text.lower()
clean_text
```

Output:

```
'hi mr  smith  i m going to buy some vegetables  tomatoes and cucumbers  from the
store  should i pick up 2lbs of black eyed peas as well '
```

Code: Remove Numbers

Input:

```
# Removes all words containing digits
clean_text = re.sub('\w*\d\w*', ' ', clean_text)
clean_text
```

Output:

```
'hi mr  smith  i m going to buy some vegetables  tomatoes and cucumbers  from the
store  should i pick up   of black eyed peas as well '
```

Preprocessing: Stop Words

Hi Mr. Smith! I'm going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?

What is the most frequent term in the text above? Is that information meaningful?

Stop words are words that have very little semantic value.

There are language and context-specific stop word lists online that you can use.

Code: Stop Words

Input:

```python
from nltk.corpus import stopwords
set(stopwords.words('english'))
```

Output:

```
{'but', 'isn', 'under', 'weren', 'those', 'when', 'why', 'few', 'for', 'it', 'of', 'down', 'ma', 'over',
'd', 'during', 'shouldn', 'did', 'above', 'below', 'myself', 'further', 'very', 'same', 'too', 'does',
'through', 'from', 'didn', 'whom', 'and', 'am', 'such', 'out', 'or', 'me', 'has', 'will', 'shan', 'on',
'then', 'here', 't', 'with', 'some', 'what', 'don', 'were', 'an', 'themselves', 'yourselves', 'off',
'being', 'more', 'they', 'ourselves', 'into', 'my', 'them', 'ain', 'a', 'wouldn', 'itself', 'i', 'hasn',
'her', 'their', 'mustn', 'our', 'herself', 'where', 'hers', 'once', 'any', 'theirs', 'before', 'most',
'other', 'not', 'himself', 'his', 'if', 'he', 'each', 'are', 'how', 'couldn', 'ours', 'doing', 'hadn',
'needn', 'again', 'these', 'wasn', 'nor', 'do', 'just', 'so', 'we', 'there', 'have', 'by', 'o', 'than',
're', 'while', 'your', 'at', 'him', 'own', 'can', 'you', 'll', 'between', 'been', 'that', 'is', 'she',
'yours', 'this', 'was', 'be', 'had', 'doesn', 'no', 'because', 'won', 'both', 'to', 'against', 'aren',
'y', 'after', 'all', 'up', 've', 'should', 'as', 'in', 'the', 'having', 'until', 'who', 'haven', 'only',
'm', 'yourself', 'about', 's', 'which', 'now', 'mightn', 'its'}
```

Code: Remove Stop Words

Input:

```
my_text = ["Hi Mr. Smith! I’m going to buy some vegetables (tomatoes and cucumbers) from
the store. Should I pick up some black-eyed peas as well?"]

# Incorporate stop words when creating the count vectorizer
cv = CountVectorizer(stop_words='english')
X = cv.fit_transform(my_text)
pd.DataFrame(X.toarray(), columns=cv.get_feature_names())
```

Output:

	black	buy	cucumbers	eyed	going	hi	mr	peas	pick	smith	store	tomatoes	vegetables
0	1	1	1	1	1	1	1	1	1	1	1	1	1

Including stop words

	and	as	black	buy	cucumbers	eyed	from	going	hi	mr	...	pick	should	smith	some	store	the	tomatoes	up	vegetables	well
0	1	1	1	1	1	1	1	1	1	1	...	1	1	1	2	1	1	1	1	1	1

Preprocessing: Stemming

Stemming & Lemmatization = Cut word down to base form

- Stemming: Uses rough heuristics to reduce words to base
- Lemmatization: Uses vocabulary and morphological analysis
- Makes the meaning of run, runs, running, ran all the same
- Cuts down on complexity by reducing the number of unique words

Multiple stemmers available in NLTK

- PorterStemmer, LancasterStemmer, SnowballStemmer
- WordNetLemmatizer

Code: Stemming

Input:

```python
from nltk.stem.lancaster import LancasterStemmer

stemmer = LancasterStemmer()

# Try some stems
print('drive: {}’.format(stemmer.stem('drive')))
print('drives: {}'.format(stemmer.stem('drives')))
print('driver: {}'.format(stemmer.stem('driver')))
print('drivers: {}'.format(stemmer.stem('drivers')))
print('driven: {}'.format(stemmer.stem('driven')))
```

Output:

```
drive: driv
drives: driv
driver: driv
drivers: driv
driven: driv
```

Preprocessing: Parts of Speech Tagging

Parts of Speech

- Nouns, verbs, adjectives, etc.
- Parts of speech tagging labels each word as a part of speech

Code: Parts of Speech Tagging

Input:

```
from nltk.tag import pos_tag

my_text = "James Smith lives in the United States."

tokens = pos_tag(word_tokenize(my_text))
print(tokens)
```

Output:

```
[('James', 'NNP'),
 ('Smith', 'NNP'),
 ('lives', 'VBZ'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('United', 'NNP'),
 ('States', 'NNPS'),
 ('.', '.')]
```

Code: Parts of Speech Tagging

Input:

```
nltk.help.upenn_tagset()
```

```
[('James', 'NNP'),
 ('Smith', 'NNP'),
 ('lives', 'VBZ'),
 ('in', 'IN'),
 ('the', 'DT'),
 ('United', 'NNP'),
 ('States', 'NNPS'),
 ('.', '.')]
```

Output:

```
DT: determiner all an another any both del each either every half la many much nary neither no some such that the them
these this those

IN: preposition or conjunction, subordinating astride among uppon whether out inside pro despite on by throughout below
within for towards near behind atop around if like until below next into if beside ...

NNP: noun, proper, singular Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos Oceanside Escobar Kreisler
Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Meltex Liverpool ...

NNPS: noun, proper, plural Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists Andalusians Andes
Andruses Angels Animals Anthony Antilles Antiques Apache Apaches Apocrypha ...

VBZ: verb, present tense, 3rd person singular bases reconstructs marks mixes displeases seals carps weaves snatches
slumps stretches authorizes smolders pictures emerges stockpiles seduces fizzes uses bolsters slaps speaks pleads ...
```

Preprocessing: Named Entity Recognition

Named Entity Recognition (NER) aka Entity Extraction

- Identifies and tags named entities in text (people, places, organizations, phone numbers, emails, etc.)
- Can be tremendously valuable for further NLP tasks
- For example: “United States” --> “United_States”

Code: Named Entity Recognition

Input:

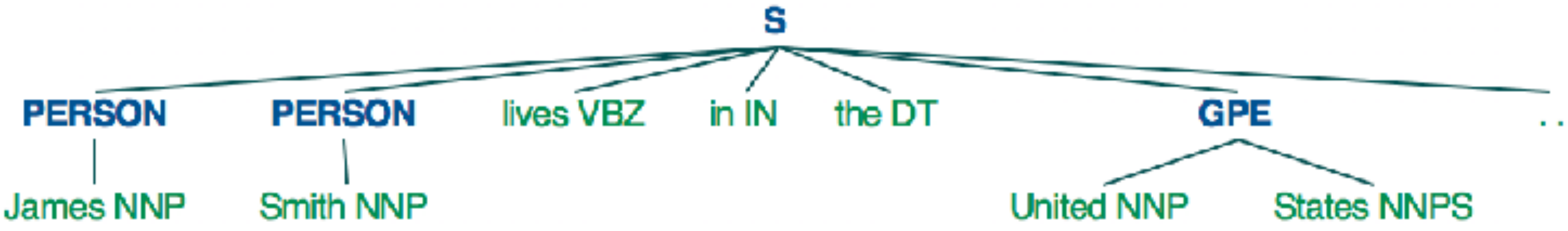
```
from nltk.chunk import ne_chunk

my_text = "James Smith lives in the United States."

tokens = pos_tag(word_tokenize(my_text)) # this labels each word as a part of speech

entities = ne_chunk(tokens) # this extracts entities from the list of words
entities.draw()
```

Output:



Preprocessing: Compound Term Extraction

Extracting and tagging compound words or phrases in text

- This can be very valuable for special cases
- For example: “black eyed peas“ --> “black_eyed_peas”
- This totally changes the conceptual meaning!
- Named entity recognition groups together words and identifies entities, but doesn’t capture them all, so you can identify your own compound words

Code: Compound Term Extraction

Input:

```
from nltk.tokenize import MWETokenizer # multi-word expression

my_text = "You all are the greatest students of all time."

mwe_tokenizer = MWETokenizer([('You','all'), ('of', 'all', 'time')])
mwe_tokens = mwe_tokenizer.tokenize(word_tokenize(my_text))

mwe_tokens
```

Output:

```
['You_all', 'are', 'the', 'greatest', 'students', 'of_all_time', '.']
```

Preprocessing Checkpoint

What have we done so far?

- Introduced Python’s Natural Language Toolkit
- Converted text into token form and count vector form
- Further cleaned the data by removing characters, using stop words, stemming, parts of speech tagging, named entity recognition and compound words

Preprocessing Review

Given the text below, what are some preprocessing techniques you could apply?

Hi Mr. Smith! I'm going to buy some vegetables (tomatoes and cucumbers) from the store. Should I pick up some black-eyed peas as well?

Tokenization
- Sentence
- Word
- N-Gram
- Regex

Remove
- Punctuation
- Capital Letters
- Numbers
- Stop Words

Chunking
- Named Entity Recognition
- Compound Term Extraction

More
- Stemming
- Parts of Speech
- Misspellings
- Diff Languages

Preprocessing Summary

- Text data is messy
  - Preprocessing must be done before doing analysis
  - Python has some great libraries for NLP, such as NLTK, TextBlob and spaCy

- There are many preprocessing techniques
  - Tokenization and organizing the data for analysis is necessary
  - Otherwise, pick and choose the techniques that makes most sense for your data and your analysis