NLP Preprocessing Techniques

Presented By: Aizan Zafar (PhD Scholar)

Research Area: Conversational AI

Supervised By: Dr. Asif Ekbal

NLP ToolKit and Preprocessing Techniques

- NLP ToolKit
  - Python libraries for NLP

- Text Preprocessing Techniques
  - Converting text to meaningful format
  - 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
  - Build 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()
```

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 meaningful format for analysis
    - Tokenization

2. Clean the data
    - Remove: capital letters, punctuation, numbers, stop words
    - Stemming
    - Parts of speech tagging
    - Correct misspellings
    - 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)) # print function requires Python 3
```

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 regex:

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

Code: Tokenization (Regular Expressions)

```
from nltk.tokenize import RegexpTokenizer

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?"

# Regexp Tokenizer with whitespace delimiter
whitespace_tokenizer = RegexpTokenizer("\s+", gaps=True)
# my_text.decode('utf-8') <--- for python2 only
print(whitespace_tokenizer.tokenize(my_text))
```

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

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.

Preprocessing Techniques

1. Turn text into meaningful format for analysis
    - Tokenization

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

Preprocessing: Remove Characters

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?

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 '
```

Tips and Tricks: Lambda

```python
# Basic example of a lambda

square_me = lambda x : x * x

print(square_me(5))
```

```
25
```

INPUT   OUTPUT

```python
# Yes it is exactly equivalent to

def square_me_too(x):
    return x * x

print(square_me_too(5))
```

```
25
```

Tips and Tricks: Lambda and maps

```
# Basic example of a map

my_numbers = [9, 3, 4, 100, 2, 1]

my_numbers_squared = list(map(square_me, my_numbers))

print(my_numbers_squared)
```

```
[81, 9, 16, 10000, 4, 1]
```

Tips and Tricks: Lambda and maps

```
# But what if you want to clean a bunch of texts?
text1 = "I'm going to buy 5 cans of beans"
text2 = "I'm going to buy 6lbs of ham"
text3 = "I'm going to 111 Dore Street"

texts = [text1, text2, text3]

# I could use a for loop... or I can use a lambda and a map
remove_numbers = lambda x : re.sub('\w*\d\w*', ' ', x)

texts = list(map(remove_numbers, texts))

print(texts)
```

```
["I'm going to buy   cans of beans", "I'm going to buy   of ham", "I'm going
to   Dore Street"]
```

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:

```
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: 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 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:

```
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: part of speech

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

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

Input:

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

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: Name Entity Recognition

Input:

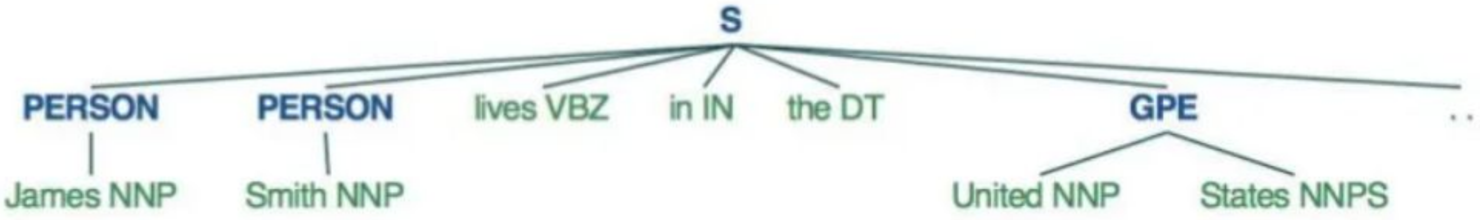
```
from nltk.chunk import ne_chunk

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 for special cases
- For example: “black eyed peas” → “black_eyed_peas”
- This totally changes the conceptual meaning!
- NER groups together words and identifies entities

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
- 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?

We're rushing our patient to the nearest hospital in Bend, Oregon. He has a traumatic brain injury and requires medical attention within the next 10 minutes!

Tokenization	Remove	Chunking	More
Sentence Word N-Gram Regex	Punctuation Capital Letters Numbers Stop Words	Named Entity Recognition Compound Term Extraction	Stemming Parts of Speech Misspellings Diff Languages

Pandas for Data Analysis Review

- Pandas is an open-source python library used for data manipulation and analysis

- It provide easy-to-use data structures and data analysis tools which can be used in a wide range of fields.

- We will only discuss some of the NLP-related frequently used pandas functions.

Pandas DataFrame

A DataFrame is a two-dimensional array with heterogeneous data.

It basically a table of data much like in Excel or SQL

	user_id	stars	reviews
0	A368Z46FIKHSEZ	5	I love these cookies! Not only are they healt...
1	A1JAPP1CXRG57A	5	Quaker Soft Baked Oatmeal Cookies with raisins...
2	A2Z9JNXPIEL2B9	5	I am usually not a huge fan of oatmeal cookies...
3	A31CYJQO3FL586	5	I participated in a product review that includ...
4	A2KXQ2EKFF3K2G	5	My kids loved these. I was very pleased to giv...

Creating Pandas DataFrame

DataFrames can be created **manually** or from **file**.

Manually:

```
import pandas as pd

new_dataframe = pd.DataFrame(
            { “column_name” : [“jack”, “jill”, “john”],
              “column_age” : [13, 14, 12],
              “column_weight” : [130.4, 123.6, 150.2] }
            )
```

```
import pandas as pd

new_dataframe = pd.DataFrame(
    {
        "column_name" : ["jack", "jill", "john"],
        "column_age" : [13, 14, 12],
        "column_weight" : [130.4, 123.6, 150.2]
    }
)
new_dataframe
```

	column_name	column_age	column_weight
0	jack	13	130.4
1	jill	14	123.6
2	john	12	150.2

From csv file:

```
import pandas as pd

file_dataframe = pd.read_csv(‘file_data.csv’)
```

```
file_dataframe = pd.read_csv('file_data.csv')
file_dataframe
```

	column_name	column_age	column_weight
0	jack	13	130.4
1	jill	14	123.6
2	john	12	150.2

Selecting specific column:

```
file_dataframe.column_name
```

```
0    jack
1    jill
2    john
Name: column_name, dtype: object
```

Basic Pandas Functionality

```
import pandas as pd
data = pd.read_csv('data.csv')
```

Selecting top and bottom rows:

```
pd.head()     Returns the first n rows.
pd.tail()     Returns the last n rows.
```

Selecting columns:

```
data['column_name'] or data.column_name
```

Selecting by indexer:

```
data.iloc[0] - first row of data frame
data.iloc[-1] - last row of data frame
data.iloc[:,0] - first column of data frame
data.iloc[:,-1] - last column of data frame
Data.iloc[0,1] – first row, second column of the dataframe
data.iloc[0:4, 3:5] # first 4 rows and 3rd, 4th, 5th columns of data frame
```

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

References

- https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/
- https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79
- https://www.kaggle.com/sudalairajkumar/getting-started-with-text-preprocessing
- https://medium.com/analytics-vidhya/text-preprocessing-for-nlp-natural-language-processing-beginners-to-master-fd82dfecf95
- https://www.slideshare.net/ankit_ppt/nlp-toolkits-andpreprocessingtechniques
- https://web.stanford.edu/~jurafsky/slp3/slides/2_TextProc.pdf

Thank You