HOCHSCHULE SRH HEIDELBERG
Intelligence in Learning

PROJECT REPORT

"Big Data Programming"

Masters of Big Data and Business Analytics

Semester II

Submitted By:

Ali Mushtaq
Anish Umesh
Jasmine Kaur
Narendra Medisetti
Fahmida

INTRODUCTION

In today's era of internet and online services, data is getting generated at a rapid rate. These data are either numerical or categorical data and has a variety of structures such as text, image, audio, and video. Online activities such as articles, website text, blog posts, and social media posts are generating unstructured textual data and Organisations analyse these textual data to understand customer activities, opinion, and feedback to successfully derive their business. To compete with big textual data, text analytics is evolving at a faster rate than ever before.

NLP is a subfield of Artificial Intelligence and is concerned with interactions between computers and human languages. It is the process of analysing, understanding, and deriving meaning from human languages for computers. The following subareas of Natural Language Processing are used for this:

- speech recognition,
- language translation,
- Segmentation of previously acquired language into individual words and sentences
- Recognize the basic forms of words and capture grammatical information
- Recognizing the functions of individual words in the sentence (subject, verb, object, article, etc.)
- Extraction of the meaning of sentences and phrases
- Recognition of sentence contexts and sentence relationships

The two significant libraries used in NLP are NLTK and spaCy**.**

NLTK is a powerful Python package that provides a set of diverse natural languages algorithms. It is open source, easy to use, large community, and well documented. NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, pre-process, and understand the written text.

SpaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. Unlike NLTK, which is widely used for teaching and research, SpaCy focuses on providing software for production usage. It is an excellent choice when conducting market research and collecting insights because of its aspect-based sentiment analysis, named-entity recognition, and optimization of conversational user interface abilities.

MOTIVATION

Natural language processing (NLP) is getting very popular today, which became especially noticeable in the background of the deep learning development. The main tasks include speech recognition and generation, text analysis, sentiment analysis, machine translation, etc. There are many tools and libraries designed to solve NLP problems such as

- NLTK
- SpaCy
- Scikit-Learn
- Gensim
- Polyglot
- Pattern

For this project we chose **SpaCy** library because it has an upper hand with respect to other libraries such as:

- It was built with production-readiness in mind, focusing more on efficiency and performance.
- It follows an object-oriented approach in handling the tasks. The text is processed in a pipeline and stored in an object, and that object contains attributes and methods for various NLP tasks. This approach is more versatile and in alignment with modern Python programming.
- It is Easy to learn and is fastest NLP framework. It doesn’t make you choose the correct algorithm. Instead, it often provides the best and most efficient algorithm for a particular task without wasting any time.
- It uses neural networks for training some models and also provides built-in word vectors.
- It provides pre-built statistical models for few of the languages which can be downloaded and used based on the requirements. It comes with the trio (Tokenization, POS and NER with a side of syntatic dependency) all packed and ready. The design successfully hides the complexity with easy to use interface.
- The Matcher allows building of complex set of rules that can be used to detect any text sequence of our choice in addition to whatever model has to offer.

FUNCTIONALITIES - SpaCy

SpaCy has several functionalities to perform NLP such as:

- Pre-processing: tokenisation, sentence segmentation, lemmatisation, stopwords
- Linguistic features: part of speech tags, dependency parsing, named entity recogniser
- Visualizers for dependency trees and named entity recogniser
- Pre-trained word vectors and models
- Flexibility: can augment or replace any pipeline component or add new components such as TextCategorizer, etc.

We have taken below features for our project:

LEMMATIZATION

Lemmatization is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.
For example, lemmatization would correctly identify the base form of 'caring' to 'care', whereas, stemming would cut off the 'ing' part and convert it to car.

'Caring' -> Lemmatization -> 'Care'
'Caring' -> Stemming -> 'Car'

Also, sometimes, the same word can have multiple different 'lemmas. So, based on the context it's used, it should identify the 'part-of-speech' (POS) tag for the word in that specific context and extract the appropriate lemma.

spaCy is a relatively new in the space and is billed as an industrial strength NLP engine. It comes with pre-built models that can parse text and compute various NLP related features through one single function call. spaCy checks whether the lemma it's trying to generate is in the known list of words or exceptions for that part of speech.

Check out the lemmatizer.py file, specifically the lemmatize function at the bottom.

```
def lemmatize(string, index, exceptions, rules):
  string = string.lower()
  forms = []
  forms.extend(exceptions.get(string, []))
  oov_forms = []
  for old, new in rules:
    if string.endswith(old):
      form = string[:len(string) - len(old)] + new
      if not form:
        pass
      elif form in index or not form.isalpha():
        forms.append(form)
      else:
            oov_forms.append(form)
    if not forms:
        forms.extend(oov_forms)
    if not forms:
        forms.append(string)
    return set(forms)
```

For English adjectives, for instance, it takes in the string we're evaluating, the index of known adjectives, the exceptions, and the rules, as you've referenced, from this directory (for English model).

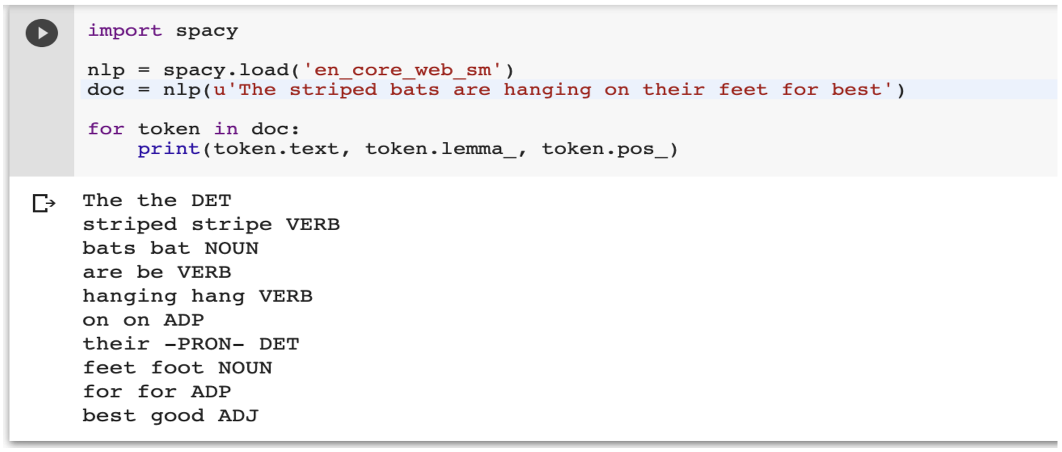
The first thing we do in lemmatize after making the string lower case is check whether the string is in our list of known exceptions, which includes lemma rules for words like "worse" -> "bad".
Then we go through our rules and apply each one to the string if it is applicable. For the word wider, we would apply the following rules:

```
["er", ""],
["est", ""],
["er", "e"],
["est", "e"]
```

and we would output the following forms: ["wid", "wide"].

Then, we check if this form is in our index of known adjectives. If it is, we append it to the forms. Otherwise, we add it to oov_forms, which I'm guessing is short for out of vocabulary. wide is in the index, so it gets added. wid gets added to oov_forms.

Lastly, we return a set of either the lemmas found, or any lemmas that matched rules but weren't in our index, or just the word itself.

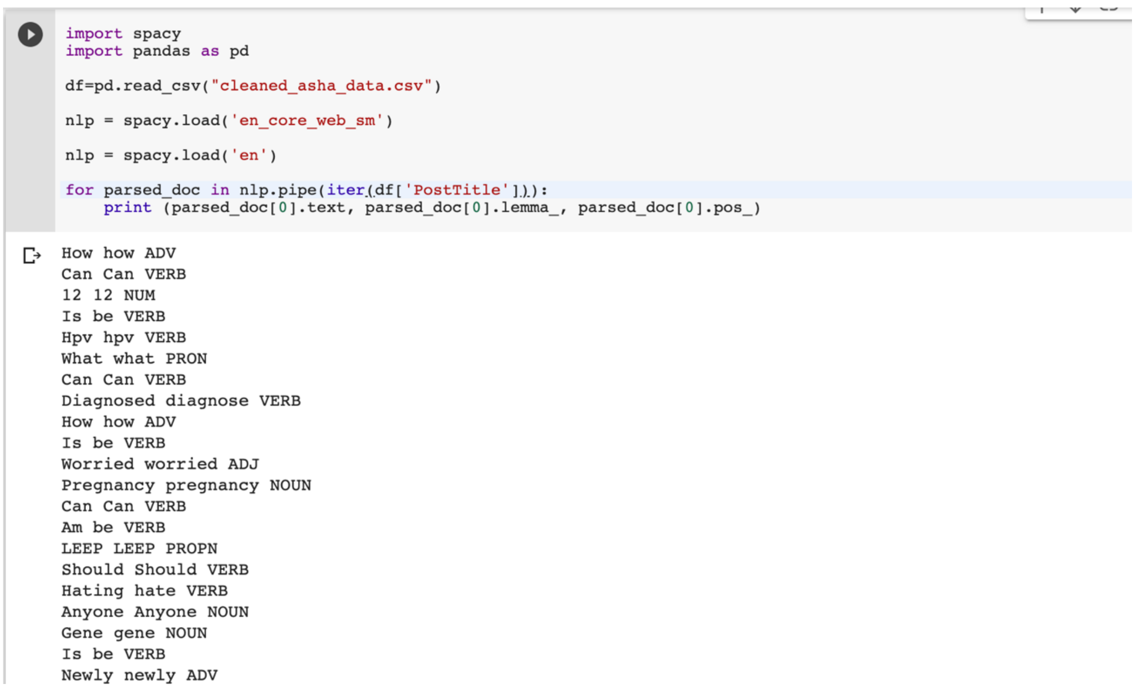


```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'The striped bats are hanging on their feet for best')

for token in doc:
    print(token.text, token.lemma_, token.pos_)
```

```
The the DET
striped stripe VERB
bats bat NOUN
are be VERB
hanging hang VERB
on on ADP
their -PRON- DET
feet foot NOUN
for for ADP
best good ADJ
```

It did all the lemmatizations the Wordnet Lemmatizer supplied with the correct POS tag did. Plus it also lemmatized ‘best’ to ‘good’.

You’d see the -PRON- character coming up whenever spacy detects a pronoun.



```python
import spacy
import pandas as pd

df=pd.read_csv("cleaned_asha_data.csv")

nlp = spacy.load('en_core_web_sm')

nlp = spacy.load('en')

for parsed_doc in nlp.pipe(iter(df['PostTitle'])):
    print (parsed_doc[0].text, parsed_doc[0].lemma_, parsed_doc[0].pos_)
```

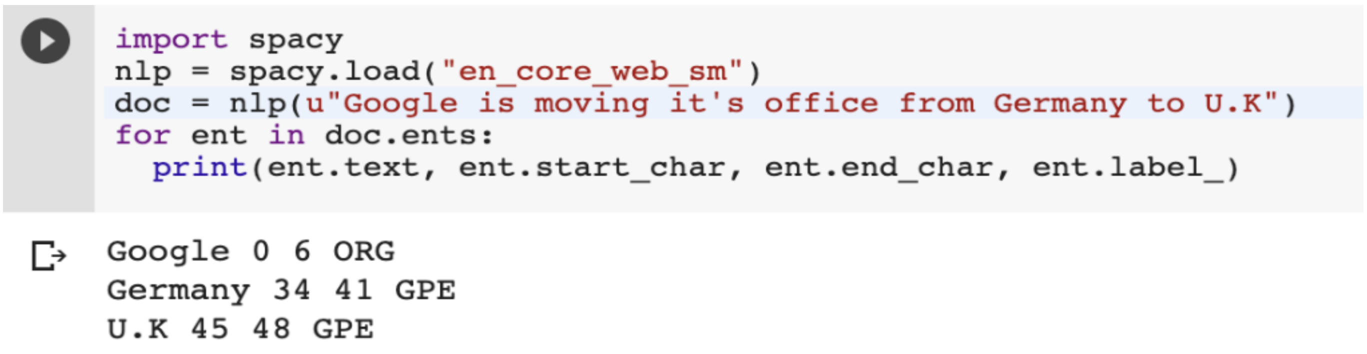
```
How how ADV
Can Can VERB
12 12 NUM
Is be VERB
Hpv hpv VERB
What what PRON
Can Can VERB
Diagnosed diagnose VERB
How how ADV
Is be VERB
Worried worried ADJ
Pregnancy pregnancy NOUN
Can Can VERB
Am be VERB
LEEP LEEP PROPN
Should Should VERB
Hating hate VERB
Anyone Anyone NOUN
Gene gene NOUN
Is be VERB
Newly newly ADV
```

We also tried the same by passing an entire column of data and SpaCy returns the output for each word in that column.

NAMED ENTITY RECOGNISATION

Name Entity Recognition (NER) task was firstly introduced at MUC-6 in 1995. Since that time, it has moved from rule-based systems to statistical systems with variety of advanced features. Named entity recognition (NER) is the first step towards information extraction that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. NER is used in many fields in Natural Language Processing (NLP), and it can help answering many real-world questions, such as:

- Which companies were mentioned in the news article?
- Were specified products mentioned in complaints or reviews?
- Does the tweet contain the name of a person? Does the tweet contain this person's location?



```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp(u"Google is moving it's office from Germany to U.K")
for ent in doc.ents:
  print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

```
Google 0 6 ORG
Germany 34 41 GPE
U.K 45 48 GPE
```

FUTURE SCOPE

The venerable NLTK has been the standard tool for the NLP in Python for some time, but spaCy has arisen as a competitor which has the goal of providing powerful, streamlined language processing. Compering these two toolkits there are some limitations can be pointed out for spaCy.

- Sentence tokenization process should be fast.
- Spacy should increase language support.
- Efficient entity recognition label, Tags.
- Media recognition (audio, video, images).

Language Limitation:
spaCy has only one support of language for English. Different languages may need different algorithms. With NLTK users can mix and match the algorithms according to the user's need, so developers that need to ensure a particular algorithm is being used will also want to stick with NLTK. spaCy has to make choice for each language and this process takes long time.

Time consuming sentence Tokenization:
spaCy performs poor in sentence tokenization. NLTK simply attempts to split the text into sentences. In contrast, spaCy constructing a syntactic tree for each sentence. It will slow the tokenization process for sentences.

Considering above limitations there is a good scope for spaCy to overcome these limitations and can be a better choice for most common users.

REFERENCES

- https://spacy.io/
- https://en.wikipedia.org/wiki/SpaCy
- https://explosion.ai/blog/introducing-spacy
- https://www.analyticsvidhya.com/blog/2017/04/natural-language-processing-made-easy-using-spacy-%E2%80%8Bin-python/

PRESENTATION LINK

https://prezi.com/view/el5rW3Qza7E31EaqjO7W/