Part 4: Step by Step Guide to Master NLP – Text Cleaning Techniques

CHIRAG GOYAL — June 20, 2021
Beginner Data Cleaning NLP Python Text

This article was published as a part of the Data Science Blogathon

Introduction

This article is part of an ongoing blog series on Natural Language Processing (NLP). In the previous part of this blog series, we complete the initial steps involved in text cleaning and preprocessing that are related to NLP. Now, in continuation of that part, in this article, we will cover the next techniques involved in the NLP pipeline of Text preprocessing.

In this article, we will first discuss some more text cleaning techniques which might be useful in some NLP tasks and then we start our journey towards the normalization techniques, **Stemming**, and **Lemmatization** which are very crucial techniques that you must know while you are working with on an NLP based project.

This is part-4 of the blog series on the Step by Step Guide to Natural Language Processing.

Table of Contents

More Text Cleaning Techniques

In the previous part of this blog series, I give a homework problem related to text cleaning and preprocessing. So, let's now discuss some of the techniques which might be useful while we doing an NLP task.

The first step after getting the text data in the text normalization process is to convert all text data into lowercase which makes all text on a level playing field i.e, no text has priority. With this step, we are able to cover each and every word available in text data for our analysis.

Removing HTML Tags

Typically, HTML tags are one of these components which don't add much value towards understanding and analyzing text i.e, not helpful for our analysis. When we collect the text data using techniques such as **web scraping** or **screen scraping**, it contained a lot of noise. So, we can remove unnecessary HTML tags and retain useful textual information for further steps of our analysis.

Removing Accented Characters

Usually, in any text data, you might have accented characters or letters, especially if you only want to analyze the English language. Hence, we have to convert and standardized these characters into ASCII characters. **For Example,** converting é to e.

Expanding Contractions

Contractions are words or combinations of words that are shortened by dropping letters and replacing them with an apostrophe. Let's have a look at some examples:

```
we're = we are; we've = we have; I'd = I would
```

In simple words, we can say that contractions are shortened versions of words or syllables. Or more simply, a contraction is an abbreviation used for representing a sequence of words.

In NLP, to resolve the constraints, we can be converting each contraction to its expanded and original form that helps with text standardization.

Removing Special Characters

Depends on the problem statement, special characters and symbols are usually non-alphanumeric characters or even occasionally numeric characters, which adds extra noise in unstructured text. Usually, to remove them we can use simple regular expressions.

Correction of typos

There are a lot of mistakes in writing English text or for other languages text, like Fen instead of a fan. Doing the correct map requires a dictionary by which we mapped words to their correct form based on similarity. This process is known as the Correction of typos.

Standardization

Often, text data contains words or phrases that are not present in any standard dictionaries.

For Example,

```
hashtags → #nlpisfun #preprocessing
acronyms → NLP, NLU, NLG
slangs → "rt", "luv", "dm"
```

The final aim of text standardization is to transform these words and phrases into their normal or standard form. To do this, we require manual efforts using data dictionaries, or directly with the help of regular expressions in python.

What is Normalization?

Due to grammatical reasons, a document can contain:

- Different forms of a particular word such as drive, drives, driving.
- Related words having a similar meaning, such as nation, national, nationality.

So, with the help of normalization techniques, we reduce the inflectional forms and sometimes derivationally related forms of a given word to a common base form.

Now, let's define the term "Normalization" formally,

Normalization is the process of converting a token into its base form. In this process, the inflection from a word is removed so that the base form can be obtained.

For Examples,

```
am, are, is => be
dog, dogs, dog's, dogs' => dog
```

If we apply the above mapping to the below sentence, then the text will be look something like this:

```
the boy's dogs are different sizes => the boy dog be a different size
```

Important Points about Normalization

1. Normalization is useful in many ways. Some of them are as follows:

- Reduces the number of unique tokens present in the text,
- Removes the variations of a word in the text, and
- Removes the redundant information too.

The popular methods which are used for normalization are Stemming and Lemmatization. However, they are different from each other.

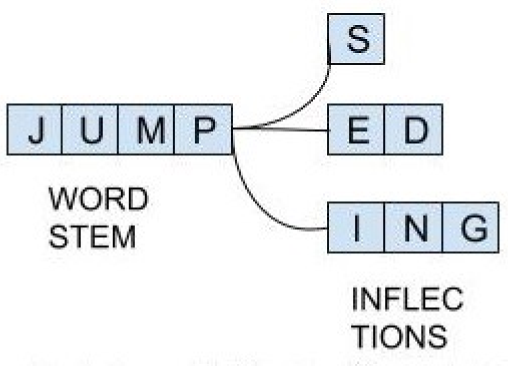
2. In the Feature Engineering step for text data, Normalization becomes a crucial step as it converts the high dimensional features to the low dimensional space, which is an ideal task for any Machine learning model.

Stemming

To understand stemming, firstly we have to understand the meaning of the word stem. **Word stems,** also known as the base form of a word, and we can create new words by attaching affixes to them in a process known as inflection.

For Example, for the stem word JUMP, we can add affixes to it and form new words like JUMPS, JUMPED, and JUMPING.

```
JUMP -> JUMPS, JUMPED, JUMPING
```



Word stem and its inflections (Source: Text Analytics with Python, Apress/Springer 2016)

What does the above figure represent?

So, the reverse process of finding the base form of a word from its inflected form is called stemming. It helps us in standardizing words to their base or root stem, irrespective of their inflections, which helps many applications like:

- Classifying text
- Clustering text, and
- Information retrieval, etc.

Let's define the stemming formally,

Usually, Stemming refers to a crude heuristic process that simply chops off the ends of words i.e, suffixes and prefixes with the aim of achieving the above-discussed goal correctly most of the time and often includes the removal of derivational affixes.

For Example, Consider the following words,

```
Words: "laughing", "laughed", "laughs", "laugh"
```

All the above words will become "laugh", which is their stem because their inflection form will be removed.

"laughing", "laughed", "laughs", "laugh" >>> "laugh"

Since the above four words don't differ much in their meaning in the context of other words, so it is a good idea to just give the word stem to our model to learn so it will have a much less word size that it needs to focus on. Therefore, it is a very powerful technique.

How did Stemming Algorithms work?

They took a language and identified some conditions on the language (suffix and prefix OR affixes). Based on language, after identifying suffix and prefix and by eliminating this they will become the stem words.

Sometimes stemming can produce words that are not in the dictionary, therefore stemming is not a good normalization technique. **For Example**, Consider the given sentence:

```
Sentence: His teams are not winning
```

After stemming the tokens that we will get are- "hi", "team", "are", "not", "winn"

From the above example, we have observed that the keyword **"winn"** is not a regular word, and the word "**hi**" changed the context of the entire sentence.

So, there are too many errors in stemming and it's not error-free because it is based on heuristics and there is no perfect rule that converts every word correctly to their stem word.

Now, discuss some other problems with the Stemming algorithm:

Over stemming

In over stemming, too much of the word is cut off (lost) or two words of different stems map to the same words. Let's consider the given four words:

```
Words: University, Universities, Universal, Universe
```

Our stemming algorithm maps all these four words to the base word as "Univers", but in reality, Universal and Universe are from the same stem while university and Universities are from the different stem.

It is just the opposite of Overstemming in which two words of the same stem are mapped to the different stems by our stemming algorithm. Let's consider the following two words:

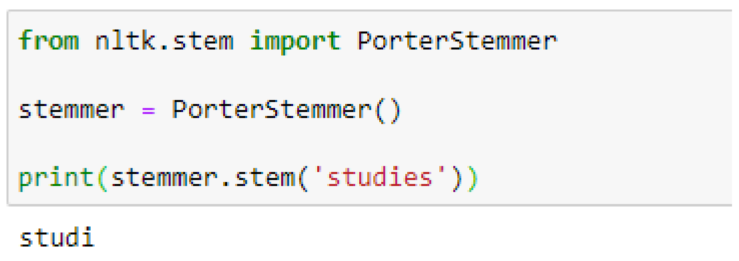
```
Words: Data, Datum
```

Our stemming algorithm mapped Data to **"dat"** and datum to **"datu"** as their stem words but in reality, both these words belong to the same stem.

Different types of stemmers in NLTK are as follows:

- PorterStemmer,
- LancasterStemmer,
- SnowballStemmer, etc.

Here is one quick example using Porter Stemmer in NLTK.



```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('studies'))
```

```
studi
```

Lemmatization

Let's understand why we required Lemmatization while we have Stemming in our hand?

To overcome the problems of Stemming i.e, reductions can produce a "root" word that isn't actually a word and does not make much sense, we introduce the concept of lemmatization.

Lemmatization is an organized & step by step procedure of obtaining the root form of the word, as it makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations).

The output of lemmatization is the root word called **lemma. For Example,**

```
Am, Are, Is >> Be
Running, Ran, Run >> Run
```

Important points about Lemmatization

1. Since Lemmatization is a systematic process so while performing lemmatization we can specify the part of the speech tag for the desired term and lemmatization will only be performed when the given word has a proper part of the speech tag associated with it.

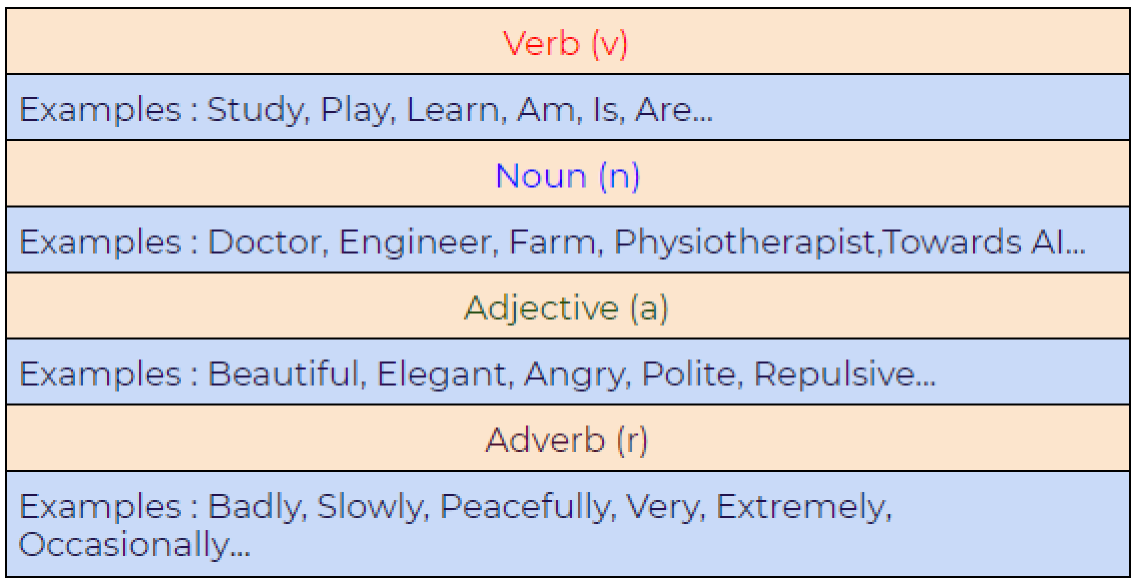
For Example, Consider the following example:

```
running as a verb -> run
running as a noun -> no conversion
```

From the above example, we have observed that if we try to lemmatize the word **running** as a **verb**, it will be mapped to **run** while if we try to lemmatize the same word **running** as a **noun** it won't be converted.

2. In Lemmatization, the root word is always a lexicographically correct word (present in the dictionary). It uses a knowledge base called **WordNet, which we will be discussed in our subsequent parts of the series**. Because lemmatization takes the help of knowledge, it can even convert words that are different that can't be solved with the help of stemmers.

3. Lemmatization takes Part Of Speech (POS) values in its consideration. Also, it may generate different outputs for different values of POS. Generally, we have four choices for POS:



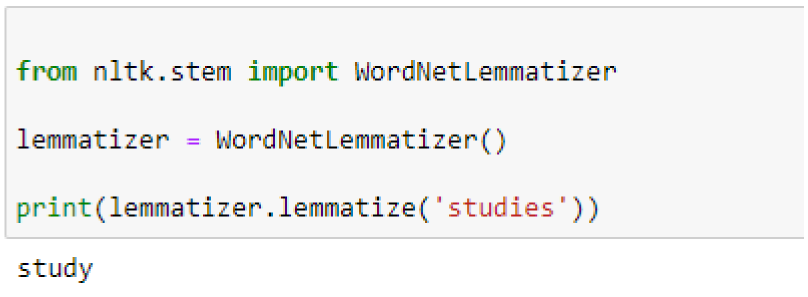
Verb (v)
Examples : Study, Play, Learn, Am, Is, Are...
Noun (n)
Examples : Doctor, Engineer, Farm, Physiotherapist,Towards AI...
Adjective (a)
Examples : Beautiful, Elegant, Angry, Polite, Repulsive...
Adverb (r)
Examples : Badly, Slowly, Peacefully, Very, Extremely, Occasionally...

Image Source: Google Images

Based on the applicability you can choose any of the below lemmatizers

- Wordnet Lemmatizer
- Spacy Lemmatizer
- TextBlob
- CLiPS Pattern
- Stanford CoreNLP
- Gensim Lemmatizer
- TreeTagger

Here is one quick example using Wordnet lemmatizer in NLTK.



```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize('studies'))
```

```
study
```

Homework Problem

Please try the following questions in Python implementation using NLTK by yourself to gain a better understanding of the topics.

- Try to implement a basic example that demonstrates how a lemmatizer works
- Try to make a Lemmatizer with default POS value
- Try to make another example demonstrating the power of lemmatizer
- Try to form Lemmatizer with different POS values

Difference between Stemming and Lemmatization

The difference between stemming and lemmatization based on some criterion are as follows:

Based on Context Consideration

Lemmatization considers the context and converts the word to its meaningful base form, whereas stemming simply chops off the ends of a word using heuristics that often leading to incorrect meanings and spelling errors.

Stemming is Typically faster but not that accurate

Stemmer operates **without knowledge of the context** and therefore stemming algorithms can not understand the difference

- They are **easier to implement,** and
- They usually **run faster**.
- Also, the less value of “accuracy” may not matter for some use-cases.

Let’s discuss the above concept with the help of the following examples:

1. Consider the word “better” which mapped to “good” as its lemma. This type of mapping is missed by stemming since it requires knowledge of the dictionary.

2. Consider the word “play” which is the base form for the word “playing”, and hence this is the same for both stemming and lemmatization.

3. Consider the word “meeting” that can behave as either the base form of a noun or a verb as **“to meet”** depending on the context. **For Examples**,

```
Sentences:
“in our last meeting”
“We are meeting again tomorrow”
```

Unlike stemming, lemmatization tries to select the correct lemma depending on the context.

Lemmatization is typically more Accurate

Lemmatization aims to achieve a similar base “stem” for a specified word. However, it always finds the dictionary word as their stem instead of simply chops off or truncating the original word. Therefore, lemmatization also considers the context of the word. That is why it more accurate than stemming.

Speed vs Accuracy tradeoff

Stemming becomes an appropriate approach if accuracy is not the project’s final goal whereas if we want accuracy on the higher side and the project is not on a tight deadline, then the best option is Lemmatization.

The process of Lemmatization aims to reach the same result as that of stemming but it takes an alternative approach that takes the part of speech in their consideration, which considers whether the word is a noun, verb, adjective, and so on. So, the process of lemmatization is more intensive and slower, but it gives more accurate results than stemming.

Chunking

In Chunking, we try to extract meaningful phrases from unstructured data in the form of text. By tokenizing an article into words, it’s sometimes hard to find some meaningful information.

It works on top of Part of Speech(POS) tagging, which we already covered in **part-3** of this series. Therefore, Chunking takes input as POS tags and gives chunks as output.

In simple words, chunking means a group of words, which breaks simple text data into phrases that give more meaningful information than individual words.

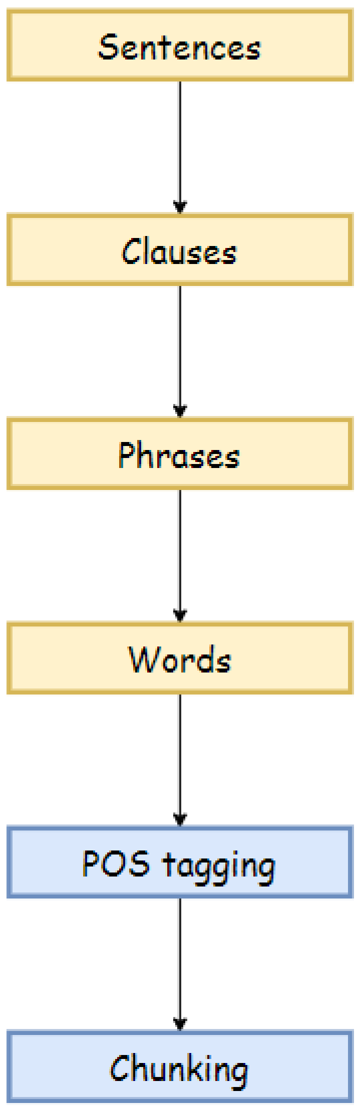


Image Source: Google Images

Before seeing an example of chunking, we have an idea about what phrases are?

Phrases are meaningful groups of words. There are five significant categories of phrases.

- Noun Phrases (NP).
- Verb Phrases (VP).
- Adjective Phrases (ADJP).
- Adverb Phrases (ADVP).
- Prepositional Phrases (PP).

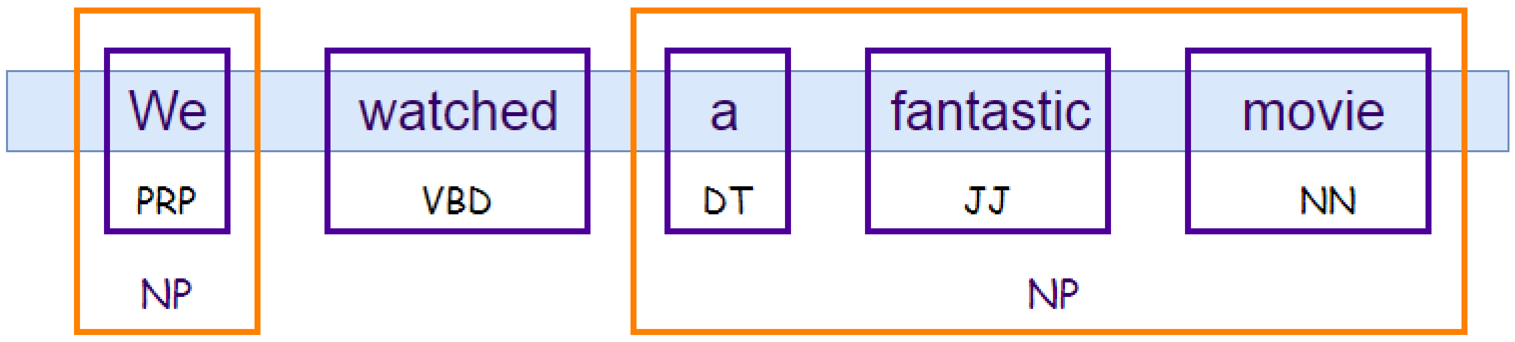
Phrase Structure Rules

- S(Sentence) → NP VP.
- NP → {Determiner, Noun, Pronoun, Proper name}.
- VP → V (NP)(PP)(Adverb).
- PP → Pronoun (NP).
- AP → Adjective (PP).

For Example,

Consider the following sentence:

```
Sentence: We watched a fantastic movie
```



In Chinking, we exclude some part from our chunk, as sometimes there are certain situations where we have to exclude a part of the text from the whole text or chunk.

But remember that in complex extractions, it might be possible that chunking can give unuseful data as the output. Therefore, In such case scenarios, chinking comes into the picture to exclude some parts from that chunked text.

This ends our Part-4 of the Blog Series on Natural Language Processing!

Other Blog Posts by Me

You can also check my previous blog posts.

Previous Data Science Blog posts.

LinkedIn

Here is **my Linkedin profile** in case you want to connect with me. I'll be happy to be connected with you.

Email

For any queries, you can mail me on Gmail .

End Notes

Thanks for reading!

I hope that you have enjoyed the article. If you like it, share it with your friends also. Something not mentioned or want to share your thoughts? Feel free to comment below And I'll get back to you. 😉

The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.

Part 2: Step by Step Guide to NLP - Knowledge Required to Learn NLP

Getting started with NLP using NLTK Library

Part 3: Step by Step Guide to NLP - Text Cleaning and Preprocessing

blogathon data cleaning NLP python

About the Author

CHIRAG GOYAL

Our Top Authors