PennState World Campus

DAAN862: Analytics Programming in Python

Lesson 12: Text Mining with re and NLTK

Lesson 12: Objectives and Overview (1 of 6)

Lesson 12: Text Mining with re and NLTK

Text mining is the process of finding useful information that hides in plain-language text. Lately, text mining has gained more attention due to the great amount of data that is generated everyday. Text mining is a very big topic, in this lesson you will get a small taste of the basics.

At the end of this lesson, students will learn how to:

Explain what text mining is

Apply Regular Expression package to text mining tasks

Perform text miing with NLTK package

By the end of this lesson, please complete all readings and assignments found in the Lesson 12 Course Schedule.

Lesson 12.1: Introduction to Text Mining (2 of 6)

Lesson 12.1: Introduction to Text Mining

Why Text Mining?....because text data has been growing fast lately.

2.5 million TB data is created in a day

The accumulated data will grow to 44 billion TB by 2020

Approximately 80% of all tex-rich data is estimated to be unstrucutured:

More than 40 million artilces in Wikipedia

More 500 million tweets a day, 200 billion a year

More than 31 million messages and 2.8 million videos created on Facebook

More than 40,000 searches per second on Google, which makes 3.5 billion per day

What can be done with text?

Parse text, breaking text data into smaller chunks according to a set of rules

Identify relationship and assertation from text

Classify text documents

Search for relevant text documents

Sentiment analysis (positive, negative, neutral, etc...)

(click tabs to learn more)

What is Natural Language Processing?

Natural language is a language used for people's daily communication such as English, German, French or Chinese etc... Such languages have evolved for thousands of years. Unlike programing language or mathematics representations, they don’t have explicit rules. Natural language processing (NLP) is to find any kind of computer manipulation of natural language such as counting, word frequencies, and understanding the meaning.

Introduction to NLP:

First, we are going to introduce some vocubulary used in NLP:

Corpus - Body of text. Example: A collection of journal articles, reviews, tweets etc..

Lexicon - Words and their meanings such as dictionaries for different languages. Sometimes, different fields may have different lexicons. For example, “Bull” means the market goes up in finance but it means an animal in English dictionary.

Token – Basic components of corpus such as words, phrases or sentences. Tokenization is a process of beaking down textual data into smaller components. The most common tokenization methods include word and sentence tokenization.

Transcript

In this video, I will introduce re, regular expression package. Let’s import it first.

We will use text2 as an example. Backslash t means tab, which will be considered as space too.

The first function I will introduce is split function. In split function, pattern equals to backslash s plus which means more than one or more spaces. String is text2.

The result is a list of words, and we can see that the string has been successfully split into words.

Although the string object has a built-in split method, you can split it with a specific number of spaces.

There is another way to split the data, you can use compile function to compile the pattern first, toAll re functions can also be the method for regular expression object created by compile function. We define regex is the results from re.compile patterns equals to multiple white spaces.

Regex method split can be used to split the string text2, which creates the same results with the top approach.

The findall method is to find all patterns in the strings. Tab will be considered as space too.

Text3 is a string contains several email address, how can we extract emails from it.

First, we want to define the email format as a pattern. Before @, the character could be A to Z, 0 to 9 dot, underscore, percent, plus or minus. It could appear one or more time, that's why here is plus sign. After @ the character could be A to Z, 0-9, dot, minus which could appear one or more time. Then it will follow the dot. After a dot, it should be characters A to Z which appears 2 to 4 times.

Now you can compile this pattern into a regular expression object called regex.

First, we try the match method which is to check if a text matches the pattern, and it will return the start index and end index if it could find such a pattern.

The findall method returns all matches of the pattern. We can see that all email addresses have been extracted from the string.

Search function only searches the location of the first match, and it returns the start and end index.

Lesson 12.2: Regular Expression

In Lesson 4, we introduced string operations, methods, and vectorized string methods in pandas. Here we introduce another package re for finding patterns in text.

Regular expressions provide a flexible way to search or match (often more complex) string patterns in text. A single expression, commonly called a regex, is a string formed according to the *regular expression* language. Let's use email format in Figure 12.1 as an example:

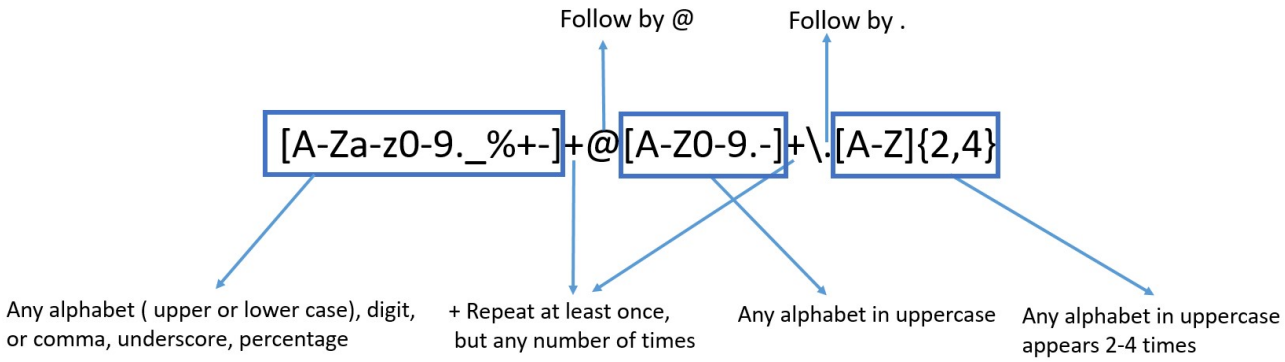

Fig 12.1 (click to enlarge)

A cheat sheet for all regular expression can be found here.

Python provides a built-in *re* module to perform regular expressions on strings. Some example will show later how functions in the *re* module work. These functions can be split into three categories: pattern matching, splitting, and substitution. The main functions of the *re* package are listed in Table 12.2.1:

Table 12.2.1: Main Functions of the re package

Function	Description
compile	Compile a regular expression pattern into a regular expression object.
ignorecase	Perform case-insensitive matching.
match	Checks If zero or more characters at the *beginning* of *string* match this regular expression.
search	Scan through string looking for the first location where this regular expression produces a match, and return a corresponding match object.
fullmatch	If the whole string matches this regular expression.
split	Split string by the occurrences of pattern.
sub	Replaces all patterns with a string or results of a function.
findall	Find all occurrences of a pattern.
escape	Escape special characters in pattern.

Figure 12.2 shows how to import the package in Python:

```
In [1]: import re
```

Fig 12.2 (click to enlarge)

Now, let's consider a sentence with words seperated by multiple spaces. You can use split function and use pattern (shown in Figure 12.2) = '\a+' (more than one space):

```
In [2]: text2 = 'Python is   \t        a   great      tool for\t data analysis'

In [3]: re.split('\s+', text2) # \s+ means more than one space.
Out[3]: ['Python', 'is', 'a', 'great', 'tool', 'for', 'data', 'analysis']
```

Fig 12.3 (click to enlarge)

You can also compile a pattern into a regular expression object, which can be used for split(), matach(), findall() and search() functions.

Figure 12.4 shows how you can compile a regular expression pattern into a regular expression object, which can be used for matching using its match(), search() and other methods, described below:

```
In [4]: regex = re.compile('\s+')   # compile the pattern

In [5]: regex.split(text2)          # split text2 by the pattern
Out[5]: ['Python', 'is', 'a', 'great', 'tool', 'for', 'data', 'analysis']

In [6]: regex.findall(text2)        # find all patterns in text2
Out[6]: [' ', '   \t        ', '   ', '      ', ' ', '\t ', ' ']
```

Fig 12.4 (click to enlarge)

Figure 12.5 shows how you can define your pattern for the email first:

```
In [8]: pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
```

Fig 12.5 (click to enlarge)

After defining your pattern you can compile the pattern and ignore the case as shown in Figure 12.6:

```
In [9]: regex = re.compile(pattern, flags=re.IGNORECASE)  # compile a pattern for
emails

In [10]: m = regex.match('wesm@bright.net, edd')     # check if a text match the
pattern
```

Fig 12.6 (click to enlarge)

After you apply match function, it will return the start index and end index for the first subtext which matches the pattern as shown in Figure 12.7:

```
In [11]: m.start()          #  the first index of the text matching with pattern
Out[11]: 0

In [12]: m.end()            # the last index of the text matching with pattern
Out[12]: 15

In [13]: 'wesm@bright.net, edd'[m.start(): m.end()] # email in the text
Out[13]: 'wesm@bright.net'
```

Fig 12.7 (click to enlarge)

Figure 12.8 shows how *findall* is used to find out how all texts match the pattern. If you apply the findall function to text3 all emails will be returned:

```
In [14]: text3 = """Dave dave@google.com
    ...: Steve steve@gmail.com
    ...: Rob rob@gmail.com
    ...: Ryan ryan@yahoo.com
    ...: """

In [15]: regex.findall(text3)
Out[15]: ['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

Fig 12.8 (click to enlarge)

Since the *match* only finds the first pattern in the text, it only returns the first email dave@google.com in the text3 as shown in Figure 12.9:

```
In [16]: m = regex.search(text3)

In [17]: m
Out[17]: <_sre.SRE_Match object; span=(5, 20), match='dave@google.com'>

In [18]: text3[m.start():m.end()]
Out[18]: 'dave@google.com'
```

Fig 12.9 (click to enlarge)

Transcript

In this video, I will introduce NLTK package, the natural language processing package. Let's import it first.

There are two main tokenization: sentence and word.

This is an example of extract sentences from text 2. You need to use sent underscore tokenize function. And as a result, text2 has been split into sentences.

The next

We will use built-in books in nltik as an example.

From nltk.book import * means to import everything under nltk.book. There are 9 books in it and their names are text1, text2, …, text9.

Here we use text2 as an example, which is the book "Sense and Sensibility".

You can check the first 9 sentences by sent plus a number.

With the text ready, you can explore the basic information.

Like how many words in text2 by using len function

How many unique words in text2 by finding the length of the set of text2

Length of each sentence.

FreqDist the function to calculate the unique words and their counts.

Since it returns a dict, you can use len to check the unique length.

You can use a for loop to find the frequent words. Here we choose the count is larger than 100 and the words should have more than five characters.

Here is the head of the results.

The next part is to normalize the works,

For example, you have “reply” in different formats. Typically, you want to convert it to lower cases and split it into a list.

Then, you can apply PorterStemmer() or lemmatization.

This is a one-line for loop, you convert all words in words1 with porterstemmer and return the results as a list.

Use sent3 as an example, we can see that it returns different results.

Lesson 12.3: NLTK Basic

pandas has a natural language processing package, called *NLTK*. You can import the packages into Python (shown in Figure 12.10):

Fig 12.10 (click to enlarge)

Let's use NLTK to perform some basic natural language tasks:

Sentence Tokenization:

Sentence tokenization is to extract sentences from a text corpus, which is the first level of tokens.

There are several ways for sentence tokenization. Basic methods include looking for delimiters between sentences such as:

A period (.)

A newline character (\n)

A semi-colon (;)

NLTK provides various interfaces for performing sentence tokenization. In Figure 12.11 we only introduce *sent_tokenize* function:

```
In [22]: text2 = '''This is the first sentense. Nothing happened.
    ...:  Is this the third one? Yes, it is'''

In [23]: nltk.sent_tokenize(text2)
Out[23]:
['This is the first sentense.',
 'Nothing happened.',
 'Is this the third one?',
 'Yes, it is']
```

Fig 12.11 (click to enlarge)

Work Tokenization:

Word tokenization is to extract words from sentences or texts. This is the most important process in NLP, especially in text cleaning or normalization. The standard NLP cleaning and normalization process include:

Split a text into a list of words

Remove special characters

Perform stemming and lemmatization on each word

Find frequency of unique words

Figure 12.12 shows both split method and NTLK function to perform word tokenization on text1:

Fig 12.12

From the example, we can see that there is a difference between the split method and nltk.wod_tokenize function. The former considers quotations as part of words, and the later one considers quotations as words.

Lesson 12.4: Text Normalization

Text normalization is a process to transform the text in some way such that it can be analyzied by other algorithms or applications as input. In other words, text normalization can be considered as text cleaning and wrangling. After we extract the words or sentence from a text, it needs additional processing such as case conversion, frequent counts, stemming and lemmatization.

In Figure 12.13, we are going to use the built-in books in nltk. You can see that nltik.book contains 9 books and their names are listed below:

```
In [27]: from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Fig 12.13 (click to enlarge)

You can get the information for the second book by using text2 and sent2 as shown in Figure 12.14:

```
In [28]: text2
Out[28]: <Text: Sense and Sensibility by Jane Austen 1811>

In [29]: sent2
Out[29]:
['The',
 'family',
 'of',
 'Dashwood',
 'had',
 'long',
 'been',
 'settled',
 'in',
 'Sussex',
 '.']
```

Fig 12.14 (click to enlarge)

Counting vocabularly of words, Frequency of words, Stemming, and Lemmatization:

(click tabs to learn more)

Counting Vobabulary of Words:

To gain basic information about the text you can use the example shown in Figure 12.15:

```
In [30]: len(text2)                  # How many words in text2
Out[30]: 141576

In [31]: len(set(text2))             # How many unique words in text2
Out[31]: 6833

In [32]: list(set(text2))[:10]       # Show 10 unique words
Out[32]:
['YOU',
 'Honiton',
 'advised',
 'slightest',
 'prevailing',
 'tranquil',
 'stiffly',
 'don',
 'mortification',
 'owning']

In [33]: len(sent2)                  # How many words in sent2
Out[33]: 11
```

Fig 12.15 (click to enlarge)

Frequency of Words:

FreqDist function is used to get unique words and their frequencies from a text and the results are stored in a dictionary (shown in Figure 12.16 A&B). After applying FreqDist to the text2 “Sense and Sensibility”, we can see 6833 unique words:

```
In [34]: dist = FreqDist(text2)     # Return a dictionary with words as keys and
their freq as values

In [35]: len(dist)                  # How many unique words
Out[35]: 6833
```

Fig 12.16 (A) (click to enlarge)

```
In [36]: words = dist.keys()        # Unique words

In [37]: list(words)[:10]           # Show head of words
Out[37]:
['[',
 'Sense',
 'and',
 'Sensibility',
 'by',
 'Jane',
 'Austen',
 '1811',
 ']',
 'CHAPTER']
```

Fig 12.16 (B) (click to enlarge)

We can find all words that contains more than 5 alphabets and frequency larger than 100 shown in Figure 12.17:

```
In [38]: freqwords = [w for w in words if len(w) > 5 and dist[w] > 100]

In [39]: freqwords[:10]
Out[39]:
['Dashwood',
 'sister',
 'before',
 'mother',
 'nothing',
 'himself',
 'however',
 'thought',
 'enough',
 'little']
```

Fig 12.17 (click to enlarge)

Stemming:

Morphemes consist of words stem and affixes (prefixes, suffixes etc.). Stemming is the process of obtaining the base form of a word. Figure 12.18 shows how stemming process works:

```
In [40]: example1 = 'reply Replying replied REPLIES'

In [41]: words1  = example1.lower().split(' ')

In [42]: words1
Out[42]: ['reply', 'replying', 'replied', 'replies']
```

Fig 12.18 (click to enlarge)

PorterStemmer is a word stemmer in nltk based on the Porter stemming algorithm (https://tartarus.org/martin/PorterStemmer/) . We can use a for loop to apply the PorterStemmer function to each word as shown in Figure 12.19:

```
In [43]: porter = nltk.PorterStemmer()

In [44]: [porter.stem(w) for w in words1]
Out[44]: ['repli', 'repli', 'repli', 'repli']
```

Fig 12.19 (click to enlarge)

Lemmatization:

Lemnatization is a similar process to stemming. Instead of obtaining the root stem, it returns the root word. The difference between root stem and root word is that you may not find the root stem in a dictionary but you always find out the root word.

The following code in Figure 12.20 shows how to use both stemming and lemmatization on sent3 of the book “Sense and Sensibility”. First, we apply PorterStemmer to the sent3:

```
In [45]: sent3
Out[45]:
['In',
 'the',
 'beginning',
 'God',
 'created',
 'the',
 'heaven',
 'and',
 'the',
 'earth',
 '.']

In [46]: [porter.stem(w) for w in sent3]
Out[46]:
['In',
 'the',
 'begin',
 'god',
 'creat',
 'the',
 'heaven',
 'and',
 'the',
 'earth',
 '.']
```

Fig 12.20 (click to enlarge)

Figure 12.20 returns ‘creat’ instead of ‘create’ if you using PorterStemmer.

WordNetLemmatizer is the lemmatization model in nltk. Figure 12.21 shows the result for applying WordNetlemmatizer to sent3:

```
In [47]: WNlemma = nltk.WordNetLemmatizer()

In [48]: [WNlemma.lemmatize(w) for w in sent3]
Out[48]:
['In',
 'the',
 'beginning',
 'God',
 'created',
 'the',
 'heaven',
 'and',
 'the',
 'earth',
 '.']
```

Fig 12.21 (click to enlarge)

It returns ‘beginning’ and ‘created’ instead of ‘begin’ and ‘creat’.

Lesson 12 References

- https://docs.python.org/3/library/re.html# (https://docs.python.org/3/library/re.html)
- http://www.cbs.dtu.dk/courses/27610/regular-expressions-cheat-sheet-v2.pdf
- Chapter 3 “Text Analytics with Python”, Dipanjan Sarkar
- Natural Language Processing with Python, https://www.nltk.org/book/ (https://www.nltk.org/book/)
- Chapter 7, Data analytics with Python
- https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#34cfd2f917b1 (https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#2846c2fb17b1)