DSC 5101 – Computer Programming in DSAI

Part II - Working with Data

HO 07 - Text Processing with NLTK

Opim Salim Sitompul

Department of Data Science and Artificial Intelligence
Universitas Sumatera Utara

DS&AI

Co-funded by the Erasmus+ Programme of the European Union

Outline I

Introduction

- Natural language is a form of written and spoken communication that has developed organically and naturally.
- Processing means analyzing and making sense of input data with computers [4].

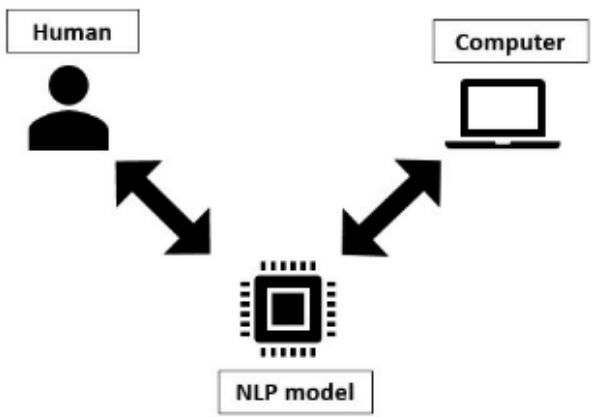

Figure 1: Natural Language Processing (Source: [4])

Introduction

- Natural language processing is the machine-based processing of human communication.
- It aims to teach machines how to process and understand the language of humans, thereby allowing an easy channel of communication between human and machines.
- Example: Google Assistant, Microsoft Cortana, Alexa and Siri.
- Natural language processing algorithms aid these technologies in communicating with humans.

Introduction

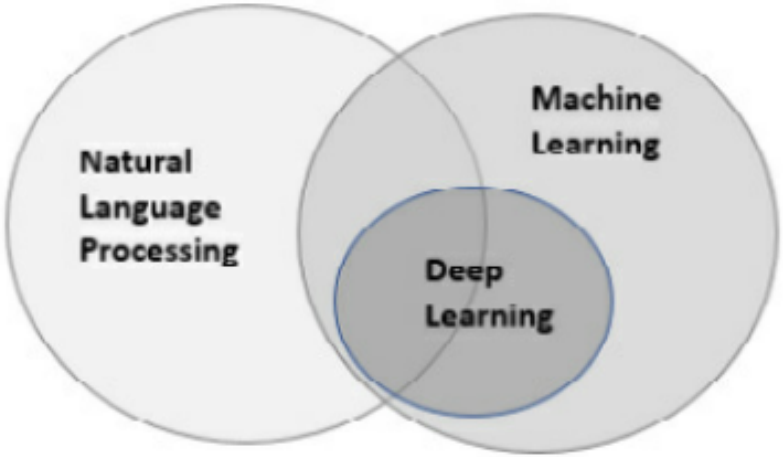

Figure 2: Venn diagram for NLP (Source: [4])

- The basic point between programming languages is the same, that is to communicate with machine using language understandable by machines.

Introduction

- However, the purpose of NLP is different: rather than having humans conform to the ways of a machine and learn how to effectively communicate with them, natural language processing enables machines to conform to humans and learn their way of communication.
- Natural language processing is a subfield of artificial intelligence along with machine learning and deep learning.
- Because it deals with natural language, NLP is at intersection of artificial intelligence and linguistics.

Introduction

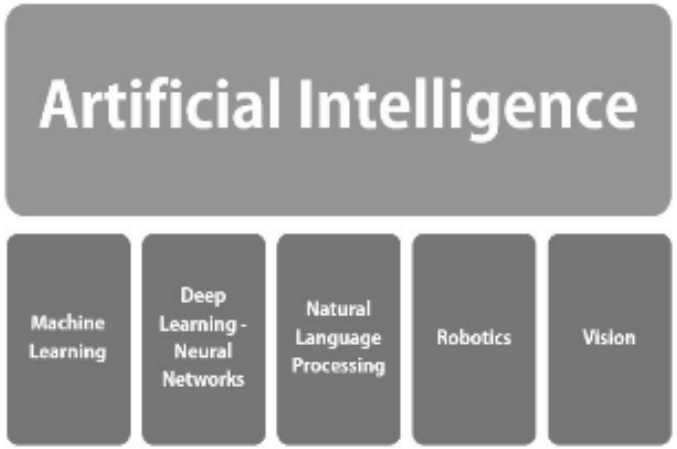

Figure 3: AI and its subfields (Source: [4])

- NLTK Capabilities:
  - Speech Recognition
  - Natural Languange Understanding
  - Natural Languange Generation

Preparing the NLTK

- To begin using NLTK there are some preparation to be made, such as downloading several data needed in text processing.
  - Downloading *punkt:*
  - *Punkt Sentence Tokenizer*. This tokenizer divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences.
  - It must be trained on a large collection of plaintext in the target language before it can be used.

```
import nltk
nltk.download('punkt')
```

Word Tokenizer

- Example:
  s = "C4.5 is a set of algorithms for classification problems in machine learning and data mining."
  tokens = nltk.word_tokenize(s)
  print(tokens) $\rightarrow$
  ['C4.5', 'is', 'a', 'set', 'of', 'algorithms', 'for', 'classification', 'problems', 'in', 'machine', 'learning', 'and', 'data', 'mining', '.']
- Result obtained is not get rid of punctuations, such as '.'

Word Tokenizer

- The following code can be used to exclude punctuations.
  from nltk.tokenize import RegexpTokenizer
  tokenizer = RegexpTokenizer(r'\w+')
  print(tokenizer.tokenize(s)) $\rightarrow$ ['C4', '5', 'is', 'a', 'set', 'of', 'algorithms', 'for', 'classification', 'problems', 'in', 'machine', 'learning', 'and', 'data', 'mining']
- However, it also remove '.' from "C4.5".

Word Tokenizer

- A better approach to remove punctuation is using *filter* of a lambda function:

```
1  def no_punctuation(s, list_of_punct):
2      from nltk.tokenize import word_tokenize,
           sent_tokenize
3      tokens = [word for sent in sent_tokenize(s
           ) for word in word_tokenize(sent)]
4      res = list(filter(lambda word: word not in
            list_of_punct, tokens))
5      return res
```

Word Tokenizer

- Calling the function such as:
  new_s = no_punctuation(s, ',.')
  print(new_s) → ['C4.5', 'is', 'a', 'set', 'of', 'algorithms', 'for', 'classification', 'problems', 'in', 'machine', 'learning', 'and', 'data', 'mining']
  ss = 'Hey, Good morning!'
  new_ss = no_punctuation(s, ',!')
  print(new_ss) → ['Hey', 'Good', 'morning']

Stop Words

- Downloading stopwords
  nltk.download('stopwords')
- Post Tagging without stop words:
  tagged = nltk.pos_tag(tokens)
  print(tagged) $\rightarrow$

[('C4.5', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('set', 'NN'), ('of', 'IN'), ('algorithms', 'NN'), ('for', 'IN'), ('classification', 'NN'), ('problems', 'NNS'), ('in', 'IN'), ('machine', 'NN'), ('learning', 'NN'), ('and', 'CC'), ('data', 'NN'), ('mining', 'NN'), ('.', '.')]

Stop Words

- Removing stopwords:

```
1 from nltk.corpus import stopwords
2 from nltk.tokenize import word_tokenize,
      sent_tokenize
3
4 def stopwords_token(sentence):
5   stop_words = set(stopwords.words('english'))
6   tokenized = sent_tokenize(sentence)
7   for i in tokenized:
8     wordsList = nltk.word_tokenize(i)
9     # removing stop words from wordList
10    wordsList = [w for w in wordsList if not w
          in stop_words]
11    #  Using POS-tagger.
12    tagged = nltk.pos_tag(wordsList)
13  return(tagged)
```

Post Tagger

- Calling post-tagger:
  tokenize = stopwords_token(sentence)
  print(tokenize) $\rightarrow$

[('C4.5', 'NNP'), ('set', 'VBD'), ('algorithms', 'RP'), ('classification', 'NN'), ('problems', 'NNS'), ('machine', 'NN'), ('learning', 'VBG'), ('data', 'NNS'), ('mining', 'NN'), ('.', '.')]

Wordnet

- Downloading WordNet:
- WordNet is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus.
- WordNet could be used alongside the NLTK module to find the meanings of words, synonyms, antonyms, and more.
- Example:

  nltk.download('wordnet')
  syns = wordnet.synsets("objective")
  print(syns[0].name()) → aim.n.02
  print(syns[0].definition()) →
  the goal intended to be attained (and which is believed to be attainable)

Wordnet

- Finding synonyms and antonyms:

```
1  def find_synonyms(word):
2    synonyms = []
3    antonyms = []
4
5    for syn in wordnet.synsets(word):
6      for l in syn.lemmas():
7        synonyms.append(l.name())
8        if l.antonyms():
9          antonyms.append(l.antonyms()[0].name
               ())
10 return (synonyms, antonyms)
```

Wordnet

synonyms, antonyms = find_synonyms("objective")
print(set(synonyms)) → {'accusative', 'aim', 'object_lens', 'documentary', 'object', 'target', 'objective_lens', 'nonsubjective', 'objective', 'object_glass'}
print(set(antonyms)) → {'subjective'}

Regular Expression

- Regular Expression (RE) is a sequence of characters that forms a search pattern.
- RE can be used to check if a string contains the specified search pattern.
- Python has a built-in package called **re**, which can be used to work with Regular Expressions.

  ```
  import re
  ```

- A good resource to learn various RE could be found in: `https://www.w3schools.com/python/python_regex.asp`
- Example: Extracting XML file contents.

Regular Expression

- Here is an .xml file:

```
<note>
<to>Students</to>
<from>Lecturer</from>
<heading>Reminder</heading>
<body>Don't forget to fill attendance list
    !</body>
</note>
```

```
def clean_words(text):
    #remove html markup
    text = re.sub("(<.*?>)","",text)
    #remove whitespace
    text = text.strip()
    return text
```

Regular Expression

- Read input file:

```
7   def read_file(filename):
8     xml_file = open(filename, 'r')
9     out = []
10    while True:
11      # Read next line
12      line = xml_file.readline()
13      # If line is blank, then you struck the
            EOF
14      if not line :
15        break;
16      out.append(line.strip())
17    xml_file.close()
18  return out
```

Regular Expression

- Main function

```
19  def main():
20      path = 'd:/Opim/Programs/Python Notebook/'
21      filen = 'test.xml'
22      filename = path + filen
23      out = read_file(filename)
24      plain_out = []
25      for line in out:
26          text = clean_words(line)
27          if text:
28              plain_out.append(text.strip())
29      print(plain_out)
```

Regular Expression

- Run program:

```
30  if __name__ == "__main__":
31      main()
```

['Students', 'Lecturer', 'Reminder', "Don't forget to fill attendance list!"]

Regular Expression

- Converting Python *dict* into *json* with parameters.

```
# a Python object (dict):
x = {
"name": "John",
"age": 30,
"city": "New York"
}

# convert into JSON:
y = json.dumps(x, indent=4, separators=(". "
    , " = "))

# the result is a JSON string:
print(y)
```

Regular Expression

- Output:

```
{
    "name" = "John".
    "age" = 30.
    "city" = "New York"
}
```

Accessing Text from the Web

- To illustrate, we'll be using a book from Free eBooks, Project Gutenberg:
  ("http://www.gutenberg.org/files/2554/2554-0.txt")
- Library needed to fetch the url is:
  import urllib.request
- The content is read using:
  with urllib.request.urlopen ("http://www.gutenberg.org/files/2554/2554-0.txt") as response:
  html = response.read()

Accessing Text from the Web

- After successful reading, we can see some information regarding the book:
  type(html) $\rightarrow$ bytes
  len(html) $\rightarrow$ 1201735
  s = html[:75] $\rightarrow$ b'\xef \xbb \xbfThe Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsk'
- This is the raw content of the book (in bytes), including many details we are not interested in such as whitespace, line breaks and blank lines [5].
- Note that EF BB BF is a UTF-8-encoded BOM. It is not required for UTF-8, but serves only as a signature (usually on Windows).

Accessing Text from the Web

- Decoding byte into "utf-8"
  s = html[:75].decode("utf-8")
  s → '\ufeffThe Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsk'
- The Unicode character U+FEFF is the byte order mark (BOM), and is used to tell the difference between big- and little-endian UTF-16 encoding.
- To get a plain string of the result, it could be decode into utf-8-sig:
  ss = html[:75].decode("utf-8-sig")
  print(ss) → 'The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsk'

Accessing Text from the Web

- Tokenization
  - To break up the string into tokens:
    string = no_punctuation(ss, ',')
    string $\rightarrow$
    ['The', 'Project', 'Gutenberg', 'EBook', 'of', 'Crime', 'and', 'Punishment', 'by', 'Fyodor', 'Dostoevsk']
- Stopwords
- Before determining stopwords, remove unnecessary punctuations:

```
1  def combine_to_str(lst):
2  str = ""
3  for l in lst:
4    str += l + " "
5  return str
```

Accessing Text from the Web

ss = combine_to_str(string) sentence = stopwords_token(ss)

sentence →

[('The', 'DT'), ('Project', 'NNP'), ('Gutenberg', 'NNP'), ('EBook', 'NNP'), ('Crime', 'NNP'), ('Punishment', 'NNP'), ('Fyodor', 'NNP'), ('Dostoevsk', 'NNP')]

References I

[1] VanderPlas J., Python Data Science Handbook: Essential Tools for Working with Data, O'Reilly, USA, 2016.

[2] Hunt, J., A Beginners Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.

[3] Hunt, J., Advance Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.

[4] Bokka, R. K., Hora, S., Jain, T. Wambugu, M., Deep Learning for Natural Language Processing: Solve your natural language processing problems with smart deep neural networks, Packt Publishing, Kindle Edition, 2019.

[5] Bird, S., Klein, E., and Loper, E. Natural Language Processing with Python — Analyzing Text with the Natural Language Toolkit, O'Reilly Media, 2009.

References II