Handling Text Data

January 4, 2022

[1]:

```
# Handling Text data
```

[3]:

```
# Removing whitespaces
text = ["  Hello Myself  ","Cooking is my skill  ","You should consider cooking␣
 ↪as a   good. Habit"]

#Strip Whitespaces
strip_space = [string.strip() for string in text]
print(text)
print("Stripped text :",strip_space)
```

```
['  Hello Myself  ', 'Cooking is my skill  ', 'You should consider cooking as a
good. Habit']
Stripped text : ['Hello Myself', 'Cooking is my skill', 'You should consider
cooking as a   good. Habit']
```

[4]:

```
# Remove perionds
remove_periods = [string.replace(".","") for string in strip_space]
print("Removed Periods :",remove_periods) # It replaces the full stop with space
```

```
Removed Periods : ['Hello Myself', 'Cooking is my skill', 'You should consider
cooking as a   good Habit']
```

[7]:

```
# Using functions
def capitalizer(string: str) -> str:
    return string.upper()

[capitalizer(string) for string in remove_periods]
```

[7]:

```
['HELLO MYSELF',
 'COOKING IS MY SKILL',
 'YOU SHOULD CONSIDER COOKING AS A   GOOD HABIT']
```

[17]:

```
# Regular expressions library
import re

def replace_letters_with_X(string: str)-> str:
    return re.sub(r"[a-zA-Z]","X",string)
```

```
[replace_letters_with_X(string) for string in remove_periods]

"""In the
real world we will most likely define a custom cleaning function (e.g.,␣
 ↪capitalizer )
combining some cleaning tasks and apply that to the text data."""
```

[17]:

```
['XXXXX XXXXXX',
 'XXXXXXX XX XX XXXXX',
 'XXX XXXXXX XXXXXXXX XXXXXXX XX X   XXXX XXXXX']
```

[20]:

```
import sys
!{sys.executable} -m pip install bs4
```

```
/usr/lib/python3/dist-packages/secretstorage/dhcrypto.py:15:
CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes
instead
  from cryptography.utils import int_from_bytes
/usr/lib/python3/dist-packages/secretstorage/util.py:19:
CryptographyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes
instead
  from cryptography.utils import int_from_bytes
Collecting bs4
  Downloading bs4-0.0.1.tar.gz (1.1 kB)
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.10.0-py3-none-any.whl (97 kB)
     |                  | 97 kB 725 kB/s eta 0:00:011
Collecting soupsieve>1.2
  Downloading soupsieve-2.3.1-py3-none-any.whl (37 kB)
Building wheels for collected packages: bs4
  Building wheel for bs4 (setup.py) … done
  Created wheel for bs4: filename=bs4-0.0.1-py3-none-any.whl size=1272
sha256=81384c312a5a53e3ac484168e9cef91d1641abf8d5615a33715b302a5f5ab6b2
  Stored in directory: /home/kirankumar/.cache/pip/wheels/75/78/21/68b124549c9bd
c94f822c02fb9aa3578a669843f9767776bca
Successfully built bs4
Installing collected packages: soupsieve, beautifulsoup4, bs4
Successfully installed beautifulsoup4-4.10.0 bs4-0.0.1 soupsieve-2.3.1
```

[24]:

```
# Parsing and Cleaning HTML
from bs4 import BeautifulSoup

#Sample HTML
html = """
<div class='full_name'><span style='font-weight:bold'>
Masego</span> Azra</div>"
```

```
"""
#Parse html
soup = BeautifulSoup(html,"lxml")

#Find the div with class "full_name", show text
soup.find("div",{"class":"full_name"}).text
```

```
<html><body><div class="full_name"><span style="font-weight:bold">
Masego</span> Azra</div>"
</body></html>
```

[24]:
```
'\nMasego Azra'
```

[40]:
```
## Removing Punctions
import unicodedata
import sys

text = ['Hi!!!! I. Love. This. Song....',
'10000% Agree!!!! #LoveIT',
'Right?!?!']

#create a dictionary of punctuation characters
punctuation = dict.fromkeys(i for i in range(sys.maxunicode)
                            if unicodedata.category(chr(i)).startswith('P'))

[string.translate(punctuation) for string in text]
```

[40]:
```
['Hi I Love This Song', '10000 Agree LoveIT', 'Right']
```

[41]:
```
"""
translate is a Python method popular due to its blazing speed. In our solution,␣
 ↪first
we created a dictionary, punctuation , with all punctuation characters␣
 ↪according to
Unicode as its keys and None as its values. Next we translated all characters␣
 ↪in the
string that are in punctuation into None , effectively removing them. There are␣
 ↪more
readable ways to remove punctuation, but this somewhat hacky solution has the
advantage of being far faster than alternatives.
It is important to be conscious of the fact that punctuation contains␣
 ↪information (e.g.,
"Right?" versus "Right!"). Removing punctuation is often a necessary evil to␣
 ↪create
features; however, if the punctuation is important we should make sure to take␣
 ↪that
into account
```

```
"""
```

[41]:

```
'\ntranslate is a Python method popular due to its blazing speed. In our
solution, first\nwe created a dictionary, punctuation , with all punctuation
characters according to\nUnicode as its keys and None as its values. Next we
translated all characters in the\nstring that are in punctuation into None ,
effectively removing them. There are more\nreadable ways to remove punctuation,
but this somewhat hacky solution has the\nadvantage of being far faster than
alternatives.\nIt is important to be conscious of the fact that punctuation
contains information (e.g.,\n"Right?" versus "Right!"). Removing punctuation is
often a necessary evil to create\nfeatures; however, if the punctuation is
important we should make sure to take that\ninto account\n'
```

[44]:

```python
#Tokenizing Text
from nltk.tokenize import word_tokenize

string = "I will go to Japan for business trip"

word_tokenize(string)
```

[44]:

```
['I', 'will', 'go', 'to', 'Japan', 'for', 'business', 'trip']
```

[45]:

```python
# Tokenize senyences
from nltk.tokenize import sent_tokenize
# Create text
string = "The science of today is the technology of tomorrow. Tomorrow is today.
 ↪"
# Tokenize sentences
sent_tokenize(string)
```

[45]:

```
['The science of today is the technology of tomorrow.', 'Tomorrow is today.']
```

[51]:

```python
# Removing Stop Words
#First download stopwords
#import nltk
#nltk.download('stopwords')
from nltk.corpus import stopwords

tokenize_words = ['I', 'will', 'go', 'to', 'Japan', 'for', 'business', 'trip']

stop_words = stopwords.words('english')

[word for word in tokenize_words if word not in stop_words]
```

[51]:

```
['I', 'go', 'Japan', 'business', 'trip']
```

[52]:

```python
'''Note that NLTK's stopwords assumes the tokenized words are all lowercased.'''
```

[52]:

```
'Note that NLTK’s stopwords assumes the tokenized words are all lowercased.'
```

[53]:

```
'''While "stop words" can refer to any set of words we want to remove before␣
 ↪process-
ing, frequently the term refers to extremely common words that themselves␣
 ↪contain
little information value. NLTK has a list of common stop words that we can use␣
 ↪to
find and remove stop words in our tokenized words
'''
```

[53]:

```
'While "stop words" can refer to any set of words we want to remove before
process-\ning, frequently the term refers to extremely common words that
themselves contain\nlittle information value. NLTK has a list of common stop
words that we can use to\nfind and remove stop words in our tokenized words\n'
```

[55]:

```
# Stemming words
# To convert tokenized words to their root forms

from nltk.stem.porter import PorterStemmer

tokenized_words = ['i', 'am', 'humbled', 'by', 'this', 'traditional', 'meeting']

porter = PorterStemmer()

[porter.stem(word) for word in tokenized_words]
```

[55]:

```
['i', 'am', 'humbl', 'by', 'thi', 'tradit', 'meet']
```

[4]:

```
#Tagging Parts of Speech
#import nltk
#nltk.download('averaged_perceptron_tagger')
from nltk import pos_tag
from nltk import word_tokenize

text_data = "Diana likes playing cricket"

text_tagged = pos_tag(word_tokenize(text_data))

text_tagged
```

[4]:

```
[('Diana', 'NNP'), ('likes', 'VBZ'), ('playing', 'VBG'), ('cricket', 'NN')]
```

[5]:

```
"""
NLTK uses the Penn Treebank parts for speech tags

Tag  Part of speech
```

```
NNP  Proper noun, singular
NN   Noun, singular or mass
RB   Adverb
VBD  Verb, past tense
VBG  Verb, gerund or present participle
JJ   Adjective
PRP  Personal pronoun
"""
```

[5]:

```
'\nTag  Part of speech\nNNP  Proper noun, singular\nNN   Noun, singular or
mass\nRB   Adverb\nVBD  Verb, past tense\nVBG  Verb, gerund or present
participle\nJJ   Adjective\nPRP  Personal pronoun\n'
```

[6]:

```
# Filter words
[word for word, tag in text_tagged if tag in ['NN','NNS','NNP','NNPS'] ]
```

[6]:

```
['Diana', 'cricket']
```

[9]:

```
# Labeling the parts of speech in tweets
from sklearn.preprocessing import MultiLabelBinarizer

tweets = ["I am eating a burrito for breakfast",
"Political science is an amazing field",
"San Francisco is an awesome city"]

tagged_tweets = []

for tweet in tweets:
    tweet_tag = pos_tag(word_tokenize(tweet))
    tagged_tweets.append([tag for word, tag in tweet_tag])

#Using one-hot encoding
one_hot_multi = MultiLabelBinarizer()
one_hot_multi.fit_transform(tagged_tweets)
```

[9]:

```
array([[1, 1, 0, 1, 0, 1, 1, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 1, 1, 0, 0, 0, 1]])
```

[10]:

```
# Show feature names
one_hot_multi.classes_
```

[10]:

```
array(['DT', 'IN', 'JJ', 'NN', 'NNP', 'PRP', 'VBG', 'VBP', 'VBZ'],
      dtype=object)
```

[3]:

```
# Train your own tagger using Brown Corpus
#import nltk
```

```python
#nltk.download('brown')
from nltk.corpus import brown
from nltk.tag import UnigramTagger
from nltk.tag import BigramTagger
from nltk.tag import TrigramTagger

# Get some text from the Brown Corpus, broken into sentences
sentences = brown.tagged_sents(categories="news")

# Split into 4000 sentences for training and 623 for testing
train = sentences[:4000]
test = sentences[4000:]
# Create backoff tagger
unigram = UnigramTagger(train)
bigram = BigramTagger(train, backoff=unigram)
trigram = TrigramTagger(train, backoff=bigram)
# Show accuracy
trigram.evaluate(test)
```

[3]:

```
0.8174734002697437
```

[6]:

```python
# Encoding Text as a Bag of words
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

text = np.array(['I love Brazil. Brazil!',
'Sweden is best',
'Germany beats both'])

count = CountVectorizer()
bag_of_words = count.fit_transform(text)

bag_of_words
```

[6]:

```
<3x8 sparse matrix of type '<class 'numpy.int64'>'
        with 8 stored elements in Compressed Sparse Row format>
```

[7]:

```python
bag_of_words.toarray()
```

[7]:

```
array([[0, 0, 0, 2, 0, 0, 1, 0],
       [0, 1, 0, 0, 0, 1, 0, 1],
       [1, 0, 1, 0, 1, 0, 0, 0]])
```

[8]:

```python
# Show feature names
count.get_feature_names()
```

[8]:

```
['beats', 'best', 'both', 'brazil', 'germany', 'is', 'love', 'sweden']
```

[ ]:

```
'''
CountVectorizer comes with a number of useful parameters to make creating bag-
of-words feature matrices easy. First, while by default every feature is a␣
 ↪word, that
does not have to be the case. Instead we can set every feature to be the␣
 ↪combination of
two words (called a 2-gram) or even three words (3-gram). ngram_range sets the
minimum and maximum size of our n-grams. For example, (2,3) will return all 2-
grams and 3-grams. Second, we can easily remove low-information filler words␣
 ↪using
stop_words either with a built-in list or a custom list. Finally, we can␣
 ↪restrict the
words or phrases we want to consider to a certain list of words using␣
 ↪vocabulary . For
example, we could create a bag-of-words feature matrix for only occurrences of␣
 ↪coun-
try names:
'''
```

[12]:

```
# Create feature matrix with arguments
count_2gram = CountVectorizer(ngram_range=(1,2),
stop_words="english",
vocabulary=['brazil'])

bag = count_2gram.fit_transform(text)
# View feature matrix
print(bag.toarray())

# View the 1-grams and 2-grams
print(count_2gram.vocabulary_)
```

```
[[2]
 [0]
 [0]]
{'brazil': 0}
```