Introduction to Natural Language Processing by DataCamp

Date .... Page ....

09/2020

Regular expressions:

- Strings with a special syntax
- Allows us to match patterns in other strings.

Applications:

(i) Find all web links in a document.
(ii) Parse email addresses, remove/replace unwanted characters.

```
import re
re.match('abc', 'abcdef') ⟶ span=(0,3), match='abc'
word_regex = '\w+'
re.match(word_regex, 'hi there!')
↳ span=(0,2), match='hi'
```

Pattern	Matches	Example
\w+	word	'Magic'
\d	digit	9
\s	space	' '
.*	wildcard	'username74'
+ or *	greedy match	'aaaa'
\S	not space	'no_spaces'
[a-z]	lowercase ~~group~~	'abcdefg'

r"\[.*]" Search anything in square brackets Hi [I] am [Antaripa]

r"[\S]+:" character :

Python's re module

re module

split: split a string on regex

findall: find all patterns in a string

search: search for a pattern.

match: match an entire string or substring based on a pattern.

Search for first occurence of '~~Antaripa~~ Saha' in

string1 = '~~An~~ let us introduce you to Antaripa Saha. She is a B.Tech student. She belongs to Saha family. Her brother name is Trishank Saha'.

match = re.search('~~cocono~~ Saha', string1)

print(match.start(), match.end()) → Will print the index
24 31

~~pat~~

Regex groups

- OR is represented using |
- We can define a group using ()
- We can define explicit character ranges using []

```
import re
match_digits_and_words = ('(\d+|\w+)')
re.findall(match_digits_and_words, 'He has 11 cats')
↳ ['He', 'has', '11', 'cats'}]
```

Regex ranges and groups

Pattern	Matches	example
[A-Za-z]+		'ABCDghza'
[0-9]		8
[A-Za-z\-\.]+	A-Z, a-z, -, .	'My-Website.com'
(a-z)	a, -, z	'a-z'
(\s+|,)	spaces or a comma	', '

use escape characters to explicitly add a special characters

```
r'(\w+|#\d|\?|!)'
import nltk.tokenize import regexp_tokenize
```

Pattern to find hashtags

```
hastags_pattern = r'#\w+'
print(regexp_tokenize(tweets[0], ~~patternt)~~ hastags_pattern)
```

Pattern that matches both mentions and hashtags

```
pattern2 = r'([@#]\w+)'
```

Bag of words CODE

```
from nltk.tokenize import word_tokenize
from collections import Counter
Counter(word_tokenize("""The cat is in the box. The cat likes the box.
        The box is over the cat.""")).most_com
↳ {'.': 3, 'The': 3, 'the': 3 ... 'the': 2}}
```

Lemmatize CODE

from nltk.stem import WordNetLemmatizer

alpha_only = [t for t in ~~lower_tokens~~ tweet tokens if t.isalpha()]

no_stops = [t for t in alpha_only if t not in english_stops

wordnet_lemmatizer = WordNetLemmatizer()

lemmatized = [wordnet_lemmatizer.lemmatize(t) for t in no_stops]

#Create bag of words

bow = Counter(lemmatized)

print(bow.most_common(10)) → 10 most common tokens.

Introduction to gensim

→ Popular open-source NLP library

Use for building document or word vectors

Performing topic identification and document comparison.

Creating a gensim dictionary:

CODE

from gensim.corpora.dictionary import Dictionary

from nltk.tokenize import word_tokenize

my_documents = ['. . . .', '. . . .', . . .]

tokenized_docs = [word_tokenize(doc.lower()) for doc in my_docum

dictionary = Dictionary(tokenized_docs)

dictionary.token2id #Taking a look at tokens and their ids

↳ {'!': 11, ',': 7, 'a': 2, 'about': 14 . . . . }

Creating a gensim corpus:

corpus = [dictionary.doc2bow(doc) for doc in tokenized_docs]

corpus

↳ [[(0, 1), (1, 1), (2, 1), (3, 1), ...], [(0, 1), (1, 1), (9, 1), (10, 1), ...], . . .]

1st doc — 2nd doc — word_id — count

- gensim models can be easily saved, updated, and reused
- Our dictionary can also be updated
- This more advanced and feature rich bag-of-words can be used in future exercises.

CODE

```
#Select id for word 'computer'
computer_id = dictionary.token2id.get("computer")
# Use that id to get the word
print(dictionary.get(computer_id))
#Save the 2nd document
doc = corpus[1]
# Sort the doc for frequency
bow_doc = sorted(doc, key = lambda w: w[1], reverse=True)
# Print the top 5 words of the document along with frequency.
for word_id, word_count in bow_doc[:5]:
        print(dictionary.get(word_id), word_count)
#Create the defaultdict
total_word_count = defaultdict(int)
for word_id, word_count in itertools.chain.from_iterable(corpus):
        total_word_count[word_id] += word_count.
```

sorting the frequency 1st element in the list.

↳ Dictionary of total word counts for all documents

```
#Sorted list of the total word dictionary
sorted_word_count = sorted(total_word_count.items(),
key=lambda w: w[1], reverse=True)

for word_id, word_count in sorted_word_count[:5]:
    print(dictionary.get(word_id), word_count)
```

What is tf-idf?

- Term frequency - inverse document frequency.
- Allows us to determine the most imp. words in each doc.
- Each corpus may have shared words beyond just stopwords.
- These words should be down-weighted in imp.
- Eg from astronomy: "Sky"
- Ensures most common words don't show up as key words.
- Keeps document specific frequent words weighted high.

Tf-idf formula

$$w_{i,j} = tf_{i,j} * \log\left(\frac{N}{df_i}\right)$$

$w_{i,j}$ → tf-idf weight of token i in document j

$tf_{i,j}$ = no. of occurences of token i in doc j.

df_i = no. of documents that contain token i

N = total no. of doc.

Tf-idf with gensim

```
from gensim.models.~~tfif~~ tfidfmodel import TfidfModel
tfidf = TfidfModel(corpus)
tfidf[corpus[1]]
  ↳ [(0, 0.174629), (1, 0.174629), (9, 0.298531), (10, 0.771693), ...]
```

What is Named Entity Recognition?

- NLP task to identify important named entities in the text
  - (i) People, places, organizations
  - (ii) Dates, states, works of art, etc.
- Can be used alongside topic identification
- Who? What? When? Where?

Stanford CoreNLP library supports:

- Integrated with Python via nltk
- Java based
- Support for NER as well as ~~cofference~~ core-ference and dependency trees.

CODE

```
import nltk
sentence = '''In New York, I like to ride . . . . .'''
tokenized_sent = nltk.word_tokenize(sentence)
tagged_sent = nltk.pos_tag(tokenized_sent)
tagged_sent[:3]  ⟶  [('In', 'IN'), ('New', 'NNP'), ('York', 'NNP')]
```

To print in tree form
```
print(nltk.ne_chunk(tagged_sent))
```
named entity

NER in nltk

CODE

```
sentences = nltk.sent_tokenize(article)
token_sentences = [nltk.word_tokenize(t) for t in sentences]
# Tag each sentence into part of speech
pos_sentences = [nltk.pos_tag(sent) for sent in token_sentences]
# Create name entity chunks
chunked_sentences = nltk.ne_chunk_sents(pos_sentences, binary=True)
# Test for stems of the tree with 'NE' tags
for sent in chunked_sentences:
    for chunk in sent:
        if hasattr(chunk, 'label') and chunk.label() == 'NE':
            print(chunk)
```

hasattr() to determine if each chunk has a 'label' and then simply use the chunk's .label() method as the dictionary key.

CODE

Date ....
Page ....

Making a ~~chart~~ pie chart for each NER categories:

```
import matplotlib.pyplot as plt
ner_categories = defaultdict(int)
for sent in chunked_sentences:
    for chunk in sent:
        if hasattr(chunk, 'label'):
            ner_categories[chunk.label()] +=1
# Create a list from the dictionary keys for the charts labels
labels = list(ner_categories.keys())
# Create a list of the values
values = [ner_categories.get(v) for v in labels]

# Create the pie chart
plt.pie(values, labels=labels, autopct='%1.1f%%', startangle=140)
plt.show()
```

What is Spacy?

- NLP library similar to gensim, with diff. implementations.
- Focus on creating NLP pipelines to generate models and corpora
- Open-source, with extra libraries and tools

→ Displacy

↓

Entity recognition visualizer

Spacy NER

```
import spacy
nlp = spacy.load('en')
nlp.entity
↳ <spacy.pipeline.EntityRecognizer at 0x7...  >

doc = nlp("""Berlin is the capital of Germany;
           and the residence of Chancellor Angela Merkel.""")
doc.ents
↳ (Berlin, Germany, Angela Merkel)
print(doc.ents[0], doc.ents[0].label_)
↳ Berlin GPE
```

5/09/2020

Why use SpaCy for NER?

→ • Easy pipeline creating
• Diff. entity types compared to nltk
• Informal language corpora
↳ Easily find entities in tweets and chat messages.

Comparing ~~Spa~~ spacy NER with nltk

CODE

```
import spacy          # Instantiate the English model
nlp=spacy.load(name='en', tagger=False, parser=False, matcher=False)
doc = nlp(article)  # Create a new document
#Print all the found entities and their labels
for ent in doc.ents:
    print(ent.label_, ent.text)
```

Multilingual NER with polyglot

Spanish NER with polyglot

CODE

```
from polyglot.text import Text
text = """El presidente de la ......"""
ptext = Text(text)
ptext.entities

# for tag
ptext.entities.tag
```

Supervised learning steps

- Collect and preprocess our data
- Determine a label (example: Movie genre)
- Split data into training and test sets
- Extract features from the text to help predict the label.
    - Bag-of-words vector built into scikit-learn.

Building word count vectors with scikit-learn

Count vectorizer of Python

CODE

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
df = ... # load data into dataframe
y = df['Sci-Fi']
X_train, X_test, y_train, y_test = train_test_split(df['plot'], y,
                                    test_size = 0.33, random_state=53)
count_vectorizer = CountVectorizer(stop_words='english')
count_train = count_vectorizer.fit_transform(X_train.values)
count_test = count_vectorizer.transform(X_test.values)
```

~~Count Vectorizer for~~

```
# Print the first ten features
print(count_vectorizer.get_feature_names()[:10])
```

Tfid Vectorizer for text classification

~~from sklearn.model_selection imp~~

CODE

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_df=0.7)
tfidf_train = tfidf_vectorizer.fit_transform(X_train.values)
tfidf_test = tfidf_vectorization.transform(X_test.values)
print(tfidf_vectorizer.get_feature_names()[:10]
print(tfidf_train.A[:5]) # First 5 vectors of
```

↓ array attribute of tfidf_train

Inspecting the vectors

CODE

```
# Create the CountVectorizer DataFrame
count_df = pd.DataFrame(count_train.A, columns =
                        count_vectorizer.get_feature_names())
# Create the TfidVectorizer DataFrame
tfidf_df = pd.Dataframe(tfidf_train.A, columns = tfidf_vectorizer.
                        get_features_names())

print(count_df.head())
print(tfidf_df.head())
# Calculate difference in columns
difference = set(count_df.columns) - set(tfidf_df.columns)
# Check whether the dataframes are equal
print(count_df.equals(tfidf_df))
```

Naive Bayes with scikit-learn

CODE

```
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
nb_classifier = MultinomialNB()
nb_classifier.fit(count_train, y_train)
pred = nb_classifier.predict(count_test)
metrics.accuracy_score(y_test, pred)
# CONFUSION matrix
metrics.confusion_matrix(y_test, pred, labels = [0, 1])
```

→ works well with count vectorizers as it expects integer inputs and also useful for multilabel classification

[0, 1] → Change them acc. to labels like 'FAKE' and 'REAL' for fake news classifier

If we use alpha in the classifier for improving op accuracy

```
~~mb~~ ~~nb~~ classifier = MultinomialNB(alpha)
```

Inspecting the model

CODE

```
# Get the class labels
class_labels = nb_classifier.classes_
# feature names
feature_names = tfidf_vectorizer.get_feature_names()
# zip the features names together with coefficient array and
# sort by weights
feat_with_weights = sorted(zip(nb_classifier.coef_[0], feature_names))
```