NLP with Python: Topic Modeling

Topic modeling in Python using scikit-learn

6 minute read

This article is **Part 4** in a **5-Part** Natural Language Processing with Python (https://sanjayasubedi.com.np/series/nlp/).

Introduction

Topic modeling is an interesting problem in NLP applications where we want to get an idea of what topics we have in our dataset. A topic is nothing more than a collection of words that describe the overall theme. For example, in case of news articles, we might think of topics as politics, sports etc. but topic modeling won't directly give you names of the topics but rather a set of most probable words that might describe a topic. It is up to us to determine what topic the set of words might refer to.

To make this example concrete, let's consider a sample output of a topic modeling algorithm that is trained to find out two topics in our dataset.

```
0: game match player win
1: government minister election
```

The output shows us two topics 0 and 1 and the corresponding words that describe the topic. From the output we can infer that the first topic refers to `sports` whereas the second one is related to `politics`. It is up to us to give the topics a concrete name.

Just like clustering algorithms, there are some algorithms that need you to specify the number of topics you want to extract from the dataset and some that automatically determine the number of topics. In this post, I'm going to use Non Negative Matrix Factorization (NMF) method for modeling. You can also simply swap the NMF with Latent Dirichlet Allocation (LDA). You can check sklearn's documentation for more details about NMF and LDA. They are available in `sklearn.decomposition` module.

Data loading

Like in the previous posts, I'm going to use BBC dataset. First download the dataset from http://mlg.ucd.ie/files/datasets/bbc-fulltext.zip (http://mlg.ucd.ie/files/datasets/bbc-fulltext.zip) and extract. The dataset consists of 2225 documents and 5 categories: business, entertainment, politics, sport, and tech.

I've used `load_files` to load the data from the directory, created a pandas Dataframe and then removed all words that are not nouns from every article. I removed "non-nouns" because topics are almost always nouns and I do not think using other words will be of any use. But feel free to experiment and find what works best for your dataset.

```python
from sklearn.datasets import load_files
import pandas as pd
import spacy
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])

random_state = 0

DATA_DIR = "./bbc/"
data = load_files(DATA_DIR, encoding="utf-8", decode_error="replace", random_state=random_state)
df = pd.DataFrame(list(zip(data['data'], data['target'])), columns=['text', 'label'])

def only_nouns(texts):
    output = []
    for doc in nlp.pipe(texts):
        noun_text = " ".join(token.lemma_ for token in doc if token.pos_ == 'NOUN')
        output.append(noun_text)
    return output


df['text'] = only_nouns(df['text'])

df.head()
```

	text	label
0	boss award executive business magazine title p...	0
1	copy bumper sale sci fi shooter game copy sale...	4
2	msp climate change control decade committee ms...	2
3	success view week track bronze injury season i...	3
4	tory rethink association candidate election ag...	2

Model training

Now that we have our data ready. Let's train a `TfidfVectorizer` for extracting features and a `NMF` model for topic modeling.

```
# number of topics to extract
n_topics = 5

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
vec = TfidfVectorizer(max_features=5000, stop_words="english", max_df=0.95, min_df=2)
features = vec.fit_transform(df.text)

from sklearn.decomposition import NMF
cls = NMF(n_components=n_topics, random_state=random_state)
cls.fit(features)
```

Our model is now trained and is ready to be used.

Results

To see what topics the model learned, we need to access `components_` attribute. It is a 2D matrix of shape `[n_topics, n_features]` . In this case, the `components_` matrix has a shape of [5, 5000] because we have 5 topics and 5000 words in tfidf's vocabulary as indicated in `max_features` property of TfidfVectorizer. Note that if TfidfVectorizer is not able to "find" `max_features` number of distinct words in your dataset, then this number will be less.

But what does it mean? Let's analyze the first row of the `components_` matrix. If you print out first row of the matrix, then you'll get a vector of 5000 entries. Each entry has a number associated with it that indicates how much does this word influcene the first topic. The higher the number the more it describes the particular topic. Let's print all the topics and their most "describing words".

```
# list of unique words found by the vectorizer
feature_names = vec.get_feature_names()

# number of most influencing words to display per topic
n_top_words = 15

for i, topic_vec in enumerate(cls.components_):
    print(i, end=' ')
    # topic_vec.argsort() produces a new array
    # in which word_index with the least score is the
    # first array element and word_index with highest
    # score is the last array element. Then using a
    # fancy indexing [-1: -n_top_words-1:-1], we are
    # slicing the array from its end in such a way that
    # top `n_top_words` word_index with highest scores
    # are returned in desceding order
    for fid in topic_vec.argsort()[-1:-n_top_words-1:-1]:
        print(feature_names[fid], end=' ')
    print()
```

```
0 user technology people computer website internet software firm pc service phone network company way
1 game match player win team injury coach season victory time champion chance club goal
2 film award star actor actress comedy nomination ceremony director movie prize year drama role
3 government minister election party leader issue people plan spokesman campaign secretary policy law
member
4 market growth analyst price share economy company sale year month firm rate rise demand
```

The topics produced by the model look pretty good. If you remember, we had 5 categories in our dataset (tech, business, entertainment, sports, politics) and the topics also correspond to these categories quite nicely. Let's change the number of topics `n_topics` to 10 and re-run everything again. Now you'll notice that the `cls.components_` matrix has the shape of [10, 5000] because there are 10 topics now. If we look at the topics

```
0 technology video device phone people mobile music generation broadband content tv network speed
gaming
1 half game win coach ball victory minute goal chance try line match penalty kick
2 film actor award actress star comedy director drama movie prize ceremony nomination role category
3 government election minister party leader issue plan people spokesman campaign secretary policy
member voter
4 growth economy rate rise figure economist price export market month quarter analyst demand spending
5 company share firm executive deal business market shareholder sale profit analyst investor stock
investment
6 champion world title final seed event set match victory round race win year time
7 album band music song chart singer artist act rock single record concert hit star
8 user software program computer website security virus mail web information internet firm pc site
9 club season player manager team boss football game striker summer thing contract injury defender
```

Now the topics are more fine grained. For example, the topic previously related to technology now seems to be divided into “tv, gaming and internet: topic 0”, “computer programs, security: topic 8”.

What if you want to identify the topic of a new text? Simply call the `transform` function of the model and it will give you a score of each topic. Choose the topic with the highest score to determine it’s topic. As an example:

```
new_articles = [
    "Playstation network was down so many people were angry",
    "Germany scored 7 goals against Brazil in worldcup semi-finals"
]
# first transform the text into features using vec
# then pass it to transform of cls
# the result will be a matrix of shape [2, 10]
# then we sort the topic id based on the score using argsort
# and take the last one (with the highest score) for each row using `[:,-1]` indexing
cls.transform(vec.transform(new_articles)).argsort(axis=1)[:,-1]
```

```
[0, 6]
```

According to the model, the first article belongs to 0th topic and the second one belongs to 6th topic which seems to be the case.

Conclusion

This post showed you how to train your own topic modeling model and use it to identify the topics in your dataset. There are other algorithms for topic modeling as well be only NMF was covered here. Feel free to switch to `LatentDirichletAllocation` to see how it does. `gensim` is another excellent library for topic modeling and has many other algorithms implemented and you might want to try that as well.

This article is **Part 4** in a **5-Part** Natural Language Processing with Python (https://sanjayasubedi.com.np/series/nlp/).

- Part 1 - Natural Language Processing with Python: Introduction
- Part 2 - NLP with Python: Text Feature Extraction
- Part 3 - NLP with Python: Text Clustering
- Part 4 - **> NLP with Python: Topic Modeling**
- Part 5 - NLP with Python: Nearest Neighbors Search

Categories: NLP

Updated: May 19, 2019

LEAVE A COMMENT