lataent-dirichlet-allocation-lda

June 11, 2025

1 Topic Modeling and Latent Dirichlet Allocation (LDA) using Sklearn

[1]:
```python
# Importing necessary libraries
import pandas as pd
import numpy as np

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, SnowballStemmer, PorterStemmer,␣
↪RegexpStemmer, LancasterStemmer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.decomposition import LatentDirichletAllocation as LDA

import string
from string import punctuation

import os
```

Data and Steps for Working with Text

We will apply LDA on the corpus that we have seen in the previous articles:

Document 1: I want to watch a movie this weekend.

Document 2: I went shopping yesterday. New Zealand won the World Test Championship by beating India by eight wickets at Southampton.

Document 3: I don't watch cricket. Netflix and Amazon Prime have very good movies to watch.

Document 4: Movies are a nice way to chill however, this time I would like to paint and read some good books. It's been so long!

Document 5: This blueberry milkshake is so good! Try reading Dr. Joe Dispenza's books. His work is such a game-changer! His books helped to learn so much about how our thoughts impact our biology and how we can all rewire our brains.

1. Data loading

[2]:
```
dir_path = r'C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice'
dir_path

filename =  [file for file in os.listdir(dir_path) if file.endswith('.txt')]
filename

combined_texts = []

for file in filename:
    file_path = os.path.join(dir_path,file)
    print(file_path)

    with open(file_path,'r') as file:
        content = file.read()
        combined_texts.append(content)

corpus = combined_texts
corpus
```

```
C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice\Document 1.txt
C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice\Document 2.txt
C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice\Document 3.txt
C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice\Document 4.txt
C:\Users\KARTIKEY SINGH\OneDrive\Documents\DS Practice\Document 5.txt
```

[2]:
```
['I want to watch a movie this weekend.',
 'I went shopping yesterday. New Zealand won the World Test Championship by
beating India by eight wickets at Southampton',
 'I don't watch cricket. Netflix and Amazon Prime have very good movies to
watch.',
 'Movies are a nice way to chill however, this time I would like to paint and
read some good books. It's been so long!',
 'This blueberry milkshake is so good! Try reading Dr. Joe Dispenza's books. His
work is such a game-changer! His books helped to learn so much about how our
thoughts impact our biology and how we can all rewire our brains.']
```

2. Common Text Pre-processing steps for both the packages:

[3]:
```
#Apply preprossing on the Corpus

#stop loss words
stop = set(stopwords.words('english'))

# punctuations
exclude = set(punctuation)

# lemmatization
leema = WordNetLemmatizer()
```

```python
# One function for all the steps:
def clean(doc):

    # convert text into lower case + split into words
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])

    # remove any stop words present
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)

    # remove punctuations + normalize the text
    normalize = " ".join(leema.lemmatize(word) for word in punc_free.split())

    return normalize

# clean data stored in a new list
clean_corpus = [clean(doc).split() for doc in corpus]
clean_corpus
```

[3]:
```
[['want', 'watch', 'movie', 'weekend'],
 ['went',
  'shopping',
  'yesterday',
  'new',
  'zealand',
  'world',
  'test',
  'championship',
  'beating',
  'india',
  'eight',
  'wicket',
  'southampton'],
 ['don’t',
  'watch',
  'cricket',
  'netflix',
  'amazon',
  'prime',
  'good',
  'movie',
  'watch'],
 ['movie',
  'nice',
  'way',
  'chill',
```

```
  'however',
  'time',
  'would',
  'like',
  'paint',
  'read',
  'good',
  'book',
  'it’s',
  'long'],
 ['blueberry',
  'milkshake',
  'good',
  'try',
  'reading',
  'dr',
  'joe',
  'dispenza’s',
  'book',
  'work',
  'gamechanger',
  'book',
  'helped',
  'learn',
  'much',
  'thought',
  'impact',
  'biology',
  'rewire',
  'brain']]
```

3. Implementation of LDA using Sklearn

1. In sklearn, after cleaning the text data, we transform the cleaned text to the numerical representation using the vectorizer. I have used both the TF-IDF and the count vectorizer here.

[4]:
```python
# Converting text into numerical representation
# Array from TF-IDF Vectorizer
tf_idf_vectorize = TfidfVectorizer( tokenizer= lambda doc: doc, lowercase=False)

# this is our converted text to numerical representation from the Tf-IDF vectorizer and Count vectorizer
tf_idf_vectorize
```

[4]:
```
TfidfVectorizer(lowercase=False,
                tokenizer=<function <lambda> at 0x0000027DE373AF20>)
```

```python
[5]: # Converting text into numerical representation
     # Array from Count Vectorizer
     cv_vectorizer = CountVectorizer(tokenizer= lambda doc: doc, lowercase =False)

     # this is our converted text to numerical representation from the Tf-IDF␣
      ↪vectorizer and Count vectorizer
     cv_vectorizer
```

```
[5]: CountVectorizer(lowercase=False,
                     tokenizer=<function <lambda> at 0x0000027DE373AB60>)
```

```python
[6]: #Array from TF-IDF Vectorizer
     tf_idf_arr = tf_idf_vectorize.fit_transform(clean_corpus)

     #Array from Count Vectorizer
     cv_arr = cv_vectorizer.fit_transform(clean_corpus)
```

```
c:\ProgramData\anaconda3\Lib\site-
packages\sklearn\feature_extraction\text.py:521: UserWarning: The parameter
'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

2. Next, we create the vocabulary:

```python
[7]: # Creating vocabulary array which will represent all the corpus
     # get the vocb list

     vocab_cv_arr = cv_vectorizer.get_feature_names_out()
     print(vocab_cv_arr)
```

```
['amazon' 'beating' 'biology' 'blueberry' 'book' 'brain' 'championship'
 'chill' 'cricket' 'dispenza’s' 'don’t' 'dr' 'eight' 'gamechanger' 'good'
 'helped' 'however' 'impact' 'india' 'it’s' 'joe' 'learn' 'like' 'long'
 'milkshake' 'movie' 'much' 'netflix' 'new' 'nice' 'paint' 'prime' 'read'
 'reading' 'rewire' 'shopping' 'southampton' 'test' 'thought' 'time' 'try'
 'want' 'watch' 'way' 'weekend' 'went' 'wicket' 'work' 'world' 'would'
 'yesterday' 'zealand']
```

```python
[8]: vocab_tf_idf = tf_idf_vectorize.get_feature_names_out()
     print(vocab_tf_idf)
```

```
['amazon' 'beating' 'biology' 'blueberry' 'book' 'brain' 'championship'
 'chill' 'cricket' 'dispenza’s' 'don’t' 'dr' 'eight' 'gamechanger' 'good'
 'helped' 'however' 'impact' 'india' 'it’s' 'joe' 'learn' 'like' 'long'
 'milkshake' 'movie' 'much' 'netflix' 'new' 'nice' 'paint' 'prime' 'read'
 'reading' 'rewire' 'shopping' 'southampton' 'test' 'thought' 'time' 'try'
 'want' 'watch' 'way' 'weekend' 'went' 'wicket' 'work' 'world' 'would'
 'yesterday' 'zealand']
```

3. Once we have the vocabulary, we build the LDA model by creating the LDA class:
    - Inside this class of LDA, we define the components such as how many topics want to retrieve (n_components) and specify the number of iterations that the model must run (max_iter)
    - Post this, using the saved LDA model, we perform fit_transform on the model on the vectorizer. This returns the topics (called X_topics) and using lda_model.components_ we obtain the topics.

4. Implementation of LDA

To implement LDA, pass the corpus: document-term matrix to the model. We had above obtained the unique words of vocabulary using both TF-IDF and Count Vectorizer. We can continue with either as have the same unique words in both the obtained vocabularies.

[9]:
```
# Implementation of LDA:
lda_model = LDA(n_components= 6, max_iter= 20, random_state= 20)



# Create object for the LDA class
# Inside this class LDA: define the components:
X_topics = lda_model.fit_transform(tf_idf_arr)

# fit transform on model on our count_vectorizer : running this will return our␣
↪topics
topic_words = lda_model.components_
```

4. Next, we view the obtained topics using the following steps:
    1. N_top_words: first define the number of words that want to print on every topic.
    2. Then, iterate through the documents i.e iterating over topic_words, which we obtained in the last step.
    3. Each of the topic_word will be represented with a probability, which will indicate the importance of that word in the topic.
    4. Now, to view the most important words we can either create a sorted array or a sorted topic distribution.
    5. Next, to view the actual words present in the array, we can use the indexes present in the vocabulary which was created above.

5. Retrive the Topics

[10]:
```
# Define the number of words that we want to print in every topic : n_top_words
n_top_words = 6

for i, topic_dict in enumerate(topic_words):

    # np.argsort to sorting an array or a list or the matrix acc to their values
    sorted_topic_dict = np.argsort(topic_dict)
```

```
    # Next, to view the actual words present in those indexes we can make the␣
↪use of the vocab created earlier
    topic_words = np.array(vocab_cv_arr)[sorted_topic_dict]

    # so using the sorted_topic_indexes we are extracting the words from the␣
↪vocabulary
    # obtaining topics + words
    # this topic_words variable contains the Topics as well as the respective␣
↪words present in those Topics
    topic_words = topic_words[:-n_top_words:-1]
    print("Topic", str(i+1), topic_words)
```

```
Topic 1 ['movie' 'good' 'watch' 'book' 'weekend']
Topic 2 ['zealand' 'test' 'beating' 'world' 'championship']
Topic 3 ['weekend' 'want' 'watch' 'movie' 'book']
Topic 4 ['watch' 'amazon' 'cricket' 'don't' 'netflix']
Topic 5 ['movie' 'good' 'watch' 'book' 'weekend']
Topic 6 ['however' 'chill' 'would' 'it's' 'like']
```

5. Last but not least, the assignment of the topics to the documents:

5. Annotating the topics the documents

[11]:
```
# To view what topics are assigned to the documents:
doc_topic = lda_model.transform(cv_arr)

# iterating over ever value till the end value
for n in range(doc_topic.shape[0]):

    # argmax() gives maximum index value
    topic_doc = doc_topic[n].argmax()

    # document is n+1
    print("Documnt", n+1, "-- Topic:", topic_doc)
```

```
Documnt 1 -- Topic: 2
Documnt 2 -- Topic: 1
Documnt 3 -- Topic: 3
Documnt 4 -- Topic: 5
Documnt 5 -- Topic: 2
```

Parameters for LDA model in sklearn

The arguments used in the sklearn package are:

1. The corpus or the document-term matrix to be passed to the model (in our example is called doc_term_matrix)
2. Number of Topics: n_components is the number of topics to find from the corpus.

3. The number of maximum iterations: max_iter: It is the number of maximum iterations allowed for the LDA algorithm to converge.