Latent_dirichlet_allocation_news_classification

August 22, 2021

0.1 Step 0: Latent Dirichlet Allocation

LDA is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.

- Each document is modeled as a multinomial distribution of topics and each topic is modeled as a multinomial distribution of words.
- LDA assumes that the every chunk of text we feed into it will contain words that are somehow related. Therefore choosing the right corpus of data is crucial.
- It also assumes documents are produced from a mixture of topics. Those topics then generate words based on their probability distribution.

0.2 Step 1: Load the dataset

The dataset we'll use is a list of over one million news headlines published over a period of 15 years. We'll start by loading it from the `abcnews-date-text.csv` file.

[1]:
```
import numpy as np
import pylab as plt
import pandas as pd
import seaborn as sns

sns.set(style='white')

data_dir = './datasets/'
```

[2]:
```
data = pd.read_csv(data_dir+'abcnews-date-text.csv', error_bad_lines=False);
print ( f"data.shape:{data.shape}")
```

```
data.shape:(999999, 2)
```

[3]:
```
# cut the data only to take first 100000 lines

# we could also take samples randomly
# That might be necessary sometimes (in ordered data)

df = data[:100000][['headline_text']];
df['index'] = df.index
```

```
df.head()
```

[3]:

```
                                       headline_text  index
0  aba decides against community broadcasting lic…      0
1     act fire witnesses must be aware of defamation      1
2     a g calls for infrastructure protection summit      2
3           air nz staff in aust strike for pay rise      3
4      air nz strike to affect australian travellers      4
```

Let's glance at the dataset:

[4]:

```
print ( f"df.shape:{df.shape}")
```

```
df.shape:(100000, 2)
```

0.3 Step 2: Data Preprocessing

We will perform the following steps:

- **Tokenization**: Split the text into sentences and the sentences into words. Lowercase the words and remove punctuation.
- Words that have fewer than 3 characters are removed.
- All **stopwords** are removed.
- Words are **lemmatized** - words in third person are changed to first person and verbs in past and future tenses are changed into present.
- Words are **stemmed** - words are reduced to their root form.

[5]:

```
#!pip3 install gensim
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
import numpy as np
np.random.seed(8848)
```

[6]:

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

```
[nltk_data] Downloading package wordnet to /Users/gshyam/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[6]: True

0.3.1 Lemmatizer Example

Before preprocessing our dataset, let's first look at an lemmatizing example. What would be the output if we lemmatized the word 'went':

[7]:
```
print(WordNetLemmatizer().lemmatize('went', pos = 'v')) # past tense to present␣
↪tense
```

go

0.3.2 Stemmer Example

Let's also look at a stemming example. Let's throw a number of words at the stemmer and see how it deals with each one:

[8]:
```
stemmer = SnowballStemmer("english")
original_words = ['caresses', 'flies', 'dies', 'mules', 'denied','died',␣
↪'agreed', 'owned',
           'humbled', 'sized','meeting', 'stating', 'siezing',␣
↪'itemization','sensational',
           'traditional', 'reference', 'colonizer','plotted']
singles = [stemmer.stem(plural) for plural in original_words]

pd.DataFrame(data={'original word':original_words, 'stemmed':singles })
```

[8]:

	original word	stemmed
0	caresses	caress
1	flies	fli
2	dies	die
3	mules	mule
4	denied	deni
5	died	die
6	agreed	agre
7	owned	own
8	humbled	humbl
9	sized	size
10	meeting	meet
11	stating	state
12	siezing	siez
13	itemization	item
14	sensational	sensat
15	traditional	tradit
16	reference	refer
17	colonizer	colon
18	plotted	plot

[9]:
```
'''
Write a function to perform the pre processing steps on the entire dataset
'''



from gensim.parsing.preprocessing import STOPWORDS
from gensim.utils import simple_preprocess
```

```
from gensim.parsing.preprocessing import STOPWORDS

# Tokenize and lemmatize


def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def preprocess(text):
    return [lemmatize_stemming(token) for token in simple_preprocess(text) if␣
 ↪(token not in STOPWORDS and len(token) > 3)]
```

[10]:
```
# tesst a random headline
sample = df.iloc[4310].values[0]
words = sample.split()

print (f"Sample doc: \t {sample}\n")
print (f"Words involved: {words}\n")
print(f"Tokenized and Lemmatized doc: {preprocess(sample)}\n")
```

```
Sample doc:      rain helps dampen bushfires

Words involved: ['rain', 'helps', 'dampen', 'bushfires']

Tokenized and Lemmatized doc: ['rain', 'help', 'dampen', 'bushfir']
```

Let's now preprocess all the news headlines we have. To do that, let's use the map function from pandas to apply `preprocess()` to the `headline_text` column

Note: This may take a few minutes

[11]:
```
processed_docs = df['headline_text'].map(preprocess)
```

[12]:
```
'''
Preview 'processed_docs'
'''
processed_docs[:10]
```

[12]:
```
0             [decid, communiti, broadcast, licenc]
1                                [wit, awar, defam]
2            [call, infrastructur, protect, summit]
3                       [staff, aust, strike, rise]
4              [strike, affect, australian, travel]
5                [ambiti, olsson, win, tripl, jump]
6            [antic, delight, record, break, barca]
7     [aussi, qualifi, stosur, wast, memphi, match]
8             [aust, address, secur, council, iraq]
```

```
9                          [australia, lock, timet]
Name: headline_text, dtype: object
```

0.4 Step 3.1: Bag of words on the dataset

Now let's create a dictionary from 'processed_docs' containing the number of times a word appears in the training set. To do that, let's pass `processed_docs` to `gensim.corpora.Dictionary()` and call it '`dictionary`'.

[13]:
```
'''
Create a dictionary from 'processed_docs' containing the number of times a word␣
↪appears
in the training set using gensim.corpora.Dictionary and call it 'dictionary'
'''

from gensim.corpora import Dictionary
dictionary = Dictionary(processed_docs)
```

[14]:
```
'''
Checking dictionary created
'''
count = 0
for k, v in dictionary.iteritems():
    print(k, v)
    count += 1
    if count > 10:
        break
```

```
0 broadcast
1 communiti
2 decid
3 licenc
4 awar
5 defam
6 wit
7 call
8 infrastructur
9 protect
10 summit
```

** Gensim filter_extremes **

`filter_extremes(no_below=5, no_above=0.5, keep_n=100000)`

Filter out tokens that appear in

- less than no_below documents (absolute number) or
- more than no_above documents (fraction of total corpus size, not absolute number).
- after (1) and (2), keep only the first keep_n most frequent tokens (or keep all if None).

```
[15]: '''
OPTIONAL STEP
Remove very rare and very common words:

- words appearing less than 15 times
- words appearing in more than 10% of all documents
'''
# TODO: apply dictionary.filter_extremes() with the parameters mentioned above
dictionary.filter_extremes(no_below=15, no_above=0.1, keep_n=100000)
```

** Gensim doc2bow **

`doc2bow(document)`

- Convert document (a list of words) into the bag-of-words format = list of (token_id, token_count) 2-tuples. Each word is assumed to be a tokenized and normalized string (either unicode or utf8-encoded). No further preprocessing is done on the words in document; apply tokenization, stemming etc. before calling this method.

```
[16]: '''
Create the Bag-of-words model for each document i.e for each document we create␣
↪a dictionary reporting how many
words and how many times those words appear. Save this to 'bow_corpus'
'''
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
```

```
[17]: '''
Checking Bag of Words corpus for our sample document --> (token_id, token_count)
'''
bow_corpus[4310]
```

```
[17]: [(67, 1), (100, 1), (436, 1), (2813, 1)]
```

```
[ ]:
```

```
[18]: '''
Preview BOW for our sample preprocessed document
'''
# Here document_num is document number 4310 which we have checked in Step 2
bow_doc_4310 = bow_corpus[4310]

for i in range(len(bow_doc_4310)):
    print("Word {} (\"{}\") appears {} time.".format(bow_doc_4310[i][0],
                                                     ␣
↪dictionary[bow_doc_4310[i][0]],
                                                      bow_doc_4310[i][1]))
```

```
Word 67 ("bushfir") appears 1 time.
Word 100 ("help") appears 1 time.
```

```
Word 436 ("rain") appears 1 time.
Word 2813 ("dampen") appears 1 time.
```

0.5 Step 3.2: TF-IDF on our document set

While performing TF-IDF on the corpus is not necessary for LDA implemention using the gensim model, it is recommended. TF-IDF expects a bag-of-words (integer values) training corpus during initialization. During transformation, it will take a vector and return another vector of the same dimensionality.

Please note: The author of Gensim dictates the standard procedure for LDA to be using the Bag of Words model.

** TF-IDF stands for "Term Frequency, Inverse Document Frequency".**

- It is a way to score the importance of words (or "terms") in a document based on how frequently they appear across multiple documents.
- If a word appears frequently in a document, it's important. Give the word a high score. But if a word appears in many documents, it's not a unique identifier. Give the word a low score.
- Therefore, common words like "the" and "for", which appear in many documents, will be scaled down. Words that appear frequently in a single document will be scaled up.

In other words:

- TF(w) = `(Number of times term w appears in a document) / (Total number of terms in the document)`.
- IDF(w) = `log_e(Total number of documents / Number of documents with term w in it)`.

** For example **

- Consider a document containing `100` words wherein the word 'tiger' appears 3 times.
- The term frequency (i.e., tf) for 'tiger' is then:
  - `TF = (3 / 100) = 0.03`.
- Now, assume we have `10 million` documents and the word 'tiger' appears in `1000` of these. Then, the inverse document frequency (i.e., idf) is calculated as:
  - `IDF = log(10,000,000 / 1,000) = 4`.
- Thus, the Tf-idf weight is the product of these quantities:
  - `TF-IDF = 0.03 * 4 = 0.12`.

[19]:
```
'''
Create tf-idf model object using models.TfidfModel on 'bow_corpus' and save it␣
↪to 'tfidf'
'''
from gensim import corpora, models
tfidf = models.TfidfModel(bow_corpus)
```

[20]:
```
'''
Apply transformation to the entire corpus and call it 'corpus_tfidf'
'''
corpus_tfidf = tfidf[bow_corpus]
```

[21]:
```python
'''
Preview TF-IDF scores for our first document --> --> (token_id, tfidf score)
'''
from pprint import pprint
for doc in corpus_tfidf:
    pprint(doc)
    break
```

```
[(0, 0.5694032272086778),
 (1, 0.40633999001577825),
 (2, 0.48769022144343),
 (3, 0.5223275076681089)]
```

0.6 Step 4.1: Running LDA using Bag of Words

We are going for 10 topics in the document corpus.

** We will be running LDA using all CPU cores to parallelize and speed up model training.**

Some of the parameters we will be tweaking are:

- **num_topics** is the number of requested latent topics to be extracted from the training corpus.
- **id2word** is a mapping from word ids (integers) to words (strings). It is used to determine the vocabulary size, as well as for debugging and topic printing.
- **workers** is the number of extra processes to use for parallelization. Uses all available cores by default.
- **alpha** and **eta** are hyperparameters that affect sparsity of the document-topic (theta) and topic-word (lambda) distributions. We will let these be the default values for now(default value is `1/num_topics`)
  - Alpha is the per document topic distribution.
    - High alpha: Every document has a mixture of all topics(documents appear similar to each other).
    - Low alpha: Every document has a mixture of very few topics
  - Eta is the per topic word distribution.
    - High eta: Each topic has a mixture of most words(topics appear similar to each other).
    - Low eta: Each topic has a mixture of few words.
- ** passes ** is the number of training passes through the corpus. For example, if the training corpus has 50,000 documents, chunksize is 10,000, passes is 2, then online training is done in 10 updates:
  - `#1 documents 0-9,999`
  - `#2 documents 10,000-19,999`
  - `#3 documents 20,000-29,999`
  - `#4 documents 30,000-39,999`
  - `#5 documents 40,000-49,999`
  - `#6 documents 0-9,999`
  - `#7 documents 10,000-19,999`
  - `#8 documents 20,000-29,999`

- #9 documents 30,000-39,999
- #10 documents 40,000-49,999

[22]:
```python
from gensim.models import LdaMulticore

lda_model = LdaMulticore(bow_corpus,
                         num_topics=10,
                         id2word = dictionary,
                         passes = 2,
                         workers=2)
```

[23]:
```python
lda_model.print_topics()
```

[23]:
```
[(0,
  '0.037*"plan" + 0.036*"council" + 0.022*"water" + 0.021*"group" + 0.020*"seek"
+ 0.015*"govt" + 0.014*"hospit" + 0.013*"urg" + 0.011*"meet" + 0.011*"fund"'),
 (1,
  '0.021*"report" + 0.021*"crash" + 0.018*"want" + 0.013*"say" + 0.012*"time" +
0.011*"ahead" + 0.011*"england" + 0.011*"elect" + 0.010*"announc" +
0.010*"nation"'),
 (2,
  '0.017*"urg" + 0.016*"appeal" + 0.015*"home" + 0.015*"question" + 0.015*"look"
+ 0.013*"abus" + 0.009*"move" + 0.009*"black" + 0.009*"presid" +
0.009*"visit"'),
 (3,
  '0.041*"iraq" + 0.035*"kill" + 0.019*"attack" + 0.017*"rise" + 0.015*"deal" +
0.015*"iraqi" + 0.013*"trade" + 0.013*"troop" + 0.013*"talk" + 0.013*"terror"'),
 (4,
  '0.020*"jail" + 0.017*"hear" + 0.017*"power" + 0.014*"worker" + 0.014*"strike"
+ 0.013*"record" + 0.013*"murder" + 0.011*"stay" + 0.011*"work" +
0.011*"second"'),
 (5,
  '0.079*"polic" + 0.022*"probe" + 0.018*"investig" + 0.014*"charg" +
0.014*"drug" + 0.013*"death" + 0.013*"test" + 0.012*"shoot" + 0.012*"coast" +
0.012*"sydney"'),
 (6,
  '0.018*"return" + 0.012*"close" + 0.012*"final" + 0.011*"year" + 0.011*"centr"
+ 0.010*"student" + 0.009*"name" + 0.008*"win" + 0.008*"celebr" +
0.007*"injuri"'),
 (7,
  '0.020*"secur" + 0.018*"prison" + 0.017*"miss" + 0.016*"chief" +
0.015*"search" + 0.014*"studi" + 0.011*"road" + 0.010*"fail" + 0.010*"servic" +
0.009*"find"'),
 (8,
  '0.035*"claim" + 0.031*"court" + 0.028*"govt" + 0.028*"face" + 0.023*"health"
+ 0.020*"charg" + 0.019*"fund" + 0.015*"budget" + 0.015*"reject" +
0.015*"accus"'),
```

```
 (9,
  '0.021*"concern" + 0.017*"welcom" + 0.016*"industri" + 0.015*"union" +
0.014*"high" + 0.014*"take" + 0.014*"govt" + 0.013*"price" + 0.012*"decis" +
0.012*"push"')]
```

0.6.1 Classification of the topics

Using the words in each topic and their corresponding weights, what categories were you able to infer?

- 0:
- 1:
- 2:
- 3:
- 4:
- 5:
- 6:
- 7:

- 8:
- 9:

0.7 Step 4.2 Running LDA using TF-IDF

[24]:
```
'''
Define lda model using corpus_tfidf
'''
# TODO
lda_model_tfidf = LdaMulticore(corpus_tfidf,
                               num_topics=10,
                               id2word = dictionary,
                               passes = 2,
                               workers=4)
```

[25]:
```
#print topics and its relative weight
lda_model_tfidf.print_topics()
```

[25]:
```
[(0,
  '0.010*"coast" + 0.008*"south" + 0.008*"gold" + 0.006*"nuclear" +
0.006*"north" + 0.006*"korea" + 0.006*"polic" + 0.005*"hospit" + 0.005*"iran" +
0.005*"west"'),
 (1,
  '0.012*"accid" + 0.011*"die" + 0.010*"woman" + 0.008*"blaze" + 0.007*"polic" +
0.006*"kill" + 0.006*"court" + 0.005*"crash" + 0.005*"firefight" +
0.005*"driver"'),
 (2,
  '0.007*"industri" + 0.007*"award" + 0.006*"govt" + 0.005*"plan" +
```

```
0.005*"doubt" + 0.004*"benefit" + 0.004*"sugar" + 0.004*"offer" + 0.004*"film" +
0.004*"council"'),
 (3,
  '0.018*"polic" + 0.013*"charg" + 0.010*"search" + 0.009*"murder" +
0.009*"miss" + 0.009*"drug" + 0.008*"court" + 0.008*"servic" + 0.006*"face" +
0.006*"arrest"'),
 (4,
  '0.009*"shoot" + 0.008*"polic" + 0.008*"jail" + 0.008*"crash" + 0.008*"kill" +
0.007*"dead" + 0.007*"iraqi" + 0.006*"fund" + 0.006*"plane" + 0.005*"govt"'),
 (5,
  '0.007*"aussi" + 0.006*"stand" + 0.006*"world" + 0.005*"say" + 0.005*"rebel" +
0.005*"council" + 0.005*"zimbabw" + 0.004*"super" + 0.004*"sale" +
0.004*"india"'),
 (6,
  '0.010*"lead" + 0.006*"bird" + 0.005*"england" + 0.005*"black" + 0.005*"open"
+ 0.005*"season" + 0.005*"make" + 0.004*"clash" + 0.004*"spot" +
0.004*"warrior"'),
 (7,
  '0.008*"kill" + 0.007*"water" + 0.006*"injur" + 0.006*"restrict" +
0.005*"teacher" + 0.005*"plan" + 0.005*"govt" + 0.005*"blast" + 0.005*"strike" +
0.005*"sheep"'),
 (8,
  '0.008*"latham" + 0.008*"iraq" + 0.006*"plan" + 0.005*"chang" +
0.005*"solomon" + 0.005*"merger" + 0.005*"unit" + 0.005*"govt" + 0.004*"council"
+ 0.004*"troop"'),
 (9,
  '0.011*"rise" + 0.008*"govt" + 0.008*"terror" + 0.006*"appeal" +
0.006*"prison" + 0.006*"council" + 0.006*"toll" + 0.005*"deal" + 0.005*"fear" +
0.005*"trade"')]
```

[26]:
```
#print topics and its relative weight
lda_model_tfidf.print_topic(1)
```

[26]:
```
'0.012*"accid" + 0.011*"die" + 0.010*"woman" + 0.008*"blaze" + 0.007*"polic" +
0.006*"kill" + 0.006*"court" + 0.005*"crash" + 0.005*"firefight" +
0.005*"driver"'
```

0.7.1 Classification of the topics

As we can see, when using tf-idf, heavier weights are given to words that are not as frequent which results in nouns being factored in. That makes it harder to figure out the categories as nouns can be hard to categorize. This goes to show that the models we apply depend on the type of corpus of text we are dealing with.

Using the words in each topic and their corresponding weights, what categories could you find?

- 0:
- 1:

- 2:
- 3:
- 4:

- 5:
- 6:
- 7:
- 8:
- 9:

0.8 Step 5.1: Performance evaluation by classifying sample document using LDA Bag of Words model

We will check to see where our test document would be classified.

[29]:
```
'''
Text of sample document 4310
'''
document_num=4310
processed_docs[document_num]
```

[29]: ['rain', 'help', 'dampen', 'bushfir']

[30]:
```
'''
Check which topic our test document belongs to using the LDA Bag of Words model.
'''

# Our test document is document number 4310
for index, score in sorted(lda_model[bow_corpus[document_num]], key=lambda tup:␣
↪-1*tup[1]):
    print("\nScore: {}\t \nTopic: {}".format(score, lda_model.
↪print_topic(index, 10)))
```

```
Score: 0.29290565848350525
Topic: 0.021*"concern" + 0.017*"welcom" + 0.016*"industri" + 0.015*"union" +
0.014*"high" + 0.014*"take" + 0.014*"govt" + 0.013*"price" + 0.012*"decis" +
0.012*"push"

Score: 0.2914659380912781
Topic: 0.079*"polic" + 0.022*"probe" + 0.018*"investig" + 0.014*"charg" +
0.014*"drug" + 0.013*"death" + 0.013*"test" + 0.012*"shoot" + 0.012*"coast" +
0.012*"sydney"

Score: 0.27559369802474976
Topic: 0.037*"plan" + 0.036*"council" + 0.022*"water" + 0.021*"group" +
0.020*"seek" + 0.015*"govt" + 0.014*"hospit" + 0.013*"urg" + 0.011*"meet" +
```

```
0.011*"fund"

Score: 0.02000635303556919
Topic: 0.020*"secur" + 0.018*"prison" + 0.017*"miss" + 0.016*"chief" +
0.015*"search" + 0.014*"studi" + 0.011*"road" + 0.010*"fail" + 0.010*"servic" +
0.009*"find"

Score: 0.02000591531395912
Topic: 0.017*"urg" + 0.016*"appeal" + 0.015*"home" + 0.015*"question" +
0.015*"look" + 0.013*"abus" + 0.009*"move" + 0.009*"black" + 0.009*"presid" +
0.009*"visit"

Score: 0.020004533231258392
Topic: 0.021*"report" + 0.021*"crash" + 0.018*"want" + 0.013*"say" +
0.012*"time" + 0.011*"ahead" + 0.011*"england" + 0.011*"elect" + 0.010*"announc"
+ 0.010*"nation"

Score: 0.020004479214549065
Topic: 0.020*"jail" + 0.017*"hear" + 0.017*"power" + 0.014*"worker" +
0.014*"strike" + 0.013*"record" + 0.013*"murder" + 0.011*"stay" + 0.011*"work" +
0.011*"second"

Score: 0.020004462450742 72
Topic: 0.035*"claim" + 0.031*"court" + 0.028*"govt" + 0.028*"face" +
0.023*"health" + 0.020*"charg" + 0.019*"fund" + 0.015*"budget" + 0.015*"reject"
+ 0.015*"accus"

Score: 0.020004458725452423
Topic: 0.041*"iraq" + 0.035*"kill" + 0.019*"attack" + 0.017*"rise" +
0.015*"deal" + 0.015*"iraqi" + 0.013*"trade" + 0.013*"troop" + 0.013*"talk" +
0.013*"terror"

Score: 0.020004458725452423
Topic: 0.018*"return" + 0.012*"close" + 0.012*"final" + 0.011*"year" +
0.011*"centr" + 0.010*"student" + 0.009*"name" + 0.008*"win" + 0.008*"celebr" +
0.007*"injuri"
```

0.8.1 It has the highest probability (x) to be part of the topic that we assigned as X, which is the accurate classification.

0.9 Step 5.2: Performance evaluation by classifying sample document using LDA TF-IDF model

[31]:
```python
'''
Check which topic our test document belongs to using the LDA TF-IDF model.
'''
for index, score in sorted(lda_model_tfidf[bow_corpus[document_num]],␣
↪key=lambda tup: -1*tup[1]):
```

```
    print("\nScore: {}\t \nTopic: {}".format(score, lda_model_tfidf.
↪print_topic(index, 10)))
```

```
Score: 0.534644365310669
Topic: 0.010*"lead" + 0.006*"bird" + 0.005*"england" + 0.005*"black" +
0.005*"open" + 0.005*"season" + 0.005*"make" + 0.004*"clash" + 0.004*"spot" +
0.004*"warrior"

Score: 0.3053092360496521
Topic: 0.007*"industri" + 0.007*"award" + 0.006*"govt" + 0.005*"plan" +
0.005*"doubt" + 0.004*"benefit" + 0.004*"sugar" + 0.004*"offer" + 0.004*"film" +
0.004*"council"

Score: 0.020009998232126236
Topic: 0.008*"kill" + 0.007*"water" + 0.006*"injur" + 0.006*"restrict" +
0.005*"teacher" + 0.005*"plan" + 0.005*"govt" + 0.005*"blast" + 0.005*"strike" +
0.005*"sheep"

Score: 0.020007168874144554
Topic: 0.011*"rise" + 0.008*"govt" + 0.008*"terror" + 0.006*"appeal" +
0.006*"prison" + 0.006*"council" + 0.006*"toll" + 0.005*"deal" + 0.005*"fear" +
0.005*"trade"

Score: 0.020006464794278145
Topic: 0.018*"polic" + 0.013*"charg" + 0.010*"search" + 0.009*"murder" +
0.009*"miss" + 0.009*"drug" + 0.008*"court" + 0.008*"servic" + 0.006*"face" +
0.006*"arrest"

Score: 0.02000533789396286
Topic: 0.008*"latham" + 0.008*"iraq" + 0.006*"plan" + 0.005*"chang" +
0.005*"solomon" + 0.005*"merger" + 0.005*"unit" + 0.005*"govt" + 0.004*"council"
+ 0.004*"troop"

Score: 0.020004617050290108
Topic: 0.009*"shoot" + 0.008*"polic" + 0.008*"jail" + 0.008*"crash" +
0.008*"kill" + 0.007*"dead" + 0.007*"iraqi" + 0.006*"fund" + 0.006*"plane" +
0.005*"govt"

Score: 0.020004604011774063
Topic: 0.007*"aussi" + 0.006*"stand" + 0.006*"world" + 0.005*"say" +
0.005*"rebel" + 0.005*"council" + 0.005*"zimbabw" + 0.004*"super" + 0.004*"sale"
+ 0.004*"india"

Score: 0.020004263147711754
Topic: 0.012*"accid" + 0.011*"die" + 0.010*"woman" + 0.008*"blaze" +
0.007*"polic" + 0.006*"kill" + 0.006*"court" + 0.005*"crash" + 0.005*"firefight"
```

```
+ 0.005*"driver"

Score: 0.02000398188829422
Topic: 0.010*"coast" + 0.008*"south" + 0.008*"gold" + 0.006*"nuclear" +
0.006*"north" + 0.006*"korea" + 0.006*"polic" + 0.005*"hospit" + 0.005*"iran" +
0.005*"west"
```

0.9.1 It has the highest probability (x%) to be part of the topic that we assigned as X.

0.10 Step 6: Testing model on unseen document

[32]:
```python
unseen_document = "My favorite sports activities are running and swimming."

# Data preprocessing step for the unseen document
bow_vector = dictionary.doc2bow(preprocess(unseen_document))

for index, score in sorted(lda_model[bow_vector], key=lambda tup: -1*tup[1]):
    print("Score: {}\t Topic: {}".format(score, lda_model.print_topic(index,␣
 ↪5)))
```

```
Score: 0.4218541979789734        Topic: 0.021*"report" + 0.021*"crash" +
0.018*"want" + 0.013*"say" + 0.012*"time"
Score: 0.4181063175201416        Topic: 0.018*"return" + 0.012*"close" +
0.012*"final" + 0.011*"year" + 0.011*"centr"
Score: 0.02000945806503296       Topic: 0.017*"urg" + 0.016*"appeal" +
0.015*"home" + 0.015*"question" + 0.015*"look"
Score: 0.020006049424409866      Topic: 0.020*"secur" + 0.018*"prison" +
0.017*"miss" + 0.016*"chief" + 0.015*"search"
Score: 0.02000446990132332       Topic: 0.035*"claim" + 0.031*"court" +
0.028*"govt" + 0.028*"face" + 0.023*"health"
Score: 0.0200043972581625        Topic: 0.079*"polic" + 0.022*"probe" +
0.018*"investig" + 0.014*"charg" + 0.014*"drug"
Score: 0.02000419981777668       Topic: 0.037*"plan" + 0.036*"council" +
0.022*"water" + 0.021*"group" + 0.020*"seek"
Score: 0.020003730431199074      Topic: 0.021*"concern" + 0.017*"welcom" +
0.016*"industri" + 0.015*"union" + 0.014*"high"
Score: 0.020003607496619225      Topic: 0.041*"iraq" + 0.035*"kill" +
0.019*"attack" + 0.017*"rise" + 0.015*"deal"
Score: 0.020003607496619225      Topic: 0.020*"jail" + 0.017*"hear" +
0.017*"power" + 0.014*"worker" + 0.014*"strike"
```

The model correctly classifies the unseen document with ‘x’% probability to the X category.