Practical 5: Topic Modelling

1. Introduction

In this practical you will be using a Python implementation (https://github.com/lda-project/lda) of Latent Dirichlet Allocation (LDA) using collapsed Gibbs Sampling technique for parameter estimation and inference. It is very fast and is designed to analyse hidden/latent topic structures of large-scale datasets including very large collections of text/Web documents. LDA was first introduced by David Blei et al.

LDA is useful for the following potential application areas:

- Information Retrieval (analysing semantic/latent topic/concept structures of large text collection for a more intelligent information search.
- Document Classification/Clustering, Document Summarization, and Text/Web Data Mining community in general.
- Collaborative Filtering
- Content-based Image Clustering, Object Recognition, and other applications of Computer Vision in general.
- Other potential applications in biological data.

2. Install the Package

- **(Recommended)** You can directly install the package via Pip using the following command

  ```
  $ pip install lda
  ```

- Alternatively, you can download the zipped code from https://github.com/lda-project/lda and build the project using the source. Please make sure that Python development headers and a working C/C++ compiler have been previously installed. The requirements can be configured using

  ```
  $ sudo apt-get install build-essential python3-dev python3-setuptools python3-numpy
  ```

  Next, unzip the tarball, change the work directory, and run Cython to generate the relevant C files.

  ```
  $ make cython
  ```

  Finally, install the package with the following command.

  ```
  $ python setup.py install
  ```

3. How to run the LDA model

In this task, we will inspect a dataset which consists of 395 Reuters News articles.

3.0. Dataset Description

- reuters.ldac: a preprocessed file in the popular LDA-C format (https://github.com/blei-lab/lda-c/blob/master/readme.txt). You can see that each line is of the form:

  ```
  [M] [term_1]:[count] [term_2]:[count] ...  [term_N]:[count]
  ```

  where [M] is the number of unique terms in the document, and the [count] associated with each term is how many times that term appeared in the document. Note that [term_i] is an integer which indexes the term; it is not a string.

- reuters.titles: the titles of news articles
- reuters.tokens: a vocabulary which stores all terms. The order is corresponding to the term index in reuters.ldac.

3.1. Load the Data

```
>>> import numpy as np
>>> import lda
>>> X = lda.datasets.load_reuters()  # to build a document-term matrix
>>> vocab = lda.datasets.load_reuters_vocab()
>>> titles = lda.datasets.load_reuters_titles()
```

Let's further inspect X:

```
>>> X.shape
>>> X.sum()
```

What do you get? Can you interpret the output?

3.2. Train the Model

Initialise the LDA model using
`lda.LDA(n_topics, n_iter=2000, alpha=0.1, eta=0.01, random_state=None, refresh=10)`

in which we need to set the following parameters

- n_topics <int>: Number of topics
- n_iters <int>: Number of sampling iterations *// default: 2000*
- alpha <float>: Dirichlet parameter for distribution over **topics** *// default: 0.1*
- eta <float>: Dirichlet parameter for distribution over **words** *// default: 0.01*
- random_state <int or RandomState>: The generator used for the initial topics *// default: None*

For example, you can run

```
>>> model = lda.LDA(n_topics=20, n_iter=1500, random_state=1)
```

to initialise a model. For training/fitting a model, use the following command

```
>>> model.fit(X)
```

3.3. Estimation of the corpus-wise topic distribution

To retrieve the estimation of the corpus-wise topic-word distributions, use the following command

```
>>> topic_word = model.topic_word_
```

For each of the 20 topics extracted by LDA, suppose we would like to inspect the top 8 topic words with the highest probability. We can do so by executing the following code

```
>>> n_top_words = 8
>>> for i, topic_dist in enumerate(topic_word):
...     topic_words = np.array(vocab)[np.argsort(topic_dist)][:-n_top_words:-1]
...     print('Topic {}: {}'.format(i, ' '.join(topic_words)))
```

Please check the outputs and try to understand their meaning. Are you able to interpret the semantic meanings of the topics?

```
Topic 0: british churchill sale million major letters west
Topic 1: church government political country state people party
Topic 2: elvis king fans presley life concert young
Topic 3: yeltsin russian russia president kremlin moscow michael
Topic 4: pope vatican paul john surgery hospital pontiff
Topic 5: family funeral police miami versace cunanan city
Topic 6: simpson former years court president wife south
Topic 7: order mother successor election nuns church nirmala
Topic 8: charles prince diana royal king queen parker
Topic 9: film french france against bardot paris poster
Topic 10: germany german war nazi letter christian book
Topic 11: east peace prize award timor quebec belo
Topic 12: n't life show told very love television
Topic 13: years year time last church world people
Topic 14: mother teresa heart calcutta charity nun hospital
Topic 15: city salonika capital buddhist cultural vietnam byzantine
Topic 16: music tour opera singer israel people film
Topic 17: church catholic bernardin cardinal bishop wright death
Topic 18: harriman clinton u.s ambassador paris president churchill
Topic 19: city museum art exhibition century million churches
```

3.4. Estimation of the per-document topic distribution

To retrieve the estimation of the per-document topic proportion and infer the topics for the i-th article, execute the following code

```
>>> doc_topic = model.doc_topic_
>>> print("{} (top topic: {})".format(titles[i], doc_topic[i].argmax()))
```

```
3 UK: Palace warns British weekly over Charles pictures. LONDON 1996-08-25 (top topic: 8)
```

i=3

Please check the outputs and try to understand their meaning. Are you able to figure out the most prominent topics for each article?

3.5. Inference for Previously Unseen Data

Another important application scenario is to perform inference on unseen data using a trained model. To simulate such a scenario, see the following toy example. We can split the first 200 articles for training and the last 10 articles for testing, as

```
>>> X_train = X[200:]
>>> X_test = X[:10]
>>> titles_test = titles[:10]
```

We can then fit our model in the training set using

```
>>> model.fit(X_train)
```

Once the model training is done, then do the inference for the test set

```
>>> doc_topic_test = model.transform(X_test)
>>> for title, topics in zip(titles_test, doc_topic_test):
...     print("{} (top topic: {})".format(title, topics.argmax()))
```

Please check the outputs and compare them with previous outputs.

3.6. Further questions for you to think about

- With the estimated per-document topic proportion, we can figure out the most prominent topics for a particular article. What if we would like to figure out the most prominent topics for the entire data set?
- The topic number setting n_topics can affect the quality of the extracted topics. Re-run the model with different topic number settings (e.g., 20, 30, 40) and see whether you can spot the effect by examining the topic distributions.