4. Text Representation and Information Retrieval

Objectives

In this lab, you will deal with **textual** data and be able to get hands-on experience with popular methods for the following concepts:

1. vector space models;
2. information retrieval;
3. topic modeling;
4. word embeddings.

In the process, you will also be exposed to some challenges that arise when dealing with large, real-world, noisy data. Real-world data is sometimes messy, and results need to be interpreted carefully. Use your judgment, don't jump to conclusions, and keep in mind that you can always preprocess the data if needed.

We expect you to be curious and proactive—we intentionally keep this handout short and want you to look up for the appropriate resources online when needed (e.g., on using Spark and various Python libraries). **Do not hesitate to make use of the TAs**, whether to ask questions or to double-check your code and analyses.

Finally, we expect you to take this lab almost as a small project. *There is no single right way to solve the questions in this lab*, and you should feel free to come up with creative approaches (creativity is rewarded).

Deliverables

Just like in the previous labs, we expect you to hand in an HTML export of the four Jupyter notebooks containing the result of your analyses,

- `lab4-1-vsm.ipynb`
- `lab4-2-lsi.ipynb`
- `lab4-3-lda.ipynb`
- `lab4-4-w2v.ipynb`

as well as any supplementary files that might be necessary to run your analyses (e.g., Python modules). Keep in mind the four criteria that we use for grading:

1. general **correctness** of the analyses;
2. **code readability** and clarity;
3. **presentation** of the notebook and the figures;
4. bonus points for **creativity**.

Datasets

You will use two different datasets. The first one is rather small, and contains EPFL courses' description in English. It is stored as a text in the file `courses.txt`. Each line contains a dictionary representing a course. The content is:

- `courseId`: unique identifier of a course (e.g., "COM-308");

- `name`: name of the course;
- `description`: full description of the course.

The second dataset is much larger (219Mb), and consists of a dump of Wikipedia for schools. It is also stored as text in HDFS on the server, under `/ix/wikipedia-for-schools.txt`. Each line is a dictionary with the following key-value pairs:

- `pageId`: unique identifier of the page;
- `title`: title of the page;
- `tokens`: text of the article as a list of pre-processed tokens.

4.1 Vector space models

In this part, you will build a tool to help craft your study plan for your first Master's semester at EPFL. To do so, we scraped for you the course descriptions (for the classes given in English) from all faculties. They are available in the `courses` dataset. This will give you a chance to test vector space models as search engines. *Vector space models* (VSM) provide a way to represent a corpus of documents in a...vector space. That is, we construct a *profile* for each document. An important assumption is that documents are represented as the set of terms they contain: the well-known *bag-of-words* assumption. As seen in class, a widely used approach is the *term-frequency inverse-document-frequency* (TF-IDF), that characterises each term as a relevance score to each document, and each document as a vector in the terms space.

However, before anything, a first, crucial step is the one of *pre-processing*. As you will find out in your engineering life, pre-processing influences a lot the performances of any model or algorithm. Moreover, it depends on the data, the method and the goal that you have. So it better be done properly!

Exercise 4.1 Pre-processing
Pre-process the corpus to create bag-of-words representations of each document. You are free to proceed as you wish. Here are some things you can consider:

- Remove the stopwords.
- Remove the punctuation.
- Remove the very frequent words.
- Remove the infrequent words.
- Stem the words (look for *stemming* online).
- Lemmatise the words (look for *lemmatization* online).
- Add n-grams to the vocabulary, where $n = 2, 3, ...$
- Any other idea is allowed and encouraged.

1. Explain which ones you implemented and why.
2. Print the terms in the pre-processed description of the IX class in alphabetical order.

Hint: We provide you with a file `stopwords.pkl`. ■

You will now implement TF-IDF for your pre-processed corpus. TF-IDF comes in different flavours (a quick Wikipedia search will give you insights). You are free to use the one you prefer, as long as you are consistent!

Exercise 4.2 Term-document matrix
Construct an $M \times N$ term-document matrix X, where M is the number of terms and N is the

number of documents. The matrix X should be sparse. You are **not** allowed to use libraries for this task (i.e., the computation of TF-IDF must be implemented by *you*.)

1. Print the 15 terms in the description of the IX class with the highest TF-IDF scores.
2. Explain where the difference between the large scores and the small ones comes from.

Hint: It is useful for this exercise and the rest of the lab to keep track of the mapping between terms and their indices, and documents and their indices. ■

Say you would like now to improve your skills and knowledge in *Markov chains* theory, and hence choose classes that teach this topic. Also, you are interested in social networks and would like to play more with Facebook data. To do so, we can use the term-document matrix to perform some queries and retrieve the most relevant documents. Recall that to compare two documents $\mathbf{d}_i$ and $\mathbf{d}_j$ in the vector space defined by a term-document matrix, a typical similarity measure is the *cosine similarity*, defined as

$$\text{sim}(\mathbf{d}_i, \mathbf{d}_j) := \cos(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i^\intercal \mathbf{d}_j}{\|\mathbf{d}_i\| \|\mathbf{d}_j\|}.$$

Exercise 4.3 Document similarity search
Search for "markov chains" and "facebook".

1. Display the top **five** courses together with their similarity score for each query.
2. What do you think of the results? Give your intuition on what is happening. ■

4.2 Latent semantic indexing

In the vector space model, we are limited by the fact that each term has its own dimension and is completely independent of the other terms. An issue of such an approach is (near-)synonymy: we would like documents including the term *social network* to show up when the query contains the term *facebook*. To address that issue, we would need a technique that automatically uncovers the hidden relations between the terms... You guessed it, we will take advantage of our favorite dimensionality reduction tool, *singular value decomposition* (SVD). By applying SVD on the term-document matrix X, we project both terms and documents onto a low-dimension latent space of concepts. In the context of information retrieval, this is known as *latent semantic indexing* (LSI). We say that we obtain a low-rank approximation

$$X \approx USV^\intercal,$$

where U is an $M \times K$ matrix, S is a $K \times K$ diagonal matrix, V is an $N \times K$ matrix and $K << N$ is the rank.

Exercise 4.4 Latent semantic indexing
Apply SVD with $K = 300$ to your term-document matrix X from the previous exercise.

1. Describe the rows and columns of U and V, and the values of S.
2. Print the top-20 singular values of X. ■

You do not necessarily want to pursue a Masters in Communication systems or Computer science. Therefore, you would like to understand better what types of courses are offered on the campus. To do so, LSI allows you to extract some *topics* from the corpus of documents.

Exercise 4.5 Topic extraction
Extract the topics from the term-document matrix X using the low-rank approximation.

1. Print the top-10 topics as a combination of 10 terms and 10 documents.
2. Give a label to each of them. ■

In the previous section, similarity between documents comes from shallow knowledge about the relationships between the terms, typically the frequency counts of words. As explained, LSI provides a way to use a deeper understanding of the corpus. To compute the similarity between a term t and a document d using the latent concept-space representation provided by LSI, we can simply compute the following product:

$$\text{sim}(t,d) := \frac{\mathbf{u}_t S \mathbf{v}_d^\mathsf{T}}{\|\mathbf{u}_t\| \|S\mathbf{v}_d\|},$$

where $\mathbf{u}_t$ and $\mathbf{v}_d$ are **row** vectors, corresponding to rows of U and V respectively.

Exercise 4.6 Document similarity search in concept-space
Implement a search function using LSI concept-space, and search for "markov chains" and "facebook".

1. Display the top **five** courses together with their similarity score for each query.
2. Compare with the previous section. ■

Finally, as you loved the subjects taught in this class, you would like to take similar courses next semester. You can compute the similarity between documents very efficiently in LSI. This can be seen as a recommender system for documents in the corpus.

Exercise 4.7 Document-document similarity
Find the classes that are the most similar to Internet Analytics.

1. Write down the equation to efficiently compute the similarity between documents.
2. Print the top 5 classes most similar to COM-308. ■

4.3 Latent Dirichlet allocation

You will now use a more advanced topic model: *latent Dirichlet allocation* (LDA). As seen in class, this probabilistic generative model allows to represent documents as mixtures of topics that are themselves distributions of words. The *Dirichlet* distribution comes into the game to encode this idea: it represents a distribution of distributions. Topics can hence generate words with certain probabilities. As a result of all this, unobserved (*latent*) topics are automatically discovered from observed words in documents.

In this section, you will experiment with the (hyper)parameters of LDA. We can start by first comparing the topics uncovered by this model with LSI from the previous section. We will use Spark for this section. Note that the version of LDA implemented in Pyspark 1.6.2 is lacking a few features. For instance, it is unfortunately not possible to retrieve the distribution of topics per document.

Exercise 4.8 Topics extraction
Using your pre-processed `courses` dataset, extract topics using LDA.

1. Print $k = 10$ topics extracted using LDA and give them labels.
2. How does it compare with LSI?

You can use the default values for all parameters. ■

The Dirichlet distributions in the LDA model have each one hyperparameter, denoted α for the distribution of topics in documents and β for the distribution of words in topics. Their role is to encode any prior belief we have about the hidden topics for the corpus at hand.

Exercise 4.9 Dirichlet hyperparameters
Analyse the effects of α and β. You should start by reading the documentation of pyspark.mllib.clustering.LDA.

1. Fix $k = 10$ and $\beta = 1.01$, and vary α. How does it impact the topics?
2. Fix $k = 10$ and $\alpha = 6$, and vary β. How does it impact the topics?

Hint: You can set the seed to produce more comparable outputs. ■

Finally, we can try to find the topics that most explain the topics covered by the courses taught at EPFL.

Exercise 4.10 EPFL's taught subjects
List the subjects of EPFL's classes.

1. Find the combination of k, α and β that gives the most interpretable topics.
2. Explain why you chose these values.
3. Report the values of the hyperparameters that you used and your labels for the topics.

■

Satisfied about your freshly crafted study plan, you are interested now in analysing how Wikipedia is structured. However, the dataset is too large to experiment a lot of different combinations. In fact, fitting the LDA model on the provided dataset requires around 5 minutes to run - and the deadline is coming... You will need your intuition!

Exercise 4.11 Wikipedia structure
Extract the structure in terms of topics from the `wikipedia-for-school` dataset. Use your intuition about how many topics might be covered by the articles and how they are distributed.

1. Report the values for k, α and β that you chose *a priori* and why you picked them.
2. Are you convinced by the results? Give labels to the topics if possible. ■

4.4 Word2Vec

In section 4.2, we saw that LSI can be a useful technique to address the issue of synonymy. In this section we will explore another technique that addresses this problem, namely *Word Embeddings* and in particular, *Word2Vec*. For the exercises in this section you will be using the EPFL courses dataset `courses.txt`.

Word2Vec finds dense vector representations for words that capture their meaning, based on the assumption that words that appear in similar contexts have similar meaning. In the Skip-Gram model of Word2Vec, the representations are learned for each word in the vocabulary by moving a sliding window through a large corpus and using the word in the centre of the window to predict the words in the context on both sides.

The vectors thus learned are found to be such that the vectors of words with similar meaning have a high cosine similarity. Thus if we represent documents and queries by aggregating the Word2Vec representations of their words, we should be able to mitigate the issue of synonymy,

as we shall see in this section.

As training a good quality Word2Vec model needs a large corpus and significant computational resources, a standard practice for using these representations in applications with limited data is to use *pretrained* vectors that have been made available by various research groups. We will use the model ID 6 from the NLPL word embeddings repository (`http://vectors.nlpl.eu/repository/`), made available by the Language Technology Group at the University of Oslo. As mentioned on the site, this model was trained on the dump of the English Wikipedia and has vocabulary of about 300,000 words.

The model has already been downloaded and put on the cluster at `/ix/model.txt` (not on HDFS). The first line in the text file contains the vocabulary size and the dimension of the word embeddings. Each subsequent line contains a word followed by its vector representation with white space as the separator.

A useful package for working with word embeddings is `gensim` (`https://radimrehurek.com/gensim/`) and this has already been installed on the cluster. The package contains methods to load the vectors into a memory efficient 'KeyedVectors' instance. See `https://radimrehurek.com/gensim/models/keyedvectors.html` for details.

Note that you would need to pre-process the text again so that the vocabulary matches that of the pre-trained word vectors. In particular, note that you should not transform the words to lower case as the pre-trained model treats upper cased and lower cased versions of a word as different. Thus 'Apple' and 'apple' are two different words with two different vectors, and there is no vector for 'markov', only for 'Markov'. Also note that you should not perform stemming or lemmatization as these operations were not performed for the pre-trained model. As the pre-trained model is trained on a large corpus these operations are not necessary since there would be sufficient instances of the different forms of a word (upper or lower case, different endings etc.) to learn a good vector representation for each.

You would also need to make a reasonable choice for a 'default' vector for those words that are in your dataset but are not present in the pre-trained model.

As a first exercise that will enable us to see how word vectors are organized in space, we will try to cluster them. Since vectors of words that are related or have similar meaning have high cosine similarity, clustering them can give a quick indication of topics being discussed in a corpus or a document. You can use the methods implemented in `sklearn.cluster` for this exercise.

Exercise 4.12 Clustering word vectors

Apply k-means to cluster word vectors of the unique words in your pre-processed dataset. Note that k-means finds clusters so that the *Euclidean distance* of each datapoint to its cluster mean is minimized, whereas here we are interested in minimizing the *cosine similarity*. However, in practice it is found that once the word vectors are normalized, the k-means algorithm usually finds reasonable clusters.

1. After performing k-means, print the top-10 words for each cluster. The top-10 words are those words in your pre-processed dataset whose vectors are most similar in cosine similarity to the cluster centroid. Choose a suitable value of k so that you are able to find meaningful clusters.
2. What are the different types of clusters that you observe ? Give labels to 10 of the clusters that are representative of the different types that you see.
3. Compare the clusters to the topics that you obtained for LSI and LDA. ■

To the use the word vectors for information retrieval we need a way to aggregate the vector representations for the words into representations for documents and queries. There are many

ways to do this, including using complex *neural networks*, but simply averaging the vectors has been observed to give good performance. In the case of documents, using a weighted average of the word vectors with the TF-IDF scores of each word as the weight gives improved performance over a simple average, as more importance is given to rare words which are usually more meaningful. Once we have the vector representations for the documents and queries, similarity is computed using the cosine similarity as in the vector space model and LSI.

Exercise 4.13 Document search
Implement a search function using representations constructed from the Word2Vec vectors, and search for "Markov chains" and "Facebook".

1. Display the top **five** courses together with their similarity score for each query.
2. Compare with what you obtain for the vector space model and LSI . ■

Since the word embedding algorithms such as Skip-Gram are more efficient in terms of memory usage as compared to LSI, we can train it on much larger corpora with a much larger vocabulary. One advantage of working with pre-trained word embeddings trained on such large corpora is that we obtain information about many words not present in our small corpus, and in particular how they relate to the words in our small corpus. This in turn allows our system to generalize better. For instance, we can work with queries which contain words outside our corpus (we can avoid computation of IDF for the words in the query without much loss in terms of results), while this would not be possible for the system that uses LSI.

Exercise 4.14 Document search with terms outside the corpus
Test the ability of your system to generalize. For each of the parts first verify that the terms in the query are not present in your pre-processed dataset.

1. Search for "MySpace Orkut" and display the top five results. Compare the results with what what you obtained for "Facebook" in the previous exercise.
2. You have been reading about the ongoing pandemic and would like to know which EPFL courses you can attend to expand your knowledge of related topics. Search for "coronavirus" and display the top five results. Do the results seem reasonable ? ■