KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS KU LEUVEN

Postgraduate Studies in Big Data & Analytics in Business and Management

Dissertation

Topic modelling for European Commission Legislations

Build a machine learning model for topics extraction and relate all legislations by their similarity. Given a new phrase, predict the topic is about and list all related legislations.

Ermal Aliraj

2021-2022

Table of Contents

Executive Summary

Build an *unsupervised machine learning model* to cluster documents based on the topics they talk about.

(1) Get the dataset

As dataset for the model are used 1585 Regulations downloaded from EC portal EUR-Lex.

(2) Load the dataset

We build a list with **all file** names and its content. The list will contain tuples *<FileName, FileContent>*

(3) Analysing the structure of the data

Using as reference Reg/2016/679 (GDPR) we did a Part of Speech (POS) tagging and come up that the most used words are of type: noun, adjective, verb, and adverb.

(4) Pre-Processing

Data are pre-processed removing stop words, words less than three characters and less informative lemmas not in category nouns, verbs, adjectives, and adverbs.

(5) Build the LDA model

We build the **LDA model** with the *Dictionary* and *Corpus BoW*.
Dictionary with all the words present in *all files* is created.
Bag of Words is created for each file using the *dictionary*.
Corpus Bag of Words is built putting all *BoWs* together.

(6) MAPPER Topic-Documents

a. **Prediction function** - An unseen phrase is given for prediction.
   i. Using the LDA model's *Dictionary*, we calculate the BoW for the unseen phrase.
   ii. We ask the *LDA model* to calculate the relation the new BoW has with each topic.
   iii. A list of tuples *<topicId, probability>* will be returned.
   iv. We rake the list by the higher probability, and we take the first element, which is the **predicted topic**.

b. **Mapper** Topic-to-Documents
   i. We use the same logic of step 8, treating as unseen phrase the full file.
   ii. The output will be the predicted topic for the given file content.
   iii. We repeat the call for each file, and we add the filename to the topic predicted by the model, building a **mapper** with the structure *<topicId, List<fileName>>*

(7) Final prediction – An unseen phrase is given for prediction

(1) We call the prediction function of step (6)a. which gives us the **predicted topic**.
(2) Using the MAPPER of step (6)b, we extract the **list of files** associated to that topic.

(8) Conclusion

Given a phrase we can retrieve all files talking about the same topic.

1 Introduction

This dissertation will address the problem when we want to categorise documents by their similarity. When we write a contract or any legal document, we may have the need to consult other similar documents created in the past and reuse, consult or compare the content.

LEOS (Legislation Editing Open Software) is an open-source software created and distributed by the European Commission for editing legal documents. Is shipped as a configured product and as such, can be configured for any document's template both in the private and public sector.

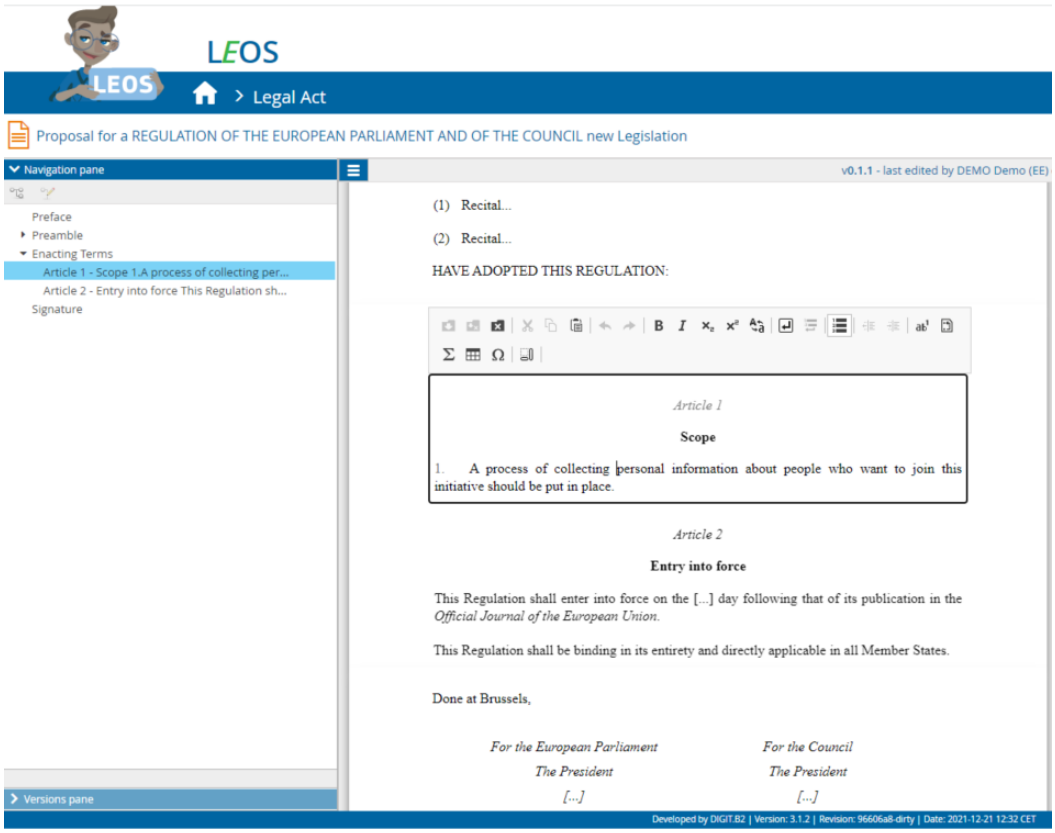
In this dissertation we are using the default configuration of LEOS for drafting legislations and the application is connected to EUR-Lex which provides the official access to EU legal documents. EUR-Lex contains all official publications for EU legal acts, published by the Publications Office of the EU also called the ***Official Journal*** of European Union.

A concrete example when using LEOS for drafting a new legislation would be, while writing the phrase *"Export IT service outside EU"*, the model lists all the laws similar to that ***topic***. For example, could be *Reg/2016/111* that talks about exporting, *Reg/2018/2001* which talks about funds or *Reg/2016/679* which talks about privacy (GDPR).

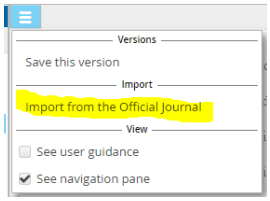
To achieve this goal, we are going to build an unsupervised machine learning model.

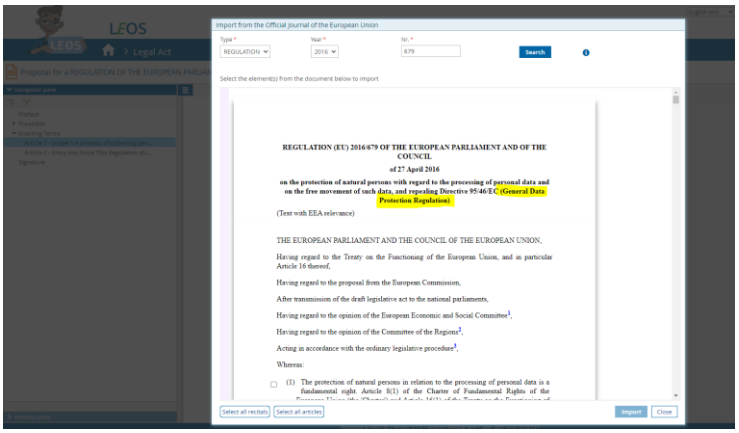
2 Business use-case

Example of creating a Regulation in LEOS. The new *Article* has been added through the *Table of Content* located in the left side of the screen and the content is being edited in the editor highlighted by the black square.

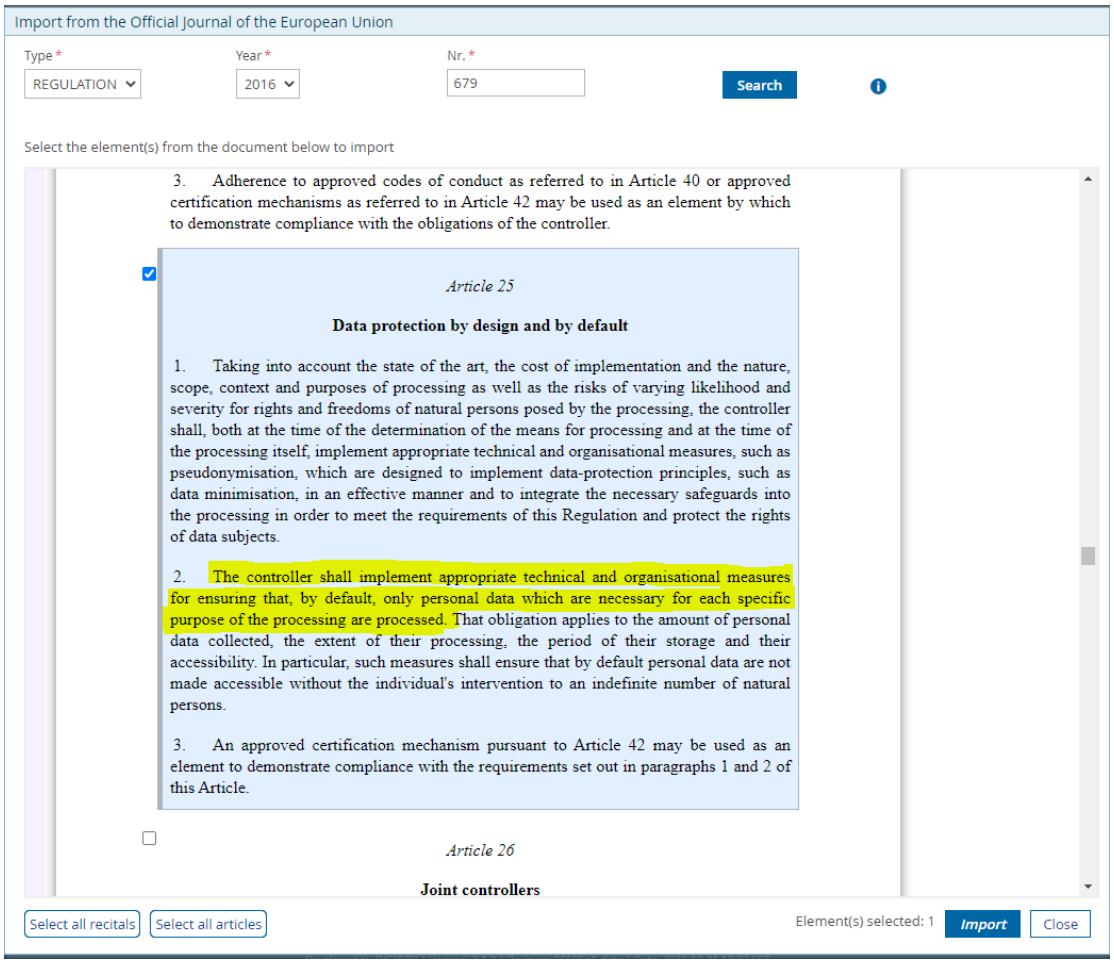


While a drafter is drafting a new article, he should know a priori old legislations talking about the same topic he is going to write. The user opens the OJ search form and searches for the regulation REG/2016/679.





The user scrolls the legislation and selects the Article he wants to import or consult.



First observation is that the user should know a priori all legislations talking about the same topic, ***Privacy***.

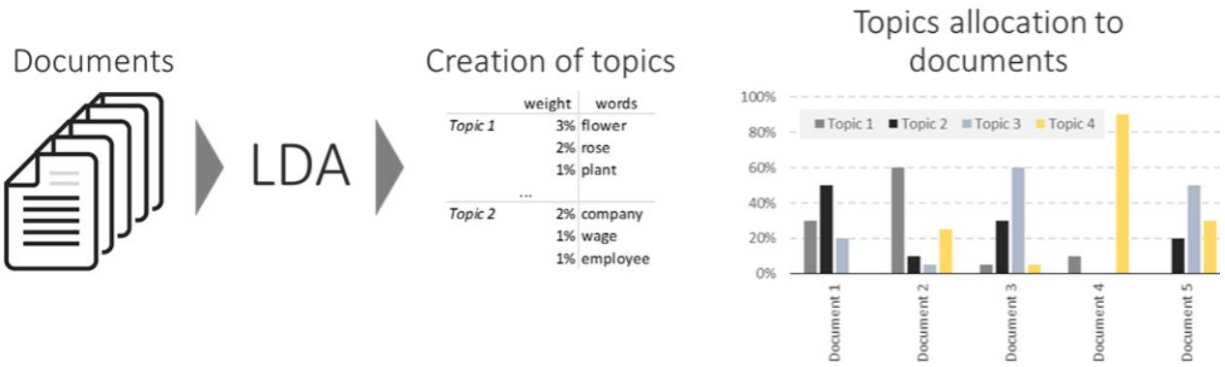
Would be ideal if while writing the phrase *"A process of collecting personal information about the people who want to join the initiative should be put in place"* the system understands that we are talking about Privacy, so lists all the legislations present in OJ talking about Privacy. The user can choose GDPR legislation REG/2013/679 or any less ranked document.

3 Problem description

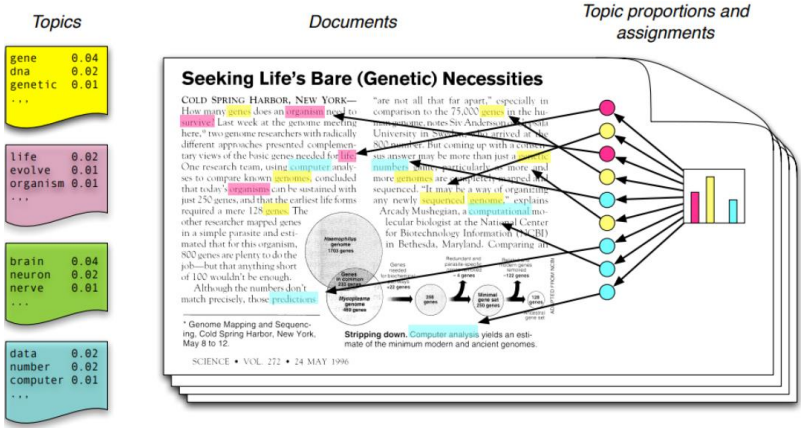
To solve this problem, we will use *Natural Language Processing* (NPL) and the *Latent Dirichlet Allocation* (LDA) technique to cluster documents by the *topics* they talk about and find similarities between them. Also, we need a way to go back from a *topic* to the *document* and for this we are going to use a map which relates a topic with a list of documents talking about that topic.

3.1 LDA - Latent Dirichlet Allocation

LDA (Latent Dirichlet Allocation) is an *unsupervised machine-learning* model that takes documents as input and finds topics as output. The model also says in what percentage each document talks about each topic.



In LDA each document is a distribution of topics, and each topic is a distribution of words.



- Each **topic** is a distribution over words
- Each **document** is a mixture of corpus-wide topics
- Each **word** is drawn from one of those topics

4 Methodology and result

4.1 Dataset

For simplicity all legislations/laws are downloaded in local, and each law is present in a different file. During the download from EUR-Lex, the file is converted in *XML AkomaNtoso* standard which is the standard used from LEOS. For this purpose, is used the open-source application *eur-lex-official-journal-sparql* created as part of this analysis.

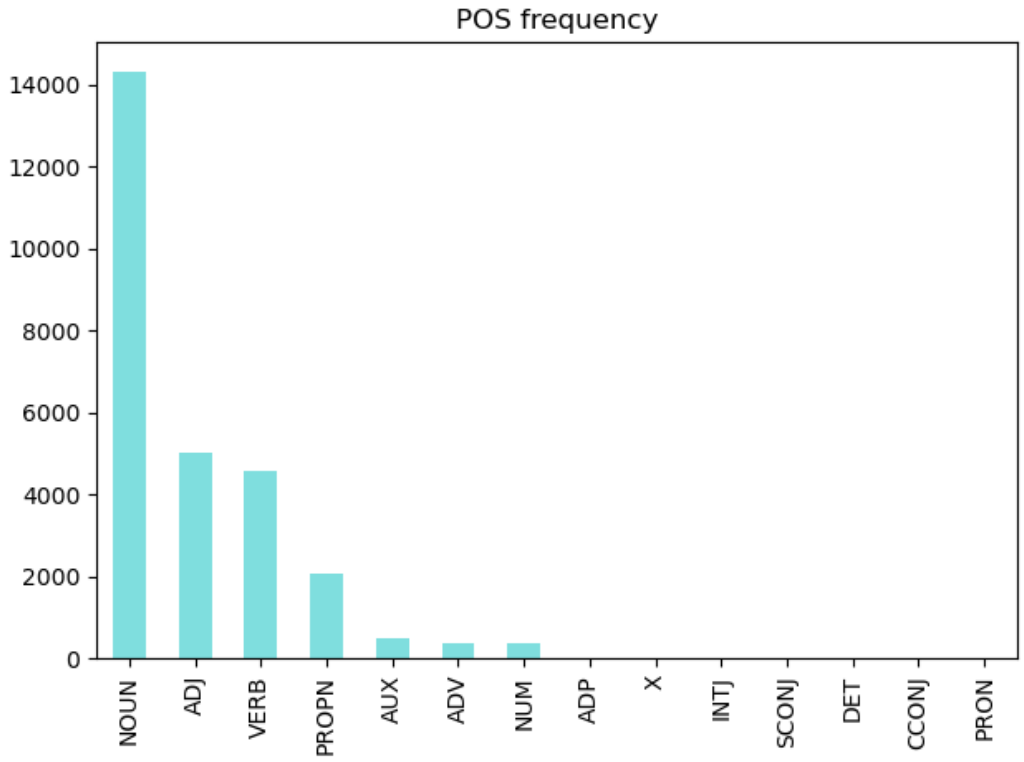
4.1.1 Understanding the dataset

The data set will be a local folder containing 1.585 regulations of the last 6 years.

Year	Nr of Regulations
2016	301
2017	284
2018	278
2019	243
2020	207
2021	272

4.1.2 Analysing a single Legislation

Let's see the Part-of-speech tagging for GDPR regulation.



The plot shows the most POS used are: "NOUN", "ADJ", "VERB", "ADV".
This info can be used during the ***Lemmatization*** of the document content to reduce dimensionality of the dictionary for topic modelling.

Choosing what POS to keep or remove depends on the use case under analysis. In the legal environment numbers can be important as they may refer to other legislations, example reg/2020/1503. In our case we decided to keep the NUM but remove lemmas long only 1 character.

Another technique can be used to further reduce the dimensionality of the dictionary by counting all the lemmas present in the document and using the quartile exclude the ones that do not appear frequently. This is not in the implementation.

4.2 Pre-processing the data

The following steps are applied for cleaning the data before their usage.

- Xml files containing regulations are read from the filesystem and with the usage of the python library *xml.etree.ElementTree* xml tags are removed, and the clean content is kept in memory for processing. A single document containing a regulation, is represented by a big string containing all the content as a single phrase.
- Stop words are removed.
- Words less than three characters are removed.
- Less informative lemmas, not in category nouns, verbs, adjectives, and adverbs are removed.

We decided to not apply the following steps:

- We could further reduce the dimensionality of the dictionary by removing words which have low frequency and very high frequency using the *quantile* but has been decided to not do so for the sake of time.
- Lemmas representing numbers are kept since can be relevant in regulations context.

4.3 Building the model

4.3.1 High Level overview

(1) A list with **all file** names and its content is built. The list will contain tuples *<FileName, FileContent>*
(2) The content is cleaned from xml tags and **pre-processed**.
(3) **Dictionary** with all the words present in *all files* is created.
(4) **Bag of Words** is created for each file using the *dictionary*. A BoW is a list of tuples *<tokenId, tokenCount>*, expressing how many times a word is present in other documents.
(5) **Corpus Bag of Words** is built putting all *BoWs* together.
(6) **LDA model** with the *Dictionary* and *Corpus BoW* is built.
(7) The topics calculated by the LDA model are printed. Each topic is a set of words followed by a probability. Distribution of topics and their words is also visualized in a chart.
(8) **Prediction function** - An unseen phrase is given for prediction.
    a. Using the LDA model's *Dictionary*, we calculate the BoW for the unseen phrase.
    b. We ask the *LDA model* to calculate the relation the new BoW has with each topic.
    c. A list of tuples *<topicId, probability>* will be returned.
    d. We rank the list by the higher probability, and we take the first element, which is the **predicted topic**.
(9) **MAPPER Topic – List of Documents** – All files are given for prediction
    a. We use the same logic of step 8, treating as unseen phrase the full file.
    b. The output will be the predicted topic for the given file content.
    c. We repeat the call for each file, and we add the filename to the topic predicted by the model, building a **mapper** with the structure *<topicId, List<fileName>>*
(10) **Final prediction** – An unseen phrase is given for prediction
    a. We call the logic of *step 8* which gives us the **predicted topic**.
    b. Using the MAPPER built in *step 9*, we extract the **list of files** associated to that topic.

4.3.2 Steps illustrated with some code

(1) Load all documents from filesystem, cleans the xml tags, lemmatize the content, and create the array "*documents*" as follows:

```
[
    [fileName1, fileContent1],
    [fileName1, fileContent2], ...
]
```

(2) Build **LdaModel** with the following data:

```
documents_words = documents[:, 1]
id2word = corpora.Dictionary(documents_words)
bow_corpus = [id2word.doc2bow(document) for document in documents_words]
lda_model = LdaModel(corpus=bow_corpus, id2word=id2word)
```

- *documents_words* List with list of words present in each *document*. All campus words
- *id2word* - **Dictionary** with all the corpus words present in *all files*.

- *bow_corpus* List with corpus BoWs. A BoW is a list of tuples *<tokenId, tokenCount>,* expressing how many times a word is present in other documents.
- *lda_model* - The LDA model will calculate and build the list of topics present in all documents.

(3) **Prediction function** - An unseen phrase is given for prediction

```
def predictTopic(lda_model, unseen_phrase):
    id2word = lda_model.id2word
    unseen_bow = id2word.doc2bow(unseen_phrase)
    predicted_topics = sorted(lda_model[unseen_bow])
    return predicted_topics[0][0]
```

a. Using the LDA model's *Dictionary*, we calculate the BoW for the unseen phrase.
b. We ask the *LDA model* to calculate the relation the new BoW has with each topic.
c. A list of tuples *<topicId, probability>* will be returned.
d. We rake the list by the higher probability, and we take the first element, which is the **predicted topic**.

(4) **MAPPER Topic – List of Documents** – All files are given for prediction.
We treat the full file as an unseen phrase, and we predict the file's topic using the prediction function of step 3. We repeat the process for each file adding its filename to the predicted topic, building a *topic_to_document* mapper:

```
topic_to_documents = defaultdict(list)
for doc in documents:
    prediction = predictTopic(lda_model, doc[1])
    topic_to_documents[prediction].append(doc[0])
```

The mapper will have the following structure:

```
[
    [topic0, [docName1, docName2, docName3]]
    [topic1, [docName4]]
    ...
]
```

(5) **Final Prediction** – An unseen phrase is given for prediction

```
unseen_phrase = 'In order to ensure a consistent level of protection for
natural persons ....'

predictedTopic = predictTopic(lda_model, unseen_phrase.split())

print("unseen phrase: ", unseen_phrase)
print("Predicted Topic", predictedTopic, "-",
lda_model.print_topic(predictedTopic))
print("Documents containing Topic", predictedTopic, "-",
topic_to_documents[predictedTopic])
```

Predicted output:

```
Predicted Topic 4 - 0.072*"datum" + 0.034*"personal" + 0.028*"processing" +
0.026*"subject" + 0.024*"authority" + 0.021*"supervisory" + 0.020*"controller" +
0.014*"protection" + 0.013*"right" + 0.013*"purpose"

Documents containing Topic 4 - ['reg_2016_679_akn_nr119seq0001.xml']
```

4.3.3 Dictionary class

Dictionary create a mapping between words and their integers ids.
Some Dictionary useful methods:

- Dictionary(documents): List of documents
- *token2id(token):* given a token, return its id.
- *cfs(tokenId):* Collection frequencies. How many instances of this token are contained in the documents.
- *dfs(tokenId):* Document frequencies. How many documents contain this token.
- *docToBow(phrase):* Creates *bag-of-words* for the given phrase. A BoW is a list of tuples<*token_id, token_count*> expressing how many times a word is present in other documents. Example:

```
bow_corpus[0][0:20]:  [(0, 1), (1, 1), (2, 3), (3, 7),. . .]
```

  - Word with id 0, found 1 time in the document
  - Word with ID 1, found 1 time in the document
  - Word with ID 2, found 3 time in the document
  - Word with ID 3, found 7 time in the document

4.3.4 LDAModel class

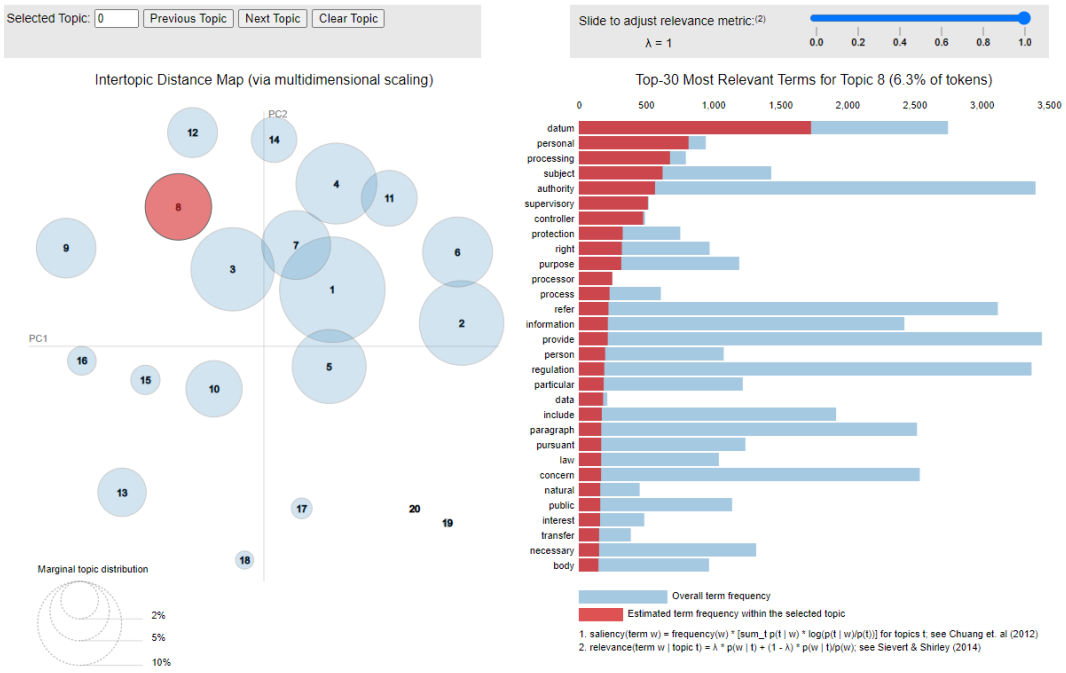
This module allows both LDA model estimation from a training corpus and inference of topic distribution on new, unseen documents. The model can also be updated with new documents for online training.

```
common_dictionary = Dictionary(common_texts)
common_corpus = [common_dictionary.doc2bow(text) for text in common_texts
# Train the model on the corpus.
lda = LdaModel(common_corpus, id2word, num_topics=10)
```

- **corpus** (iterable of list of (int, float), optional) – Stream of document vectors or sparse matrix of shape (num_documents, num_terms). If you have a CSC in-memory matrix, you can convert it to a streamed corpus with the help of gensim.matutils.Sparse2Corpus. If not given, the model is left untrained (presumably because you want to call update() manually).
- **num_topics** (int, optional) – The number of requested latent topics to be extracted from the training corpus.
- **id2word** ({dict of (int, str), gensim.corpora.dictionary.Dictionary}) – Mapping from word IDs to words. It is used to determine the vocabulary size, as well as for debugging and topic printing.

4.4 Topics distribution

Using *pyLDAvis.gensim_models* we can see the distributions of 20 topics for legislations of the year 2016.



Topic 8 in the image is the one talking about “datum”, “personal”, “processing”, “subject”, “authority”, “supervisory”, etc. Pretty like Privacy topic, so GDPR.
The word “datum” has been found around 2.750 (in light blue) times in all documents and 1.750 (in red) times is related to this topic.

An index for model accuracy is to observe the intersection of the circles (topics). If the topics are too much over each other’s means, topics share a lot of shared words so the prediction will not be accurate enough.

5 Conclusion and recommendations

First conclusion we have is that our trained model is able to predict the topic that an unseen phrase is talking about. Also, the model can extract the list of documents talking about the same topic.

Build a more accurate model

During this implementation we understood this is not the most accurate approach because we are associating only one topic to a regulation, but often regulations talk about different topics. Rather than choosing the most ranked topic for a document, and for the unseen phrase, would be better if we build a matrix with the probabilities as follow:

Document nr.	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9
Document 1	0,07	0,06	0,076	0,068	0,072	0,32	0,066	0,064	0,065	0,064
Document 2	0,303	0,009	0,013	0,123	0,139	0,206	0,011	0,578	0,007	0,201
Document 3	0,561	0,001	0,001	0,002	0,002	0,002	0,407	0,023	0,002	0,001
Document 4	0,564	0,001	0,42	0,001	0,002	0,123	0,196	0,06	0,007	0,007
Document 5	0,001	0,012	0,179	0,219	0,005	0,011	0,007	0,023	0,101	0,007
Document 6	0,134	0,012	0,001	0,345	0,56	0,104	0,003	0,007	0,012	0,104
...										
Unseen Phrase	0,001	0,009	0,232	0,123	0,08	0,231	0,001	0,543	0,105	0,005

This way we know that our unseen phrase is talking about 3 different topics: *Topic2*, *Topic5* and *Topic7* and when it's time to extract, we extract Document 4 and 5 for Topic2; Document 1 and 2 for Topic5; and Document 2 for Topic7. For time limitation this solution is not implemented.

Topics accuracy

Fine-tuning LDA is not an easy task. Number of topics are set manually so you need to see all topics to know if your model makes sense or not. Testing different pre-processing methods interactively for improving the topics can be done. For this analysis the number of latent topics is set to 20 but this also can be fine-tuned interactively.

5.1 Machine learning model solving the business use-case

All started with finding a “smart way” to extract documents while drafting legislation, or any type of legal document, in LEOS.

The machine-learning model we built can be considered a first version for solving this use-case. Later can be improved and re-trained with more data.

For European Commission usage, or any agency that drafts legislations, the ideal would be to train the model with all the regulations present in the Official Journal since year 1953.

Nevertheless, LEOS is not used only by the European Commission. As an open source and fully configurable, it can serve different business cases. Courts, Statal Agencies, or any other institutions which handle legal documents can use LEOS. In this case the machine learning model should be trained with their data of interest. Example in case of a Court, the model will be trained with old sentences and/or laws.

Also, any private sector who handles legal documents can use LEOS. A case could be a *Renewable Energy Company* which handles a lot of contracts. In this case the machine learning model will be trained with old contracts. In case the company is interested in the international laws, only that specific category of laws will be extracted from EUR-Lex for training the model.

From the LEOS application screen, while the user is drafting the document, after he inserts a reasonable number of words, the system will call the model and will list in the screen the predicted legislations. This process should be fired each time a new word/phrase is added in the editor because the semantic context can change, hence the topic to be extracted.

6 Bibliography

- Python project with LDA Model explained in this study: https://github.com/ermalaliraj/python-lda-topic-modeling-ec-laws
- Open-source application used for downloading regulations form EUR-Lex. https://github.com/ermalaliraj/eur-lex-official-journal-sparql
- LEOS, Open-source web application for drafting any legal document: https://joinup.ec.europa.eu/collection/justice-law-and-security/solution/leos-open-source-software-editing-legislation
- EUR-Lex, the portal of European Commission: https://eur-lex.europa.eu/homepage.html
- Topic Modelling by Félix Revert: https://towardsdatascience.com/the-complete-guide-for-topics-extraction-in-python-a6aaa6cedbbc
- Topic Modelling by Susan Li: https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
- Dictionary and BoW: https://radimrehurek.com/gensim/corpora/dictionary.html

DR. SEPPE VANDEN BROUCKE: SEPPE.VANDENBROUCKE@KULEUVEN.BE
https://feb.kuleuven.be/permanente-vorming/bigdataanalytics