Internship: Project

Topic Modelling/Clustering and Labelling

Yorick Estievenart

Academic year 2022-2023
Computer Science
October 18, 2023

UMONS
Université de Mons

sinch

Contents

1 Introduction

Machine learning models require more and more data to be trained, particularly in the field of Natural Language Processing (NLP). Manually labelling data takes up a considerable amount of time that could be devoted to a more useful task. One way of tackling this problem is to use Topic Modelling/Clustering to help us distribute the data into clusters and then label the data using these clusters as classes [1].

- **Topic Modeling** is defined as a statistical and probabilistic technique used to automatically identify topics or themes in a collection of documents.
- **Topic Clustering** is defined as a text analysis technique that enables similar documents or textual data to be grouped together according to their content.

This document will be used to answer the following two questions: How can we correctly group a set of documents (i.e. a corpus of text) by subject? And how can this help us label a given dataset? It will be divided in two parts. First, we will focus on the scenario when the input data are documents and classes (represented by keywords) into which we want to classify the documents. The second scenario is when the input data are only documents. We will need to extract classes for each group of documents.

[1] Code related to this document can be found at this url. Each section is represented by a separate folder.

2 Datasets

Before searching and applying models to try to solve the problems of labelling data, we need to have data. Table 2 shows the following benchmark datasets that will be used during the project. These datasets have been extracted using the OCTIS library and preprocess by rewriting manually each ambiguous class name by a more interpretable one. The manually extracted scores can be found in the next section.

Name	**# Docs**	**# Train/Test Docs**	**# Classes**
BBC_News	2225	1780/445	5
20NewsGroup	16309	13047/3262	20
DBLP	54595	43676/10919	4
M10	8355	6684/1671	10

2.1 Keywords

Keywords used for each class of each datasets can be seen in Tables 2 and 1.

Topic	**Topic Index**	**Keywords**
rec.motorcycles	0	motorcycles
sci.space	1	space
talk.religion.misc	2	religion
comp.graphics	3	computer graphics
soc.religion.christian	4	christianity
comp.sys.mac.hardware	5	mac hardware
rec.sport.hockey	6	hockey
rec.sport.baseball	7	baseball
comp.sys.ibm.pc.hardware	8	pc hardware
misc.forsale	9	for sale
rec.autos	10	cars
sci.electronics	11	electronics
comp.os.ms-windows.misc	12	windows
sci.med	13	medicine
sci.crypt	14	cryptography
talk.politics.mideast	15	middle east
comp.windows.x	16	windows x
alt.atheism	17	atheism
talk.politics.misc	18	politics
talk.politics.guns	19	guns

Topic	**Topic Index**	**Keywords**
2	0	archaeology
3	1	computer science
4	2	financial economics
0	3	agriculture
7	4	petroleum chemistry
9	5	social science
1	6	biology
8	7	physics
6	8	material science
5	9	industrial engineering

Table 1: Keywords for the 20NewsGroup dataset (left) and the M10 dataset (right).

Topic	Topic Index	Keywords
business	0	business
sport	1	sport
entertainment	2	entertainment
tech	3	technology
politics	4	politics

Topic	Topic Index	Keywords
2	0	computer vision
0	1	database
3	2	data mining
1	3	artificial intelligence

Table 2: Keywords for the BBC News dataset (left) and the DBLP dataset (right).

2.2 Distribution of classes

Distribution of classes for the datasets can be seen on Figure 2.2. It can be seen that the DBLP and M10 datasets have more constant documents lengths than the 20NewsGroup and the BBC News datasets. In addition, the BBC News and DBLP datasets are maybe more appropriate for testing since that have a fewer number of classes. We would mainly focus on the BBC News dataset for our primary experiments and then compare the results on multiple datasets.

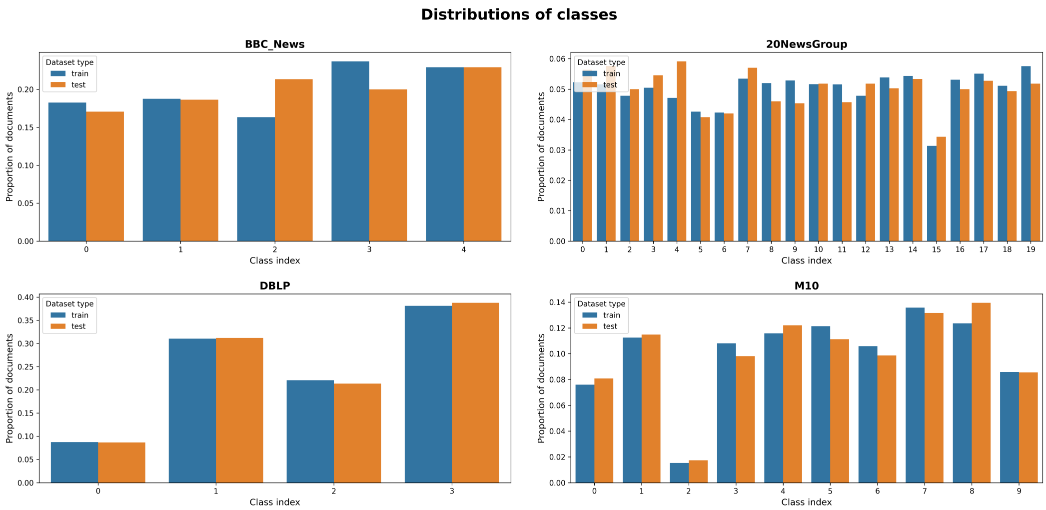


Figure 1: Distribution of the classes for the datasets.

2.3 Distribution of documents lengths

Distribution of documents lengths for the datasets can be seen on Figure 2.3. The DBLP and M10 datasets shows shorter documents, which can potentially be harder to classify.

2.4 Most frequent words

The 10 most frequent words for each datasets can be seen on Figure 2.4.

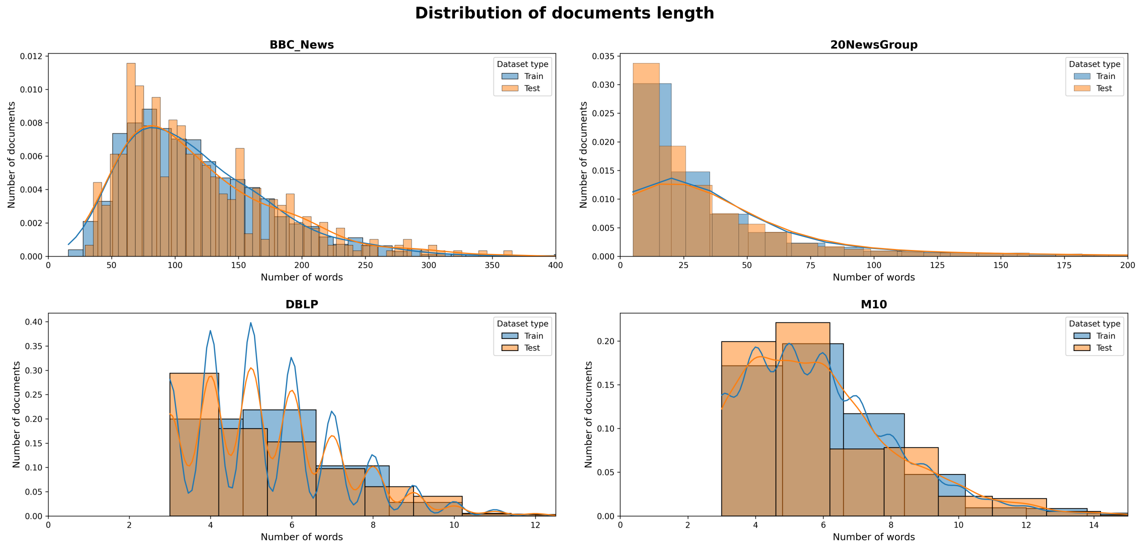


Figure 2: Distribution of the documents length for the datasets.

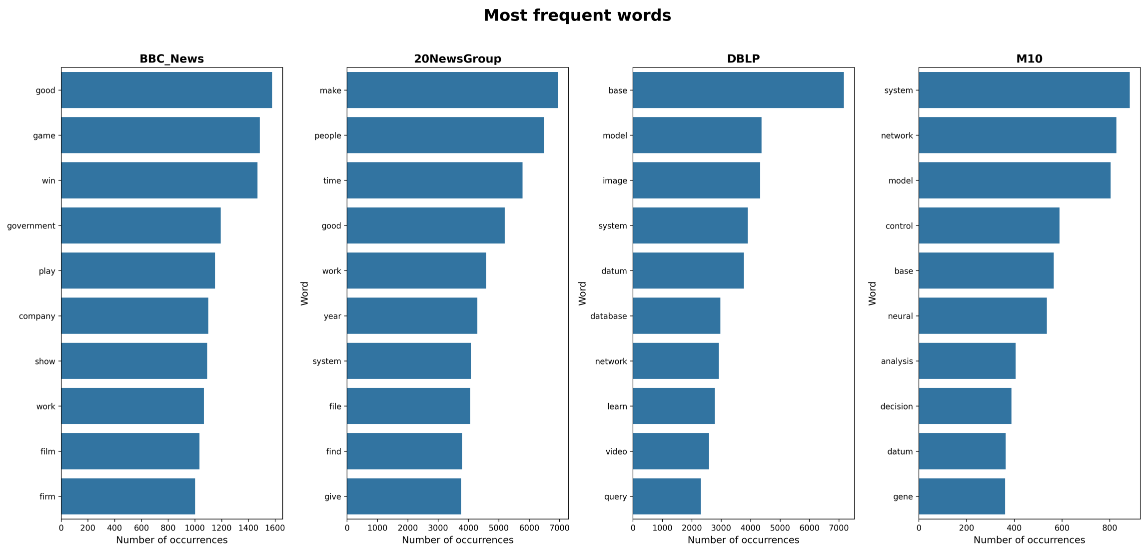


Figure 3: Most frequent words used in each datasets.

2.5 Distribution of words occurences

Distribution of words occurences for the datasets can be seen on Figure 2.5. The general rule is that most words don't appear very often. In addition, with the exception of dataset M10, no dataset contains words that appear only 2 or 3 times. This can be explained be the fact that the dataset has already been processed and therefore it contains no stopwords or punctations.

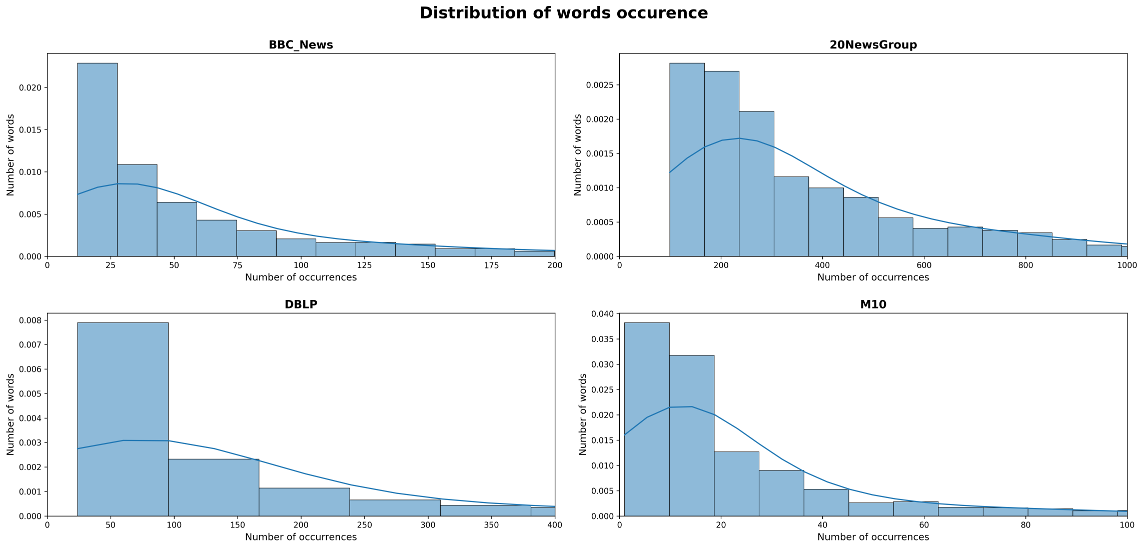


Figure 4: Distribution of the occurences of words for each datasets.

3 Full inputs scenario

This section deals with the scenario in which we receive not only input documents, but also the classes into which we want to classify the documents. Each class is represented by the keywords defined in Section 2.1 but depending on the sithuation, these can be changed and adapted for the analysis. In this section, we will investigated the model Lbl2Vec and its variants that use transformers.

3.1 Lbl2Vec

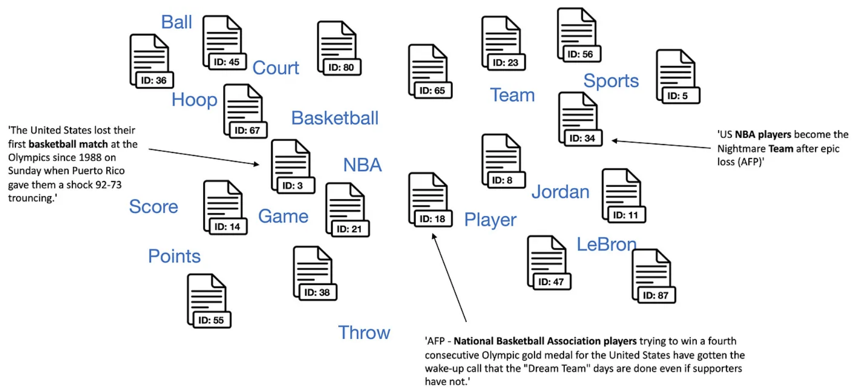
3.1.1 Theory

Lbl2Vec [1] is an unsupervised algorithm to classify documents in defined classes based on a predefined knowledge, which is the keywords associated with each classes. At a high level, this algorithm performs the following tasks:

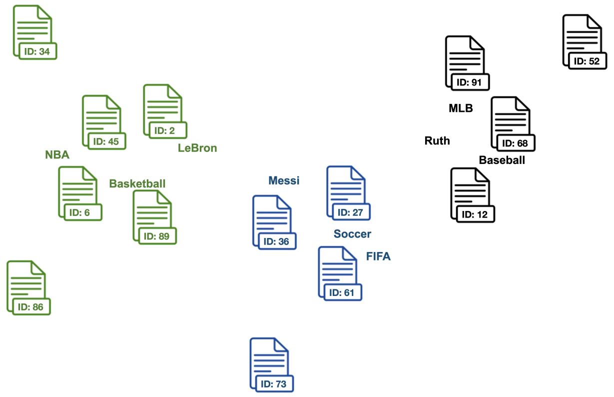
1. Use manually defined keywords for each category of interest.

Basketball	Soccer	Baseball
NBA	FIFA	MLB
Basketball	Soccer	Baseball
LeBron	Messi	Ruth
...	...	...

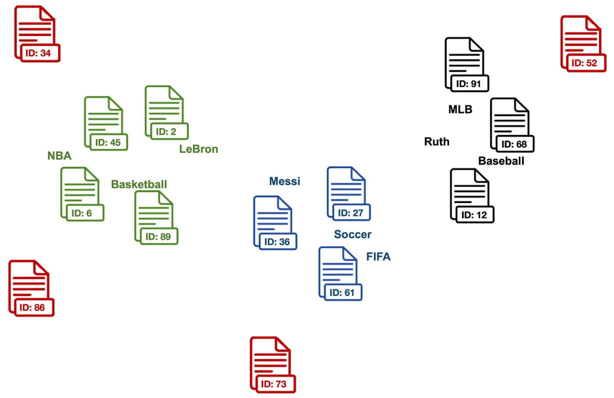
2. Create jointly embedded document and word vectors. The idea behind embedding vectors is that similar words or text documents will have similar vectors. Therefore, after creating jointly embedded vectors, documents are located close to other similar documents and close to the most distinguishing words.



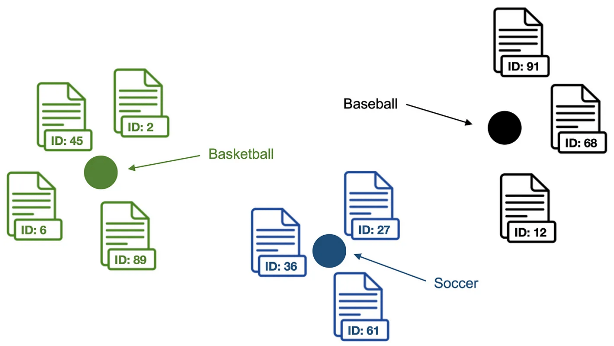
3. Find document vectors that are similar to the keyword vectors of each classification category. We compute cosine similarities between documents and the manually defined keywords of each category. Documents that are similar to category keywords are assigned to a set of candidate documents of the respective category.



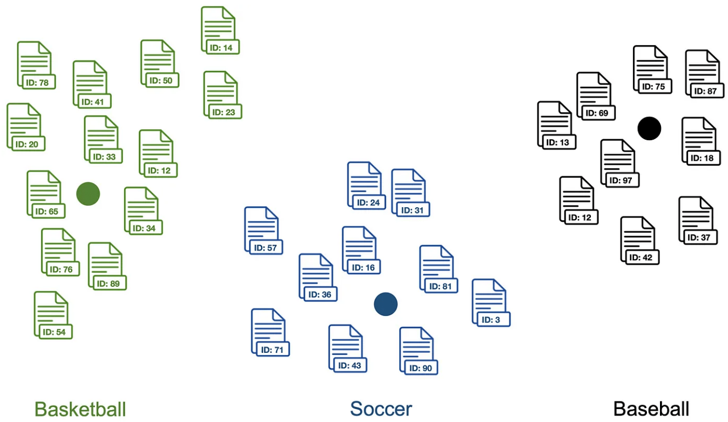
4. Clean outlier documents for each classification category using an algorithm called Local Outlier Factor (LOF).



5. Compute the centroid of the outlier cleaned document vectors as label vector for each classification category.



6. The algorithm computes label vector ⇔ document vector similarities for each label vector and document vector in the dataset. Finally, text documents are classified as category with the highest label vector ⇔ document vector similarity.



Remark: the original paper provides a library to use efficiently the models. But two problems arises. First, we have no freedom on the inner model. We cannot set a random state to the inner model, and it is impossible to extract the embeddings created during the inference. Second, it produces sometimes weird results. For example, it has a high F1 score on the BBC News dataset with Doc2Vec as embeddings, while having very bad F1 score with a transformer embedding. In order to solve these problems, we decided to rewrite the model from scratch (you can find it on the Github repository as DaCluDeK (Data Clustering with Defined Keywords)).

3.1.2 Embeddings models

Two types of embeddings models can be used. One using Doc2Vec and one using a transformer-based models. Avantages and disadvantages of these two architecture can be seen in Table 3. Both models have been used and compared in this section, usually in a 2D settings for visualisation purposes. Doc2Vec can directly generate 2D embeddings, but this is not the case with transformer-based models. To solve this issue, we use a dimensionality reduction method called Principal Compoment analysis (PCA). The transformer-based model used is *all-MiniLM-L6-v2*.

Doc2Vec		**Transformer-based**	
Advantages	**Disadvantages**	**Advantages**	**Disadvantages**
Fast to compute	Can only use keywords seen during training Bad results Random	Can use any keywords Better embeddings leading to better results	Computationally expensive

Table 3: Advantages and disadvantages of the Doc2Vec architecture compared to transformer-based models.

3.1.3 Doc2Vec - Unknown keywords problem

One important problem with the use of Doc2Vec-based models is that it can only use keywords seen during training. This can be solved in an unsupervised way using the similarity between two words. Knowing that, when using Doc2Vec-based models, instead of using the keywords of the datasets, we will compute the similarity between the keywords and all the words contained in every documents. The keywords used will be the words mainly similar to the predefined keywords. To do this, the package *spaCy* will be used. Here is how similarity is computed in spaCy:

1. Word Vectors: Each word in the input text is represented as a high-dimensional vector in a continuous vector space.

2. Sentence or Document Vectors: spaCy combines the word vectors of the individual words in the sentence or document to create a vector representation for the entire sentence or document. This is typically done by taking the average or weighted average of the word vectors.
3. Cosine Similarity: Once sentence or document vectors are obtained, spaCy computes the similarity between them using cosine similarity. Cosine similarity measures the cosine of the angle between two vectors and provides a value between -1 (completely dissimilar) and 1 (perfectly similar).

3.1.4 Embeddings

Example of 2D embeddings create for Doc2Vec and a transformer-based model can be seen on Figure 3.1.4. It can be seen that the transformer-based model has a better ability to find embedding since they are more clustered. But we have to keep in mind that we are in a 2D setup and it is possible that Doc2Vec create also good embedding in greater dimension.

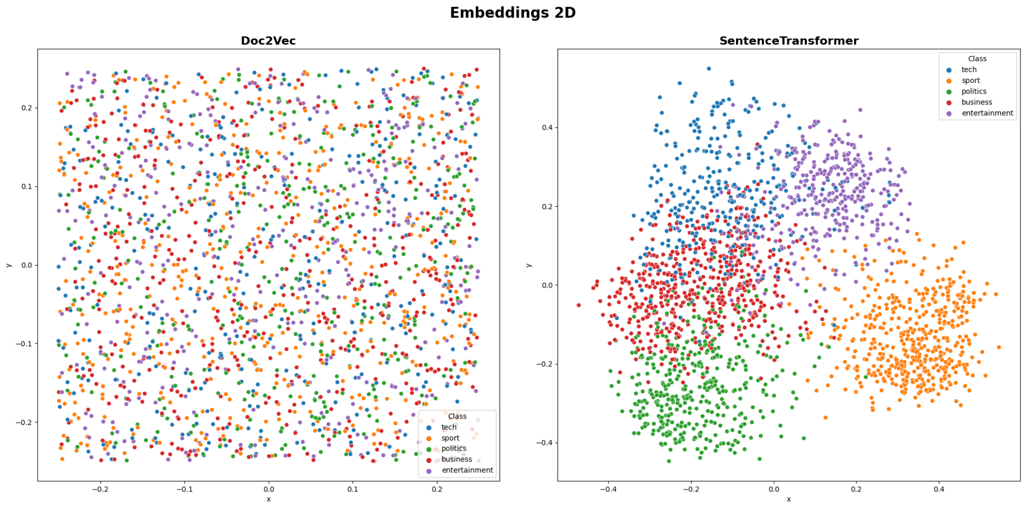


Figure 5: 2D embeddings generated by Doc2Vec and a transformer-based model on the BBC News dataset.

3.1.5 Performances using most frequent words

Performances of the two models on the BBC News dataset can be seen on Figure 3.1.5. The keywords used are the 5 most frequent words by class, and they have been extracted in a supervised way. They can be found in Table 4. The metric used is the F1 score. It can be seen that the transformer-based model presents better scores than Doc2Vec, but still very unbalanced. The score on the ”tech” class is way worse than the one obtained on the ”politics” class.

business	**entertainment**	**politics**	**sport**	**tech**
company	film	government	win	game
firm	good	election	game	technology
market	award	party	play	phone
rise	music	labour	player	mobile
sale	show	plan	good	service

Table 4: 5 most frequent keywords by class for the BBC News dataset.

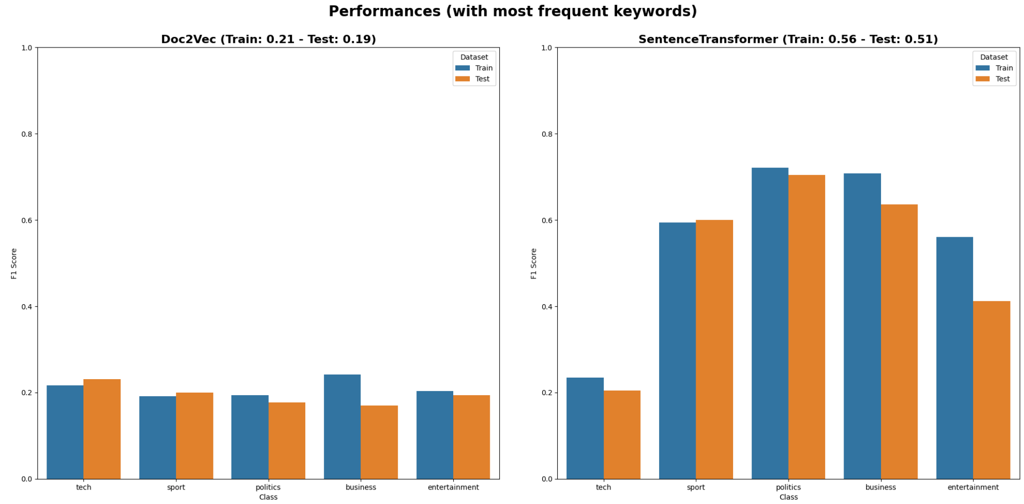


Figure 6: F1 score obtained after applying Doc2Vec and a transformer-based model on the BBC News dataset using the most frequent words of each class.

3.1.6 Performances using most similar words

The main problem with the previous result is that we retrieve the most frequent words by class in a supervised way. Nevertheless, our problem is related to unsupervised learning and we need to use another method. A solution is to use the keywords defined in Section 2.1 and to use the spaCy algorithm as explained in Section 3.1.3. The results can be seen in figure 3.1.6. We use the 5 most similar keywords to the predefined keywords, and they can be found in Table 5. The question is still: why do the F1 score is bad on the ”tech” class and how to solve it ?

business	**entertainment**	**politics**	**sport**	**tech**
business	entertainment	political	sport	technology
corporate	television	democracy	tennis	technological
company	theatre	democratic	rugby	innovation
marketing	gaming	debate	athletic	innovative
industry	multimedia	liberal	football	engineering

Table 5: 5 most similar keywords by class for the BBC News dataset.

3.1.7 Performances using most important words

To determine what could be the best keywords to use to improve the score, we use a supervised algorithm to extract the most important words. We applied a *TfidfVectorizer* to the documents before running a *Logistic Regression* model. The model predicts the classes with a F1 score of 0.99, which means that the model will be able to extract important words accurately. We extract the 100 most important keywords and then use them to predict the classes using our model. The results can be seen on Figure 3.1.7. What can be seen is that the scores are more balanced (the ”tech” class is not as good as the other), but the average score remains the same. The scores for Doc2Vec didn’t improved. This can mean that our unsupervised method to find the keywords is as good as an supervised method.

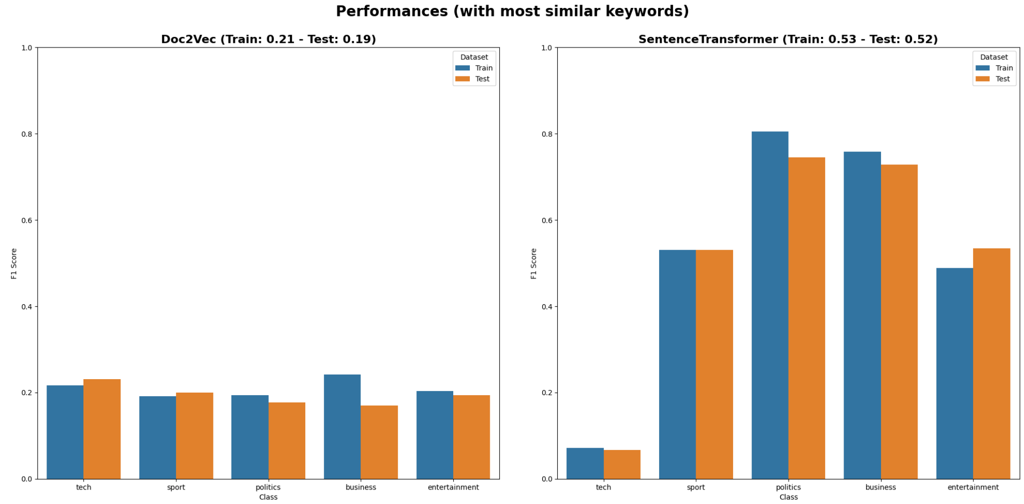


Figure 7: F1 score obtained after applying Doc2Vec and a transformer-based model on the BBC News dataset using the most similar words of each class.

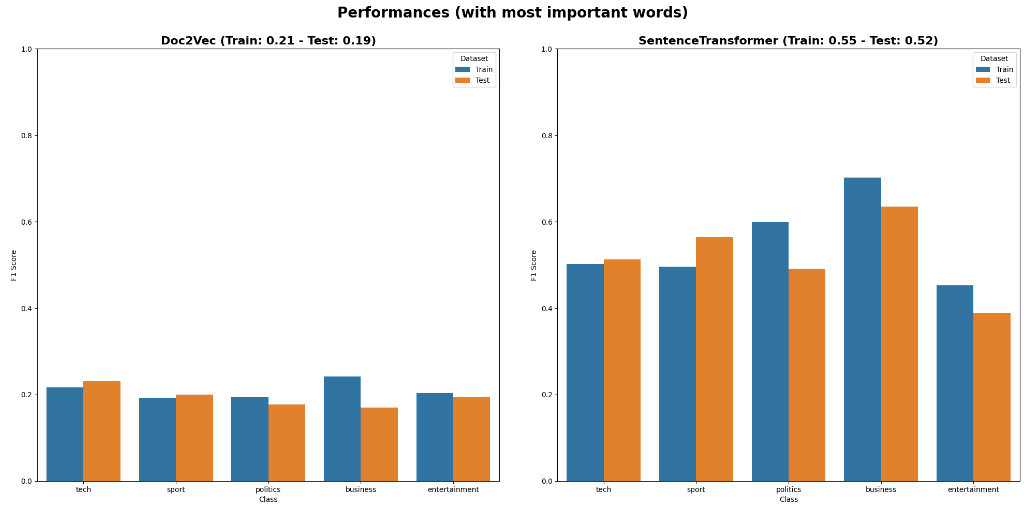


Figure 8: F1 score obtained after applying Doc2Vec and a transformer-based model on the BBC News dataset using the 100 most important words of each class.

3.1.8 General performances

As the previous observation suggest, we will see the performances of the models on each dataset using the method of most similar keywords, and by using the 10 most similar. The results can be seen on Figure 3.1.8. We also plotted the F1 score obtained with a random method i.e. by choosing randomly a class. We can see that Doc2Vec is often worst than this random classifier, which can be explained by its poor embedding performances. What is interesting is that the transformer-based model produce better results than the random classifier. It can be said that the results are quite good since we are in a unsupervised settings and that the number of classes are quite high. But compare to a supervised method, the results are still quite bad.

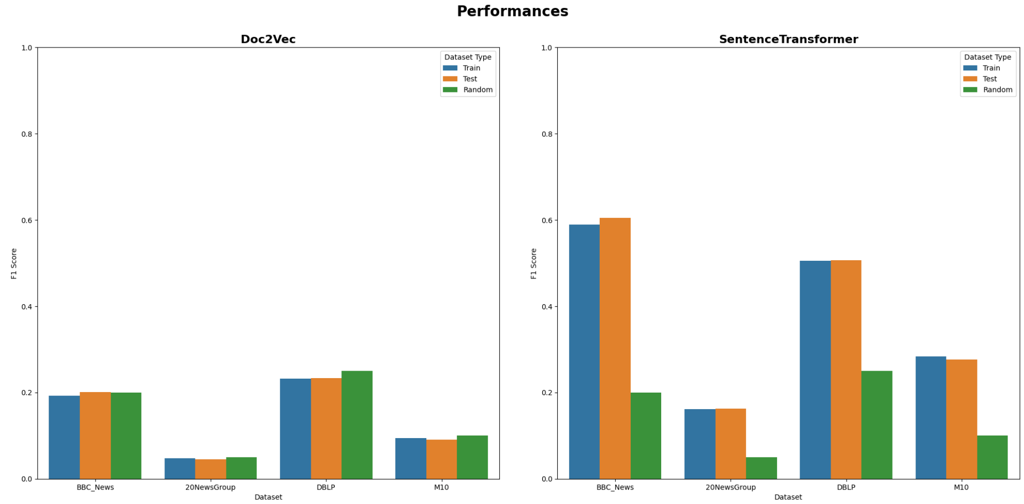


Figure 9: F1 scores obtained after applying Doc2Vec and a transformer-based model on all the datasets.

4 Reduced inputs scenario

This section deals with the scenario in which we receive only input documents, and not the classes as seen previously. This means that we also have to extract the different topics.

4.1 Evaluation

Before we start gathering information about main method used in this scenario. We need to know how to evaluate these models. As we are not in classification into classes, we cannot use basic metrics such as F1 score, accuracy, etc. The evaluation metrics should be related to the topics extracted by the model and/or their correlation with the true labels in the dataset. Without using these true labels, we can define two metrics:

- Topic Coherence (TC): assess how well a topic is 'supported' by a text set (called reference corpus).
- Topic Diversity (TD): defined as the percentage of unique words for all topics. The measure ranges from [0, 1] where 0 indicates redundant topics and 1 indicates more varied topics.

Now, to compare the correlation between the extract topics and the true labels of the datasets, we can use the top N words for each topics/labels and defined a metric that count the number of similar words and provide a value between 0 (totally different topic) and 1 (the same topic). This metric will be weighted by the number of each top words to take into account the number of words inside the documents. A simple example of this metric can be seen on Figure 4.1. This process will be done for each true labels for a given topic and the highest score will be kept since a topic is only related to one class and not every class. It will also be done for each topic and the final score correspond to the average of each individual score.

The main problem with this metric is that we don't take into account the fact that every highest score could be always related to the same class i.e. that all the extracted topics are associated to the same class. We make this assumption by taking the diversity metric into account. If the diversity metric is high, we can suppose that the topic are sufficiently diverse to be associated to different classes.

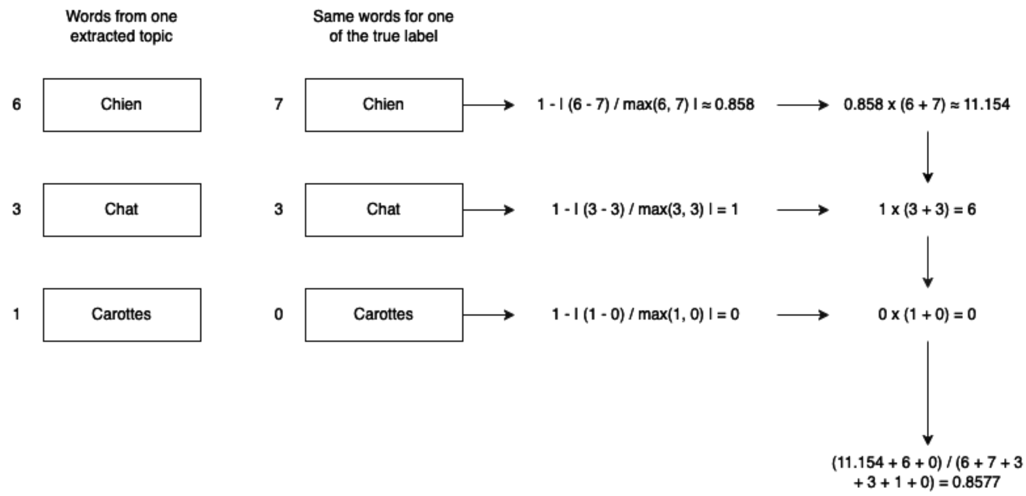


Figure 10: Example of the application of this weighted metric for one given topic and one given true labels.

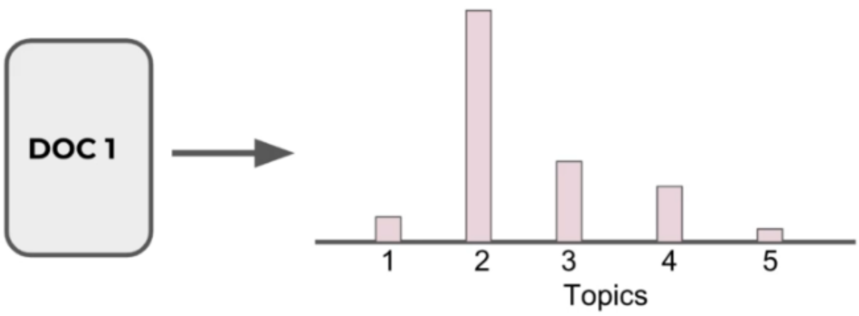
4.2 Latent Dirichlet Allocation (LDA)

4.2.1 Theory

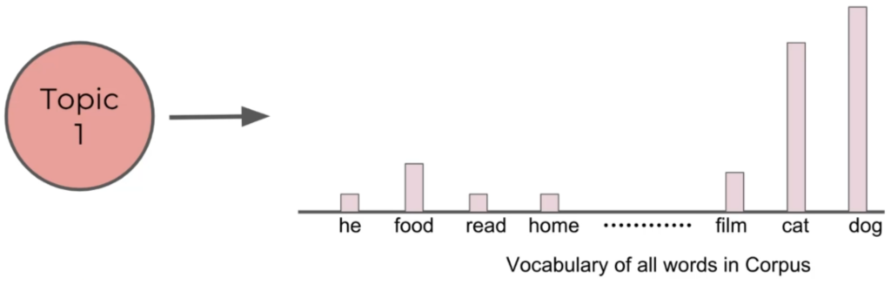
Assumptions

For this model to work, multiple assumptions are done:

- Each document is just a collection of words or a "bag of words" (BOW). Thus, the order of the words and the grammatical role of the words (subject, object, verbs, . . .) are not considered in the model.
- Documents with similar topics use similar groups of words.
- We can eliminate words that occur in at least 80%-90% of the documents, without losing any information.
- We need to specify beforehand the number of topics we want. This number is represented by the parameter K.
- Documents are probability distributions over latent topics which signifies certain document will contain more words of a specific topic.



- Topics themselves are probability distribution over words.



Algorithm

Once these assumptions have been made, the LDA algorithm works in the following way:

1. For each document, assign randomly each word of the document to one of the K topics.
2. For each document d and for each word w of the document, compute:
   - $\mathbb{P}(\text{topic t} \mid \text{document d})$: the proportion of words in document d that are assigned to topic t.
   - $\mathbb{P}(\text{word w} \mid \text{topic t})$: the proportion of assignments to topic t over all documents that come from this word w. Tries to capture how many documents are in topic t because of word w.
3. Update the probability for the word w belonging to topic t, as

$$\mathbb{P}(\text{word } w \text{ with topic } t) = \mathbb{P}(\text{topic } t \mid \text{document } d) * \mathbb{P}(\text{word } w \mid \text{topic } t) \tag{1}$$

4.3 BERTopic

4.3.1 Theory

BERTopic works in the following way:

1. Convert the documents into embeddings using any Sentence Transformer. The default transformer used is Sentence-BERT.
2. Reduce the dimensionality of the embeddings using the Uniform Manifold Approximation and Projection (UMAP) algorithm.
3. Group the reduced embeddings into clusters using an algorithm called HDBSCAN (Hierarchical DBSCAN).
4. Extract the topics from each clusters using their c-TF-IDF scores.
   - Term Frequency (TF): how frequently a term/word appears in a document, calculated as:

$$TF = \frac{\text{Number of times term appears in a document}}{\text{Total number of terms in the document}} \tag{2}$$

   - (Class-Based Term Frequency (CTF): for a term within a specific class or category is the number of times that term appears in documents belonging to that class. It measures how frequent a term is within documents of a particular class.)
   - Inverse Document Frequency (IDF): Inverse document frequency measures how unique or rare a term is across the entire corpus. The purpose of IDF is to give more weight to terms that are relatively rare in the corpus, as they are likely to carry more meaningful information. It is calculated as:

$$IDF = \log \frac{\text{Total number of documents in the corpus}}{\text{Number of documents containing the term}} \tag{3}$$

- TF-IDF Score: It quantifies how important a term is within a specific document while considering its rarity across the entire corpus. It is calculated as:

$$\text{TF-IDF} = TF * IDF \tag{4}$$

4.4 Comparison

5 Conclusion

References

[1] T. Schopf, D. Braun, and F. Matthes. Lbl2vec: An embedding-based approach for unsupervised document retrieval on predefined topics. In *Proceedings of the 17th International Conference on Web Information Systems and Technologies*. SCITEPRESS - Science and Technology Publications, 2021.