

Topic Modelling & Clustering - Observations

Yorick Estievenart

University of Mons

Overview

Introduction

Datasets

Full inputs scenario

Lbl2Vec

Reduced inputs scenario

TF-IDF + KMeans

TF-IDF

KMeans

Link between clusters and classes

Semi-supervised assignment

Keyword selection

Directions

Lbl2TransformerVec VS Lbl2Vec

Lbl2Vec V2

Motivation

Unclear class names

Introduction

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.

- ▶ **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.

(Code related to this document can be found at this url. Each section is represented by a separate folder.)

Introduction

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**:

- ▶ When the input data are documents and classes (represented by keywords) into which we want to classify the documents.
- ▶ When the input data are only documents. We'll need to extract classes for each group of documents.

NB: Class names are not relevant. Only the keywords associated with them are.

Datasets

Datasets

Before searching and applying models to try to solve the problems of labelling data, we need to have data. Table 7 shows the following benchmark datasets that will be used during the project.

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

Keywords

Keywords used for each class of the BBC News and 20NewsGroup datasets can be seen in Tables 1 and 2.

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

Table: BBC News

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

Table: 20NewsGroup

Keywords

Keywords used for each class of the DBLP and M10 datasets can be seen in Tables 3 and 4.

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

Table: DBLP

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: M10

Class distribution

Distribution of classes for the datasets can be seen on Figure 10. It can be seen that the DBLP and M10 datasets have **more constant documents lengths** than the 20NewsGroup and the BBC News datasets. Also, **the BBC News and DBLP datasets are maybe more appropriate for testing since that have a fewer number of classes.**

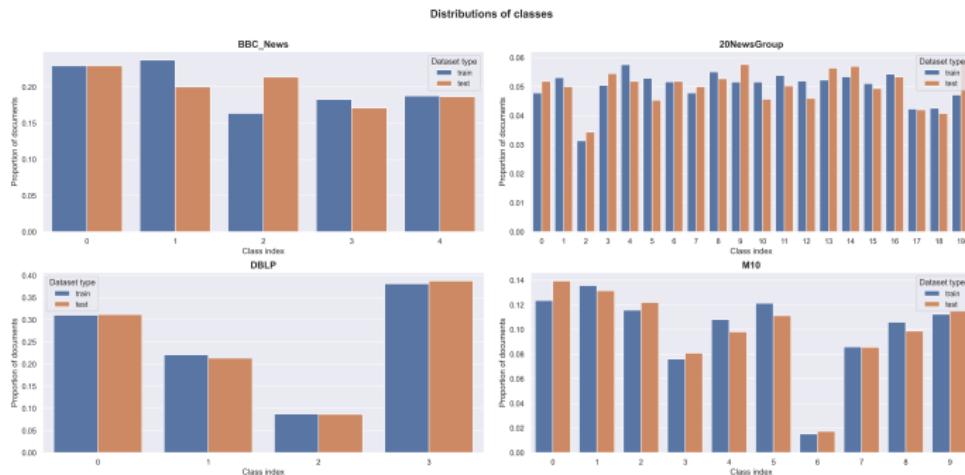


Figure: Distribution of the classes for the datasets.

Documents length distribution

Distribution of documents lengths for the datasets can be seen on Figure 11. The DBLP and M10 datasets shows shorter documents, which can potentially be harder to classify.

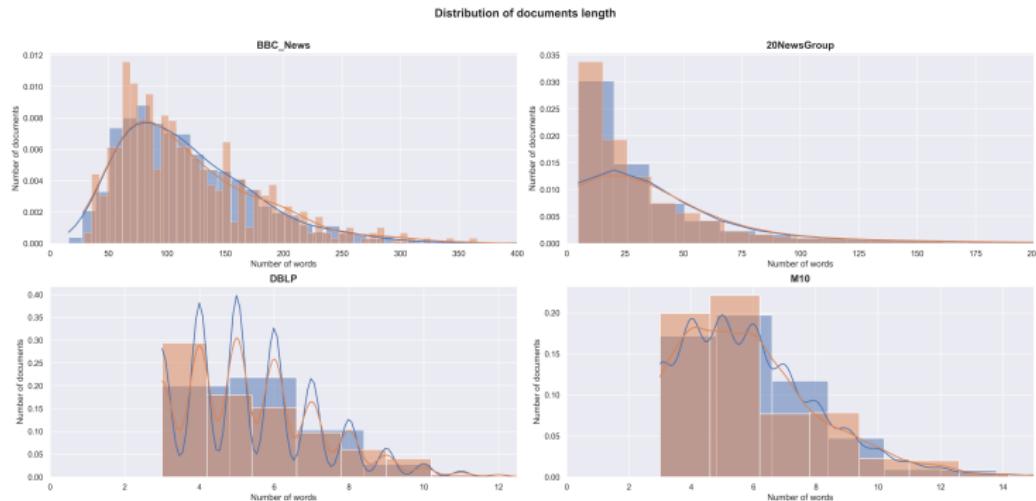


Figure: Distribution of the documents length for the datasets.

Most frequent words

The 15 most frequent words for each datasets can be seen on Figure 12.

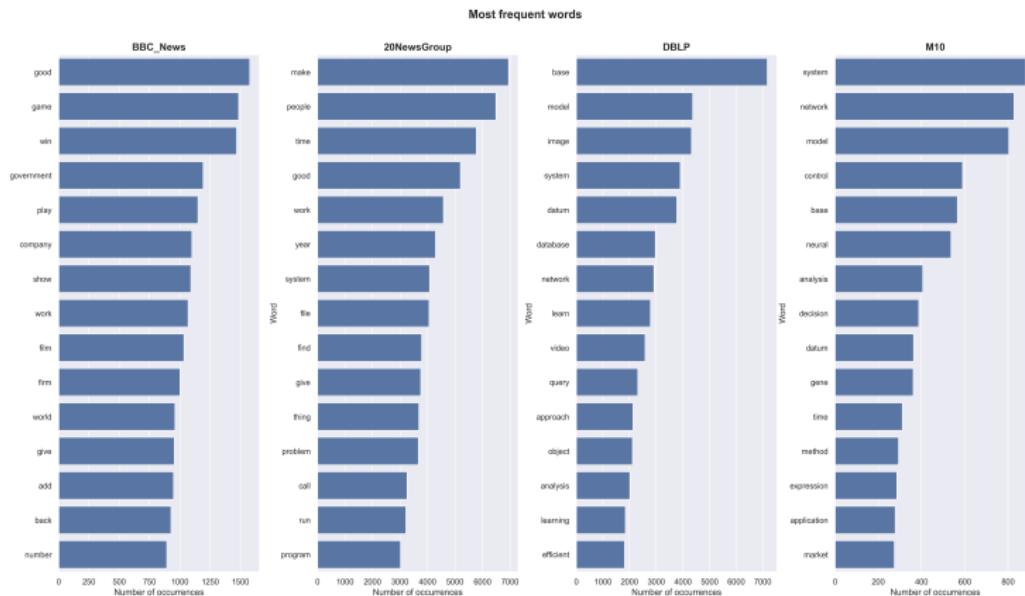


Figure: Most frequent words used in each datasets.

Distribution of words occurrences

Distribution of words occurrences for the datasets can be seen on Figure 13. The general rule is that most words don't appear very often. In addition, with the exception of data set M10, no dataset contains words that appear only 2 or 3 times.

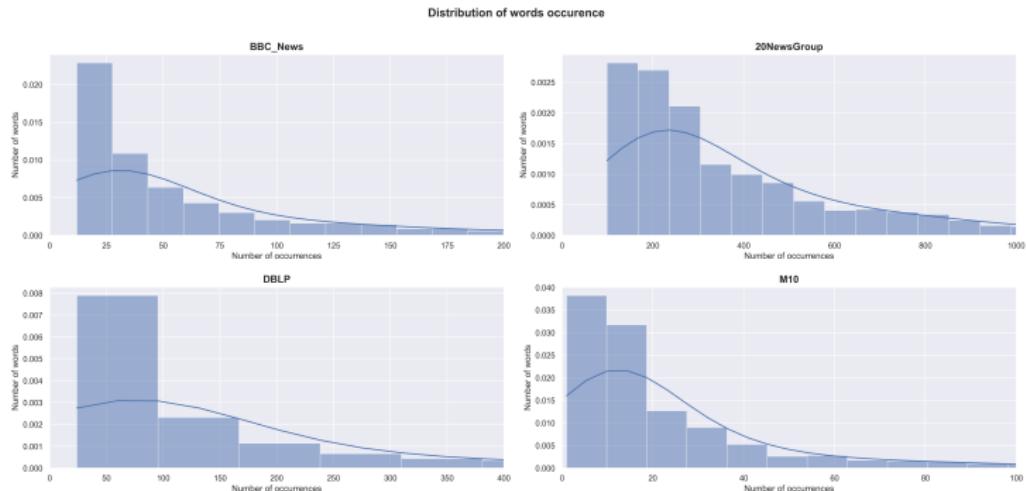


Figure: Distribution of the occurrences of words for each datasets.

Full inputs scenario

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 the previous section.

In this section, multiple models will be investigated such as:

- ▶ Lbl2Vec (and Lbl2TransformerVec)

Lbl2Vec

Lbl2Vec - Theory

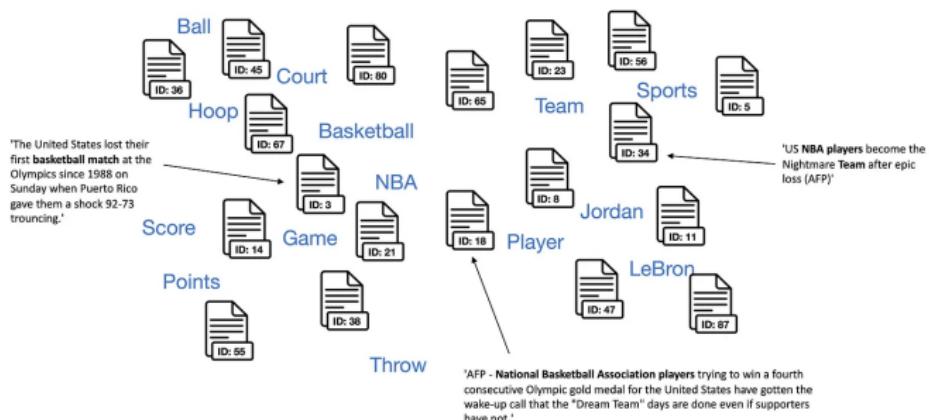
Lbl2Vec: unsupervised algorithm to classify documents in defined classes based on a predefined knowledge, which is the keywords associated with each classes. At a high level, this algorithm performs the following tasks:

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

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

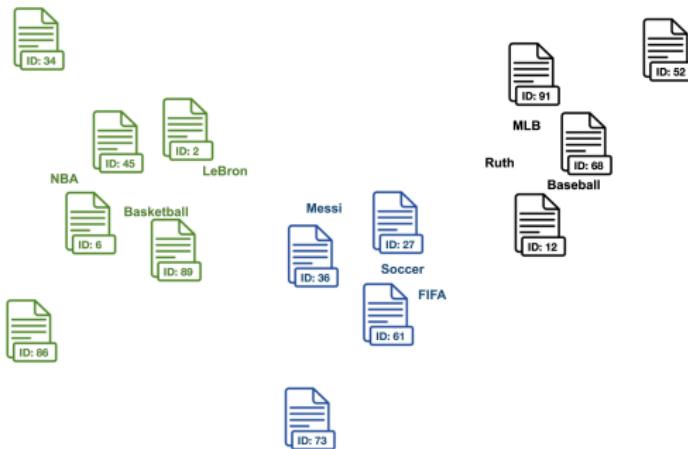
Lbl2Vec - Theory

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.



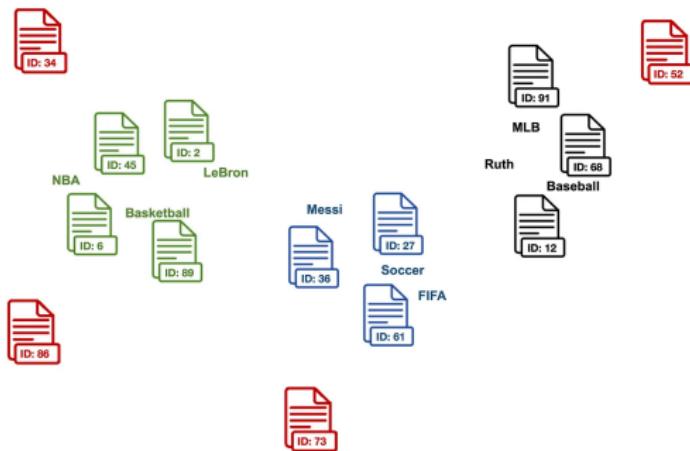
Lbl2Vec - Theory

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.



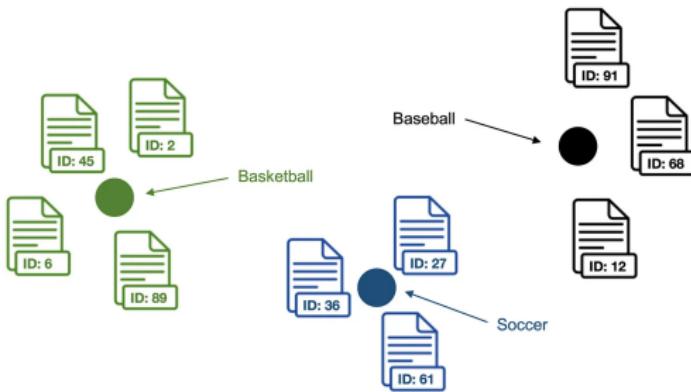
Lbl2Vec - Theory

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



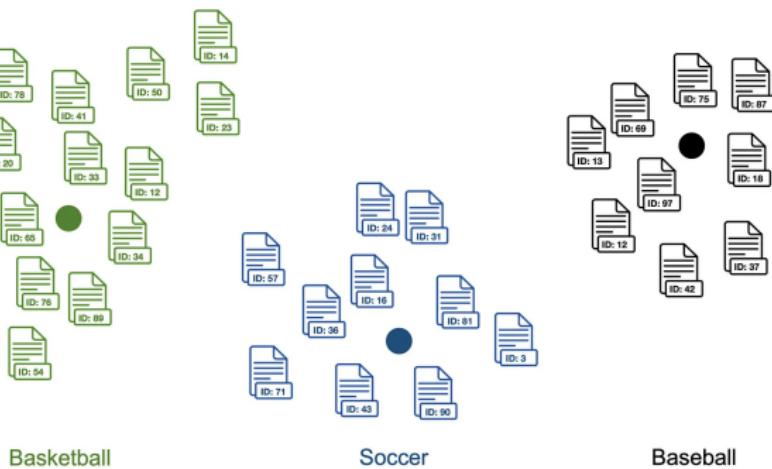
Lbl2Vec - Theory

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



Lbl2Vec - Theory

6. The algorithm computes label vector \Leftrightarrow 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 \Leftrightarrow document vector similarity.



Lbl2TransformerVec - Theory

From the **Lbl2Vec documentation**:

Lbl2TransformerVec learns word vectors, document vectors and label vectors using transformer-based language models during training. Using state-of-the-art transformer embeddings may not only yield to better predictions but also eliminates the issue of unknown keywords during model training. While the Doc2Vec-based model can only use keywords that Lbl2Vec has seen during training, the transformer-based Lbl2TransformerVec model can learn label vectors from any set of keywords. ... However, using transformers instead of Doc2Vec is much more computationally expensive, especially if no GPU is available.

Unknown keywords problem

The problem explained previously with Doc2Vec-based model that it can only use keywords seen during training 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 with 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.

Performances

A comparison of F1 scores between the Lbl2Vec and Lbl2TransformerVec can be seen on Figure ??.

Reduced inputs scenario

Reduced inputs scenario

TODO

TF-IDF + KMeans

TF-IDF + KMeans

The main goal here is to extract vectors from the documents using a TF-IDF vectorizer. Then apply a clustering algorithm called KMeans to the vectors to extract clusters. Finally, compare the found clusters with the "true" classes to see if there is any equivalence. All the experiments have been performed on the BBC News dataset.

TF-IDF

Term Frequency (TF): how frequently a term/word appears in a document, calculated as:

$$TF = \frac{\text{Number of times term appears in a document}}{\text{Total number of terms in the document}} \quad (1)$$

(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.)

TF-IDF

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:

$$\text{IDF} = \log \frac{\text{Total number of documents in the corpus}}{\text{Number of documents containing the term}} \quad (2)$$

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:

$$\text{TF-IDF} = \text{TF} * \text{IDF} \quad (3)$$

TfidfVectorizer

For this experiment, we use the *TfidfVectorizer* from scikit-learn. An example of the output of this vectorizer and a comparison with the *CountVectorizer* from scikit-learn can be seen on Figure 32.

Count Vectorizer

	blue	bright	sky	sun
Doc1	1	0	1	0
Doc2	0	1	0	1

TD-IDF Vectorizer

	blue	bright	sky	sun
Doc1	0.707107	0.000000	0.707107	0.000000
Doc2	0.000000	0.707107	0.000000	0.707107

Figure: Difference between the outputs for *TfidfVectorizer* and *CountVectorizer* from scikit-learn

KMeans

K-means: Unsupervised learning algorithm that seeks to partition a set of N data points into K clusters, where K is a user-defined parameter. The algorithm works iteratively to assign each data point to one of the K clusters and update the cluster centroids to minimize the within-cluster variance or distance. The complete algorithm can be seen in Algorithm 1.

Algorithm 1 Training of KMeans clustering.

Input: K which is the number of clusters we want to extract.

Output: data points assigned to every clusters.

```
1: centroids  $\leftarrow K$  random points
2: while none of the cluster assignments change do
3:   for data points  $p$  do
4:     distance  $\leftarrow$  smallest distance between  $p$  and each centro $\tilde{d}$ 
5:     Assign  $p$  to the cluster with the smallest distance
6:   end for
7:   for each cluster  $c$  do
8:     centroids[ $c$ ]  $\leftarrow$  average of assigned points
9:   end for
10:  end while
```

KMeans

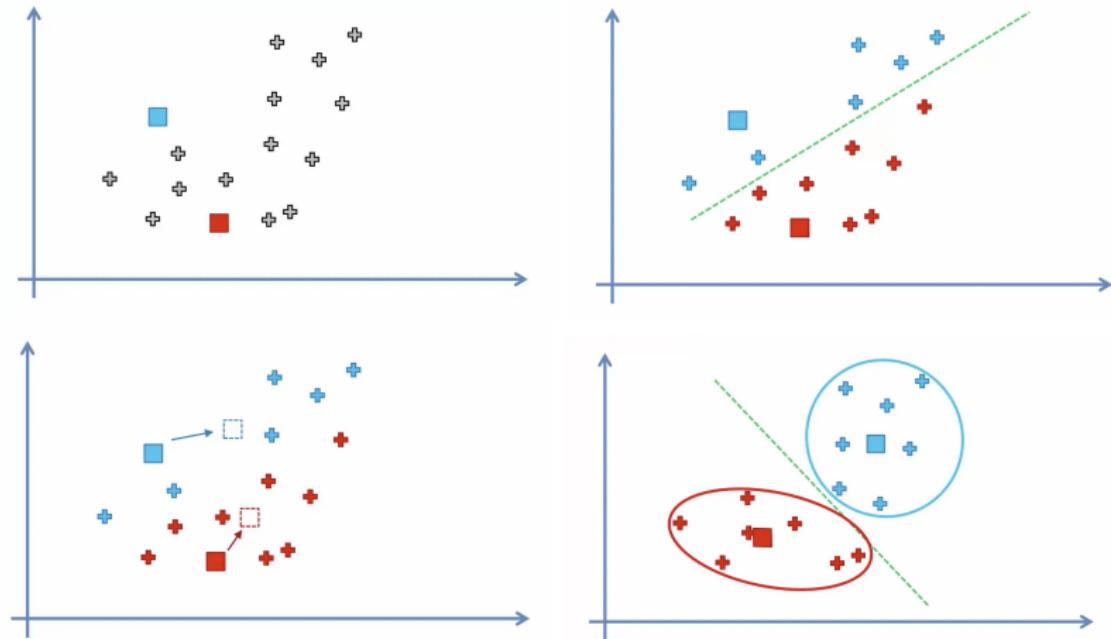


Figure: Steps for KMeans clustering algorithm from left to right.

Link between clusters and classes

After applying the algorithm, we try to assign each found clusters to the true classes by computing their centroids and assigning each clusters with the nearest class. To view the results in a two-dimensional space, we apply PCA on the data points. The results for the train and tests sets can be seen in Figures 36 and 37.

Link between clusters and classes

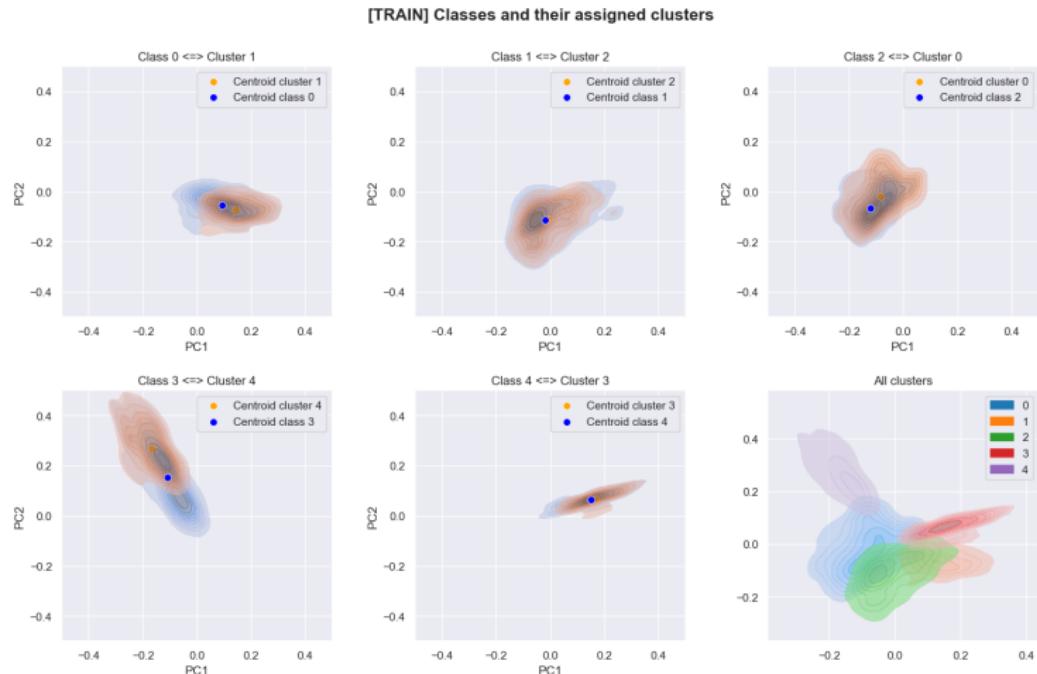


Figure: Classes and their assigned clusters for the train dataset.

Link between clusters and classes

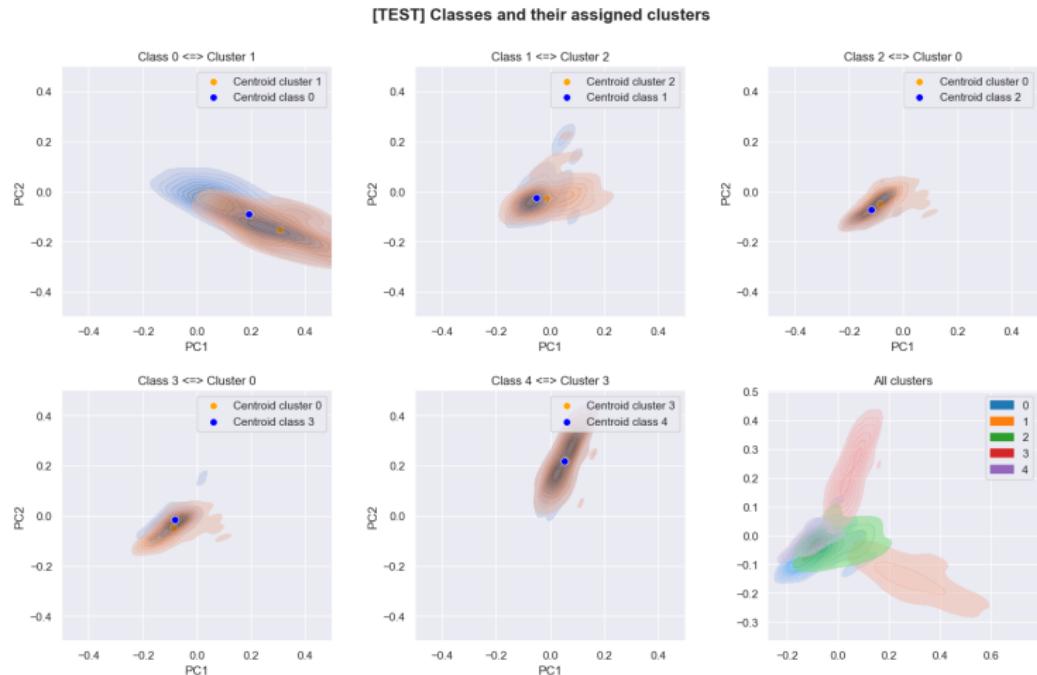


Figure: Classes and their assigned clusters for the test dataset.

Link between clusters and classes

We can already see a problem. For the test set, the cluster 0 is assigned twice. By using the train assignment, the resulting F1 score by class can be seen in Figure 38.

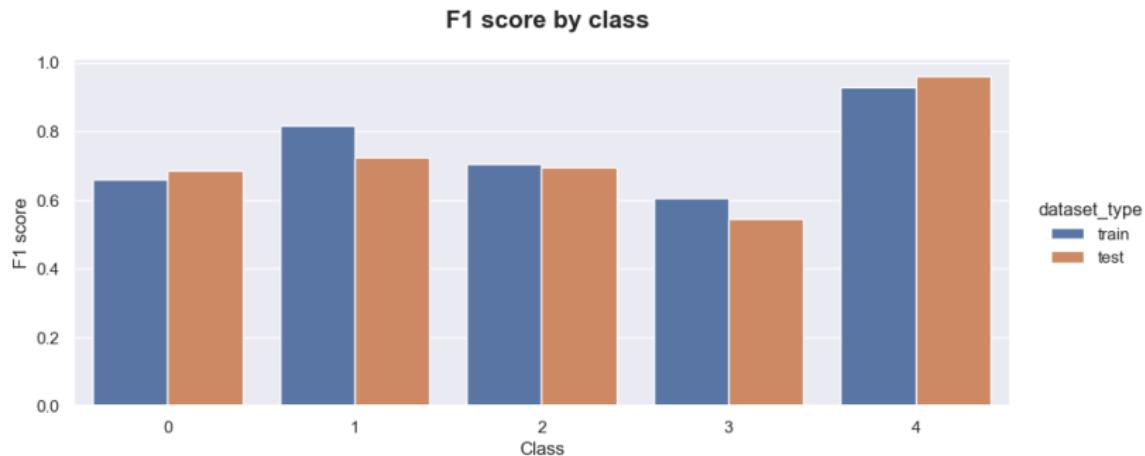


Figure: F1 scores for each class after using the train assignment of clusters.

Link between clusters and classes

In addition to the cluster 0 being assigned twice for the test set, other problems/questions arise:

- ▶ To find the centroids of the classes so that we can assign the clusters, we need to have the classes assigned to each point → supervised! So we can use this method to find clusters, but this method cannot be used to associate clusters to classes.
- ▶ How to associate clusters to the true wanted classes (if they are provided)?

In the remaining section, we will ask ourselves if there is a way to associate a class with a cluster with less knowledge? For example, with some keywords or a few documents. Therefore, we are more in a semi-supervised scenario than in a unsupervised one.

Semi-supervised assignment

For testing purpose, we will focus on class 0. We are going to see the difference in the centroid of this class by using the full knowledge, i.e. every labelled documents, and a limited knowledge, i.e. few labelled documents. The results can be seen in Figure 41.

Semi-supervised assignment

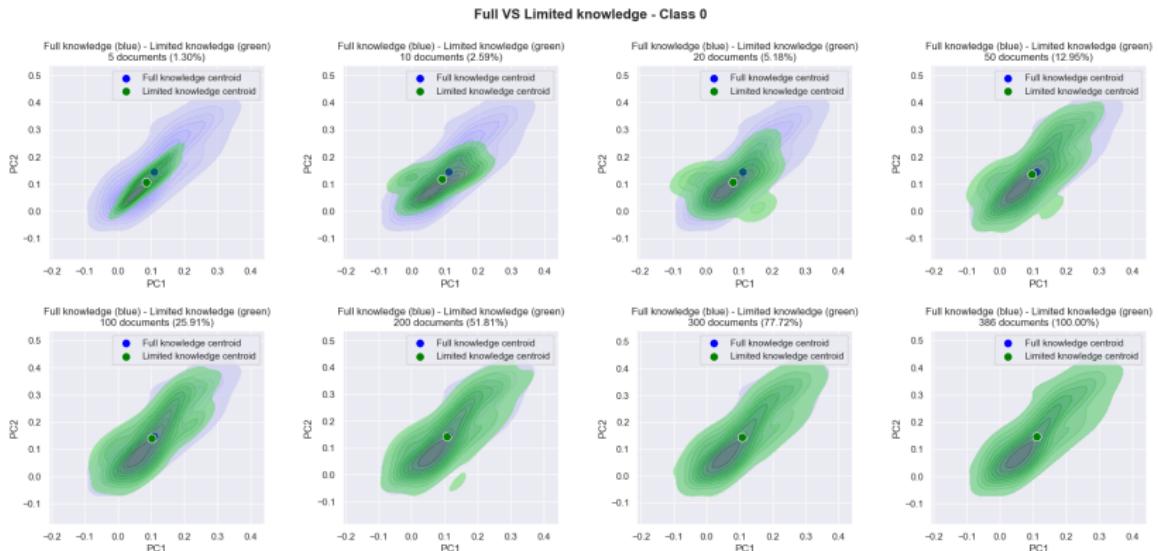


Figure: Difference in the centroids of class 0 between a full knowledge and a limited one.

Semi-supervised assignment

This result is interesting but shows us that it still cause problems:

- ▶ We are in a semi-supervised configuration which is still not what is expected.
- ▶ Still need more or less 10% of all the documents per class to be close to the centroid using supervised method.

These important problems made us move to another models.

Predefined keywords

An important input for this model to work correctly are the keywords chosen at the beginning. They have a baseline to work with, we use the BBC News dataset and extract in a supervised way the 10 most relevant words for each classes using the *TfidfVectorizer* from scikit-learn. We then compute the F1 score of the model on 5 iterations to take into account the slight randomness of the model. The result can be seen in Figure 43.

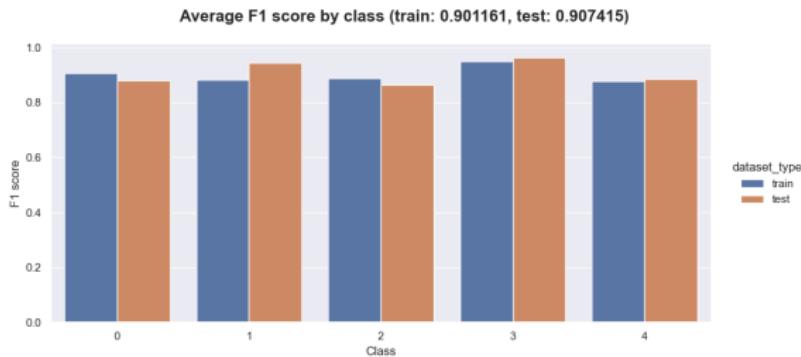


Figure: Average F1 score by class using the most relevant words by class in a supervised way.

Keyword similarity

The problem with the previous method is that we need to know the classes in advance associated to each documents to extract the keywords. What we want is to extract the keywords in an unsupervised way. The first method we use is explained below.

1. Group every word in all the documents to create a vocabulary.
2. Find the similarity score between every word and the name of the classes.
3. For each class, extract only the words with the highest scores.

Problem

We need to have a well defined name for every class.

Keyword similarity

We also compute the F1 score by class using 2 most relevant keywords for each class and 10 iterations because the model seems somehow random. Results can be found on Figure 45. We see that we have still a good F1 score, even though we have a decrease of 10%.

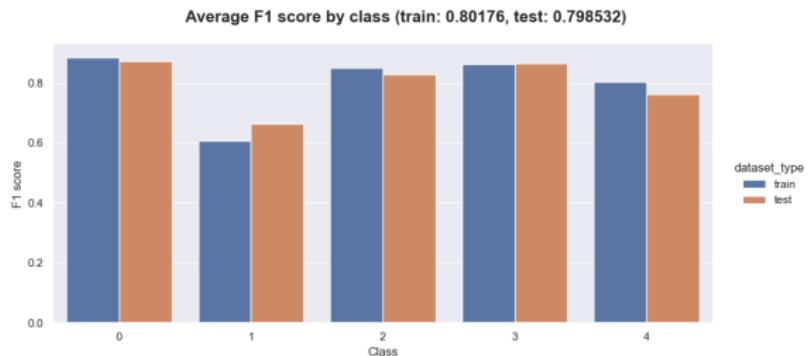


Figure: Average F1 score by class using 2 most relevant words by class in an unsupervised way.

Question

Why is the model random?

Impact of the number of keywords

We wanted to understand how the number of keywords impact the accuracy of the predictions. Results can be seen on Figure 46. It seems that we can reach an F1 score almost as close as with the predefined keywords.

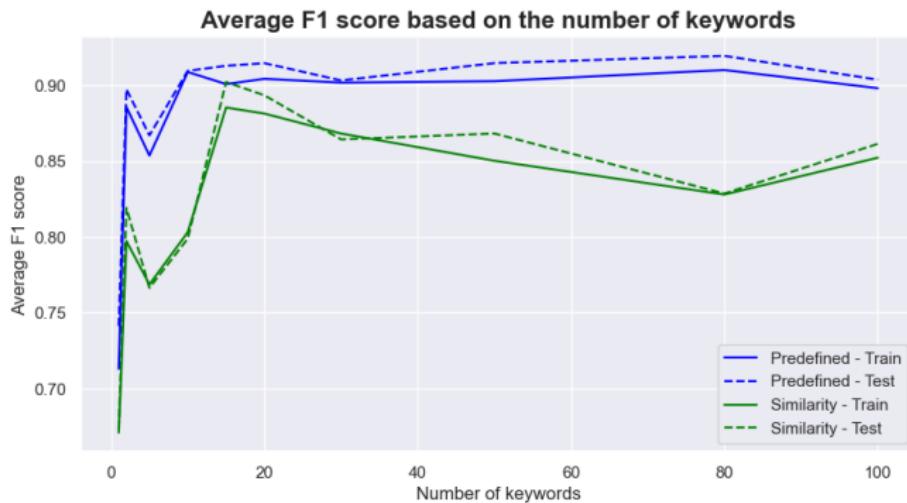


Figure: Impact of the number of keywords on the average F1 score using 5 iterations.

Impact of the number of keywords

The part from 10 to 25 seemed the more interesting. Therefore, we decided to focus on this part. Results can be seen on Figure 46. With this figure, we can assume that, for this dataset, a number of keywords between 14 and 24 is enough to have good predictive performance and a small difference with the supervised method.

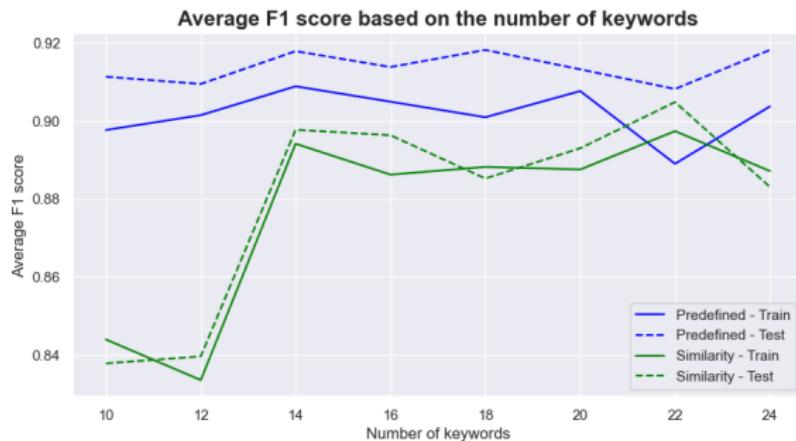


Figure: Impact of the number of keywords on the average F1 score using 5 iterations.

Directions

- ▶ Use **Lbl2TransformerVec** instead of **Lbl2Vec** (The plain Lbl2Vec model uses Doc2Vec, whereas Lbl2TransformerVec uses transformer-based language models to create the embeddings).
- ▶ Test with other datasets where the class names are not well defined (e.g. 20NewsGroup).
- ▶ What to do with not well defined class names.

Lbl2TransformerVec VS Lbl2Vec

Results of a comparison of the two models on 20NewsGroup and BBC News datasets can be seen in Figure 49. Strangely, the transformer version does not yield the best results, but it still solve the problem with unknown keywords. The fact that both method have bad results on the 20NewsGroup dataset can be possibly explained by the labels giving not as much explanation as in the BBC News dataset.

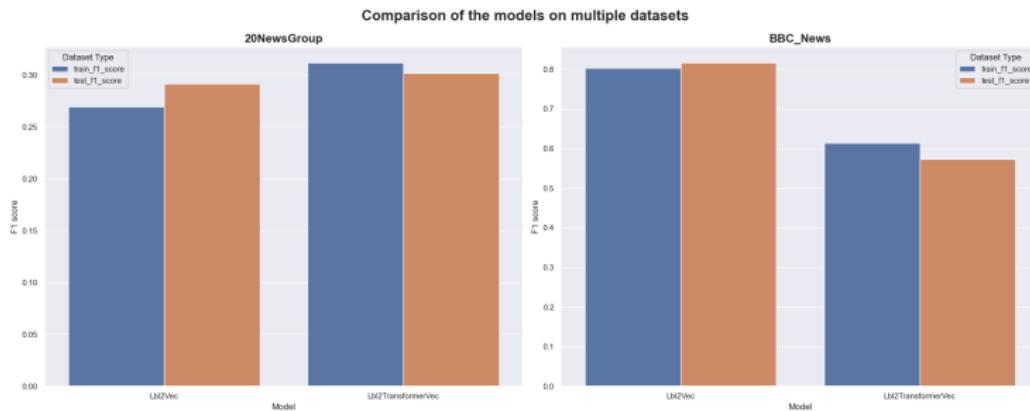


Figure: Difference in F1 score between Lbl2Vec and Lbl2TransformerVec applied on different datasets.

Lbl2Vec V2

Motivation

Motivation

All results are always computed on part of the problem, such as some datasets, a reduced number of keywords, etc. Testing everything together need a GPU. That is why everything is grouped into a single model that is available at the following url.

Using this architecture, we will be able to:

1. Determine how to solve the problem with unclear class names.
2. Determine the best number of keywords and how it adapts to every dataset.
3. Determine other methods than the Spacy similarity for finding keywords.
4. Compare the models (Lbl2Vec & Lbl2TransformerVec) on multiple datasets.

Order is important, because some problems can't be solved without solving others first.

Unclear class names

Unclear class names

Logically, if we want to classify into classes in a unsupervised way, we need to give better description to the classes. This require manually labelling each class. We have to keep in mind that the objective here is to avoid giving a description to each class, but only the label. To this end, our four benchmark datasets have been updated with the following class names: