Topic Modelling & Clustering - Observations

Yorick Estievenart

University of Mons

UMONS
Université de Mons

sinch

Overview

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.

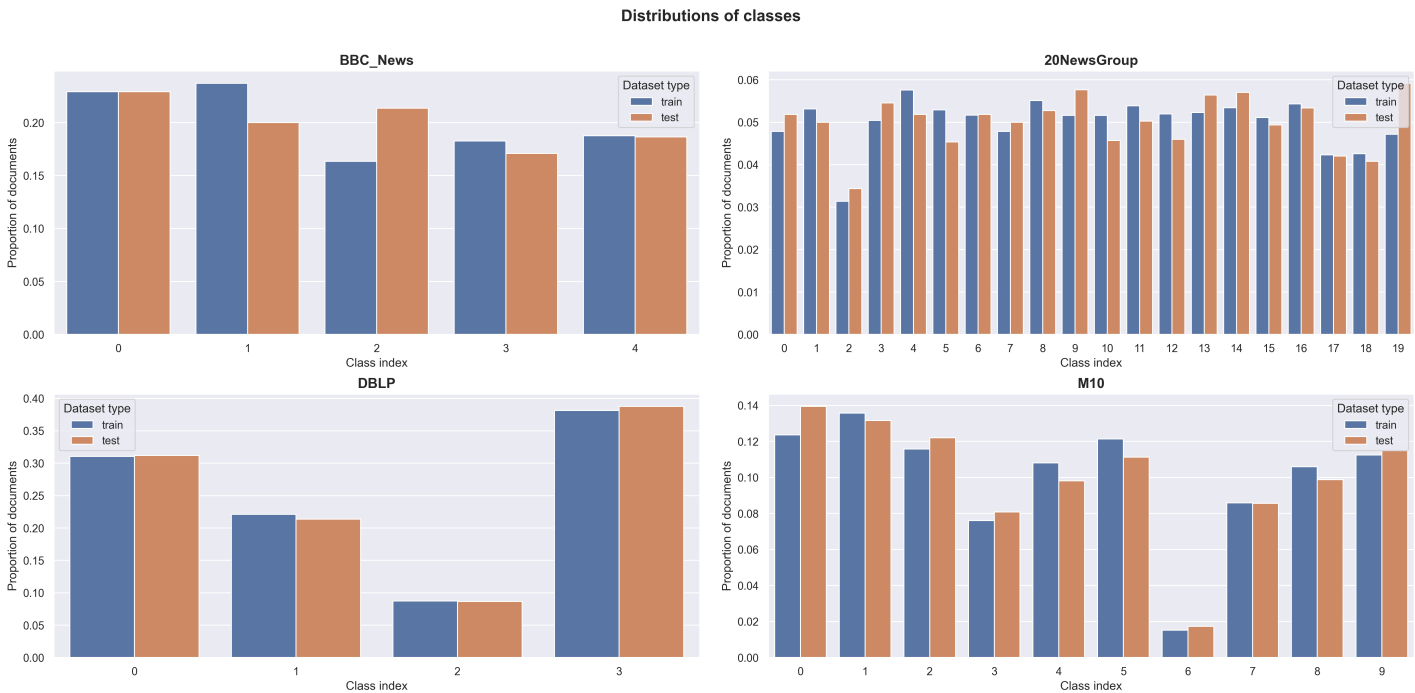


Figure: Distribution of the classes for the datasets.

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.

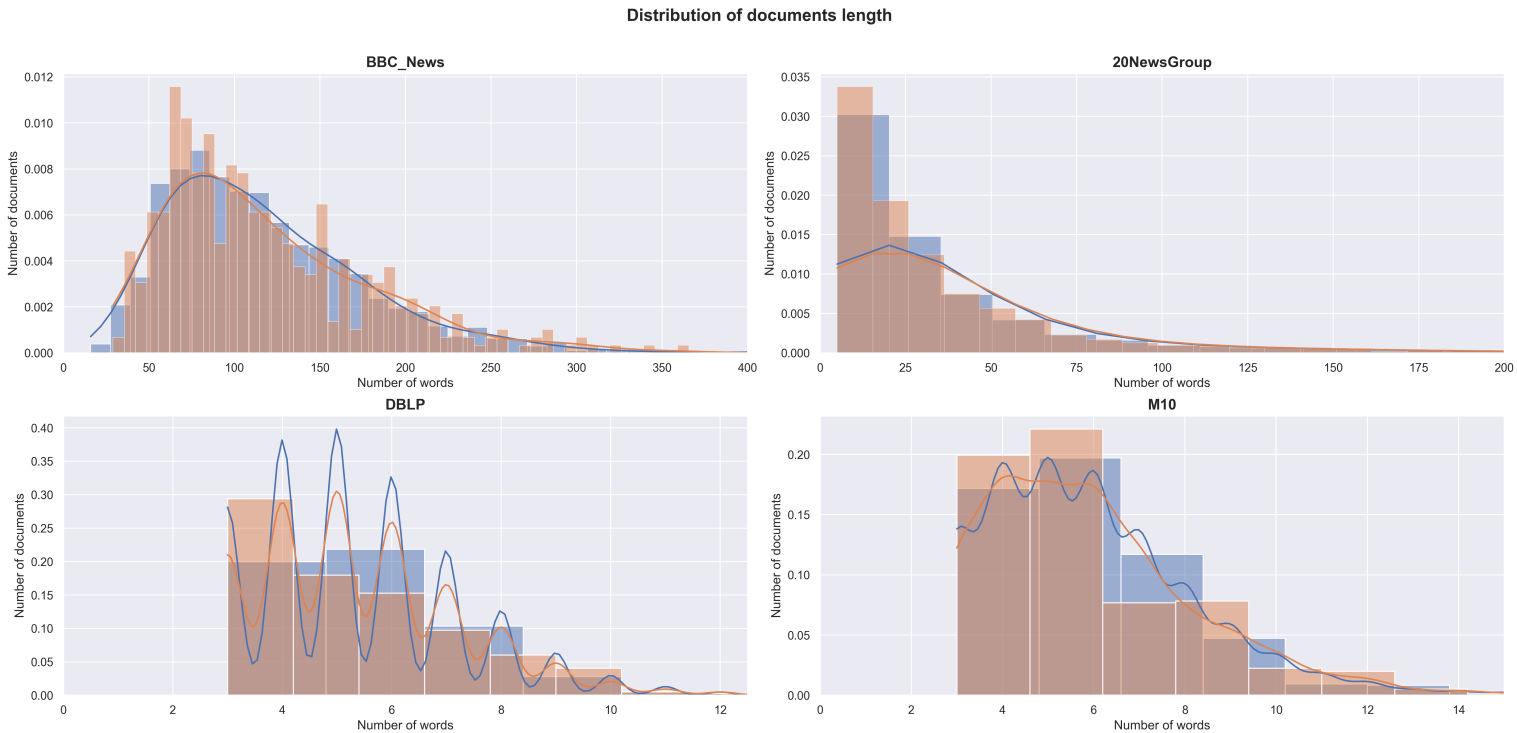


Figure: Distribution of the documents length for the datasets.

Most frequent words

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

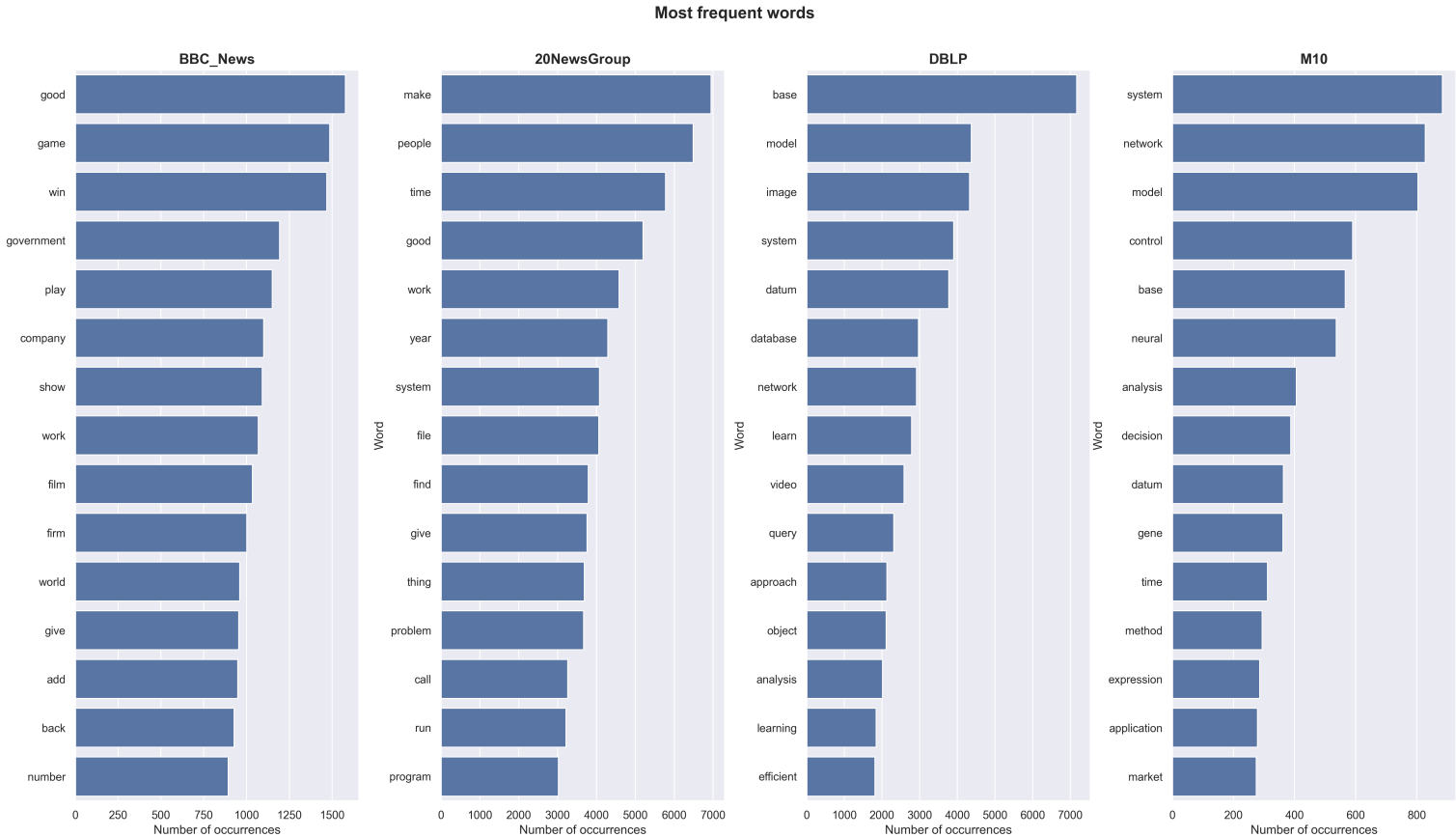


Figure: Most frequent words used in each datasets.

Distribution of words occurences

Distribution of words occurences 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.

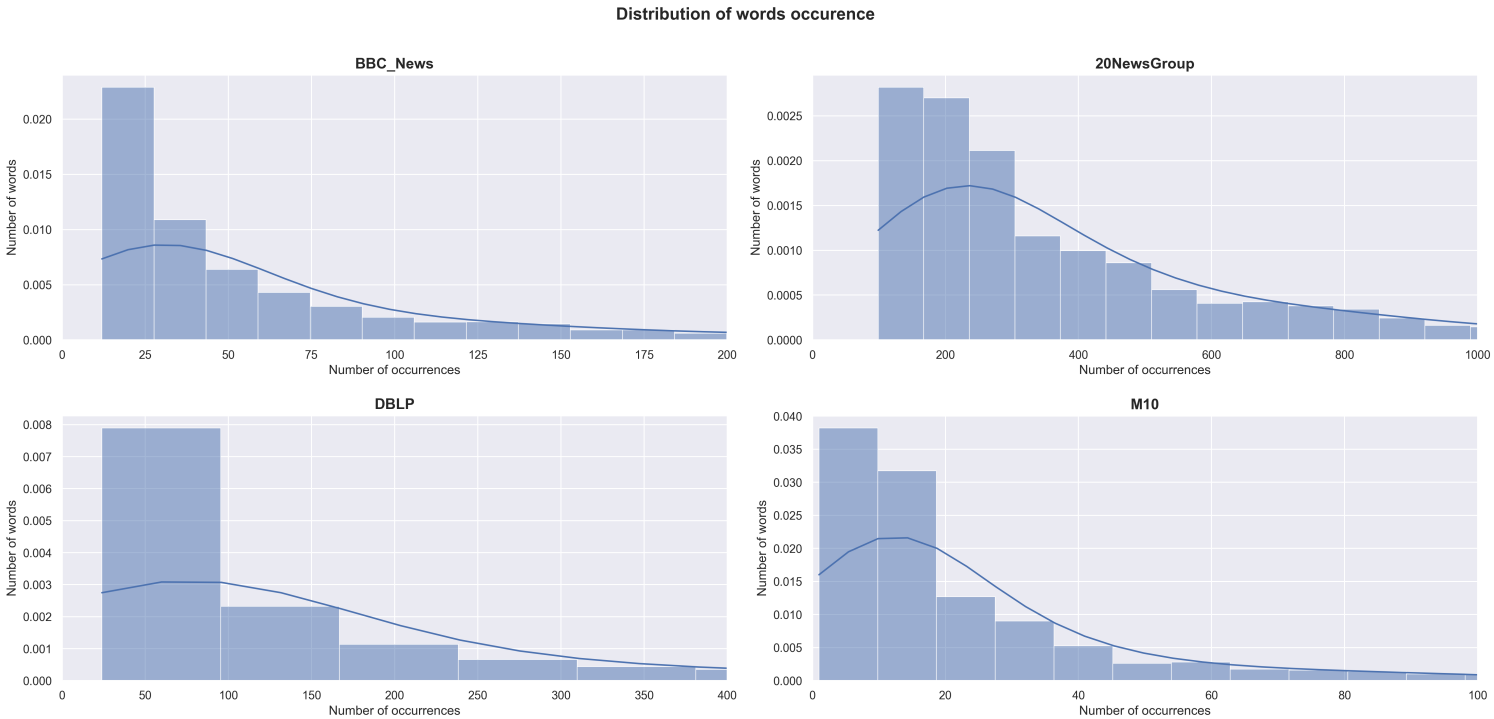


Figure: Distribution of the occurences of words for each datasets.

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
- Lbl2TransformerVec

Lbl2Vec

Lbl2Vec - Theory

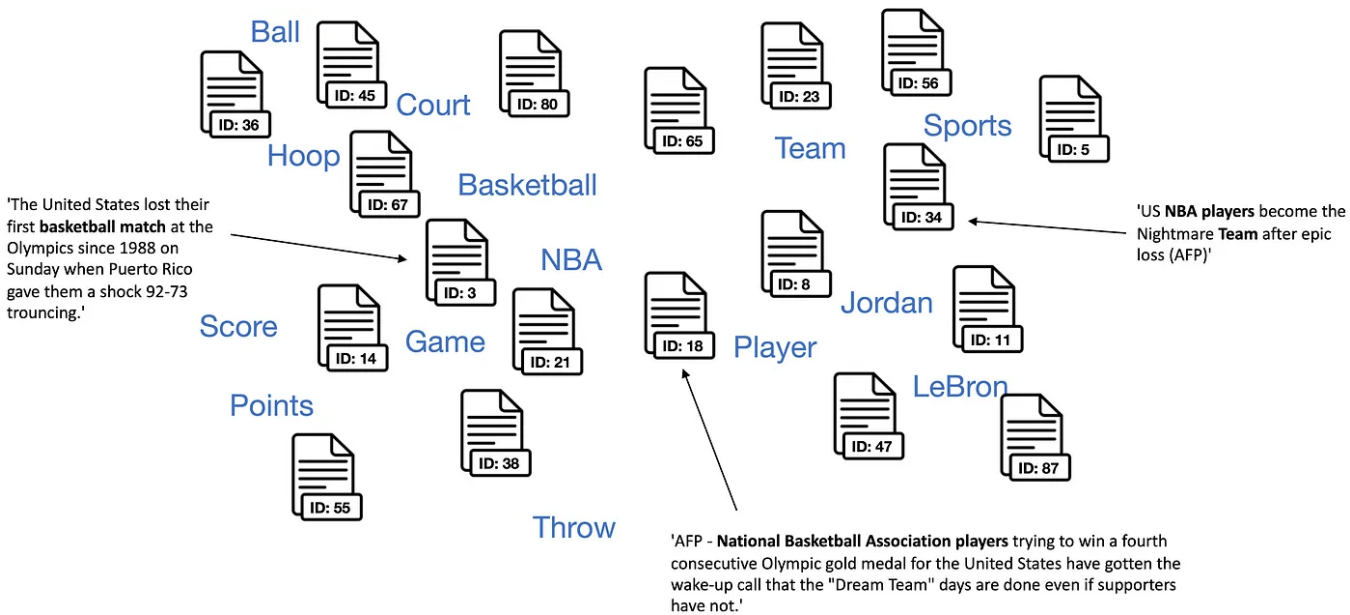
Lbl2Vec: unsupervised algorithm to classify documents in defined classes based on a predefined knowledge, which is the keywords associated with each classes. At a high level, this algorithm performs the following tasks:

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

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

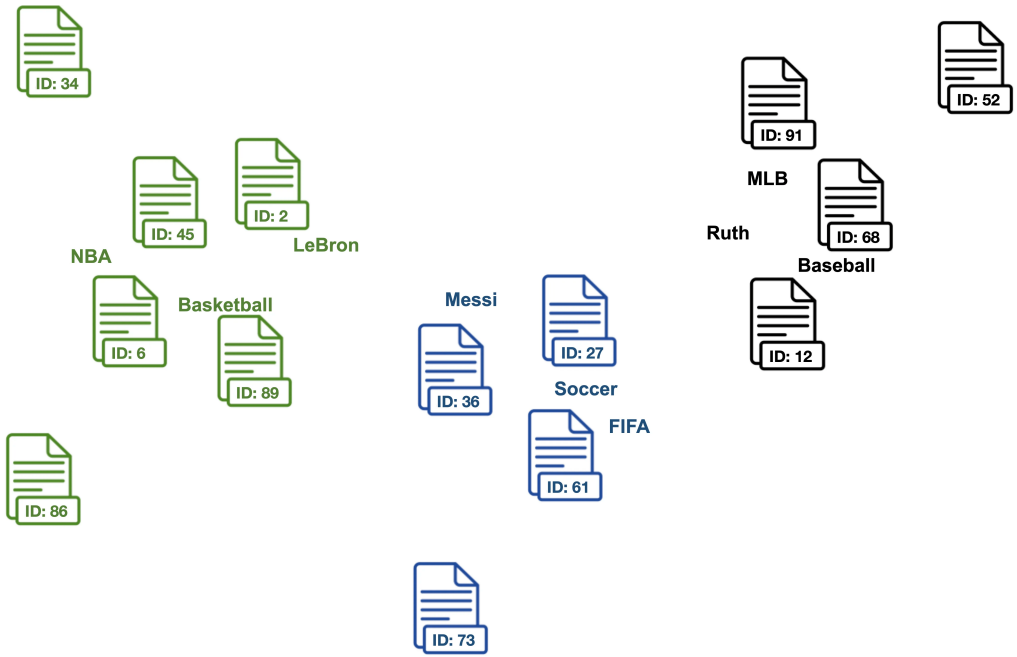
Lbl2Vec - Theory

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.



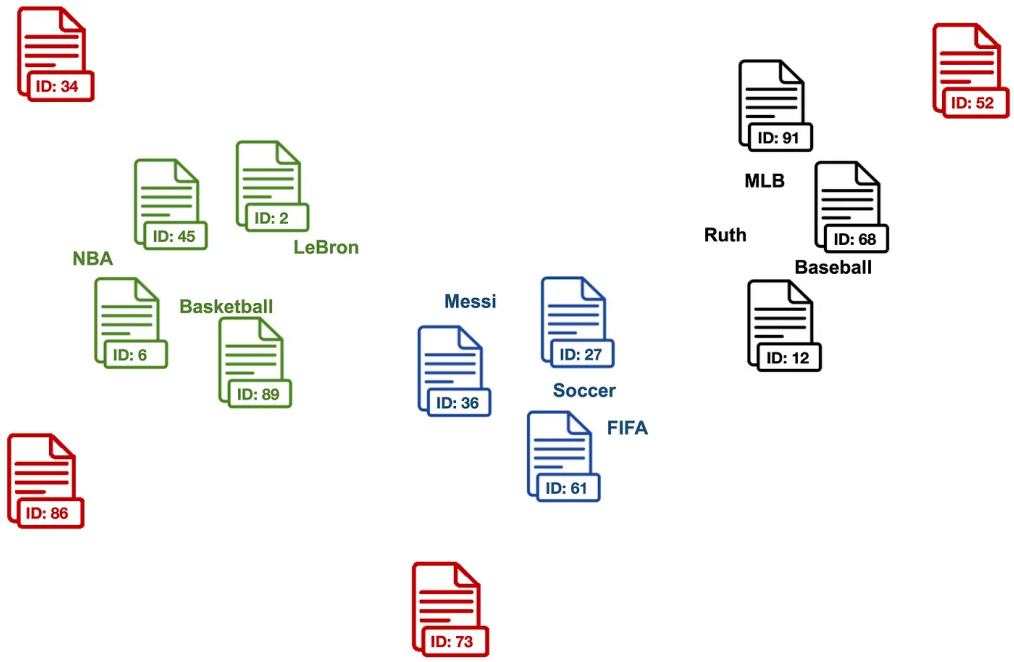
Lbl2Vec - Theory

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.



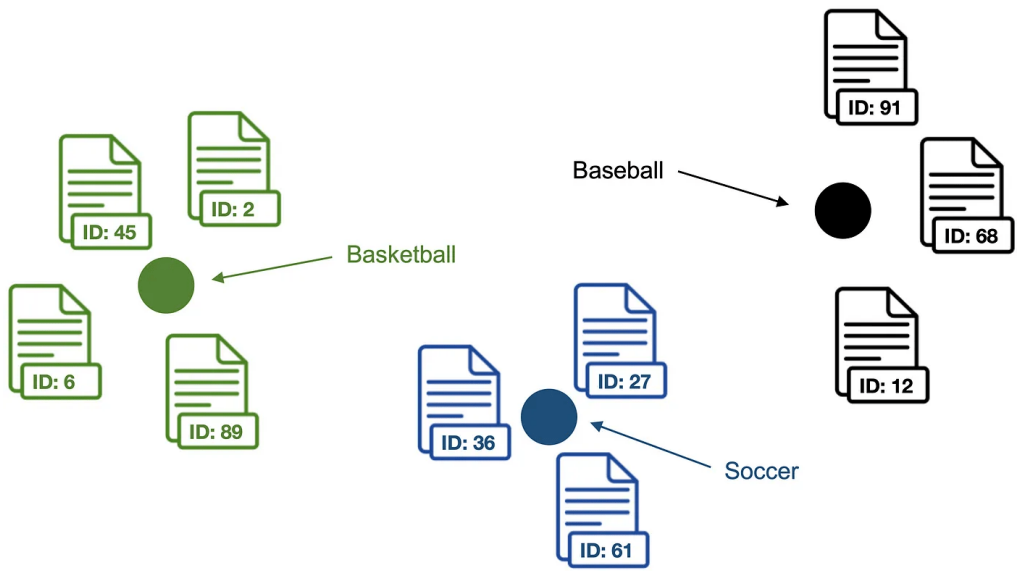
Lbl2Vec - Theory

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



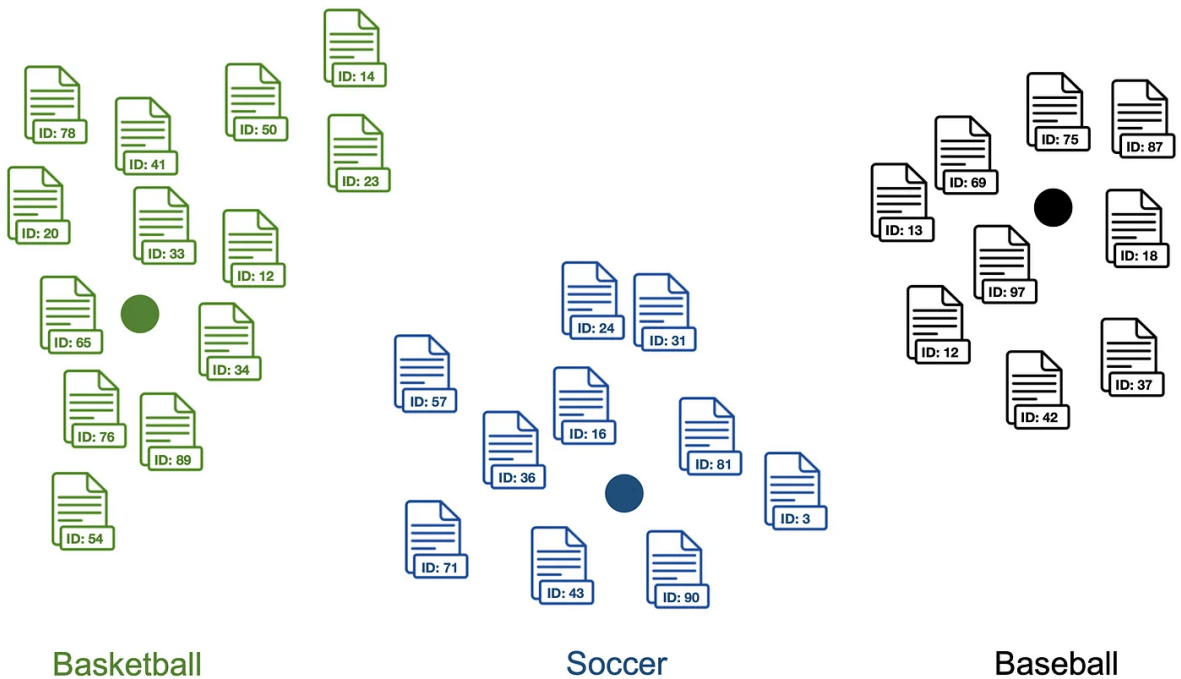
Lbl2Vec - Theory

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.



Unknown keywords problem

There is one problem with the use of Doc2Vec-based models is 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.

Unknown keywords problem

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).

Doc2Vec randomness

To have an objective F1 score, we compute an average on multiple iterations. But why are the Doc2Vec-based models random? After searching in the code, it seems that **the inner Doc2Vec model is random** and there is not possibility to add a seed. We will therefore continue to average the result on multiple iterations where we fit a new model in each iteration and show the variance of the predictions.

Now the question is, how to choose the number of iterations? We need to find a trade-off between convergence and the time taken to compute all the iterations.

Doc2Vec randomness

Figure 27 shows the F1 score obtained by applying the Lbl2Vec model to the BBC News dataset as a function of the number of iterations. We can see that after 30 iterations, the mean results does not seems to change. We will assume that this is also the case for other datasets.

The problem is that 30 iterations is computationally expensive, this is why we will take **10** iterations as a general rule, which is under a difference of 0.02 with the true mean.

Doc2Vec randomness

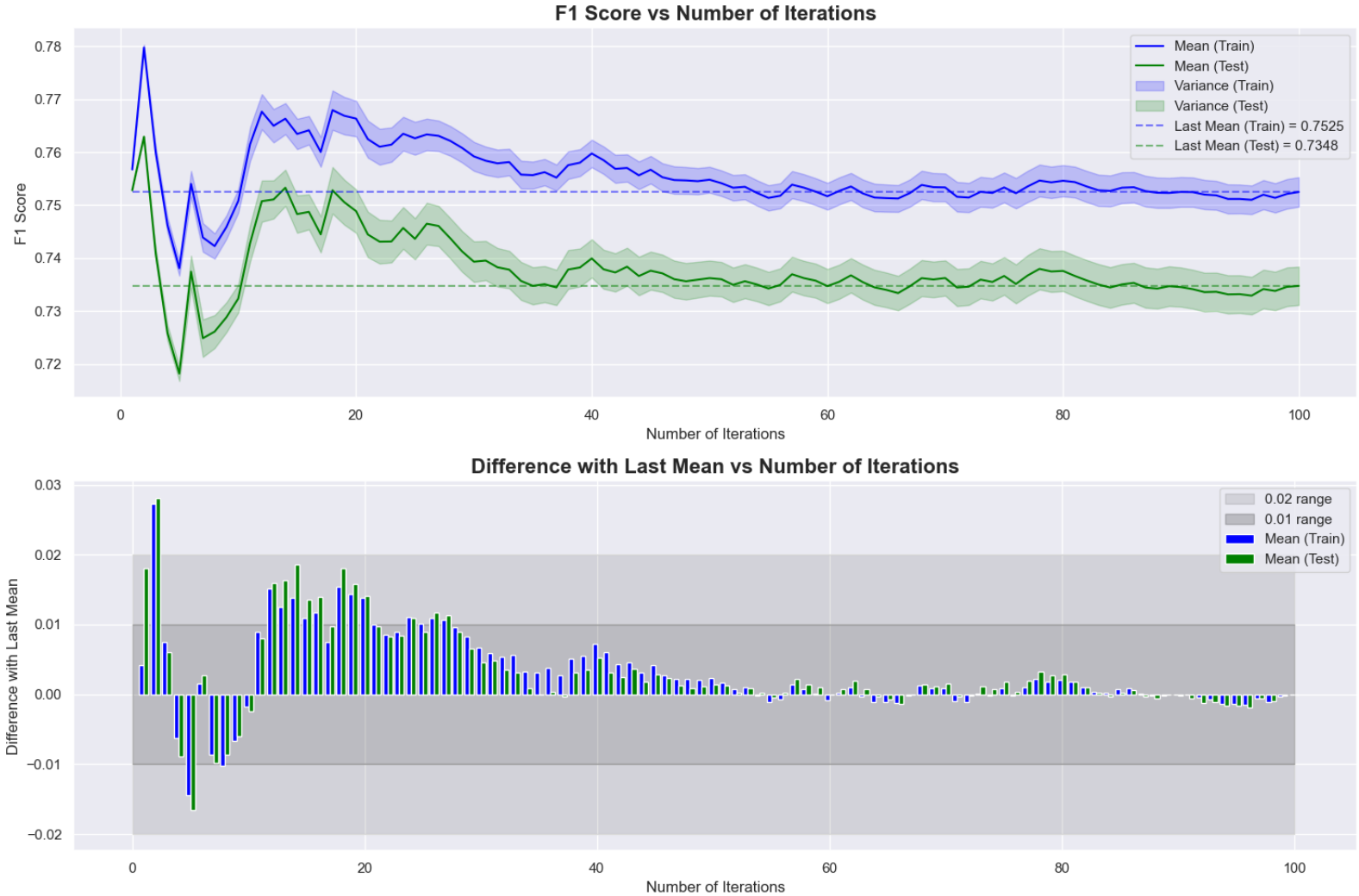


Figure: Analysis of the F1 scores obtained on the BBC News dataset with the model Lbl2Vec as a function of the number of iterations.

Number of keywords (spaCy)

One important question with this strategy used with Doc2Vec-based model is: how to choose the number of similar keywords, and how this number varies according to the dataset used? We are here trying to understand if there is a general rule or it really depends of the dataset used. Results of F1 scores obtained as a function of the number of keywords for the four datasets can be seen in Figure 29.

Here, we only set the number of iterations to two, since there are a lot of combination (for each dataset, for multiple iterations and for multiple numbers of word) and therefore it would take to much time to compute everything exactly.

Number of keywords (spaCy)

What can be seen on this picture is that after a threshold of around 20 keywords, the F1 scores for every datasets does not seems to vary much. We can conclude that **20** is an appropriate number of keywords to use.

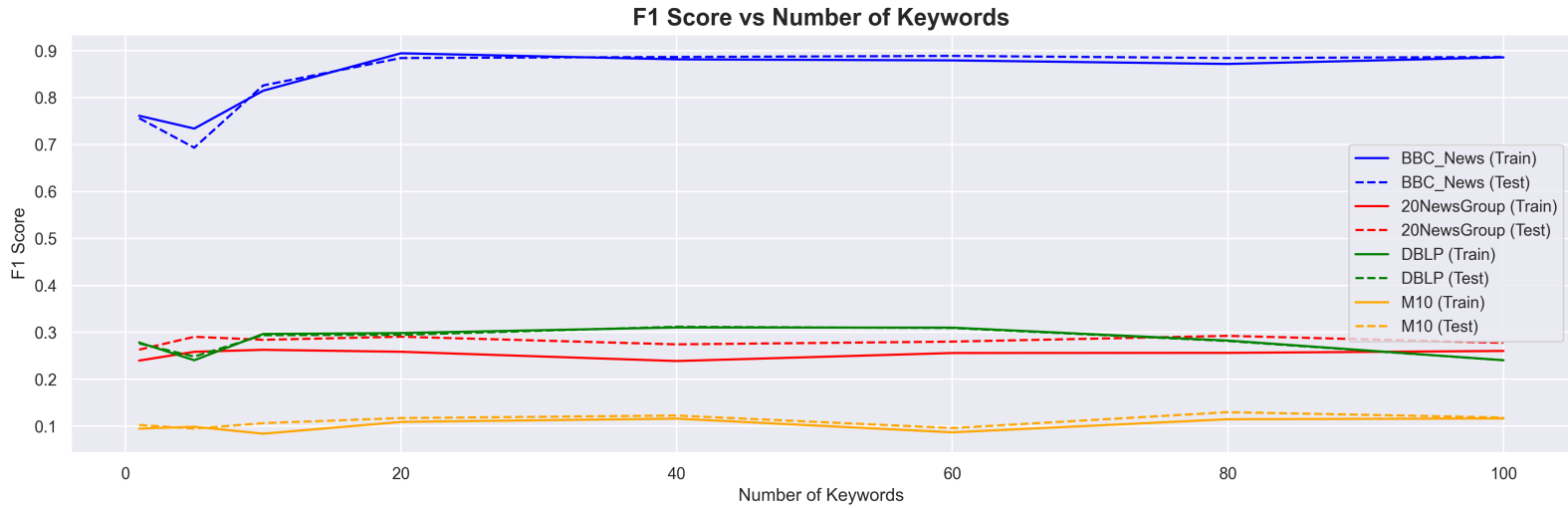


Figure: Analysis of the F1 scores obtained on all four datasets with the model Lbl2Vec as a function of the number of keywords. The keywords chosen were 1, 10, 20, 40, 70 and 100.

Assignment confidence

Another thing we wanted to see is the probability assigned to each documents by the model. Results can be seen on Figure 30.

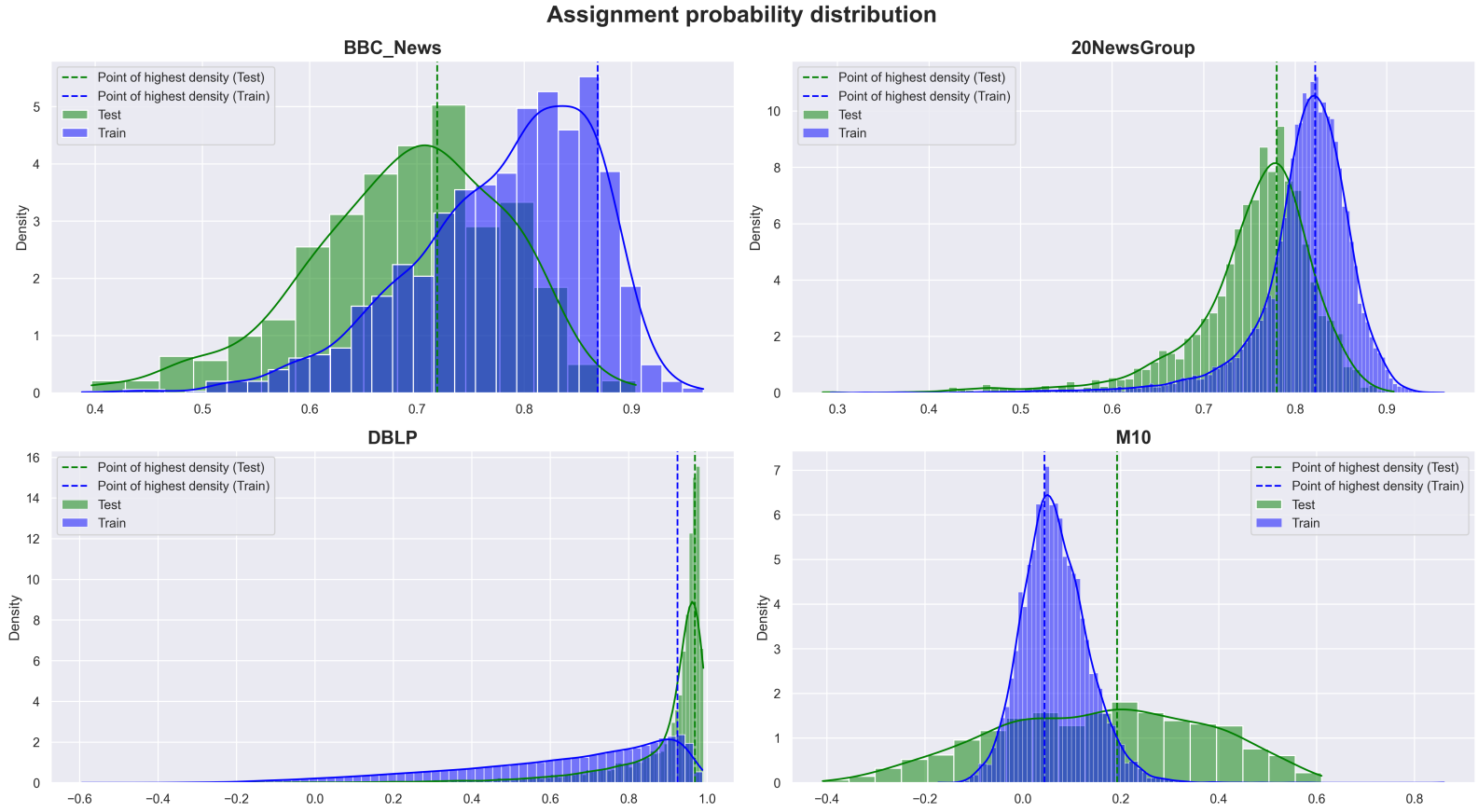


Figure: Histogram of the highest probability assigned to each document.

Assignment confidence

On this figure, what can be seen first is that the model seems **less confident on the test set than on the train set**, except for the DBLP and M10 datasets. Second, it is totally sure of its prediction on the DLBP dataset, while being totally unsure on the M10 dataset. This has to be put in relation with the performance on both datasets. The fact that the highest probability for the M10 are quite low is possible since there are 10 classes. A better study for this 2 datasets should be done since they present a stranger behavior.

Performances

Results of F1 scores obtained by applying the Lbl2Vec on multiple datasets can be seen on Figure **??**.

Limitations

According to lbl2vec's paper:

- It it is hard for our Lbl2Vec approach to distinguish between related topics.
- The Doc2Vec-based models can only use keywords seen during training (the use of spaCy is one of the solutions).

According to observations:

- The randomness of Doc2Vec can be problematic if we have to execute the model for a large amount of iterations before the F1 score converge.

Improvements:

- Use state-of-the-art transformer embeddings instead of Doc2Vec.

Lbl2TransformerVec

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.

Performances

TODO

Assignment confidence

TODO

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:

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

(**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:

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

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:

$$\text{TF-IDF} = \text{TF} * \text{IDF} \tag{3}$$

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 44.

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.

centroids $\leftarrow$ K random points
while none of the cluster assignments change **do**
    for data points p **do**
        distance $\leftarrow$ smallest distance between p and each centroÃ¯d
        Assign p to the cluster with the smallest distance
    end for
    for each cluster c **do**
        centroids[c] $\leftarrow$ average of assigned points
    end for
end while

KMeans

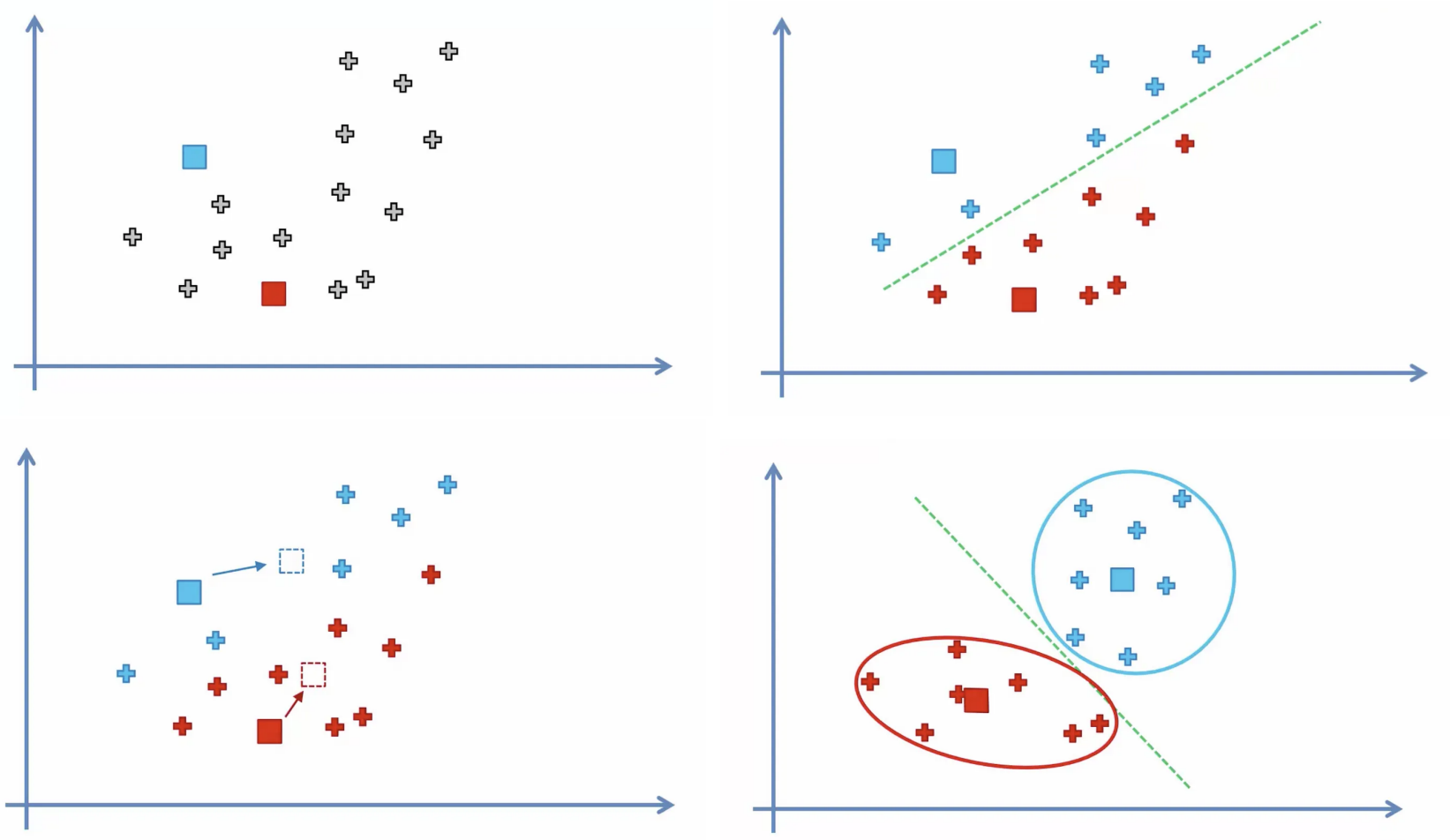


Figure: Steps for KMeans clustering algorithm from left to right.

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 48 and 49.

Link between clusters and classes

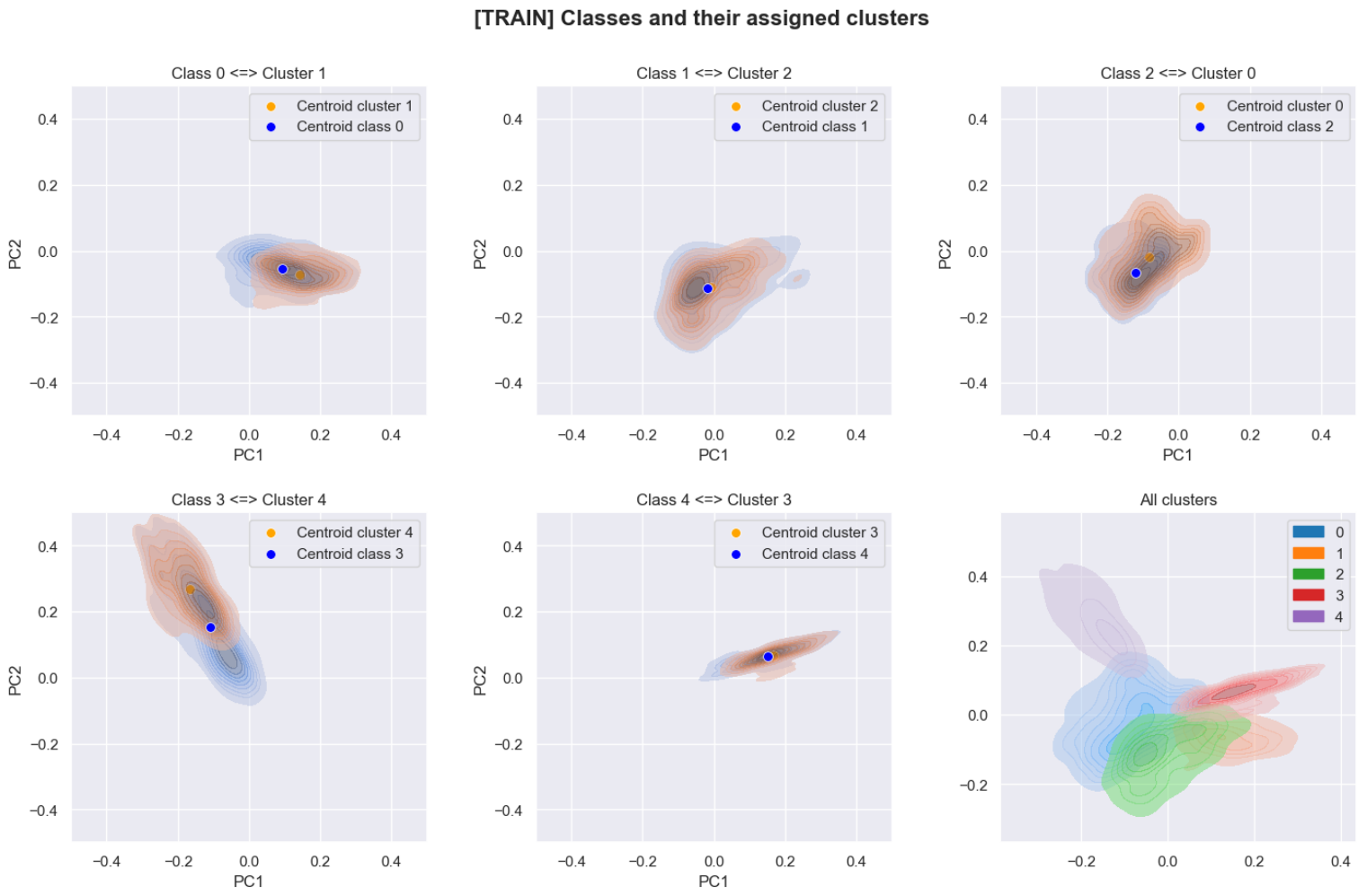


Figure: Classes and their assigned clusters for the train dataset.

Link between clusters and classes

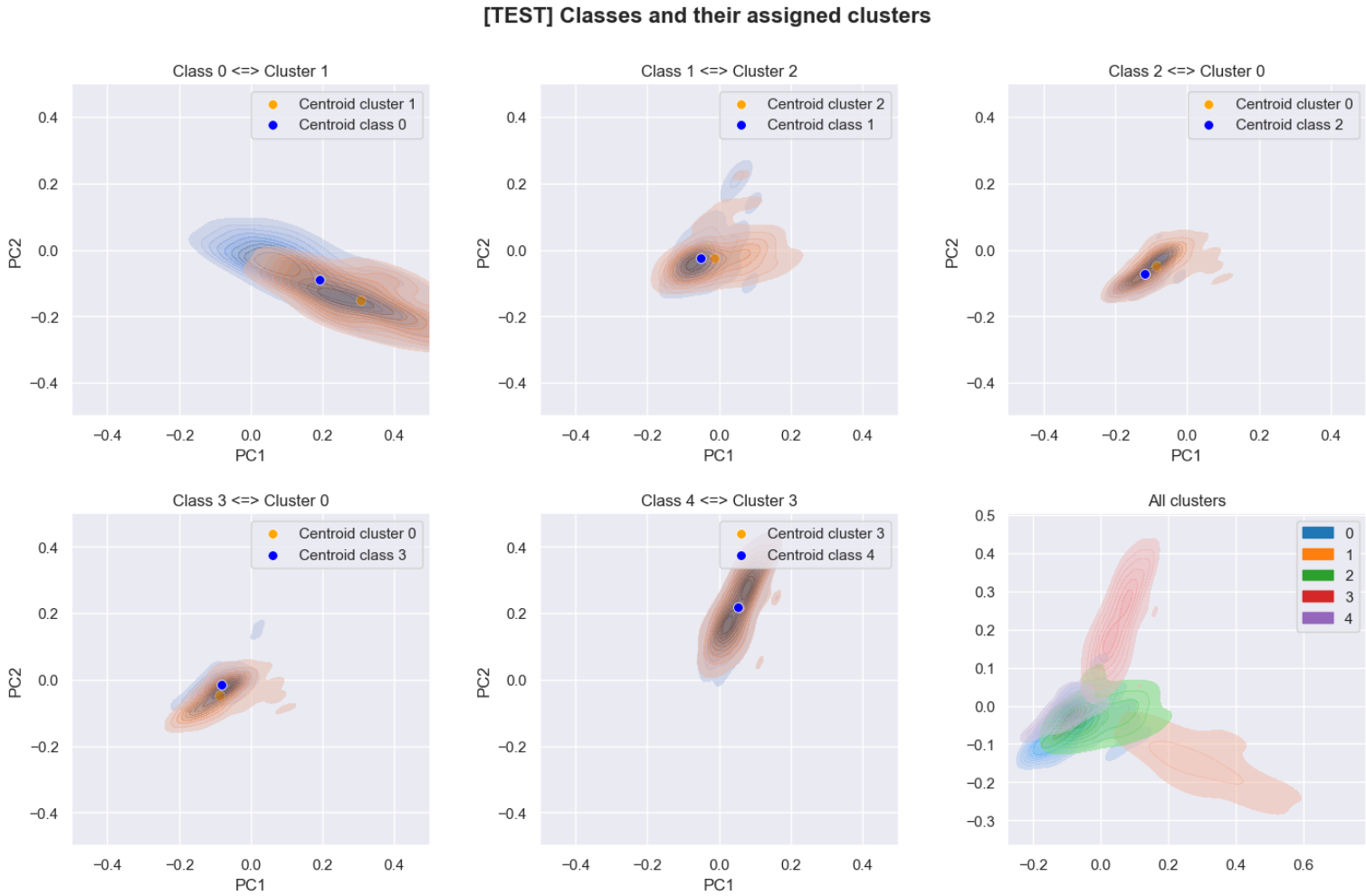


Figure: Classes and their assigned clusters for the test dataset.

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 50.

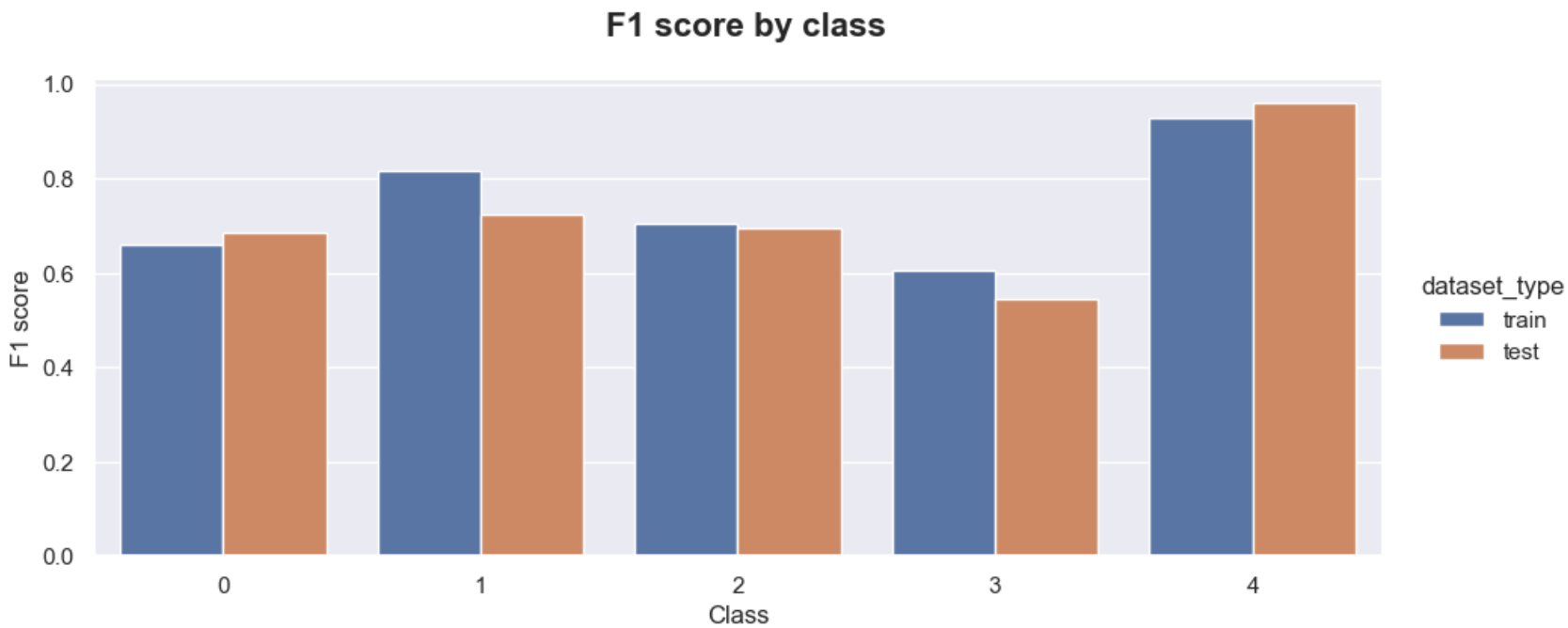


Figure: F1 scores for each class after using the train assignment of clusters.

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 $\rightarrow$ 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 53.

Semi-supervised assignment

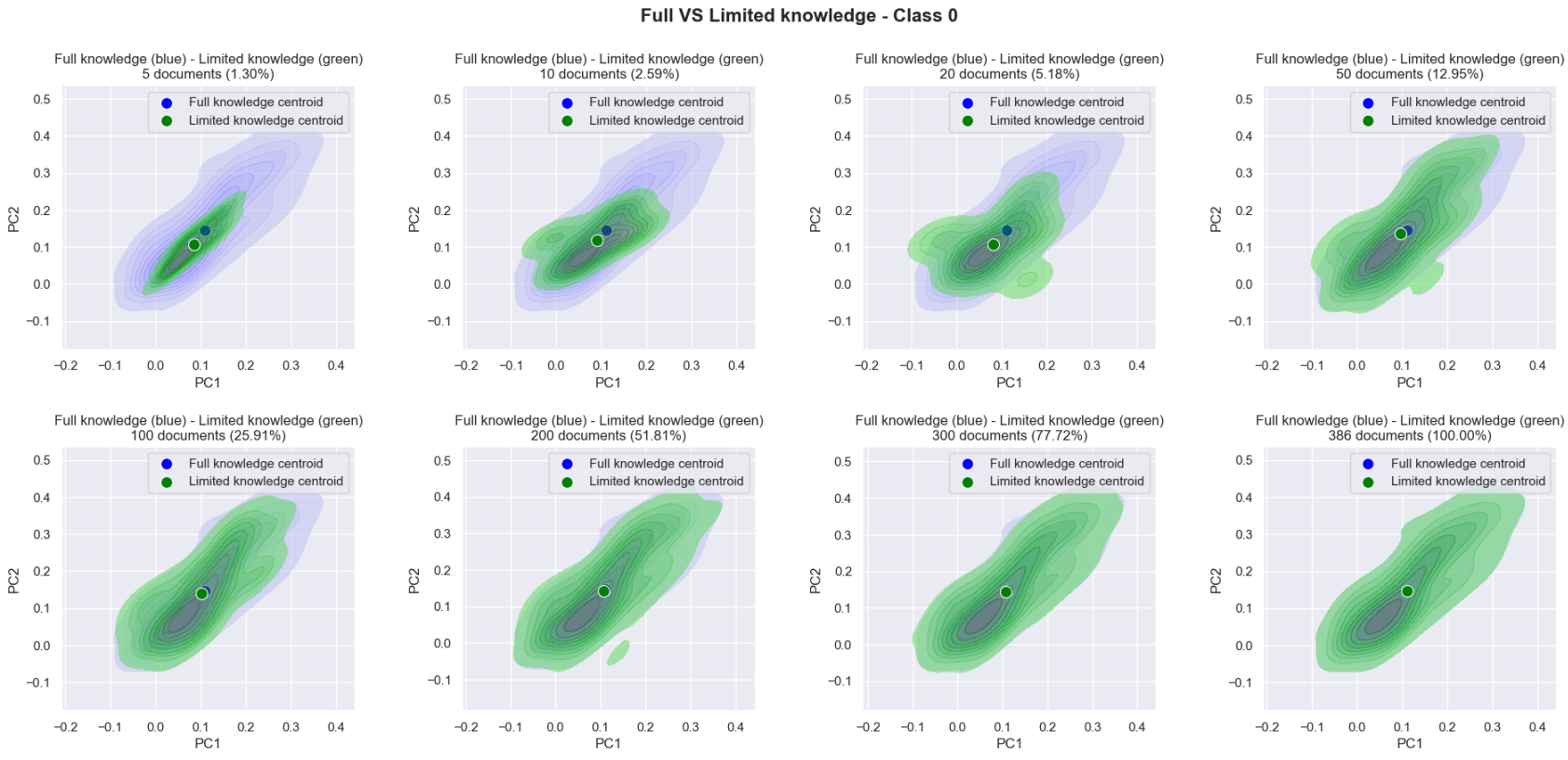


Figure: Difference in the centroids of class 0 between a full knowledge and a limited one.

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.