Topic Modelling & Clustering - Observations

Yorick Estievenart

University of Mons

UMONS
Université de Mons

sinch

Overview

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.

Introduction

This internship 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?

Overview

Benchmark

Table 6 shows the following benchmark datasets that are used during the project.

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

Benchmark

Basic informations about those datasets can be seen in Figures 8, 9, 10 and 11. On this figures, we can see the distributions of classes, the distributions of documents length, the most frequent words, and the distribution of words occurences. What can be seen is:

- The DBLP and M10 datasets have more constant documents lengths than the 20NewsGroup and the BBC News datasets.
- BBC News and DBLP datasets are maybe more appropriate for testing since that have a fewer number of classes.
- The distribution of words occurences is always higher for test than for train but this can be explain by the fact that the test part contains less words than the train part. Therefore, the words appears less often.

Benchmark

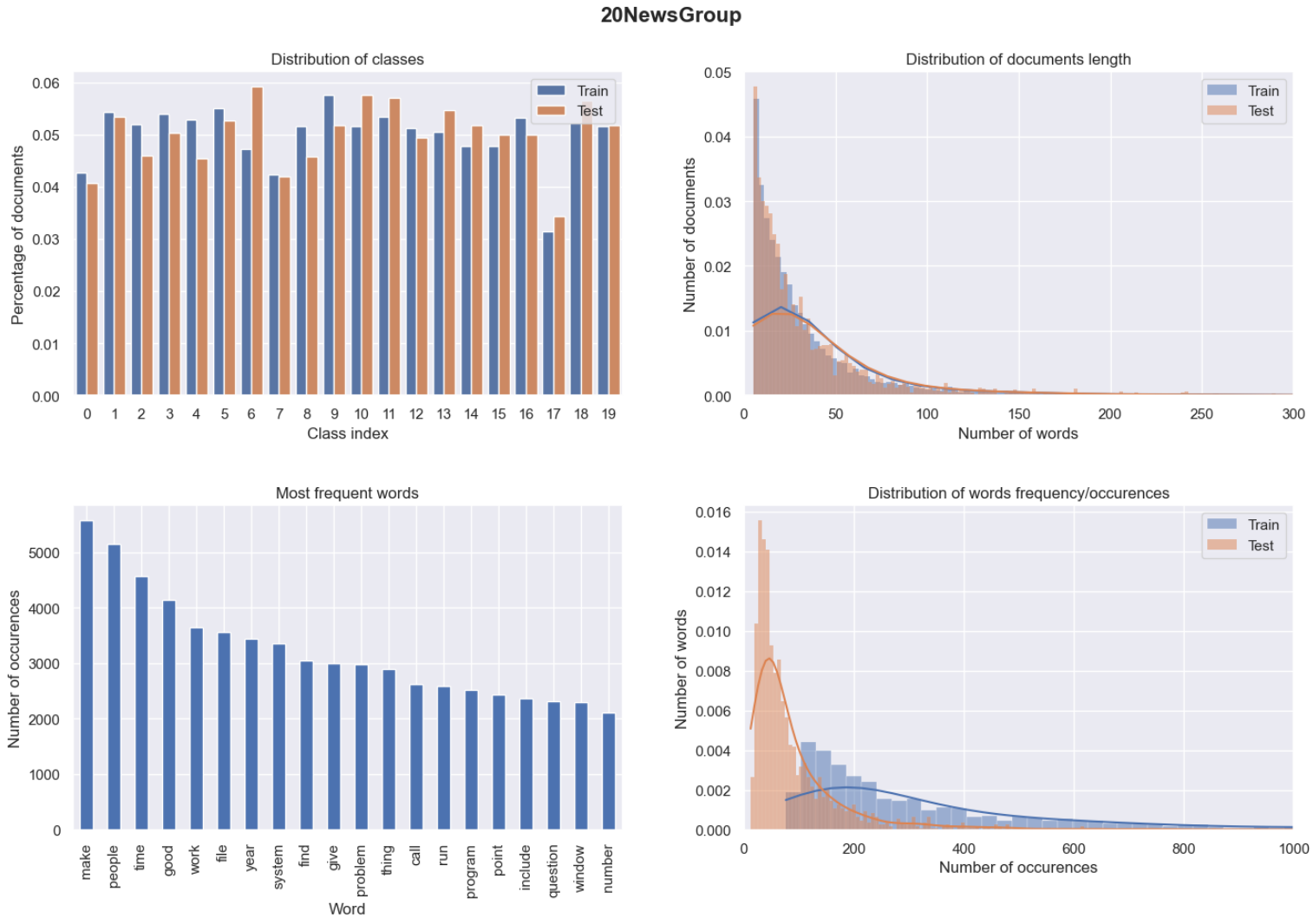


Figure: Information for the 20NewsGroup dataset.

Benchmark

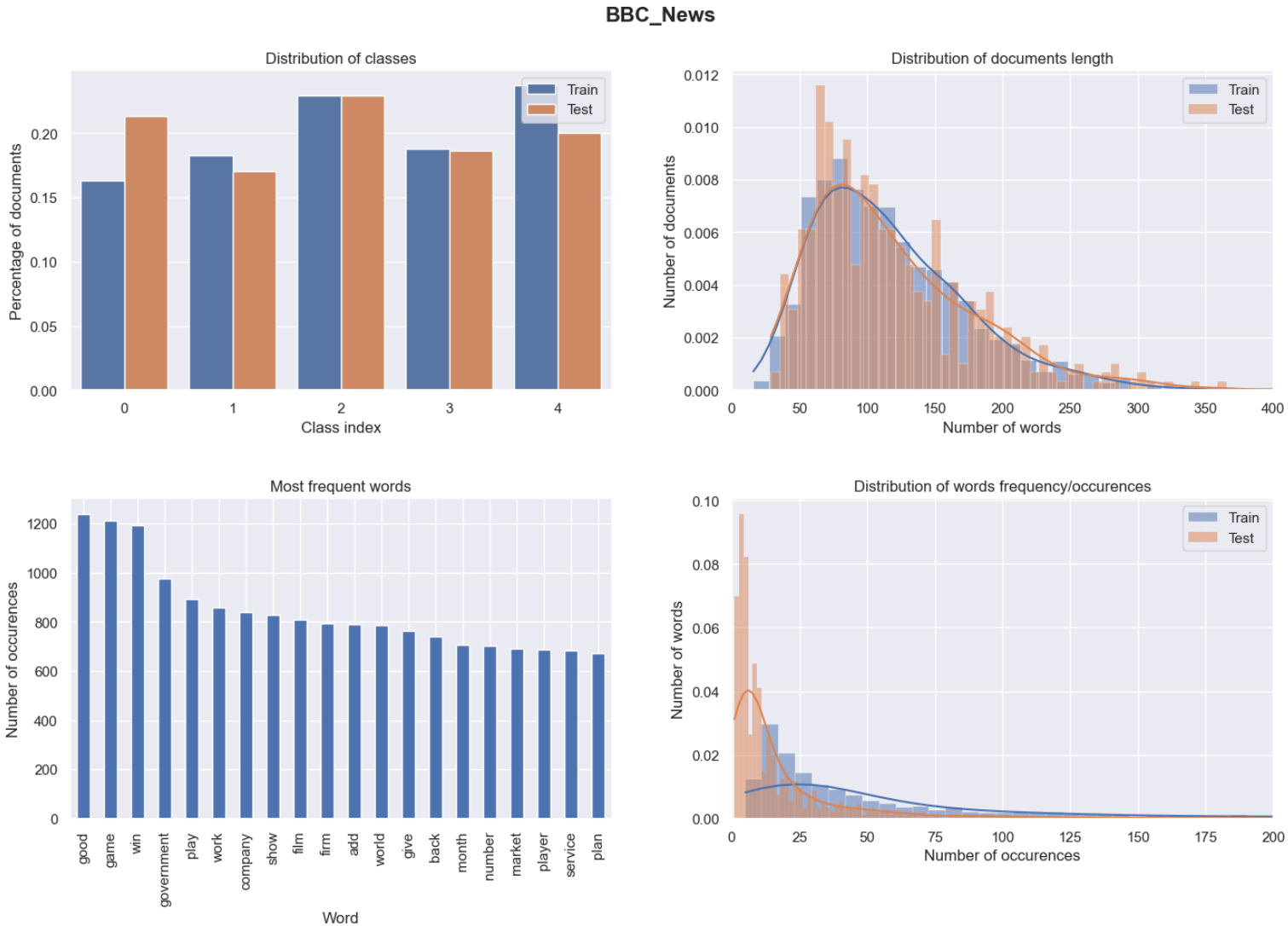


Figure: Information for the BBC News dataset.

Benchmark

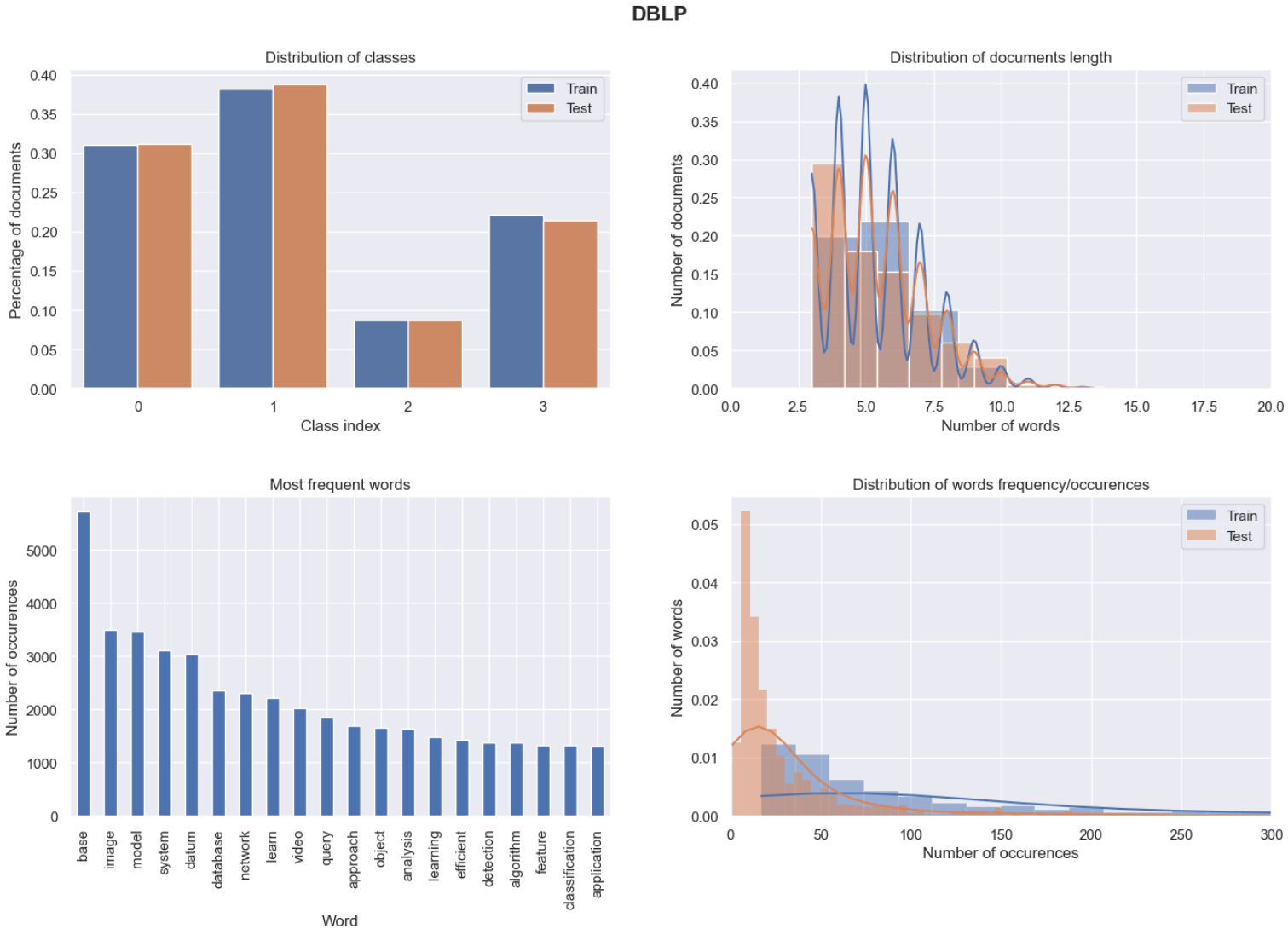


Figure: Information for the DBLP dataset.

Benchmark

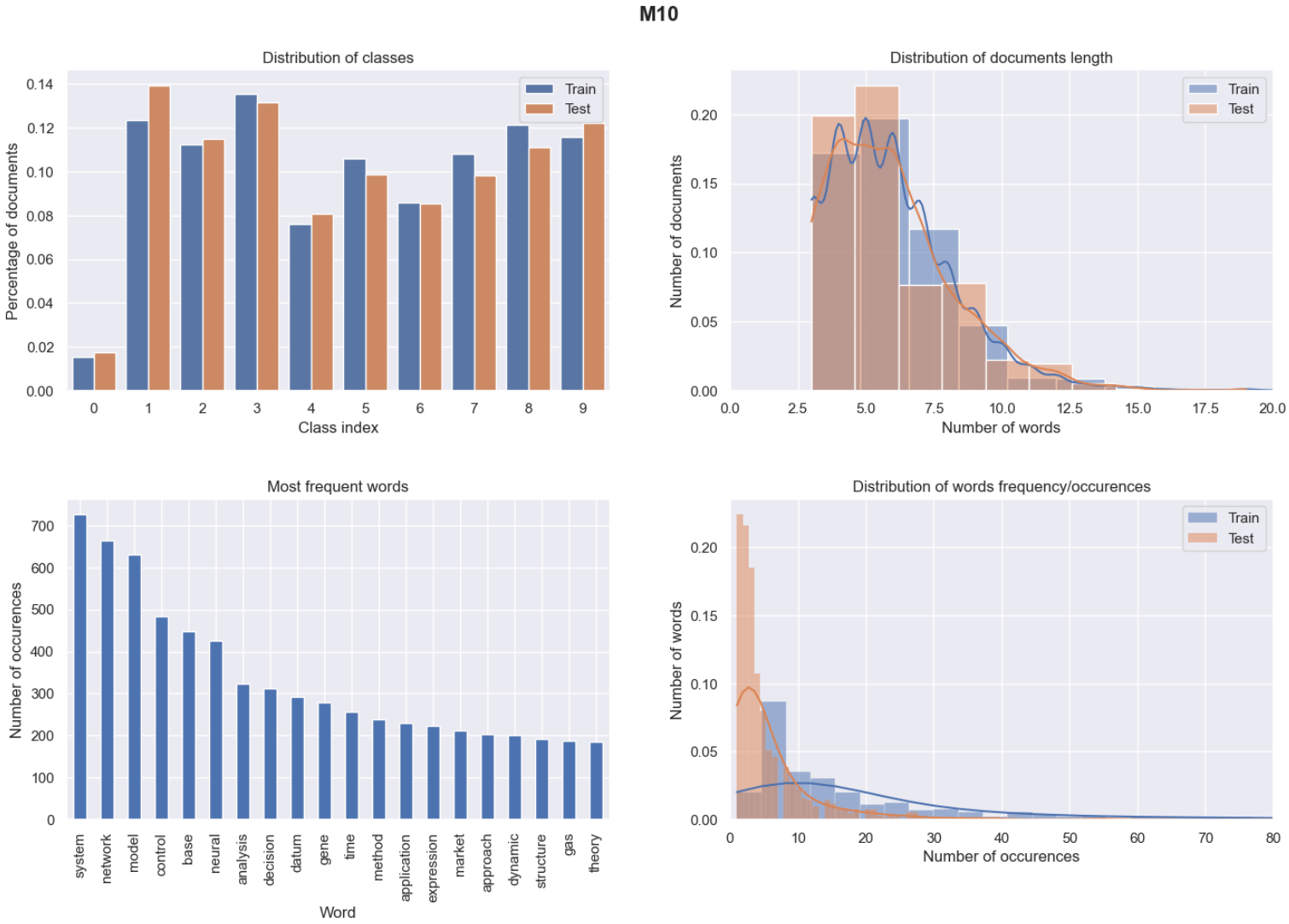


Figure: Information for the M10 dataset.

Overview

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 16.

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 ← K random points
while none of the cluster assignments change do
    for data points p do
        distance ← 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] ← average of assigned points
    end for
end while
```

KMeans

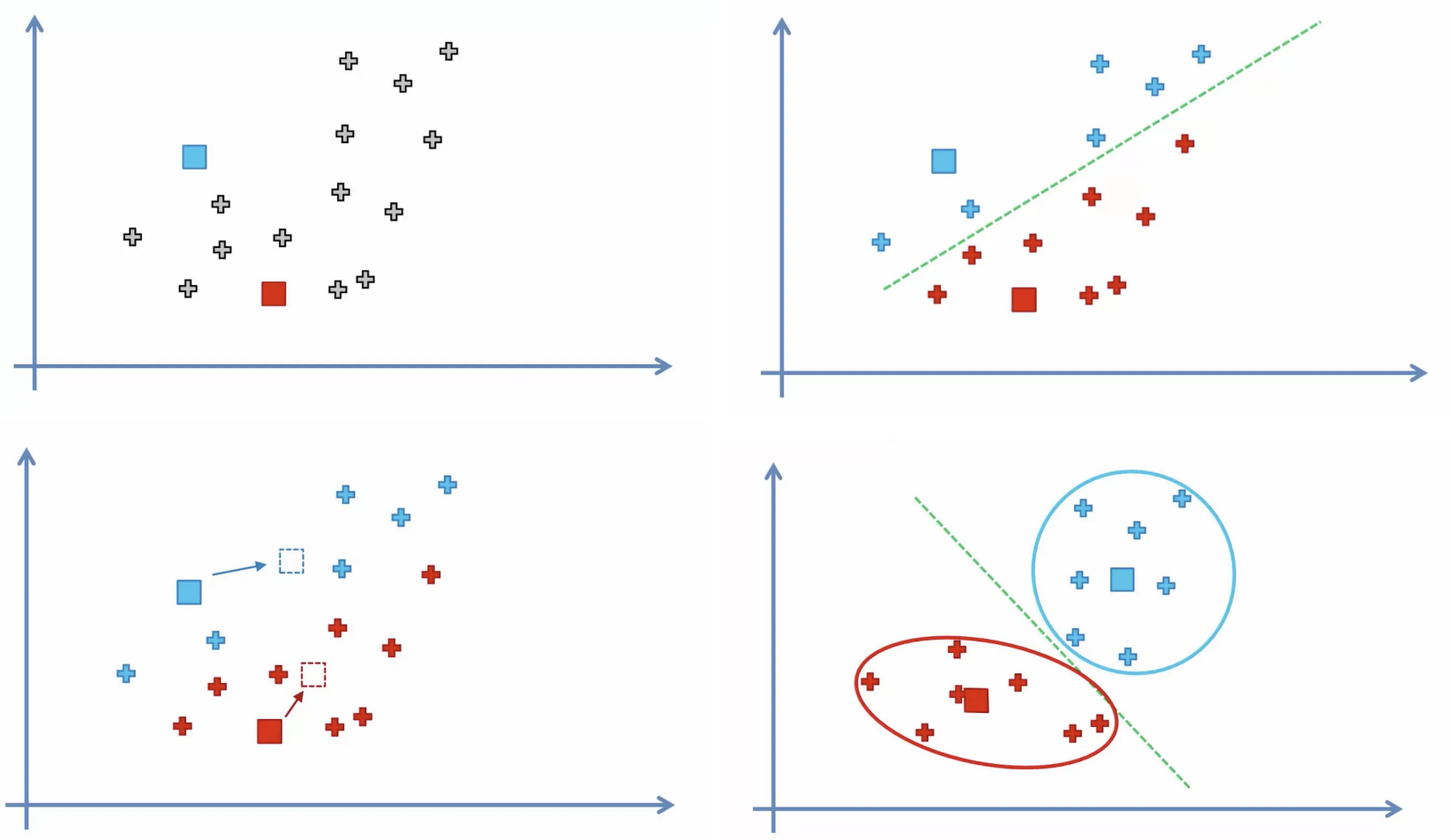


Figure: Steps for KMeans clustering algorithm from left to right.

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 20 and 21.

Link between clusters and classes

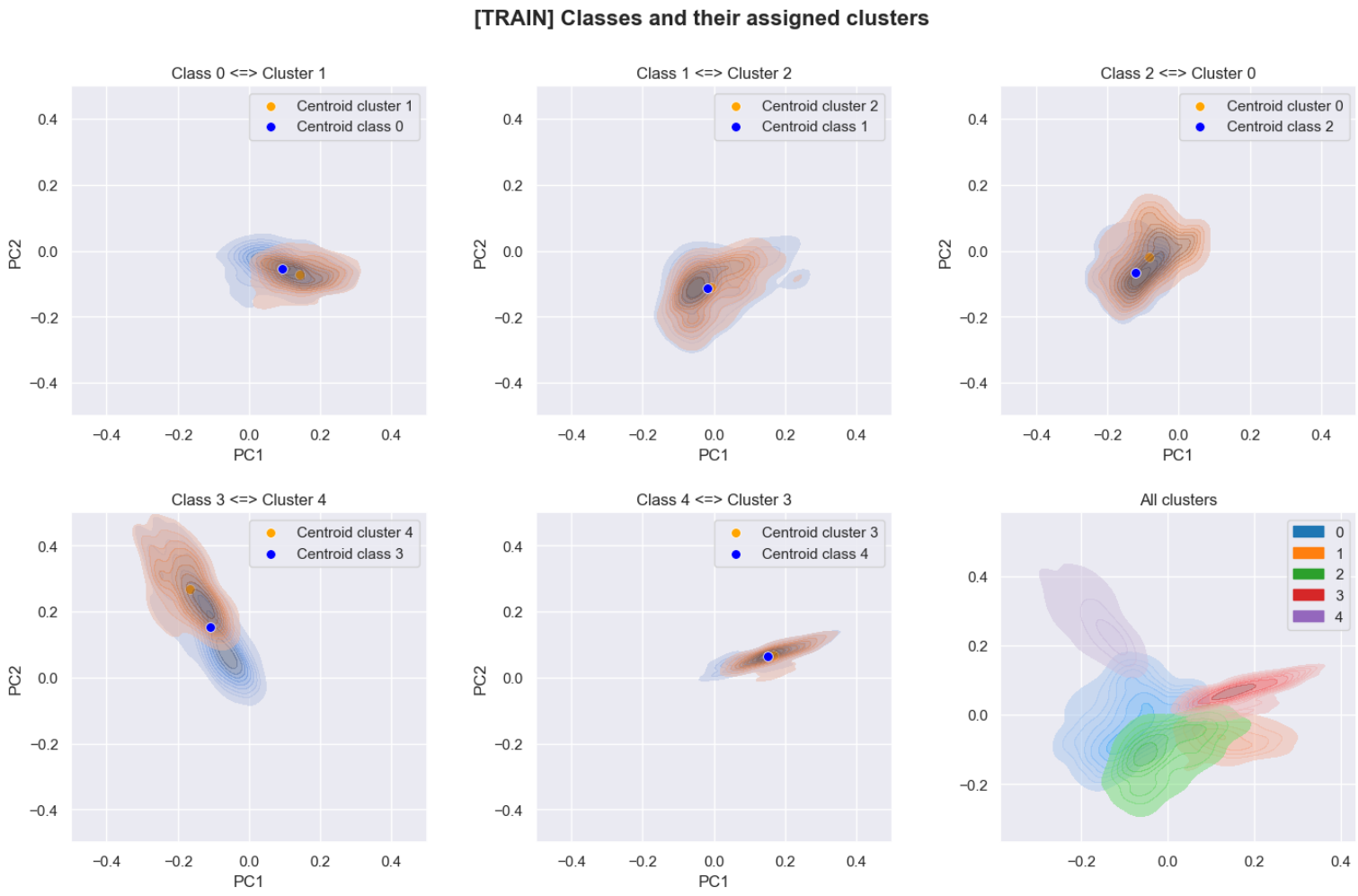


Figure: Classes and their assigned clusters for the train dataset.

Link between clusters and classes

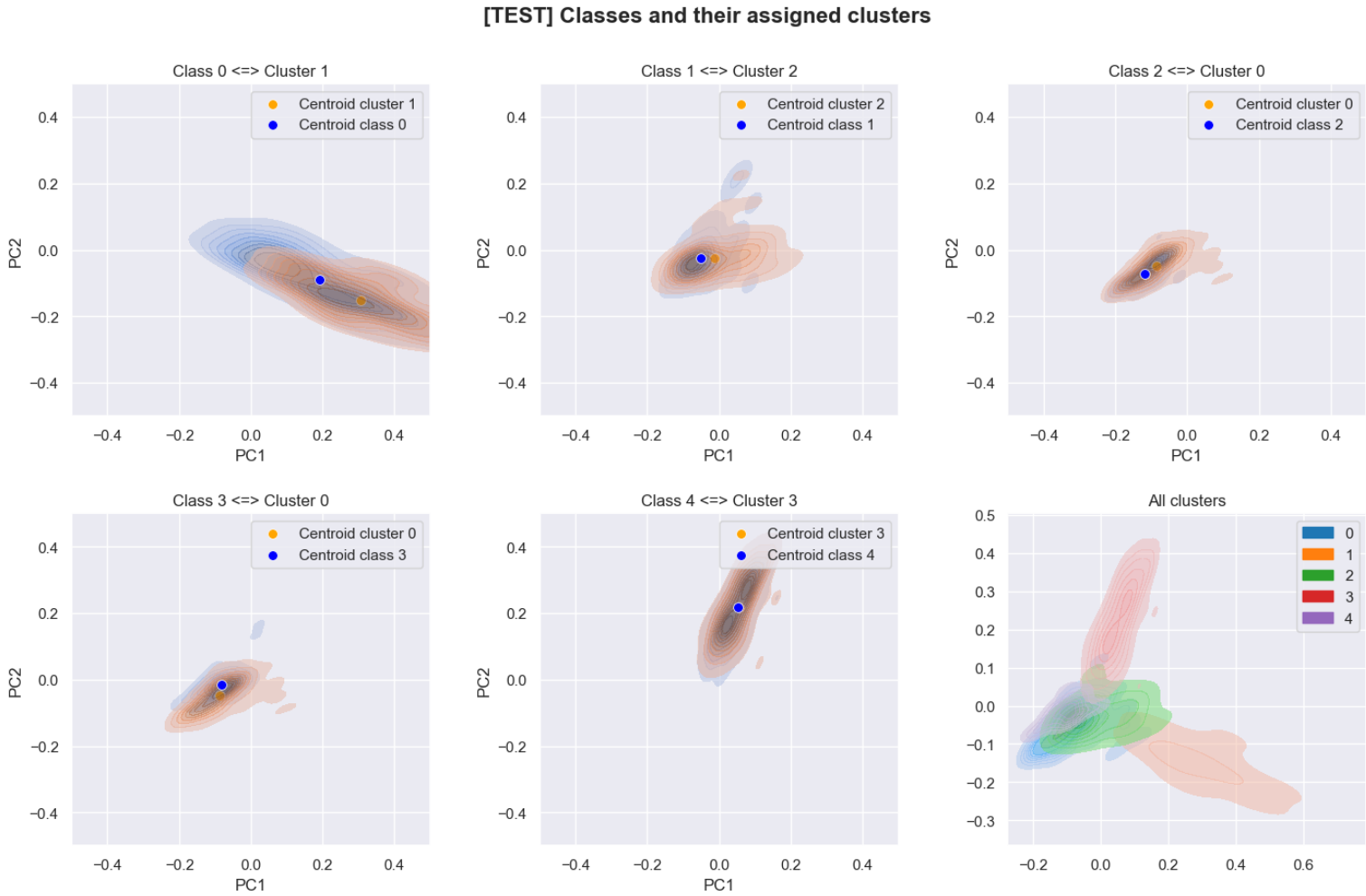


Figure: Classes and their assigned clusters for the test dataset.

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 22.

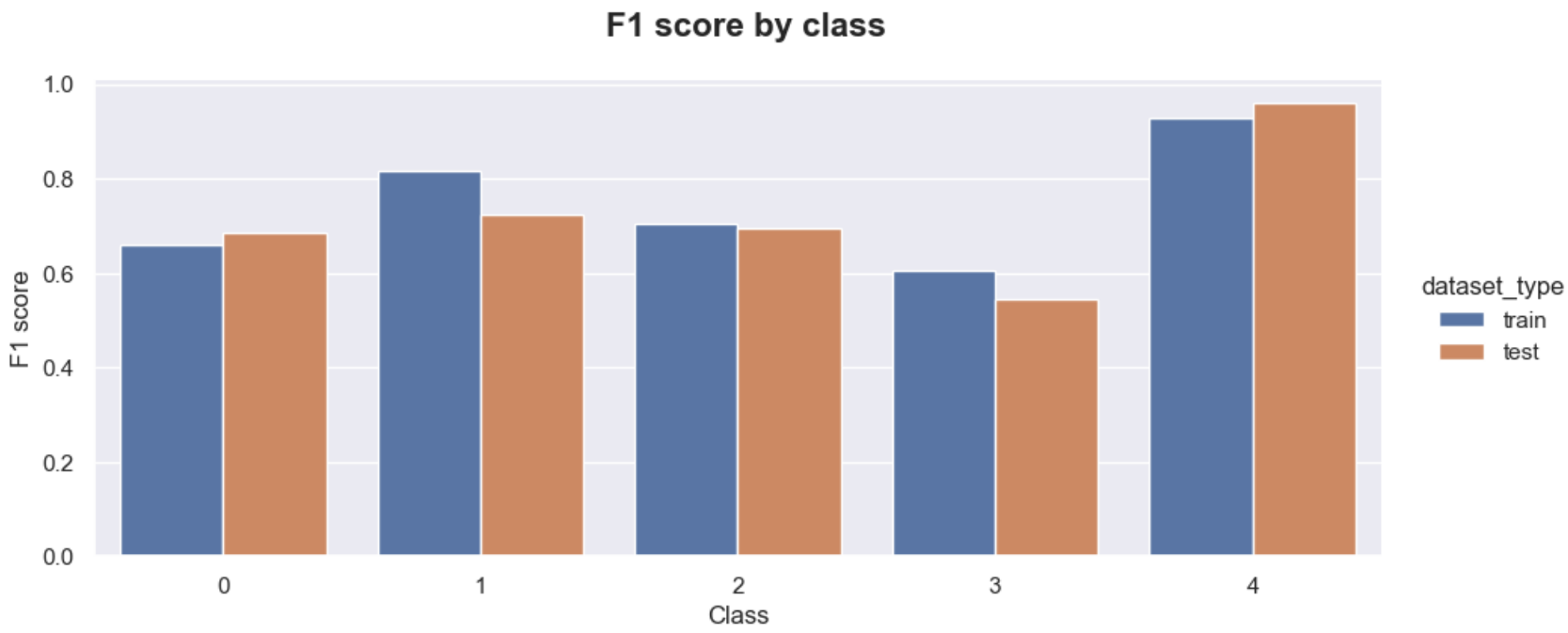


Figure: F1 scores for each class after using the train assignment of clusters.

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 25.

Semi-supervised assignment

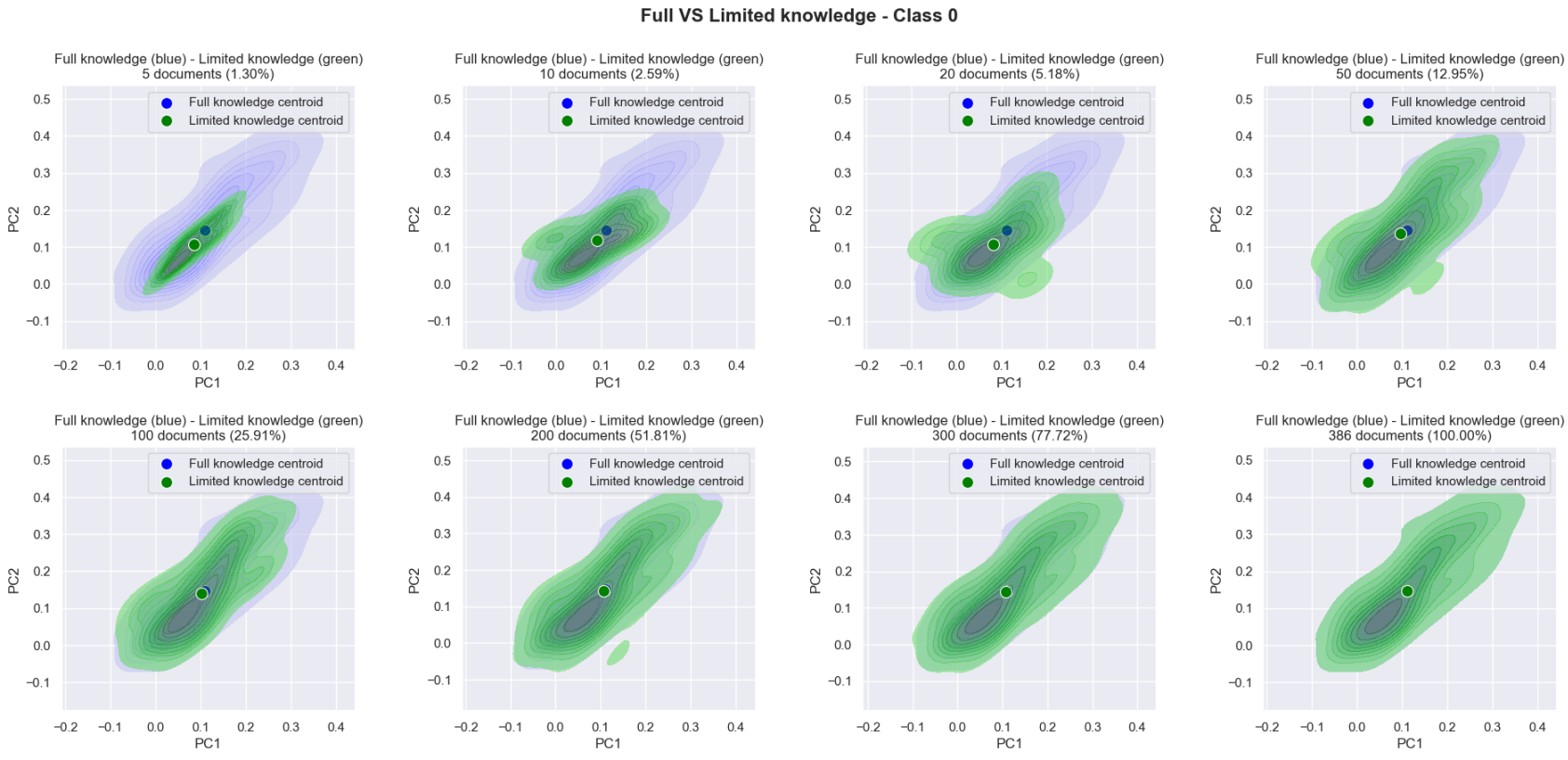


Figure: Difference in the centroids of class 0 between a full knowledge and a limited one.

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.

Lbl2Vec

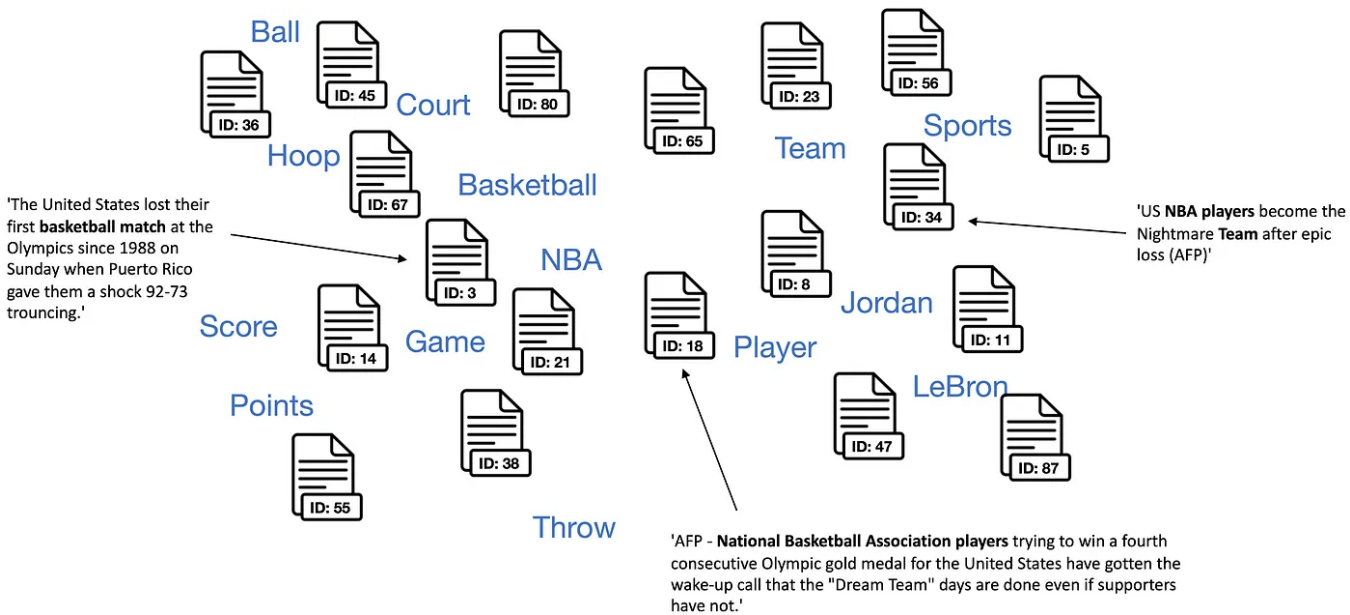
Lbl2Vec: unsupervised algorithm to classify documents in defined classes based on a predefined knowledge, which is the keywords associated with each classes. At a high level, this algorithm performs the following tasks:

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

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

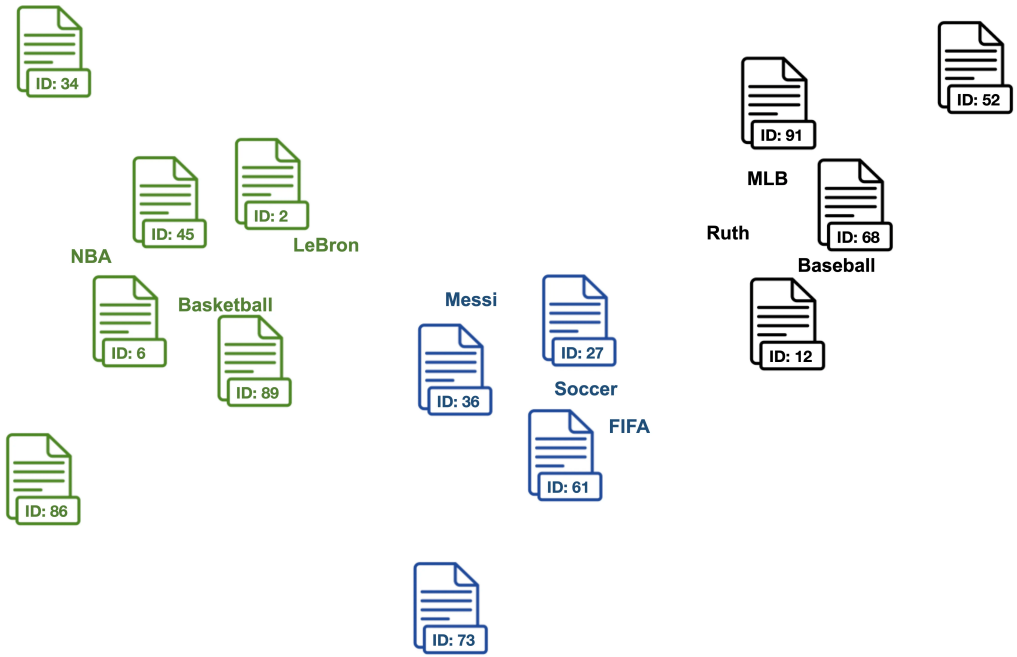
Lbl2Vec

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.



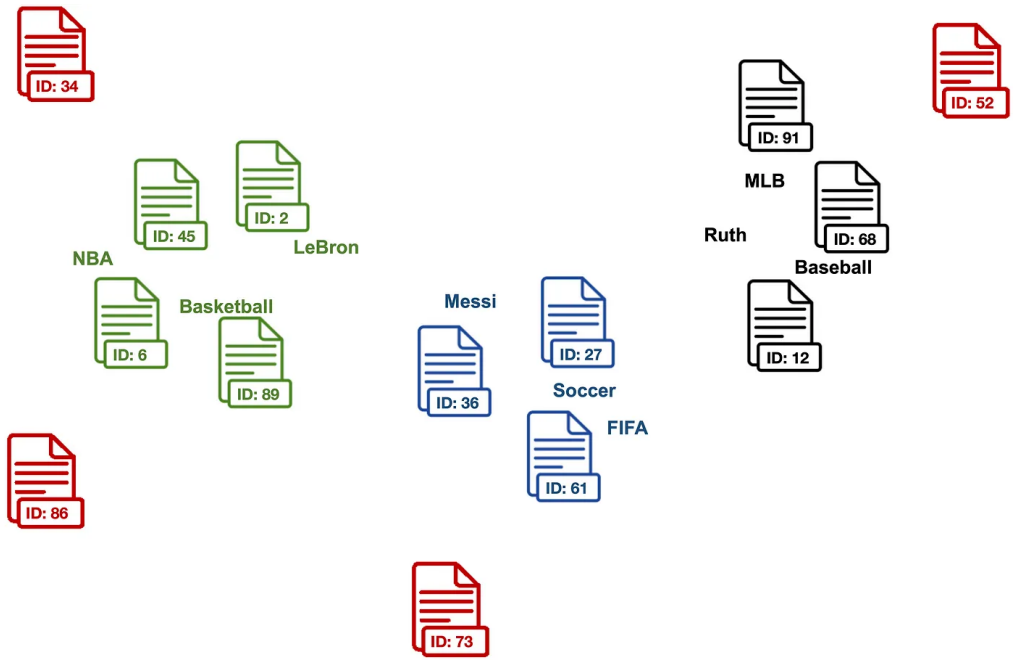
Lbl2Vec

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.



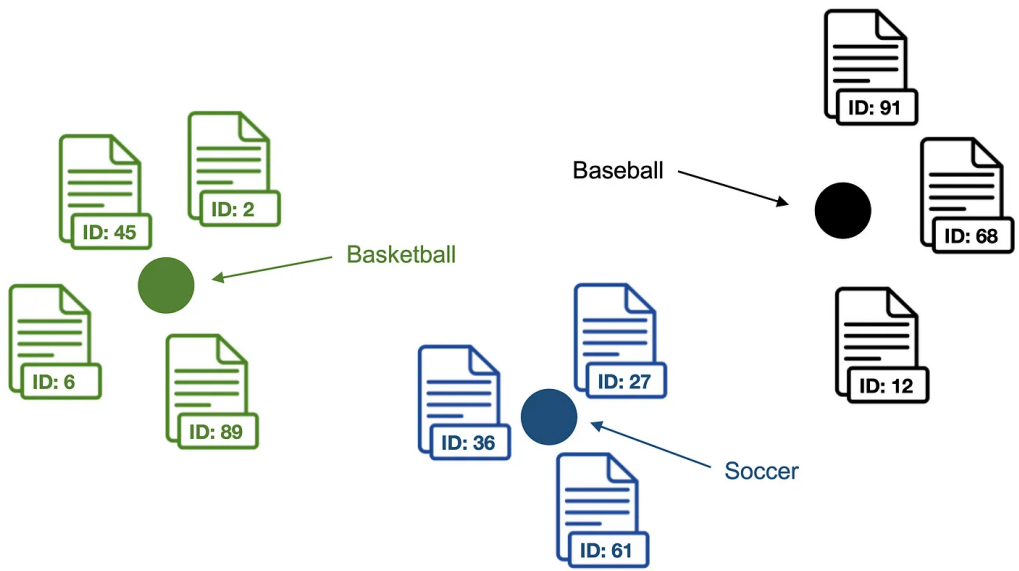
Lbl2Vec

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



Lbl2Vec

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



Lbl2Vec

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.