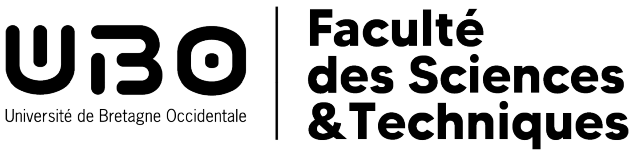
Augmented Clustering of Publication Themes

Encadré par : Mr. De Loor Pierre

Réalisé par : MELLOUKY Mohamed - NGUYEN Tien Dung



2023 - 2024

Outline

1 Introduction **2**

2 Algorithms Used **3**

2.1 Latent Dirichlet allocation (LDA) 3

2.2 BERTopic 4

2.3 Principal Component Analysis (PCA) 5

2.4 t-distributed Stochastic Neighbor Embedding (t-SNE) 5

3 Data Collection Methodology **6**

3.1 HAL API 6

3.2 Data Processing 6

4 Topic Extraction **8**

4.1 Topic Extraction with LDA Algorithm 8

4.2 Topic Extraction with BERTopic 9

5 Post processing **10**

5.1 Author Vector Computation From Last Name Token 10

5.2 Author Vector in Relation with Articles 10

5.3 GloVe Vector Embedding 12

5.4 Dimensionality Reduction 12

6 Data Visualization **13**

6.1 Visualization (plot 2D 3D) 13

7 Results **14**

7.1 Results of Simple Author Vector Computation 14

7.2 Results of Author vector computed from their contribution 14

8 Tasks Assignment **16**

Chapter 1

Introduction

The Lab-STICC is made up of several multi-disciplinary teams working in a variety of fields ranging from computer science sub-domains to digital sciences. The Lab is organized into several divisions, each of which contains several teams. The Interaction division comprises three teams: COMMEDIA, INUIT and RAMBO.

The COMMEDIA team connects psychology and computer science to enhance system engagement. The INUIT team works on evaluating and improving immersive technological solutions for increased naturalness. The RAMBO team explores machine learning techniques for autonomous interactive systems. The main objective of the Interaction division is to conduct research and develop innovative technologies that contribute to improving interactions between users and systems.

Because of the multidisciplinary nature of the Interaction division and the fact that new technologies evolve really fast, the boundary between each team becomes blurry. As a result, researchers' interests overlap. That's why, every five years, the division carries out a reallocation of team members. In this report, we will investigate unsupervised machine learning techniques for finding clusters of Interaction researcher members. This will enable us to think about how reallocation can be carried out optimally. First, we'll briefly present the algorithms used in our project. Next, we'll present the data collection process. We'll then look at data pre-processing. We will then look at how we used the Latent Dirichlet Algorithm (LDA) to find relevant topics. To calculate the distance between authors (Interaction members), we came up with several ideas that will also be discussed. Finally, we present our results.

Chapter 2

Algorithms Used

2.1 Latent Dirichlet allocation (LDA)

In natural language processing, latent Dirichlet allocation (LDA) is an algorithm that uses a Bayesian network to extract topics automatically from documents.

Following the intuition that the probability distribution over words in a topic is skewed, so that only a small set of words have high probability, the model extracts topics from documents by using the probability distribution of relevant words. The model does not have understanding of any word. However, semantic understanding is not what we really focus on in this project. Since the aim of this project is to find the relation/distance between researchers of three teams, in terms of interest or works relatedness, the word probability distribution per topic learned by this model can be useful.

If we consider the word probability distribution of each topic a vector, this topic terms vector is unique to each topic. Besides, each researcher's interest is not just one topic, but a combination of different topics. Thus if we can find a way to express the interest of each author in the form of a weighted sum of all topic terms vectors, we can see the distance in their interest. Moreover, we can also visualize the topic terms vector of relevant topics in the same space as authors "interest vector."

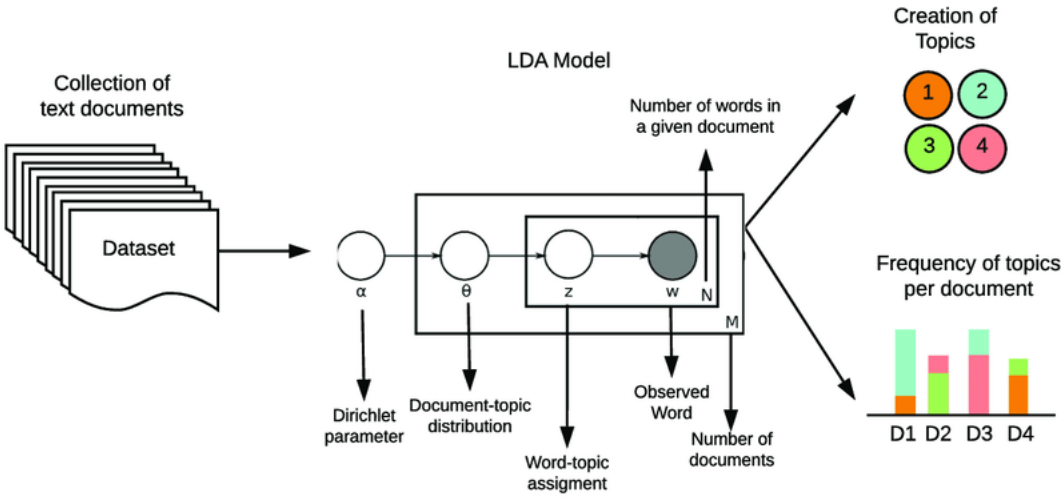


Figure 2.1: LDA plate structure (source: Analytics Vidhya)

In the figure above, W is the observed word. Only W is grayed out because it is the only observable variable. And the others are latent variables. Those are learned and update each time the model make a new observation (word/token). Amongst those latent variables, we are particularly interested in two matrix (document topic and topic word/term), which can give us two different approaches to compute the author interest vector

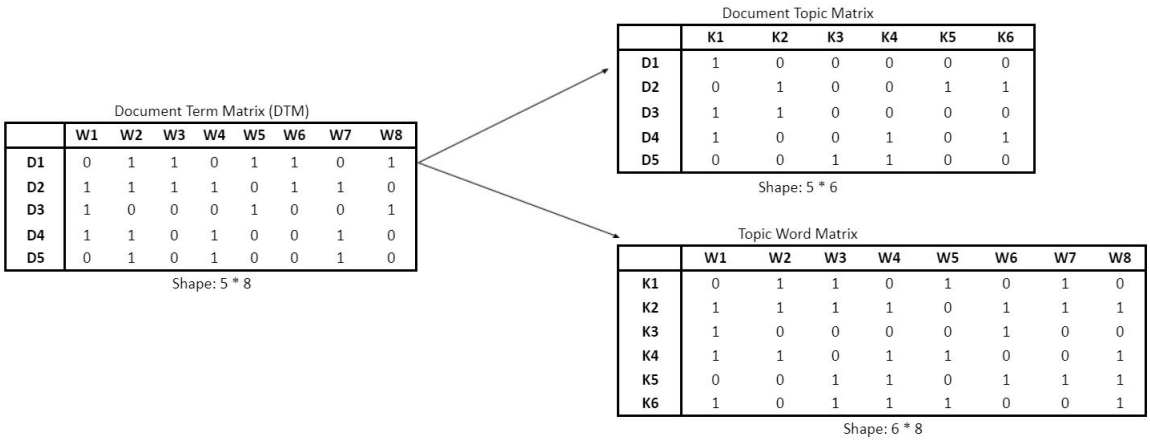


Figure 2.2: LDA variables (source: Analytics Vidhya)

The Document Term matrix contains the occurrence of every word in each document. Each line corresponds to a document, and each column corresponds to a word in the vocabulary (every word in all documents).

Before training, the number of topics K to find is determined by user. The Document Topic matrix list the probability distribution of K most relevant topics for each document.

Each topic also links to a word probability distribution. It is stored in the Topic Word or Topic Terms matrix, of which each line corresponds to a topic, and each column corresponds to the relevant of a word to the topic. However, due to the fact that this model does not learn the meaning of topics or words, each topic is represented by a number (ID). It's the job of user to find the most suitable keyword for each topic.

2.2 BERTopic

BERTopic is a topic modeling framework that allows users to create their version of a topic model. BERTopic is flexible and allows multiple implementation choices for each stage of the topic modeling process.

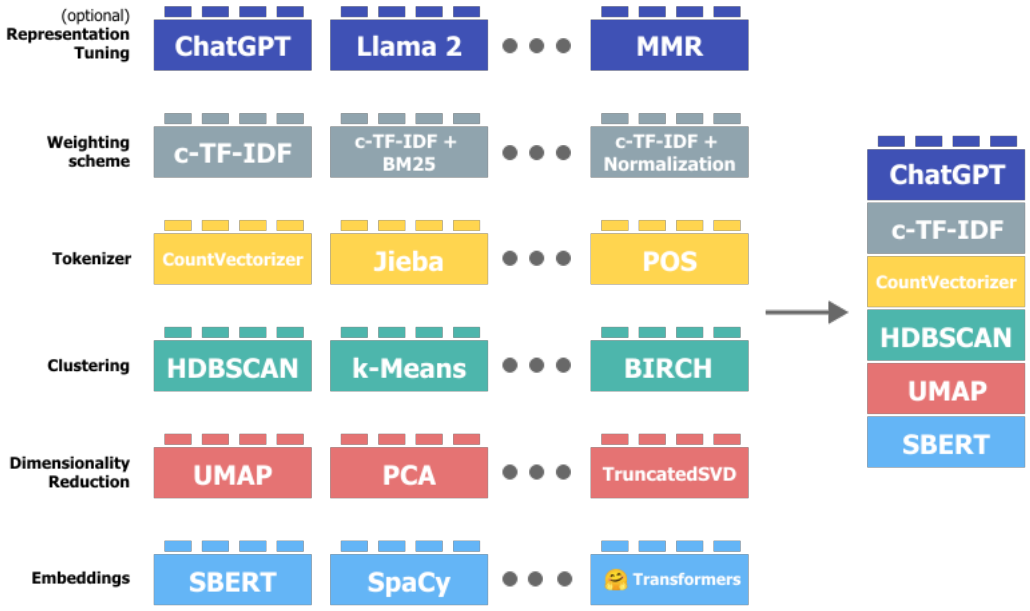


Figure 2.3: BERTopic modular structure (source: BERTopic)

BERTopic employs transformer and vector embedding, which allow the model to extract interpretable topics whilst keeping important words in the topic descriptions.

However, due to BERTopic models demand for very large dataset to be able to extract meaningful topics (more than 1000 documents), we will not prioritize its implementation. Because the data for provided is scarce (metadata of 138 articles).

2.3 Principal Component Analysis (PCA)

When a point is projected on a vector, its projection is a point in line with the vector. If the point is perpendicular with the vector, its projection is always vector zero (or no information about the position of the point is preserved). So depending on the angle between the vector and the points in the reference space, the projection of those points can preserve or loose a lot of information. Thus the aim of PCA is to find the vector to project every point in the data onto, while maximize the amount of information preserved (principal component).

When a principal component is found, the algorithm can continue to find another principal component that is perpendicular to the found ones. As long as the number of new dimensions is lower that that of the original, we can project every data point onto the new reference space created by new principal components. Depending on the number of dimension we want to preserve, the process of finding the principal component can continue until reaching the original dimension.

2.4 t-distributed Stochastic Neighbor Embedding (t-SNE)

Unlike PCA which is deterministic, t-SNE is a machine learning algorithm. The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of points in such a way that close points are assigned a higher probability. Second, t-SNE defines a mapping from that high dimensional distribution to a lower one, while minimizing the Kullback–Leibler divergence between the two distributions with respect to the locations of the points in the map.

The nondeterministic nature of this algorithm makes it inconsistent. Each launch gives a different result. Despite that, this algorithm is more promising than PCA, because it tries to minimize the divergence in position distribution of the points in space. While PCA only focus on "information", which makes the interpretation of new axis difficult.

Chapter 3

Data Collection Methodology

3.1 HAL API

HAL stands for *Hyper Article en Ligne* which means *Hyper articles online*. This is an open repository in which academics and researchers can deposit their publications, articles and conference papers online, freely accessible to academia. The HAL API allows specific data to be obtained via HTTP requests. The available fields are well documented in the **official HAL API documentation**.

For the purposes of our project, we used the following API URL: **https://api.archives-ouvertes.fr/search/?**. Using **https://api.archives-ouvertes.fr/docs/search/?schema=fieldsfields** we could check the available fields.

We developed a python script that retrieves the title, abstract, keywords, authors, year of publication and language of all articles published between 2020 and 2023. We retrieved the year of publication because the intention was to visualize the evolution of interest among Interaction division members over the four years. However, this was not done due to time constraints. The language field was retrieved because it was intended to be used to filter French articles from those published in English. However, we found that this field was not useful in our case because, in some cases, the API returns the title and abstract of an article in both languages (French and English).

The python script is located in the *data_collection* folder, which has this structure:

```
data_collection/
          collect.py
          data
                  LAB-STICC_COMMEDIA_data.csv
                  LAB-STICC_INUIT_data.csv
                  LAB-STICC_RAMBO_data.csv
          params.py
          writer.py
```

Where *data* is the folder in which the script stores the CSV files returned by the API. In fact, we have the *collect.py* script which contains a *Data_collector* singleton class that contains the functionalities for retrieving data from the API using HTTP requests. In addition, we have a *writer.py* script which is a python class that writes the results of *Data_collection* objects to CSV files. It implements the strategy design pattern to improve code re-usability.

3.2 Data Processing

Our first observation when trying the LDA topic model on raw data is that if an article's metadata lacks the abstract, it's topics will become irrelevant, and contain only names of authors. Because without abstract, one metadata is made up predominantly from nothing else but author's names

Our second observation is that some articles have only French or both English and French. This dilutes the topics found because one word now is presented in two version, English and French.

3.2.1 Data Cleaning

So the first step is to find all missing abstracts and English version of some article's metadata. If there isn't an English version available, we will try to translate it into English, as accurate as possible.

Our first approach is trying to automate the process by using GPT4All pre-trained model, since there a some models that are really good at understanding and translating languages.

The process is straightforward: we first choose a model and download it, then we can open a chat session to ask the model to do the work for us.

```
representation_model = GPT4All("gpt4all-13b-snoozy-q4_0.gguf")
...
    with model.chat_session():
        response = model.generate(prompt=f"Please translate this text to english, don't add anything
            else:\n{text}", temp=-5)
        print(response)
```

We tried multiple prompts, changing the model's parameters, change to bigger models, etc. Even with instruction based model, the response sometimes contains a rephrase of the question, or a summary at the end, or random greeting. Due to inconsistency in the model's responses, we decided to do everything manually.

3.2.2 Data Transform

Next step is to transform text data into data structure that can be feed to a topic modeling model.

The LDA model only takes in a dictionary or vocabulary of all words in the document set, and a corpus containing word frequency for each word in each document. Thus we need some steps to prepare data for LDA model:

- Remove special characters.
- Replace characters with accent into normal characters, convert characters to lower case.
- Tokenize each word.
- Remove stop-word that does not have much meaning like is, and, or, etc.
- Convert word into their original form or stem (to avoid diluting the dataset: only compute instead of computation or computing).
- Finally the dictionary of all words and a corpus for all document are created from the tokens selected.

Chapter 4

Topic Extraction

4.1 Topic Extraction with LDA Algorithm

After the data pre-processing stage, we now have a dictionary that associates an integer ID with each word. We have also the corpus, which is represented as a collection of pairs of integers, where each pair consists of a word's integer ID and its frequency. The LDA algorithm implemented by the gensim python library needs these two variables, in addition , it needs the number of topics we want to obtain and the number of passes. The number of passes represents the number of iterations the algorithm performs during training.

In our code base, you may notice that we run the LDA algorithm four times. We run the LDA algorithm for each team in the division and the fourth time for the data from all three teams together. We made this decision so that we could also visualize the distance between authors on the same team. As all the points in space in this case represent members of the same team, we won't have any clusters. However, it is always useful to check whether the members of a team are concentrated in a certain region of space, and if so, which authors are far from this region.

This is an example of a call to the LDA constructor that invokes algorithm learning. The argument names are self-explanatory and their structure is explained above.

```
lda_model_all = models.LdaModel(corpus_all, num_topics=NUM_TOPICS, id2word=dictionary_all, passes=
    NUM_PASSES)
```

You may wonder why we don't mention data splitting before model training. In fact, splitting data into training and test data is a good supervised learning technique, allowing machine learning specialists to measure how well their models generalize on unseen data. However, in our context, we don't have the notion of labels on our data, so we don't have a measure of a model's effectiveness.

We varied the value of the number of passes between 100, 1000, 10000 and 100000. In the first three runs (100, 1000, 10000), the algorithm gives almost the same probability distribution of words per topic. Training took 7.0 s for 100 passes and 1 minute and 21 second for 1000 passes and 12m 7s for 10000 passes on a machine with the following configuration: Intel I7-11th generation processor, 16 Gb Ram, NVIDIA MX450. However, training was run on the CPU running Ubuntu 22.04 LTS. Nevertheless, the training lasted 105 minutes and 20 seconds for 100,000 passes on the same machine. And it gave a completely different distribution of words per subject.

Here's an example of the word probability distribution found for a topic for each number of passes.

```
# NUM_PASSES = 100
(0, '0.024*"robot" + 0.015*"system" + 0.015*"physical" + 0.014*"group" + 0.012*"rehabilitation" +
    0.012*"exercise" + 0.010*"human" + 0.010*"pain" + 0.009*"rgb" + 0.009*"movement"'),

# NUM_PASSES = 1000

(0, '0.024*"robot" + 0.015*"system" + 0.015*"physical" + 0.014*"group" + 0.012*"rehabilitation" +
    0.012*"exercise" + 0.010*"human" + 0.010*"pain" + 0.009*"rgb" + 0.009*"movement"'),

# NUM_PASSES = 10000

(0, '0.024*"robot" + 0.015*"system" + 0.015*"physical" + 0.014*"group" + 0.012*"rehabilitation" +
    0.012*"exercise" + 0.010*"human" + 0.010*"pain" + 0.009*"rgb" + 0.009*"movement"'),

# NUM_PASSES = 100000

(0, '0.024*"virtual" + 0.023*"agent" + 0.021*"leadership" + 0.019*"interaction" + 0.014*"medical" +
    0.014*"assistant" + 0.013*"human" + 0.013*"caregiver" + 0.011*"follower" + 0.011*"situational"')
    ,
```

4.2 Topic Extraction with BERTopic

After having done testing and visualizing the results of LDA, we tried BERTopic - the second most interesting algorithm for the objective of this project.

The model modularity and extreme flexibility makes its learning curve steep, and hard to master. Thus we opted for the safer options: BERTopic basic model, and one with *SentenceTransformer* as vector embedding model, *Hdbscan* as clustering algorithm and use *Term Frequency - Inverse Document Frequency (TF-IDF)* to emerge topics from clusters of documents.

However, typical BERTopic model requires thousands of documents to be able to extract meaningful topics. The metadata we used contained only 138 articles descriptions. Thus the performance is poor, as shown in the figure below:

	Topic	Count	Name	Representation	Representative_Docs
0	-1	14	-1_the_and_leadership_agent	[the, and, leadership, agent, wordplay, medica...	[Designing Speech with Computational Linguisti...
1	0	68	0_the_of_to_and	[the, of, to, and, in, learning, for, we, on, by]	[Computer Vision and Robot Navigation in 3D En...
2	1	44	1_the_of_to_and	[the, of, to, and, in, virtual, reality, for, ...	[Virtual Workspace Positioning Techniques duri...
3	2	12	2_the_activity_recognition_smart	[the, activity, recognition, smart, of, and, i...	[A Survey of Human Activity Recognition in Sma...

Figure 4.1: BERTopic basic model result

	Topic	doc
0	-1	Commonsense Reasoning for Identifying and Unde...
1	0	Improving Neural Architecture Search by Mixing...
2	1	Indifferent or Enthusiastic? Virtual Audiences...

Figure 4.2: BERTopic's result with SentenceTransformer, Hdbscan, and TF-IDF

Chapter 5

Post processing

5.1 Author Vector Computation From Last Name Token

The visualization requires post-processing the LDA results. LDA returns a word probability distribution by topic. The LDA result can be represented by a $K \times M$ matrix, where K is the number of topics and M is the number of words. More concretely, each word in the topic_term matrix represents a token. Since the author's name is a set of tokens (usually two or three), we can use this matrix to calculate a vector representing each author, which we call the author vector. However, we only take into account the author's last name, as it is very rare to have two identical last names. We made this choice because it was difficult to predict the number of tokens an author's name contains.

If we consider the following matrix as a topic_term matrix produced by an LDA algorithm

$$\begin{array}{c|ccccc} & W_0 & W_1 & W_2 & \dots & W_n \\ T_1 & 0.1 & 0.01 & 0.2 & \dots & 0.02 \\ T_2 & 0.01 & 0.01 & 0.2 & \dots & 0.3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ T_k & 0.5 & 0.2 & 0.01 & \dots & 0.01 \end{array}$$

Let W_2 be a token representing an author's last name. Therefore, its vector would be expressed as a linear combination of topic vectors, as follows:

$$\vec{W_2} = 0.2\vec{T_1} + 0.2\vec{T_2} + \ldots + 0.1\vec{T}$$

As we can see, the topic vectors can also be expressed as linear combinations of word vectors. Consequently, the author vector expressed as a linear combination of words takes the form :

$$\sum_{i=1}^{N} c_i \mathbf{W}_i$$

where N is the number of words, $\mathbf{W}_i$ is the word vector, and c_i is a scalar in $\mathbb{R}$.

5.2 Author Vector in Relation with Articles

The author vector calculated above is a simple word distribution probability. Therefore, we can use that distribution as a author vector and visualize it in a 2D or 3D space (after dimensionality reduction, as the author vector is highly dimensional). In this section, we considered the author's interest in relation with his/her contribution to the articles. As far as we know, the first author of an article is the one who contributes the most, so we thought of adding a weight to the first author who is more important than the other authors. This weight represents the author's interest in the article's topics.

One article corresponds to a probability distribution of topics. One author usually works on multiple articles, thus, they in turn also correspond to a topic probability distribution. We go further and breakdown that topic probability distribution into a weighted sum of all the topic vectors, which are made of a probability distribution of words. As a result, we obtain the same type of author vector as the first method.

We follow two approaches:

1. The first, which we call the static approach, considers that the interest of the first author of an article is twice as great as the interest of the other authors.

2. The second approach is more dynamic. The weight of authors is calculated using a formula deduced from two assumptions.

5.2.1 Static approach

We described in the previous section that we can express the author vector (A) as a linear combination of document vector components (D).

To elaborate further: Let A be the author vector, Let D be the document vector,

$$A = \alpha D + \beta D + \ldots + \beta D$$

Here, α and β represent coefficients that capture the contributions of an author to articles. α is the coefficient of the document for which it is the first author. β the coefficient of all other documents

We have chosen to normalize the coefficients in the linear combination of document vectors that compose the author vector. Normalization is a mathematical process that ensures the sum of these coefficients equals 1.

$$2\alpha + \sum_{i}^{\text{n-1}} \beta_i = 1$$

by expanding this sum, we obtain the values of $alpha$ and $beta$ as follows:

$$\alpha = \frac{2}{n+1}$$

$$\beta_i = \frac{1}{n+1} \quad \text{for } i = 1, 2, \ldots, n-1$$

We therefore use these equations in our code base, in particular in the *get_authors_vector_from_doc* function.

5.2.2 Dynamic approach

We argued that giving the first author twice as much weight of interest as the other authors is random and not justifiable. However, this choice was intuitive and simply aimed at favoring the first contributor to add as a scale. We therefore tackled this question using our new approach.

We have thought about generalizing the values of α and β. Therefore, we made two assumptions:

- The sum of all author's contribution to one article is 1.
- The sum of contributions of each author to all of their articles is roughly the same as other author.

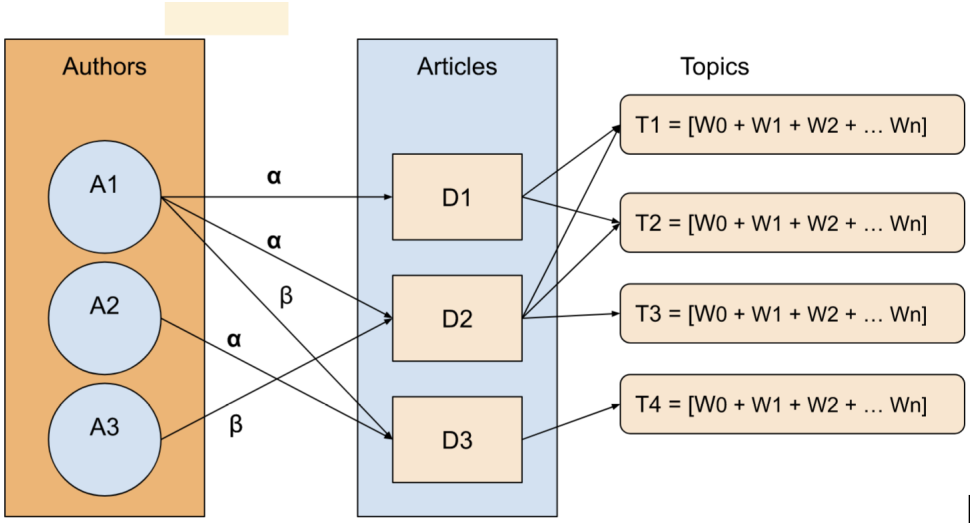


Figure 5.1: Relation between authors, documents and topic's word distribution

From the figure above, the two assumptions, and some data, we can compute Alpha and Beta that roughly contribute the contribution of authors. Below are some of the measures needed to compute Alpha and Beta:

- The total number of arcs is the total number of author names mentioned in the front page of all article: 599
- Number of researchers and Professors: 21 and 28
- Number of first author from three teams: 45 researchers, 14 professors
- Number of other author (second to last): 7 researcher and 166 professors
- There are 138 articles in total
- There are 185 authors mentions in total (49 from the lab)

This is the equation to compute those weights:

$$45\alpha + 7\beta = \frac{21}{185} \times 138$$
$$14\alpha + 166\beta = \frac{28}{185} \times 138$$

5.3 GloVe Vector Embedding

Due to the capacity to represent words meaning and their relation, word embedding vectors allows us to infer the meaning of a combination of words. For example, if we add two vectors representing the word *adult* and *male*, we will have a new vector that is close to the vector of the word *man*. Thus our idea of expressing relations between authors, and their interest (keywords, topics).
Since author vector can be represented as a word probability distribution, as well as topic vector. If we convert every word into a embedding vector, we can compute a weighted sum of all those embedding vector. We theorize that the resulted weighted sum will roughly represent the authors interest, as well as topics, in a "semantic space". Additionally, we can also find the appropriate word for each topic for the visualization.
However, we later found out that there is no vector representation for complex concepts, or compound words. The resulted nearest words for topics are only simple words like *virtual*, *robot*, *system*, *learning*, etc. We decided not to go deeper since that hampers half of our purpose for this approach.

5.4 Dimensionality Reduction

In the previous sections, we described how we calculated the author vector. The author vector represents an author in an n-dimensional space. However, the author vector is a high-dimensional vector, and to visualize it, we need to reduce its dimension to three or two dimensions.

In our project, we used two dimensionality reduction algorithms. PCA and t-SNE. The former is a deterministic algorithm that gives the same result when run on the same input data with the same parameters. The latter is a stochastic algorithm that aims to preserve the relative similarity between data points. It evaluates the similarity of points in the original high-dimensional space and attempts to reflect these similarities in a lower-dimensional space, which is useful for visualization.

We have implemented the PCA dimensionality reduction using the PCA class in the *decomposition* module of the *scikit-learn* python library. However, our function *pca_dim_reduction* also adds some additional data checking to ensure that the PCA constructor receives data in the desired format and to avoid errors and bugs in addition to certain user logs, such as the original shape of the data and the reduced shape of the data.

We've done the same with the t-SNE algorithm. We have encapsulated the dimensionality reduction process in the *tsne_dim_reduction* function, which uses the scikit-learn *manifold* module and uses a default perplexity of 5.

The result of both functions is an n_component-dimension vector to be used for visualization.

Chapter 6

Data Visualization

6.1 Visualization (plot 2D 3D)

In the previous section, we discussed dimensionality reduction. In fact, this process was specifically developed to visualize higher-dimensional vectors in two- or three-dimensional space.

The plan was to use these vectors to create an interactive visualizer. The latter features zoom and a slider to visualize the evolution of the author's interest over time. we were unable to achieve these objectives due to insufficient time constraints. We do, however, have a static visualization created using the *matplotlib* python library. It shows how the authors are arranged in 2D or 3D space.

In our project, we created two functions. One for 2D visualization and one for 3D visualization.

By default, it shows how the authors are distributed in 2D space as simple points without annotations. Each team is color-coded in this visualization. This gives a general idea of how teams overlap or are grouped together.

As we couldn't see any clear clusters, we decided to create a 3D visualization. We also decided to create a 3D visualization. It may be useful to examine how authors are distributed in 3D space. So we re-run the dimensionality reduction algorithms to reduce the dimensions to 3D vectors and created a *plot_data_3d* function that has the same functionality as the simple *plot_data* function.

Here is the prototype of the *plot_data* function:

```
def plot_data(data, author_names, color, label, title, xlabel="Dim1", ylabel="Dim2", plot_annotation
    =False)
```

The visualization outputs will be presented and discussed in the results section.

Chapter 7

Results

7.1 Results of Simple Author Vector Computation

In the previous sections, we explained how we calculated the simple author vector and how dimensionality reduction is performed using PCA and t-SNE. In this section, we present the results of visualizing this simple approach with the two dimensionality reduction methods.

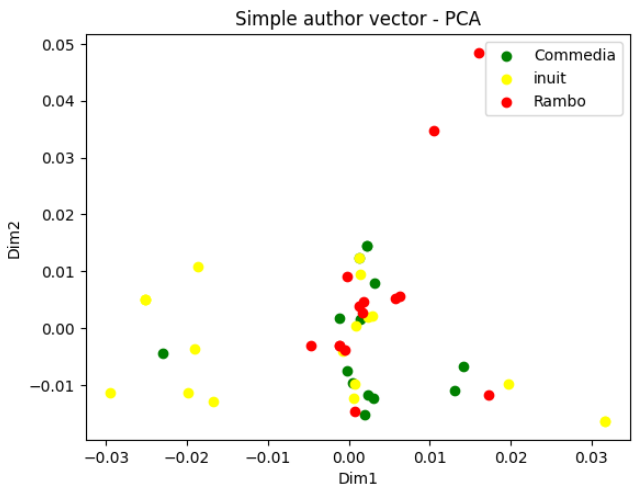


Figure 7.1: Simple author vector visualized using PCA

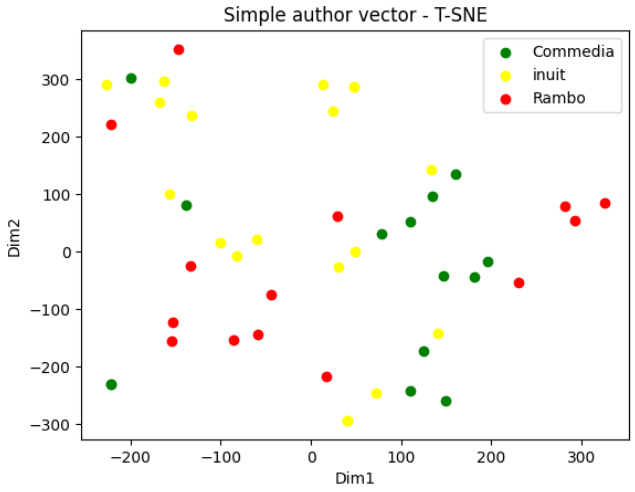


Figure 7.2: Simple author vector visualized using t-SNE

We can see that the clusters are not very clear. However, we are still able to define some boundaries between team members. This visualization also suggests that team members' interests overlap. We can also see that the PCA does not give clear clusters. However, we believe that by zooming in (using an interactive visualizer), we could get a clearer view of the clusters.

7.2 Results of Author vector computed from their contribution

7.2.1 Static weights

In this section, we present results computed using static weights. Particularly, we attribute the first author twice the contribution of the other authors. The same as the previous section, we visualize in 2-dimensions using PCA and t-SNE.

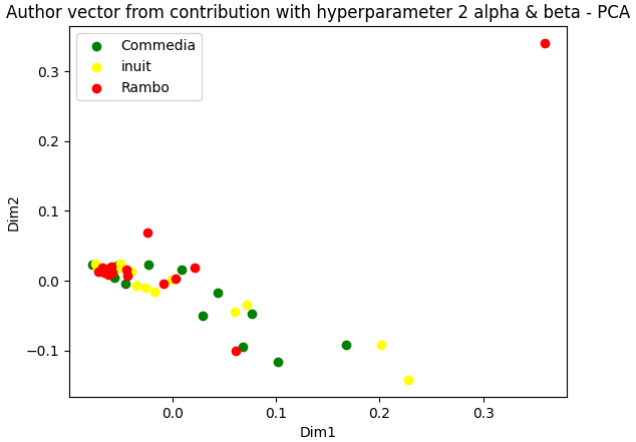


Figure 7.3: Author vector from static contribution PCA

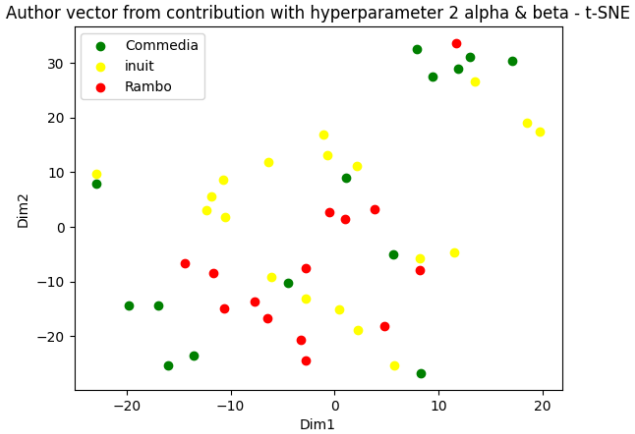


Figure 7.4: Author vector from static contribution t-SNE

7.2.2 Author vector computed from dynamic author contribution

In the contrast with the previous approach, here the α and β takes two distinct values computed to achieve fair contribution between all authors (to add an equilibrium).

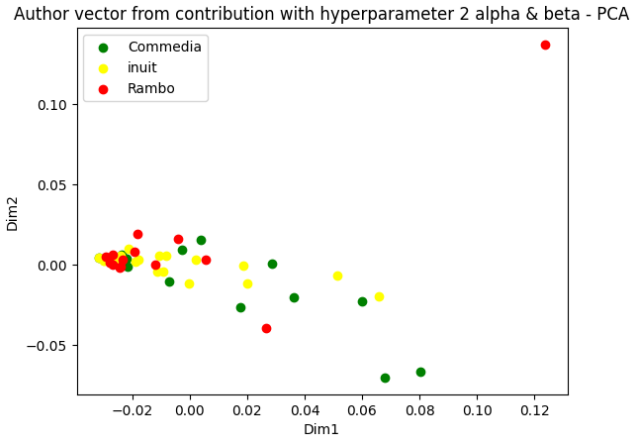


Figure 7.5: Author vector from a dynamic contribution - PCA

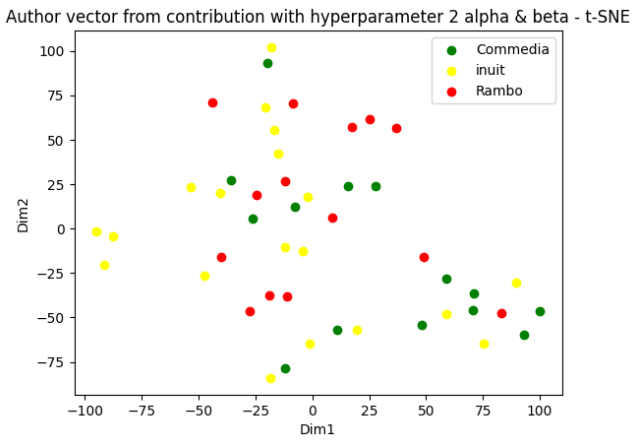


Figure 7.6: Author vector from a dynamic contribution - t-SNE

With the contribution of the first author roughly three times of the authors. We see no real difference between the first method that uses the the static contribution.

7.2.3 Author vector and Glove vector embedding

In this approach, we combined the author vector that we have found ourselves and the GloVe vector embedding to perhaps emerge the semantics aspect of author interest. In this section, we present the visualization result that we found in figure 7.7 and 7.8.

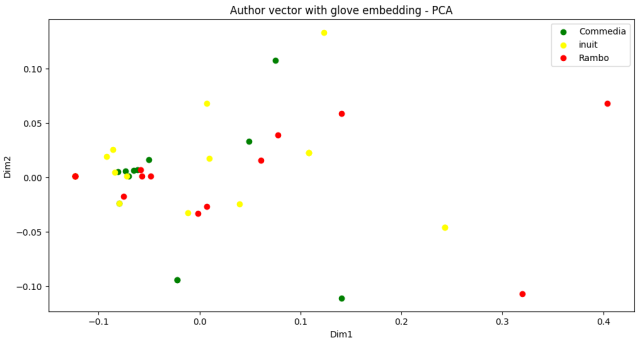


Figure 7.7: Author vector & Glove vector embedding - PCA

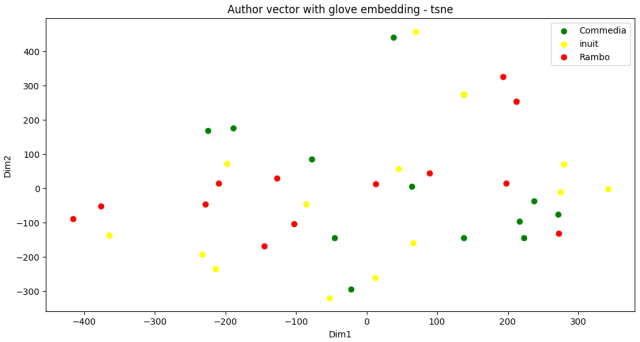


Figure 7.8: Author vector & Glove vector embedding - t-SNE

We see that this approach does not have really good clustering effect. In fact, we get results close to what we get with the previous approaches. Since our initial purpose is to use this approach to visualize authors and topics in the same space to visualize each author's interest. However, finding relevant keywords to represent topics (in order to visualize them on a 2d space) was proven unreliable. So deem this approach less effective than the previously presented approaches.

Chapter 8

Tasks Assignment

Task	**Student**
Data Collection	MELLOUKY Mohamed
Data Preprocessing	NGUYEN Tien Dung
Topic Extraction with LDA Algorithm	NGUYEN Tien Dung
Postprocessing: Computing the author vector	NGUYEN Tien Dung + MELLOUKY Mohamed
Extra/optional step: convert authors vector to GloVe vectors	NGUYEN Tien Dung
Dimensionnality reduction	MELLOUKY Mohamed
Visualization (plot 2D 3D)	MELLOUKY Mohamed

Table 8.1: Task Assignment

Conclusion

In most cases, we observe considerable overlap between clusters, as well as a large number of outliers. We conclude that the three teams are not perfectly arranged, or that what we have seen is the result of the lack of a large number of data instances. Indeed, as we know, any topic model requires a larger data set than the one we used.

If we had more time, we could first create an interactive visualizer that would allow us to zoom in and out to get a clearer view of the clusters, in addition to a slider that would allow us to visualize the evolution of the author's interest over the four years. In addition, we could increase the quantity and quality of data by adding article content to our dataset.

Besides, we could also work on fool-proof author's full name recognition.