M2 MALIA
Network Analysis for Information Retrieval

Elyes KHALFALLAH & Mohammed Ali EL ADLOUNI

16/03/2025

Contents

1 Introduction

The increasing quantity of text data currently available online demands efficient methods of information retrieval and analysis. In most fields of academic research, business intelligence, and legal records for example, it is crucial to be able to process and mine informative content from large text corpuses. Network analysis presents a powerful framework for studying the structure and relations among corpuses all while allowing clustering, classification, and search optimization.

This report presents our steps treating a corpus of scientific IT texts using network-based techniques. It is a procedural study that begins with the acquisition of data and pre-processing, continues with corpus structuring, and finally search engine construction. Each step uses generic Natural Language Processing (NLP) techniques, including text normalization, feature extraction, and similarity measurement. Our is to build an effective document search engine capable of supporting relevant information discovery on the basis of Term Frequency-Inverse Document Frequency (TF-IDF) weighting and similarity calculations.

2 Data acquisition and Treatment

The dataset given to us, plainly titled `data_projet.csv`, is a collection of texts containing the name of their venue, their abstract, authors, number of citations, references, title, year of publication, id, and class :

1. `venue` : the name of the venue where the presentation took place
2. `abstract` : the abstract of the presentation
3. `authors` : the authors of the presentation
4. `n_citations` : the number of citations of the presentation
5. `references` : the references of the presentation
6. `title` : the title of the presentation
7. `year` : the year the presentation took place
8. `id` : the id of the presentation
9. `class` : the class associated to the presentation

The dataset is stored in a tab-separated values (TSV) format and is imported into a Pandas DataFrame for processing. Initial exploratory steps include inspecting the structure of the dataset, checking for missing values, and identifying unique document categories.

3 Corpus Structuring and Pre-processing steps

Before starting to build the search engine, it's important that our data be properly formatted and pre-processed. First and foremost, to actually build the text we would be working on, a new column was created named `text`, being the concatenation of the work's title followed by its abstract. At the beginning of our project, we wanted to include `venue` as well, but after some discussion with

our professor, we came to realize that this would too strongly bias our search engine, given that works presented at the same or similar venues often resembled each other.

As for pre-processing methods, we've used a number of simple yet strong techniques :

1. **Punctuation Removal**: All non-alphanumeric characters are eliminated to avoid discrepancies in text matching.
2. **Lowercasing**: Text is converted to lowercase to ensure uniformity in word representation
3. Whitespace Normalization: Extra spaces are removed to maintain consistency in text formatting
4. **Stopword Removal**: Common words such as "the", "and", and "is" are discarded, as they do not contribute significant meaning in retrieval tasks.
5. **Lemmatization**: Words are reduced to their base forms (e.g., "running" $\rightarrow$ "run") to enhance the accuracy of similarity measures.
6. **Outlier Removal**: Extremely rare and overly frequent words are identified based on percentile thresholds and removed to reduce noise in the dataset. This was a debated topic during the writing of our code, as removing outliers theoretically improves search result quality, but comes at the risk of completely ruining similarity calculations. Often, while working with outlier removal, if an outlier was searched for, all similarity results would return as null, and searches would be unsuccessful.

These pre-processing steps transform raw textual data into a structured form, making it suitable for indexing and retrieval. A visual analysis of word distributions further informs the decision-making process regarding stopword handling and frequency thresholds.

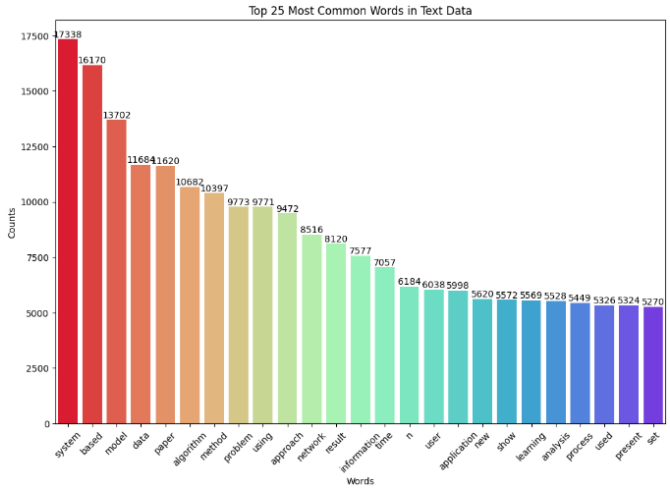

Figure 1: Top 25 most common words in text data

4 Search Engine

4.1 Textual embeddings

In this section, we describe the core of the information retrieval system is the construction of a document-term matrix using TF-IDF weighting. This process involves:

- **Term Frequency (TF) Matrix**: A numerical representation of the frequency of words in each document.
- **TF-IDF Matrix**: An adjusted version of the TF matrix that accounts for the importance of words across the entire corpus.

4.1.1 Impact of Stopwords on TF and TF-IDF

The inclusion, or not, of stopwords in our text directly influences the results on TF and TF-IDF :

- **TF Matrix**: Performs optimally when stopwords are removed, as high-frequency, low-information words risk dominating the representation.
- **TF-IDF Matrix**: Can function with or without stopword removal. Retaining stopwords in TF-IDF-based approaches allows for a more balanced weighting of common terms across documents.

4.1.2 Dimensionality Reduction and Clustering

After vectorizing the text data, we reduce its dimensionality to extract meaningful patterns while mitigating the sparsity of high-dimensional feature spaces. Principal Component Analysis (PCA) is applied to compress the TF-IDF representation while preserving as much variance as possible. This step also aids visualization and clustering tasks by revealing latent document structures.

For clustering, we employ the K-Means algorithm, which groups similar documents based on their transformed feature representations. The choice of the number of clusters is guided by methods such as the elbow method or silhouette analysis, ensuring that clusters are well-separated and meaningful. Other clustering techniques, such as hierarchical clustering or DBSCAN, could be considered depending on the corpus characteristics and the need for flexible, density-based groupings.

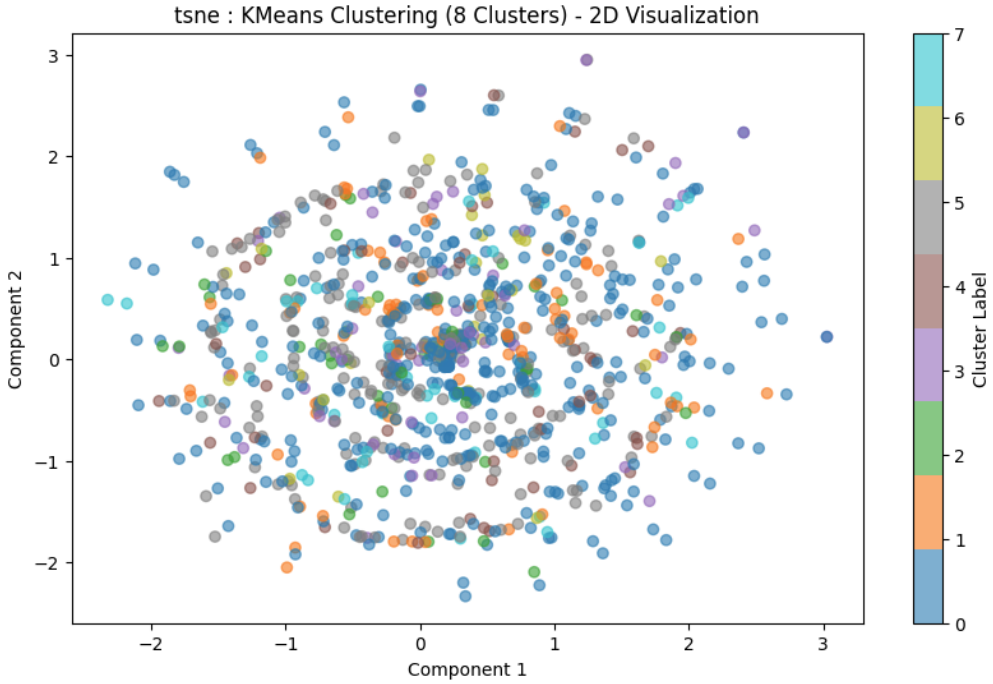

Figure 2: TSNE representation

4.1.3 Topic Modeling with Latent Dirichlet Allocation (LDA)

To uncover hidden thematic structures within the corpus, we apply Latent Dirichlet Allocation (LDA). This probabilistic model assumes that each document is composed of multiple topics, each represented by a distribution over words. By identifying these topics, LDA provides insights into the dominant themes present in the corpus.

While LDA is widely used for topic modeling, alternative approaches exist. Non-negative Matrix Factorization (NMF) offers a more deterministic way of extracting topics by factorizing the document-term matrix. More recent deep learning-based models, such as BERTopic, leverage transformer embeddings and dynamic topic modeling to capture more nuanced relationships between documents.

4.2 Node embeddings and Clustering

It was now time to try to vectorize the graphs we've generated up until now.

4.2.1 Comparing Louvain and Spectral Methods for Network Analysis

When analyzing networks, two common approaches for community detection and graph partitioning are the Louvain method and spectral clustering. Both techniques aim to group nodes based on structural properties, but they differ significantly in methodology, computational complexity, and interpretability.

Louvain Method: A Heuristic Approach for Community Detection
The Louvain method is a hierarchical, modularity-based algorithm designed to detect communities in large networks efficiently. It works by iteratively optimizing modularity, a measure of how well nodes are clustered compared to a random graph.

In contrast, **spectral clustering** is based on eigenvalue decomposition of the graph Laplacian, leveraging spectral graph theory. Instead of using modularity, it formulates clustering as an eigenproblem

Initially, we considered spectral clustering due to its solid theoretical foundations and ability to provide meaningful embeddings of the network. However, in practice, it quickly became computationally prohibitive for our dataset. The main bottleneck was the eigenvalue decomposition of the Laplacian matrix, which scales poorly with large graphs. Since spectral clustering requires an $O(n^3)$ operation for eigenvector computation, it became infeasible for our use case. While Louvain has some limitations, such as instability across runs and a tendency to miss small communities due to the resolution limit, its speed and efficiency made it the better choice for our project.

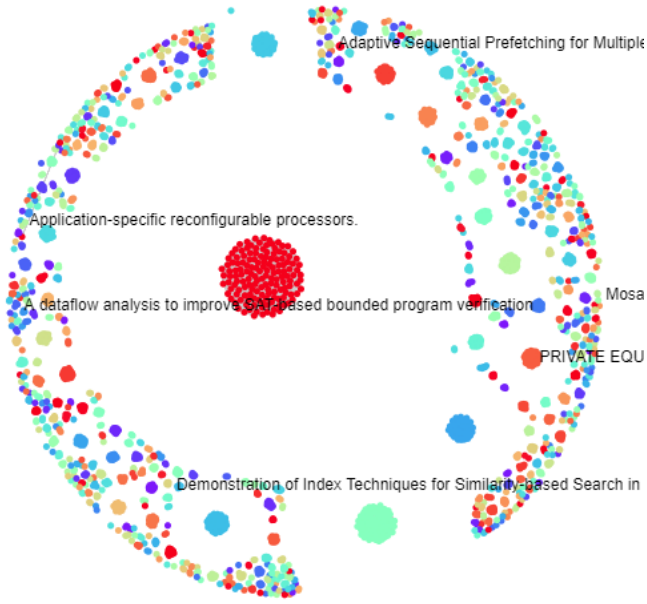

Figure 3: Graph Representation with Louvain method

4.2.2 Challenges with Node2Vec and Alternative Approaches

To numerically represent textual data, we stuck to using TF-IDF vectorization. By doing so, TF-IDF ensures that unique or domain-specific words contribute more to the document's representation than frequently occurring filler words.

As for graph node vectorization, we considered Node2Vec, a popular algorithm for learning node embeddings in graphs, however, implementing Node2Vec led to dependency issues, particularly with NumPy versions and related libraries. Many modern machine learning frameworks rely on specific versions of NumPy, SciPy, and network-related libraries like NetworkX. This resulted in a dependency hell situation, where fixing one package would break another. Due to these challenges, we explored alternative methods for computing graph embeddings and landed on the Laplacian Eigenvalue method. Instead of relying on biased random walks, this approach leverages the Laplacian matrix of the graph, which encodes structural information about the network. By computing its eigenvalues and eigenvectors, we obtain a lower-dimensional representation of nodes that preserves connectivity properties.

The Laplacian Eigenvalue method has several advantages over Node2Vec:

- **Fewer dependencies**: It relies primarily on NumPy and SciPy, avoiding the need for additional graph embedding libraries.
- **Mathematical interpretability**: The eigenvalues and eigenvectors provide a clear representation of graph structure based on spectral graph theory.
- **Deterministic behavior**: Unlike Node2Vec, which involves randomness in the walk process and requires multiple runs for consistency, spectral methods produce stable embeddings.

4.3 Similarity Computation

The search engine retrieves relevant documents using two different similarity measures:

- **Cosine Similarity**: Measures the angle between vector representations of documents, effectively capturing their semantic similarity.
- **Euclidean Distance**: Calculates the straight-line distance between document vectors. However, in high-dimensional spaces, Euclidean distances become less effective due to the "curse of dimensionality," which causes all documents to appear equidistant.

5 Supervised classification

In this section, we perform a network analysis for information retrieval by leveraging textual data and graph-based deep learning. The following subsections describe our approach in detail.

We begin by loading the dataset `data_text.csv`, which contains 40,596 documents along with various metadata (e.g., `text`, `venue`, `abstract`, `authors`, `n_citation`, `references`, `title`, `year`, `id`, and `class`). After ensuring that documents with empty text fields are removed (some textual data were totally removed because of the preprocessing), we proceed to convert the raw textual data into numerical features.

Two vectorization schemes are employed:

- A term-frequency (TF) representation using `TfidfVectorizer` with `use_idf=False`.
- A full TF-IDF weighted representation.

The resulting document-term matrix, particularly the TF-IDF matrix, has a dimensionality of approximately $(39\,980 \times 54\,973)$, providing a high-dimensional feature space for subsequent analysis.

5.1 Graph Construction and Subgraph Extraction

To capture the similarity between documents, we construct a weighted graph where each node represents a document. The adjacency matrix A is computed as:

$$A = X_{\text{tfidf}} \times X_{\text{tfidf}}^{\top},$$

where X_{tfidf} is the TF-IDF document-term matrix. This multiplication yields a sparse matrix reflecting pairwise cosine similarity among documents.

Due to the large number of nodes, we extract a random subset corresponding to 30% of the nodes (approximately 11994 documents) to form a manageable subgraph. The submatrix A_{subset} is obtained by indexing A with the selected subset of node indices.

To prepare the graph for spectral learning and ensure proper propagation in the neural network, the sparse adjacency matrix is normalized. The normalized matrix is converted to a SciPy sparse format and then to a PyTorch sparse tensor to efficiently support sparse operations during model training.

5.2 Graph Convolutional Network (GCN) Architecture

Graph Convolutional Networks (GCNs) operate through a message-passing mechanism between nodes. At each layer, every node aggregates and transforms information from its immediate

neighbors. Thus, as more convolutional layers are stacked, the message is passed progressively from node to node, allowing each node to access an increasingly broader context.

A two-layer Graph Convolutional Network is employed for document classification. Key components include:

- **Dropout:** Applied after the first layer for regularization.
- **Output Layer:** The final layer outputs raw logits which are transformed into log-probabilities using a log-softmax function.
- In our implementation, the input dimension corresponds to the number of terms ($n_{\text{features}} = 54\,973$), the hidden dimension is set to 16, and the output dimension is equal to the number of classes (8 in this case).

5.3 Training Procedure

Due to computational constraints, we use only 30% of the available nodes for training, splitting them into 80% for training and 20% for evaluation. The training loop is executed for 200 epochs with the following settings:

- **Loss Function:** Negative Log-Likelihood Loss (NLLLoss).
- **Optimizer:** Adam with a learning rate of 0.01 and weight decay of 5×10^{-4}.
- **Metrics:** Both training loss and accuracy are monitored and plotted over epochs.

5.4 Evaluation and Discussion

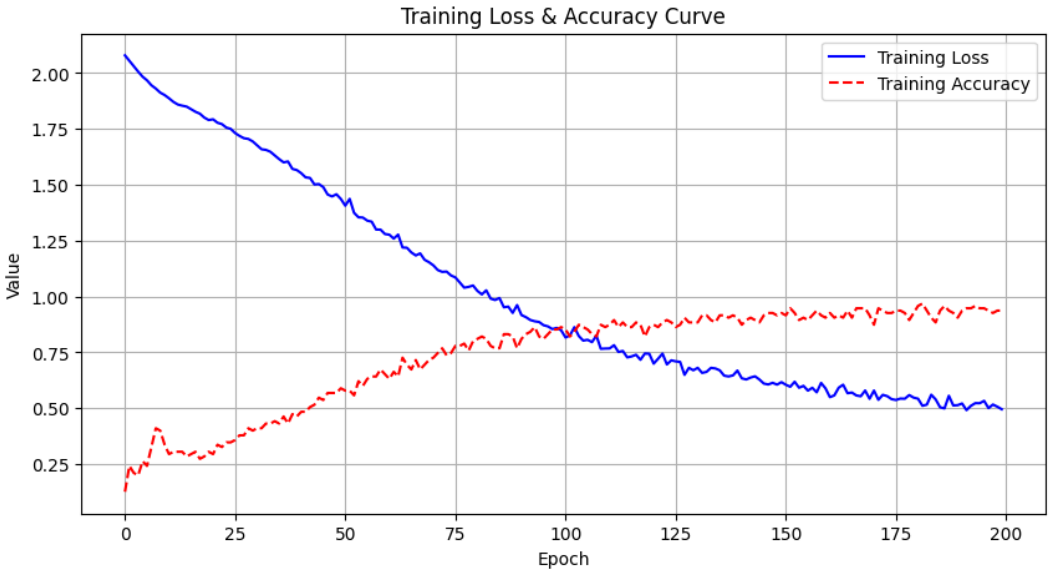

Figure 4: Training loss and accuracy curves over epochs.

The training curves indicate a steady decrease in loss and an increase in training accuracy, eventually reaching near-perfect accuracy on the training set.

After training, the model is evaluated on the held-out evaluation set. Despite achieving high training accuracy, the evaluation accuracy is 33.3%, suggesting that the model may be overfitting to the small training subset. This gap highlights the need for potential use of a larger training set, or exploring more robust regularization techniques.

6 Conclusion

In this report, we explored the application of network analysis techniques for information retrieval in a large scientific corpus. Our approach combined traditional NLP methods, such as TF-IDF vectorization and similarity computation, with graph-based techniques, including community detection and node embeddings.

Despite initial challenges with computational efficiency, particularly in spectral clustering and Node2Vec implementation, we successfully adapted our methodology to prioritize scalability and interpretability. The use of the Louvain algorithm for clustering proved to be a robust alternative, allowing for efficient network partitioning without excessive computational overhead. Additionally, our search engine demonstrated the effectiveness of TF-IDF weighting and cosine similarity in retrieving relevant documents.

While our system performs well within the given dataset, future work could explore deep learning-based embeddings, such as transformer models, to further improve semantic understanding. Additionally, hybrid approaches combining graph embeddings with traditional text vectorization may enhance retrieval accuracy. This study highlights the potential of network-based methods in information retrieval and opens avenues for further exploration in large-scale text mining.