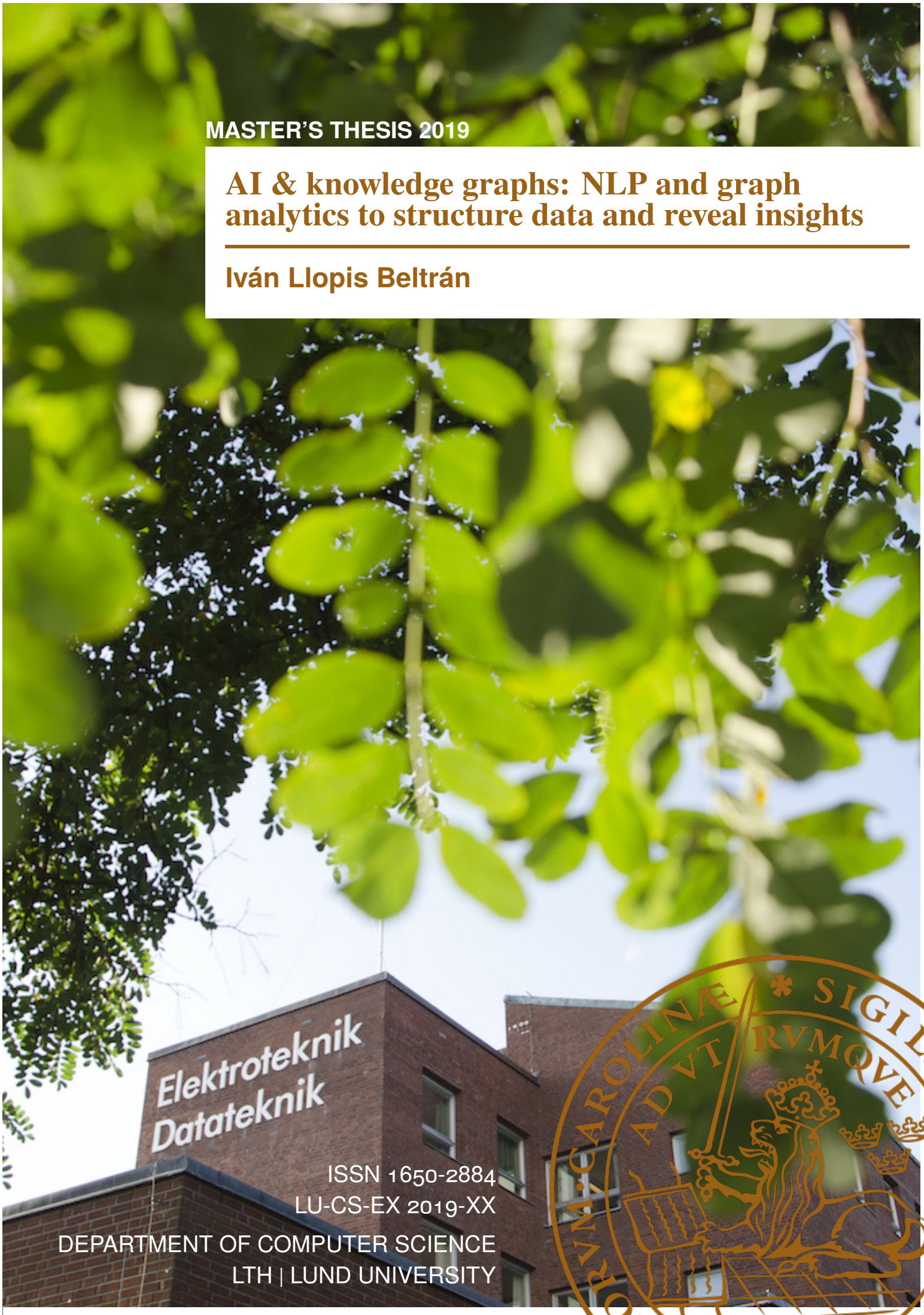

MASTER'S THESIS 2019

AI & knowledge graphs: NLP and graph analytics to structure data and reveal insights

Iván Llopis Beltrán

ISSN 1650-2884
LU-CS-EX 2019-XX

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-XX

AI & knowledge graphs: NLP and graph analytics to structure data and reveal insights

Iván Llopis Beltrán

AI & knowledge graphs: NLP and graph analytics to structure data and reveal insights

(A LaTeX class)

Iván Llopis Beltrán
ivl122ll-s@student.lu.se

August 22, 2019

Master's thesis work carried out at Volvo Cars Corporation.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

This Master's thesis attempts to solve the problem of document relation modelling. Normally, documents are classified by tags. However, rare is the case where document relations are tagged in the database. The study in this project attempts to estimate relations among text documents by means of topic modelling techniques, such as *term frequency inverse document frequency*, *latent semantic analysis*, *latent dirichlet allocation* and *word embeddings*. The Master's thesis presents a study of document similarity and jumps into implementation details over a real use-case at Volvo Cars. We program a graphical user interface in Python to interact with the system. We present the results with a comparison of the techniques in the evaluation method, which we base on a recall score and the rank biased precision metrics over a validation set. The results are enhanced by exploring clustering techniques with graph theory in order to discover communities within the documents.

Keywords: artificial intelligence, knowledge graphs, natural language processing, topic modelling, document similarity, word embeddings, graph algorithms

Acknowledgements

I would like to thank Fabien Batejat, who has been my supervisor at Volvo Cars Corporation and Konstantin Lindström, intelligence engineer at VCC, for their inestimable help and support during this Master's thesis. Their guidance, insight and experience have made the difference during this entire project. They have made me better as a person and as an engineer. This would not have been possible without them.

I would like to thank Pierre Nugues, my supervisor at Lund University, for his guidance supervising this Master's thesis. His unquestionable brilliance and experience in the field made this project an enhancing experience.

I would also like to thank Deniz Eca Aktan and Ali Ebrahimzadeh for their help on the technical side and their indefatigable will and good mood every single day of work.

Also, to my colleagues at Volvo Cars for making this such an amazing experience full of truly smart minds, but even more, great coworkers.

Finally, to my mum. The reason behind everything. To whom I owe my past, my present and my future. My inspiration and my courage. To you.

Contents

Chapter 1
Introduction

We live in the Information Age. We have passed from static sources of knowledge, i.e. books, to a dynamic continuously growing connected system spinning around the information technology. Huge powerful databases are being built every day. Text documents are one of the most typical forms of communication in the world. Institutions amass enormous amounts of documents that we need to categorize. Knowing the relations among them, if they exist, may also help having a better understanding of the information.

1.1 The problem

There are a lot of applications that need human supervision over text documents in our daily lives. In this Master's thesis, we will attempt to solve the problem of document relation and topic characterization and we will apply it to a real use-case at Volvo Cars Corporation.

Volvo Cars Corporation is constantly searching for new trends that could be an incorporation in future projects. One way to do this job is by exploring and evaluating thoroughly information appearing in the news. The information is processed and used inside the company in many data-systems. The process starts by reading articles that appear in a news feed and classify them in categories that are suitable for the company. The next step goes through combining similar or related articles that infer potential concepts that could be developed as projects. This is a mentally demanding task for a human and it requires the ability to identify and relate different pieces of information together. The complexity of carrying the task grows as the amount of information available increases. It is worth exploring solutions in order to provide with autonomy the process of information integration and enhancing the data.

A potential solution to this problem is the techniques found in *topic modelling*. This field of *natural language processing* has proven very powerful for processing and analyzing text documents.

1.2 Previous work

The world of topic modelling is something that fortunately has been treated previously. The first ones started by simply counting words (the so-called *bag of words*) as a way to discover if language had an internal pattern (Harris, 1954). This subject evolved to weight word occurrences depending on the commonality of a word in a corpus. The *term frequency-inverse document frequency* (TF-IDF) technique takes advantage of information retrieval to weight the significance of a word in a text (Ramos, 2003).

Some other techniques were developed during the last decades. They are still relevant and of great interest in the field. Methods, such as *latent semantic analysis* (LSA), *latent Dirichlet allocation* (LDA) or *explicit semantic analysis* (ESA) are highly used for computing document similarity and detecting document relevance (Deerwester et al., 1990; Blei et al., 2003; Gabrilovich and Markovitch, 2007).

Another big advance in natural language processing is *Word embeddings*, which is a method for text vectorization that is trained by analyzing the pattern of word appearances in a corpus (Mikolov et al., 2013). Some work has also been done using the power of a knowledge base, like the document published by Schuhmacher and Ponzetto (2014), in which they use a large publicly available knowledge base to compute document similarity.

1.3 Graph Databases

Graph databases are a type of database that represent their information as interlinked elements. An element of the database is a node. Nodes can be connected to other nodes in the database through the so-called *relationships*. These databases do not allow only to have framework where we can store our documents, but also the relations we find throughout the thesis. Furthermore, the fact that is a *graph* allows for the usage of graph algorithms, which can be really useful for analytics over the database. Graph technologies are rapidly gaining in popularity by changing the way companies work with their data.

1.4 Objectives

The objective in this Master's thesis is to study algorithms to solve the problem of document characterization, where we will apply topic modelling techniques to find the main themes that define a document. Besides this, we will use the topics to automatically predict the relations existing among the documents. A way to do this is to score texts by similarity and establish relations based on these scores. We will attempt to provide a way to analyze commonalities between documents that could help the user understand why the documents are related. We will also design a graphical user interface to present the documents and their recommendations. Finally, we will explore the capabilities of graph algorithms to reveal insights about the internal structure of the data, such as clusters of documents or common words that define those clusters.

Chapter 2
Approach

The field of artificial intelligence is growing at an outstanding velocity. The number of problems where it is applied is broader every day. Its applications go from predicting the stock market evolution to create pieces of art generated while emulating another artist's style (Gatys et al., 2016). Most algorithms existing in *natural language processing* (NLP) rely on analyzing text to find an internal structure and the harmony that exists among its words. Processing this structure aims for comprehension of human language.

An example of what we can do with NLP is attempting to automatically predict the meaning of sentences. This usually falls into the classification field, where a machine evaluates texts and classifies them among a discrete amount of categories, as for instance: predicting 'positive' or 'negative' posts in a review-aggregation portal of movies (Pang and Lee, 2008). This example is called *sentiment prediction* and it is very common in NLP. An agent can also process text with the objective of analyzing the occurrences of words and their frequency. This gives a sense of the internal structure of the language. If a group of words occur consistently together, they belong to the same context or perhaps these words create a theme. This field is called *topic modelling*, which will be the main pillar of this Master's thesis.

2.1 Topic Modelling

Topic modelling consists of the extraction of topics that describe a document, normally by means of statistical methods. If the topics are predefined, the concept extraction is a supervised categorization task. If the topics are not predetermined, it becomes unsupervised inference. One way to find relations among documents is through similarity of topics. Texts that talk about the same topics are likely to be related, whilst documents speaking about completely different ones are prone to not be. For example, if we read one document about *guitars* and one document about *pianos*, we can identify *instruments* as a topic in common.

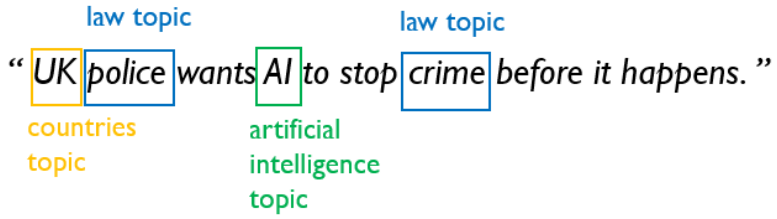

Figure 2.1: Example of an ideal topic modelling categorization where three topics have been identified.

There are fundamentally three aspects that shape the type of topics discovered: the number of topics, the dataset and the algorithm used to infer them. The number of topics is the amount of latent dimensions to be found during inference. The content of the dataset also shapes the topics. The more varied the content is, the more general the topics tend to be. In the previous example, the topic identified could have been *music* instead of *instruments* if the dataset included more instances of words related to that topic. Another tool to find relations is *knowledge-bases*, where complex structured information is part of the system's database. However, that falls outside the scope of this Master's thesis.

The use-case where we will apply the study on is unsupervised, where the relations must be inferred and the topics must be discovered. In Section 2.2 we will name different techniques and we will explore their possibilities on document characterization.

2.2 Algorithms

There were many techniques mentioned in Section 1.2 related to topic modelling. We will study some of those techniques to compare documents and compute document similarity. Specifically: term frequency - inverse document frequency, latent semantic analysis, latent Dirichlet allocation and word embeddings. We explain the theoretical background in this section, whilst the implementation details and software will be explained in Chapter 3.

2.2.1 Term Frequency - Inverse Document Frequency

This technique is a derivation from the *bag of words* model. A *bag of words* (BoW) is a representation where a document is characterized as a set that describes the occurrence of tokens in the document (Harris, 1954). We can take as an example the next two texts:

Text 1 EU puts weight behind Wi-Fi over 5G for connected cars. As the industry decides what connectivity should be used for connected cars, the EU appears to be putting its weight behind Wi-Fi over 5G.

Text 2 European Union (EU) has agreed to cut carbon emissions from cars by 37.5% within a decade. The new targets are part of a wide EU push to reduce total greenhouse gas emissions.

The texts need to pass through a process of tokenization, where the terms appearing in the texts are extracted. For simplicity, the typical stop words have been filtered out. Then

the bags of words represent the occurrences of those terms in the documents. The bags of words for the documents in the example look like:

```
BoW_Text1 = {"5G": 2, "appears": 1, "behind": 2, "cars": 2,
  "connected": 2, "connectivity": 1, "decides": 1, "EU": 2,
  "industry": 1, "puts": 1, "putting": 1, "used": 1,
  "weight": 2, "Wi-Fi": 2}

BoW_Text2 = {"cars": 1, "EU": 2, "37.5%": 1, "agreed": 1,
  "carbon": 1, "cut": 1, "decade": 1, "emissions": 2,
  "European": 1, "gas": 1, "greenhouse": 1, "new": 1, "part": 1,
  "push": 1, "reduce": 1, "targets": 1, "total": 1, "Union": 1,
  "wide": 1, "within": 1}
```

Term Frequency - Inverse Document Frequency, or tf-idf, is a technique that also models documents by terms. A document in tf-idf is represented by a vectorization of the tokens appearing in that document. The terms are weighed by their relevance to the document in the corpus. This is done by calculating the *term frequency* and the *inverse document frequency*.

Term frequency is the number of times that a word occurs in the document. The higher the frequency, the higher the impact the word is likely to have in the general concept of the document. The formula for calculating the term frequency varies, as there several ways to compute this. Typically:

$$tf(t,d) = \frac{f_{t,d}}{number\ of\ words\ in\ d}, \tag{2.1}$$

where $tf(t,d)$ refers to the *term frequency* factor of a term t in a document d in the tf-idf and $f_{t,d}$ refers to the frequency of the term t in the document (the number of occurrences).

Inverse term frequency is the part of the computation that weights the words regarding the amount of documents in which they appear. Explained in a different form, a term that appears in every single document is not plausible to be defining in a text. On the contrary, a term that appears in a small amount of documents in the corpus will most probable be a term that says a lot about its content. Mathematically, it takes the form:

$$idf(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}, \tag{2.2}$$

with $idf(t,D)$ being the inverse document frequency for a term t, N being the number of documents in the corpus and the denominator being the number of documents in the corpus that include the term t.

The combination of both aspects is the Term Frequency - Inverse Document Frequency:

$$tfidf(t,d,D) = tf(t,d) \cdot idf(t,D). \tag{2.3}$$

The modelling of a document using TF-IDF will output a vector of terms that are weighted regarding the term frequency in the document and the inverse document frequency in the corpus. The higher the weight of a term, the most significant the term is for that specific document in the corpus. This is useful to find keywords in a text. The use of keywords are central in search engines and for summarizing content and relating documents. The tf-idf representation for the previous example is:

```
tfidf_Text1 = {"5G": 0.3849, "appears": 0.1924, "behind":
   0.3849, "cars": 0.0, "connected": 0.3849, "connectivity":
   0.1924, "decides": 0.1924, "EU": 0.0, "industry": 0.1924,
   "puts": 0.1924, "putting": 0.1924, "used": 0.1924,
   "weight": 0.3849, "Wi-Fi": 0.3849}

tfidf_Text2 = {"cars": 0.0, "EU": 0.0, "37.5%": 0.2182, "agreed
   ": 0.2182, "carbon": 0.2182, "cut": 0.2182, "decade": 0.2182,
   "emissions": 0.4364, "European": 0.2182, "gas": 0.2182,
   "greenhouse": 0.2182, "new": 0.2182, "part": 0.2182, "push":
   0.2182, "reduce": 0.2182, "targets": 0.2182, "total": 0.2182,
   "Union": 0.2182, "wide": 0.2182, "within": 0.2182}
```

Note that the words *cars* and *EU* have weight zero in the tf-idf representation as both documents mention or speak about them. Hence, they are not representative or relevant for the document in the example. However, words like *Wi-FI*, *5G* or *connected* are actually relevant for text 1. Tf-idf expresses documents by vectors of tokens with weights associated to their relevance. These vectors can now be compared among themselves to compute document similarity. The methods to compute document similarity will be explained in Section 2.3.

2.2.2 Latent Semantic Analysis (LSA)

The previous method explained a way to vectorize documents based on a word-by-word fashion. A document including the words 'car' and 'human' will have a total different representation than a document including the words 'automobile' and 'person'. Each word is a different position in the dictionary. This means that there does not exist a semantic sense in the document vectorization.

Latent Semantic Analysis (LSA) is a technique that reduces the dimensionality of the tf-idf vectors in order to infer more general topics. The methodology starts with a tf-idf transformation. Then a term-document matrix is constructed with the weights from the tf-idf vectorization. The rows of the matrix correspond to terms in the dictionary of the corpus and the columns correspond to documents (see Table 2.1). The next step is a rank lowering of the dimension of the term-occurrence matrix. The technique performs a singular value decomposition (SVD) over the document vectorization (Deerwester et al., 1990). Then it trims the singular values that have the lowest impact up to the number of output topics decided by the user.

The result is an approximation of the previous system with minimal error. The document vectors have now become a semantic space that represent more general concepts than singular words. A characterization using LSA with two latent topics over the texts in the previous example is:

```
lsa_Text1 = {"Topic #0": 5.5136, "Topic #1": 2.1447}

lsa_Text2 = {"Topic #0": 2.7568, "Topic #1": -4.2895}
```

where the latent topics are:

```
Topic #0: 0.435*"eu" + 0.363*"cars" + 0.290*"5g" + 0.290*"
   connected" + 0.290*"weight" + 0.290*"behind" + 0.290*"wi-fi"
```

	Text 1	Text 2
5G	0.3849	0.0
agreed	0.0	0.2182
appears	0.1924	0.0
behind	0.3849	0.0
...	...	...
wide	0.0	0.2182
Wi-Fi	0.3849	0.0
within	0.0	0.2182

Table 2.1: Tf-idf term-document matrix of the corpus of documents in the example.

```
   + 0.145*"emissions" + 0.145*"appears" + 0.145*"used" +
   0.145*"puts" + 0.145*"industry" + 0.145*"putting" + 0.145*"
   decides" + 0.145*"connectivity" + 0.073*"european" + 0.073*"
   gas" + 0.073*"part" + 0.073*"greenhouse" + 0.073*"targets"

Topic #1: -0.373*"emissions" + 0.187*"behind" + 0.187*"connected
   " + 0.187*"wi-fi" + 0.187*"weight" + -0.187*"european" +
   -0.187*"gas" + -0.187*"decade" + 0.187*"5g" + -0.187*"
   greenhouse" + -0.187*"part" + -0.187*"within" + -0.187*"
   reduce" + -0.187*"targets" + -0.187*"total" + -0.187*"union"
   + -0.187*"wide" + -0.187*"agreed" + -0.187*"new" + -0.187*"
   carbon"
```

An issue with latent semantic analysis, just like what happens with TF-IDF, is that the topics are not capable of capturing polysemy. A word with multiple meanings is treated equally regardless of the context. This limitation appears in almost all of the techniques explained in this chapter, however it is mitigated by the fact that normally words have a predominant meaning or context.

2.2.3 Latent Dirichlet Allocation (LDA)

Latent Dirichlet allocation, or LDA, is a generative statistical type of topic modelling. It is based on topic extraction, just like latent semantic analysis. LDA is very similar to this last one. Latent Dirichlet allocation is actually a type of probabilistic latent semantic analysis. It learns to group similar words into topics probabilistically and establishes the representation of documents as a vectorized distribution over such topics.

This technique was developed by David Blei, Andrew Ng and Michael I. Jordan in 2003 (Blei et al., 2003). It is very used in natural language processing and topic modelling applications, which is this Master's thesis' main focus. Latent Dirichlet allocation will be used for document vectorization, as well as the previous explained techniques. The number of topics will be one of the parameters to tune during the process.

2.2.4 Word Embeddings

The two previous techniques offered methods to vectorize entire documents based on the singular words in the corpus and general concepts extracted from the texts (named as *topics* in LSA). Each element in a vector representation was defined by a floating value. The technique presented in this section approaches the same concept by providing a vectorization for singular words individually: the *word embeddings*. The first proper representation for words in this context was created by Tomas Mikolov in Mikolov et al. (2013). This model used in topic modelling is named *Word2vec*. The training of this technique can be done as continuous bags of words or as skip grams.

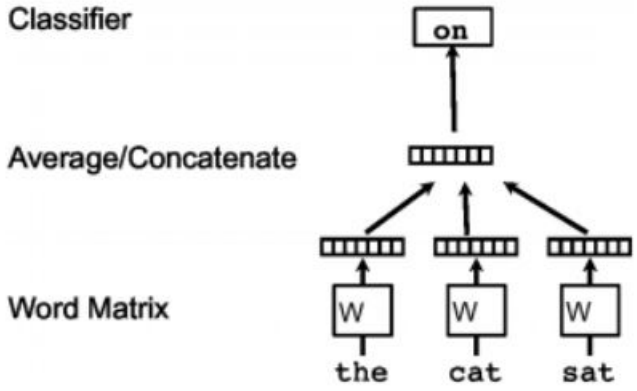

Figure 2.2: Training schema for Word2Vec embeddings as continuous bag of words. *From: https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e*

The size of the vector (called *embedding* from now on) depends entirely on the user: a term can be represented by a vector of three elements or three thousand. The properties yielded by those will change completely, as words are embedded with certain properties.

Word Embeddings are trained based on the frequency and context in which they appear. This allows the estimation of the strength of the relationships among words, being stronger in those words that are likely to appear together and weaker in those that are unrelated or unlikely to appear in the same sentence. The aim of word embeddings is to identify semantic regularities of terms in human language. This fact does not solve the polysemy issue, but puts words in context by giving similar encodings to words that consistently occur together (Fig. 2.3).

The word embeddings have also relational properties: any transformation relating two words will keep a similar semantic relation for other two words in a different context (Fig. 2.4). For example, the same vector defines the path to go from the word embedding of *male* to the word embedding of *female* as to go from the word embedding of *king* to the word embedding of *queen*. It is the same semantic relationship, which is marked as the same dimensional transformation in the vector space. Using the same philosophy, the vector that connects the word embeddings for *cat* and *dog* will connect the word embeddings for *tiger* and *wolf*. The semantic relations will depend on the dimensionality of the embeddings and the corpus used during training.

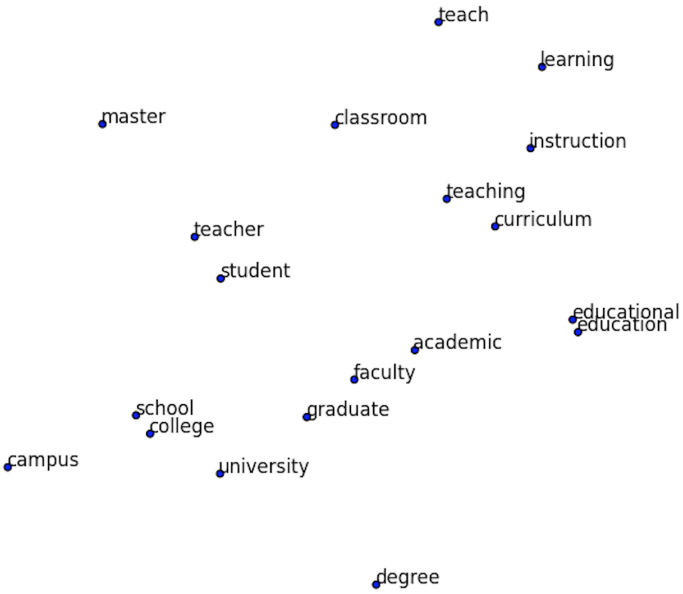

Figure 2.3: 2D projection of a portion related to academic terms of a vector space using word embeddings. The projection was created using t-Distributed Stochastic Neighbor Embedding (t-SNE).
From: https://www.datacamp.com/community/tutorials/lda2vec-topic-model

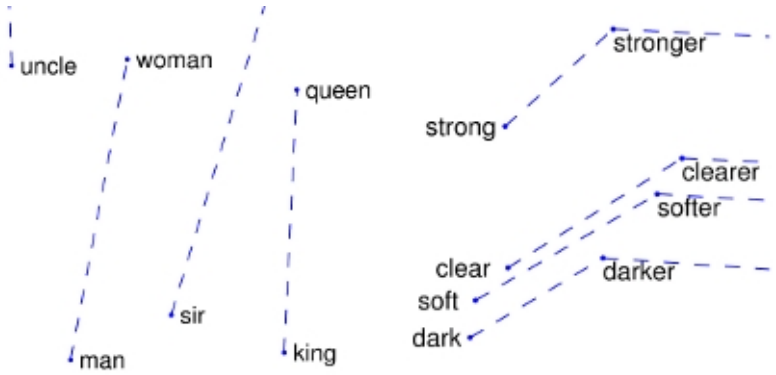

Figure 2.4: Semantic relationships in word embeddings: gender transformation (left) and comparative-superlative transformation (right).
From: https://github.com/stanfordnlp/GloVe

The first formal word embeddings model publicly available ever created was Tomas Mikolov's Word2vec in 2013, followed by Stanford's Global Vectors (GloVe) in 2014 (Mikolov et al., 2013) (Pennington et al., 2014).

2.3 Document Similarity

The explanations given up to this point are about techniques to obtain vectors from documents. The next step is how to use these representations to calculate similarity. The similarity between two vectors can be evaluated in different ways. The ones used for this Master's thesis are: *cosine similarity* and *Euclidean distance*.

2.3.1 Euclidean distance

Euclidean distance is a direct representation of the absolute magnitude that separates the endpoints of two vectors in a vector space (see Fig. 2.5). This magnitude is represented by the norm of the vector that connects both cursors:

$$Euclidean\_dist(\vec{v_1}, \vec{v_2}) = \|\vec{v_1} - \vec{v_2}\|_2. \tag{2.4}$$

In terms of topic modelling, each location in the vector space would be a specific meaning that represents the document. Documents that speak about similar topics are likely to point around the same region in this space. Obviously, a document usually does not speak only about one single topic, however this premise typically upholds.

2.3.2 Cosine similarity

There are other ways to make a comparison among vectors. In Section 2.3.1 we compared the distance from one end to the other end of the pointer locations of the vectors. Another magnitude that can be measured for comparison is the angle that is formed between them (see Fig. 2.5). This magnitude is a representation of how much two vectors differ. In the case of application to document embeddings, the cosine similarity symbolizes the difference of general meaning of documents at their core. The cosine similarity can be calculated as:

$$Cosine\_sim(\vec{v_1}, \vec{v_2}) = \frac{\vec{v_1} \cdot \vec{v_2}}{\|\vec{v_1}\| \cdot \|\vec{v_2}\|}. \tag{2.5}$$

Following this description, one could assume each possible direction in the n-dimensional vector space as one meaning. In this case, the length of the vectors do not have an impact in the cosine similarity, as the angle does not change. However, as seen in Section 2.2, each word in the embedding space occupies one single position. A word embedding will represent a different token when applying a transformation. Hence, the length of the vector does have an impact on word embeddings, even if the cosine similarity remains the same. Other methods to compute text similarity are the ones described in Wu and Palmer (1994); Resnik (1995); Lin (1998), which address semantic similarity in texts by different means.

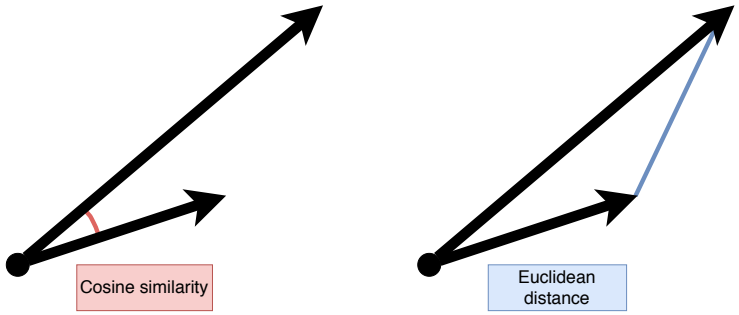

Figure 2.5: Vector distance measurements: *Cosine similarity* (left) and *Euclidean distance* (right) distance representation between a couple of 2D vectors.

2.4 Knowledge graphs

A knowledge graph is a type of linked database where its entities are represented by nodes (Ehrlinger and Wöß, 2016). Nodes can have *properties* attached, represented as key-value pairs. Nodes can also have a *label*. Labels help to keep the information structured. One can connect two nodes in the database through a *relationship*. For example, we can have a node that represents a student named "James Balaguer" with label *Person* and a node that represents the city of Lund with label *City* . We can have them connected through a relationship of type :LIVES_IN to indicate that James lives in the city of Lund (see Fig. 2.6).

Figure 2.6: Example of a simple graph database with two nodes connected by a relationship.

The database can have more than one node with the same label. The properties of an entity are attached to the entity, so we can have nodes and relationships with common key-value attributes (we can have two students named *James Balaguer* for example). Nodes can share any amount of relationships. Relationships are directional. They can have properties and labels associated too. Properties can be used to provide information about a relationship (see Fig. 2.7). Graph databases are related to cloud computing and big data. Their information retrieval is not dependent of the size of the database. Another property is that graph theory methods are also available, which will be used to perform clustering of documents. Graph theory also allows for analysis of relevance of documents in the database. These type of methods are called *centrality algorithms*.

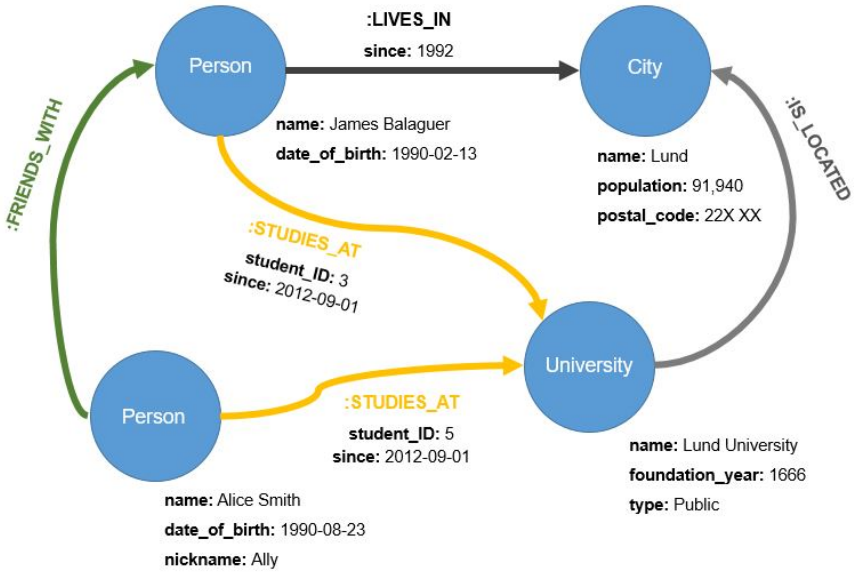

Figure 2.7: Example of a simple graph database with different types of nodes and relationships, with properties attached.

The software used during this Master's thesis is Neo4j, which is one of the most popular graph database platforms. It uses the Cypher Query Language for querying information. The similarity study leads to establishing relations among articles, which are be represented as relationship links in the graph database. The article connections sometimes form communities, which will be explored in Section 3.2.5.

Chapter 3

Implementation

This chapter focuses on the implementation of the techniques explained in Chapter 2 applied to the Volvo Cars Corporation use case. The implementation of this solution has been done in Python and can be separated into four main sections: *Data processing*, *Implementation of algorithms*, *Knowledge graph* and *Graphical user interface*. A picture of the system's architecture appears in Fig. 3.1.

The code is available at: *https://github.com/ivllopis/MScThesis/*.

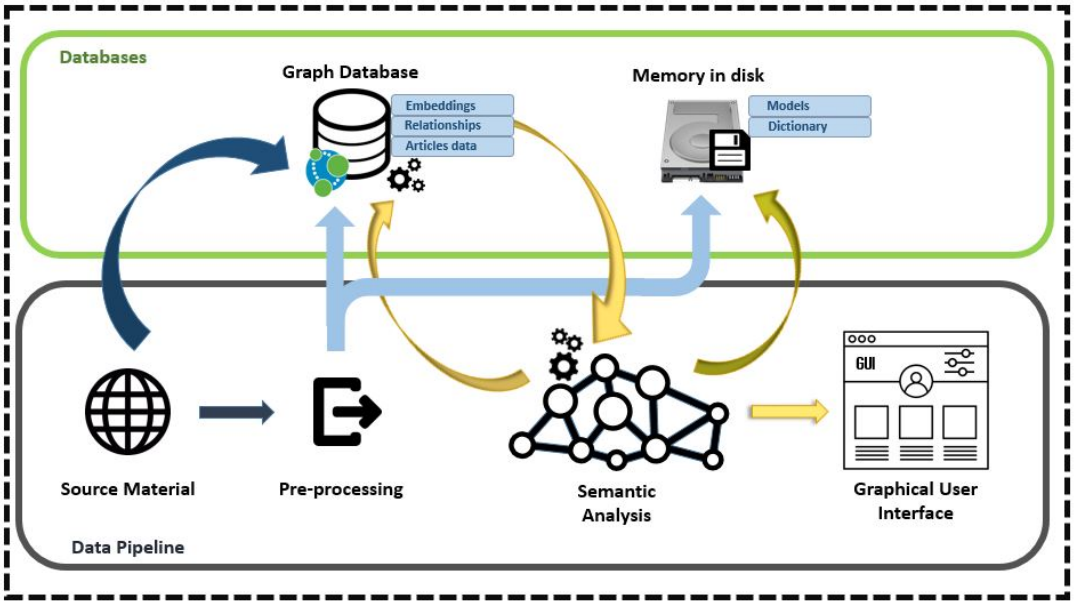

Figure 3.1: System architecture of the Master's thesis project.

3.1 Data Processing

The preparation of data is a common task in any digital project that uses data. The quality of this data turns out to be a key factor. Specially in artificial intelligence, obtaining a clean stream of input data is actually as important as the algorithms themselves. The documents must be processed and analyzed by means of the algorithms explained (see Section 2.2) in order to be matched by similarity. This section explains the processing methods that we have used on the news articles.

3.1.1 Managing data

The input data to this system is news articles obtained with a RSS aggregator. The information is received by means of an *application programming interface* (API). The *request* call to the API returns the documents' information in a structured format through the *response*. The fields of information from the *response* that we use in the system is:

- Title: String representing the title of a document.
- Content: String representing the source material.

The graph database will serve as placeholder for the documents and the solutions from the algorithms. The relevant information regarding documents are stored as nodes in the graph, while the predicted relations by the algorithms are represented as relationships. We have assigned a type (label) *Article* to the nodes storing documents. Each Article node contains the following information as properties of the node:

- Title: title of the article (*string*).
- Content: content of the document extracted from the source (*string*).
- Processed: content of the document after being processed and tokenized. This is the list of tokens that provides the document with meaning (*list of strings*).
- Processed_stem: content of the document after being processed, tokenized and stemmed. This is the list of tokens that provides the document with meaning (*list of strings*).
- Keywords: keywords extracted from the document using tf-idf (*list of strings*).
- Community: cluster discovered with *community finding* algorithms (*string*).
- Url: URL to the source material of the news article.
- Id: ID of the node in the database.

Neo4j APOC library is a tool designed to extend the functionality of Neo4j. The data integration process requires this library to use the *periodic commit* command, which is used to load information by batches into the database. The pieces of information that we require will be retrieved with queries to the database. The models trained during implementation are saved in disk (see Fig. 3.1). For the analysis of community finding (see Section 3.2.5), the *Graph Algorithms Playground* library is also required.

3.1.2 Cleaning the data

The information does not normally come in a format well suited for direct analysis. The processing method presented in this section is summarized in the next steps: markup extraction, lower casing, tokenization, extraction of punctuation marks and stop words, parsing special keys and lemmatization.

Markup extraction. The content obtained from the API response is HTML code. This source material includes markups that do not contain valuable information about the meaning of the content. We have used the *html* library in Python for decoding HTML entities into characters. For markup extraction, we have used *regular expressions* to identify and filter out XML tags.

Lower casing. Most words mean the same regardless of their form. Lower-casing allows unifying in the same token words that are capitalized, upper-cased and lower-cased. For that purpose, we have used the native method for string variables called *lower()*.

Tokenization. This is the process of splitting a string into a list of tokens, or terms. We have used the tokenizer included in the Natural Language Toolkit (NLTK) library.

Extraction of punctuation marks and stop words. *Punctuation marks* and *stop words* are two types of tokens do not contain valuable information for topic modelling. Articles, prepositions and pronouns are examples of stop words. We have used the list of stop words available in NLTK and the list of punctuation marks included in Python to filter those tokens out from the processed content.

Parsing special keys. There are certain tokens that do not own a significant meaning themselves, but the context that they represent. An example of these are amounts of money, dates or numbers. These tokens are typically parsed into special keys, or tokens. We have performed this type of parsing using regular expressions (see the code for details).

Lemmatization. We have used stemming to study if lemmatization has an impact in the results of the implemented algorithms. For that purpose, we have used a stemmer included in the Natural Language Toolkit called *Porter Stemmer*.

An example of the outcome of this methodology applied on a document is:

```
Original content:
<div><img title="Stupid Smart Cities With Molly Sauter" src="
   https://cdn-images-1.medium.com/fit/t/1638/2048/1*
   Ng3VxdA_KnxaB7KMJyKIKg.jpeg"><br><div><div>
<p>Medium</p>
<p>'Stupid Smart Cities' With Molly Sauter</p>
<p>Is your city the next VC guinea pig in the technocratic
   experiment to grow cities and extract their value?</p>
</div>
</div></div>

Processed content:
['medium', 'stupid', 'smart', 'citi', 'molli', 'sauter', 'citi',
'next', 'vc', 'guinea', 'pig', 'technocrat', 'experi', 'grow',
'citi', 'extract', 'valu']
```

3.1.3 Creating the dictionary

There are two important elements that we use when creating the topic models: the *dictionary* and the *bag of words* (BoW). The dictionary is a collection of unique tokens that describe the terms used in the corpus. Gensim has a dictionary object in their library (*gensim.corpora.dictionary*) with special functions:

- The collection frequencies (*cfs*): amount of instances of a certain token in the corpus.
- The document frequencies (*dfs*): amount of documents that contain a certain token.
- Filters of several classes: filters that allow to remove tokens by frequency or by id.
- Bag of words (*doc2bow*): transformation of a document into its bag of words.

The implementation starts by processing the documents as explained in Section 3.1.2. The processed documents (*corpus*) are then introduced to Gensim's dictionary object with:

```
import gensim.corpora.dictionary.Dictionary as Dictionary
dictionary = Dictionary(documents=corpus, prune_at=2000000)
```

The dictionary is mainly used for transforming documents into bags of words as well as for tracking token frequencies (number of instances) in the corpus. We filter out the tokens with frequency one or two as those tokens will not be significant in the corpus. This filtering is done using the *cfs* function mentioned before (see the code for more details). The output of a dictionary in Gensim looks like this:

```
Loading dictionary...
Dictionary loaded.
Dictionary(6878 unique tokens: ['2025', '32', '3d', 'abstract',
   'accord']...)
Loading corpus...
Corpus loaded.
MmCorpus(374 documents, 6878 features, 97868 non-zero entries)
```

The transformation of a document into its bag of words is done with the dictionary function in Gensim *doc2bow*. This BoW is a list of tuples $(id, frequency)$ where the *id* refers to the identification number of a token in the dictionary and the *frequency* refers to the number of times that the token appears in the document. Following the example in Section 3.1.2, its bag of words would look like this:

```
Pre-processed content:
['medium', 'stupid', 'smart', 'citi', 'molli', 'sauter', 'citi',
'next', 'vc', 'guinea', 'pig', 'technocrat', 'experi', 'grow',
'citi', 'extract', 'valu']

Bag of words:
[(159, 1), (296, 3), (297, 1), (298, 1), (299, 1), (300, 1),
(301, 1), (302, 1), (303, 1), (304, 1), (305, 1), (306, 1),
(307, 1)]
```

3.2 Implementation of algorithms

The algorithms explained are numerous, while having several variations each. *Gensim* is an open source NLP library for topic modelling that provide already built-in implementations for several of the techniques mentioned in Section 2.2 (Rehurek and Sojka, 2010). We have decided to use pre-trained models for word embeddings as the amount of data required to obtain reliable models is out of range in the use case used in this Master's thesis. The usage of libraries and pre-trained models has been distributed in the following way:

- For the techniques TF-IDF, LSA and LDA, the implementations are done using Gensim's open source library.
- For word embeddings, several models have been studied using pre-trained embeddings from GloVe, BERT and Flair.

3.2.1 Term Frequency - Inverse Document Frequency

TF-IDF uses the frequency of tokens in the corpus to weight the values obtained in the *bag of words* (see Section 2.2.1). The implementation of this algorithm is performed in Gensim by creating an instance of the model *tfidf*:

```
from gensim.models import TfidfModel

#convert corpus to BoW format
bow = [dictionary.doc2bow(document) for document in corpus]

#fit TF-IDF model
model_tfidf = TfidfModel(bow)
```

The way to apply this model to a document is by simply passing a text in BoW format to the instance we have just created:

```
#TF-IDF transformation of a BoW document
document_tfidf = model_tfidf[document_bow]
```

If we apply this to the example in Section 3.1.3, the TF-IDF transformation is:

```
Bag of words:
[(159, 1), (296, 3), (297, 1), (298, 1), (299, 1), (300, 1),
(301, 1), (302, 1), (303, 1), (304, 1), (305, 1), (306, 1),
(307, 1)]

TF-IDF transformation:
[(159, 0.0677), (296, 0.2169), (297, 0.1068), (298, 0.2546),
(299, 0.1110), (300, 0.4141), (301, 0.2659), (302, 0.3592),
(303, 0.1385), (304, 0.4141), (305, 0.4141), (306, 0.1385),
(307, 0.3271)]
```

where each element is a tuple $(id, weight)$ representing the identification number of the token and the weight associated with the TF-IDF transformation, respectively.

The library also provides a class for calculating the cosine similarity among vector representations in the form of lists of tuples (such as TF-IDF, LSA or LDA). This class is called *Similarity* in Gensim (*gensim.similarities.docsim.Similarity)*. The parameter *corpus* in this class refers to the LSA, LDA or TF-IDF transformation of the corpus and *num_features* to the length of the dictionary. The result is a matrix M, where each element M[i,j] represents the cosine similarity between the document with index *i* and the document with index *j*. The index of a document is stored as a node property called *id* in the graph database (see Section 3.1.1).

3.2.2 Latent semantic analysis & latent Dirichlet allocation

The procedure for LSA and LDA is similar to the process for TF-IDF. The starting point is the same: they require a dictionary and the bag of words of the corpus. Gensim provides also built-in functions to calculate the LSA model and the LDA model without any further difficulty. The statistical processes and the dimensionality reduction that they require are done automatically when creating the models:

```
from gensim.models import LsiModel, LdaModel

#fit LSA model
lsa = LsiModel(corpus, id2word = dictionary, num_topics =
   ntopics)

#fit LDA model
lda = LdaModel(corpus, id2word = dictionary, num_topics =
   mtopics)
```

There are a few parameters that one can tune for these models when creating them:

- **corpus** – Stream of tokenized documents. These tokens must appear in the dictionary, otherwise they will be ignored.
- **num_topics** – Number of topics to identify (latent dimensions).
- **id2word** – Optional. ID to word mapping. Dictionary to be used.
- **chunksize** – Optional. Number of documents to be used in each training chunk.
- **distributed** – Optional. If True, distributed mode (parallel execution on several machines) will be used.

Then a decomposition is created for the number of topics preset, where topics for LSA tend to be more specific and for LDA more general. The cosine similarity among documents can be computed as explained at the end of Section 3.2.1. Some of the topics inferred during the analysis with *latent semantic analysis* are:

```
First 10 topics found with LSA:
------------------------------

[(0,
  '0.533*"amountofmoney" + 0.301*"usa" + 0.228*"citi" + 0.221*"
     data" + 0.159*"use" + 0.124*"compani"'),
 (1,
  '0.528*"amountofmoney" + 0.341*"usa" + -0.226*"data" + -0.189*"
     use" + -0.135*"ai" + -0.126*"compani"'),
 (2,
  '0.478*"citi" + 0.343*"00" + -0.271*"emiss" + -0.247*"use" +
     -0.227*"data" + -0.152*"co2"'),
 (3,
  '0.434*"data" + -0.411*"emiss" + 0.248*"ai" + -0.218*"co2" +
     -0.200*"electr" + -0.174*"citi"'),
 (4,
  '0.294*"data" + -0.281*"compani" + 0.261*"00" + 0.232*"ai" +
     0.218*"citi" + 0.212*"emiss"'),
 (5,
  '-0.419*"debt" + -0.344*"trillion" + -0.256*"80" + -0.250*"60"
     + -0.223*"70" + -0.211*"50"'),
 (6,
  '-0.499*"data" + 0.442*"ai" + 0.199*"use" + 0.177*"car" +
     -0.164*"compani" + 0.148*"case"'),
 (7,
  '0.366*"citi" + -0.221*"say" + -0.203*"said" + 0.182*"scooter"
     + 0.174*"vehicl" + 0.161*"urban"'),
 (8,
  '-0.265*"compani" + -0.254*"franchis" + -0.231*"00" + -0.214*"
     busi" + -0.207*"ai" + 0.196*"said"'),
 (9,
  '0.412*"car" + -0.261*"citi" + 0.217*"data" + 0.201*"vehicl" +
     -0.165*"compani" + -0.144*"said"'),
 (10,
  '0.240*"market" + 0.237*"growth" + -0.226*"said" + -0.219*"
     compani" + 0.214*"europ" + -0.203*"use"'),
 (11,
  '0.210*"franchis" + -0.209*"scooter" + -0.186*"00" + -0.178*"
     said" + -0.177*"growth" + 0.175*"urban"')]
```

The more topics one seeks for in the computation of LSA and LDA, the more specific those topics will be, in principle. This is a statistical computation and the procedure that extracts those topics by combining words can sometimes lead to nonsensical or non-physically existing topics. These topics still make sense mathematically to encapsulate the documents into (for more information, see Section 2.2.2).

3.2.3 Word embeddings

The document relation for this model is done through word embeddings. We have opted for using *GloVe*, a popular dictionary of embeddings trained over a corpus of billions of words (Pennington et al., 2014). The semantic meaning is inferred into word embeddings from the frequency of occurrence of such words. The more probable that two tokens are to appear together in a sentence, the more similar their embeddings will be, while respecting some other side relations among these embeddings (see Section 2.2.4 for more information). GloVe is a so-called *static embedding* model, where the vectors obtained for the words are fixed, i.e. not context sensible. For this reason, we will explore some variants using some dynamic word embedding models, such as *BERT* (Devlin et al., 2018) and *Flair* (Akbik et al., 2018).

We store these token representations as nodes into the knowledge graph. These embeddings can be downloaded from their website (*https://nlp.stanford.edu/projects/glove/*). The nodes are of type *Word* and have two properties:

- Name: the name of the token.
- Embedding: vector representation of the embedding for that word.

Combination of word embeddings

Word embeddings are representations of tokens. Since the documents have already been processed, the remaining tokens in the corpus are those which confer meaning to the texts. In order to obtain vectors that represent documents, we require some sort of combination method that takes into account the embeddings from the words that compose a document. There are some approaches regarding this:

- Addition: adding the word embeddings will create a vector with the value of the individual words of the text. This is a similar approach as how humans read texts, which is in fact the stacked composition of its words. This method, however, will experiment trouble when calculating the Euclidean distance of texts with significant difference in length, as their representations will point to locations quite apart in the vector space.
- Addition with weights: the same approach, however with weights attached when computing the summation. An example of this could be using the weights from the TF-IDF decomposition. The idea is to emphasize the embeddings of the document's keywords. The previous problem with the length still appears, as we do not apply normalization to the vectors.
- Average: the average of the embeddings. This is a technique widely used in the field. Calculating the average of the embeddings belonging to the words composing a text to create the vector representation of the document fights the problem of documents with different length.

Comparison of words

An alternative way to compute document distances is to match words by pairs and calculate the total Euclidean distance. For example, we could calculate the Euclidean distance among subjects, verbs and objects of two texts in order to evaluate the document distance between them. We could match words in pairs by meaning instead. Let us assume two sentences:

"Autonomous vehicles could free up some parking space in cities." –Sentence 1.

"Self-driving cars could make city congestion a lot worse". –Sentence 2.

The subjects of these two sentences are *autonomous vehicles* and *self-driving cars*. The predicates speak about *parking space in cities* and *city congestion*. We can compute the distance among the word embeddings in the vector space for these words and add the values to compute the distance between texts. If we had a third sentence to compare:

"Oxford investigates effects of digital technology on well-being". –Sentence 3.

The distance between *autonomous vehicles* and *self-driving cars* is smaller than between *Oxford* and *autonomous vehicles*, just the same as the distance between *parking space in cities* and *city congestion* is likely to be smaller than between *parking space in cities* and *digital technology*. This leads up to an Euclidean distance smaller between sentence 1 and sentence 2 than between sentence 1 and sentence 3. A possible issue is how to deal with documents with different lengths or with different grammatical structure. Kusner et al. (2015) have an interesting way to compute document distances from word embeddings that cope with such problems. This procedure has been applied to compute the Euclidean distance among documents in one model, where the Relaxed Word Moving Distance (RWMD) has been used. The rest of the models have been computed using the average of the embeddings using different combinations of words from the texts as input: the entire processed corpus or the keywords appearing in the title as well as using different types of embeddings: BERT, GloVe, Flair or combinations. The results can be checked in Chapter 4.

When you use a static word embedding model the representation is unique, so we could save the vector in the graph and query it whenever we need it. This is different for BERT and Flair embeddings. The token representation changes with the sentence, the rest of the tokens around it will infer a specific meaning, modifying the representation of the embedding (Devlin et al., 2018) (Akbik et al., 2018). This gets rid of the polysemy problem that we talked about. However, since these embeddings are computed "on the fly", we will see that the results are not outstanding and that GloVe generally obtain the best results. In some of the models we will attempt to improve them by combining the embeddings from GloVe, BERT and Flair. For that, the library of Flair embeddings allow to combine different types of embeddings. This method stacks the embeddings. Check the code for more detail.

3.2.4 Ensemble method

The models implemented assess similarity among the documents existing in the dataset. For each document, we have sorted all the other documents by similarity and assigned a rank to that relation: the document with the highest similarity score receives *rank 1*, the document with the second highest similarity score receives *rank 2*, and so on. What we have established is a recommendation system where we have the similarity among documents ordered by rank. These results have been stored as relationships in the graph. The relationships are labeled with the algorithm that we have used to predict the relation and have the *rank* attached as a property of that relation (see Fig. 3.2).

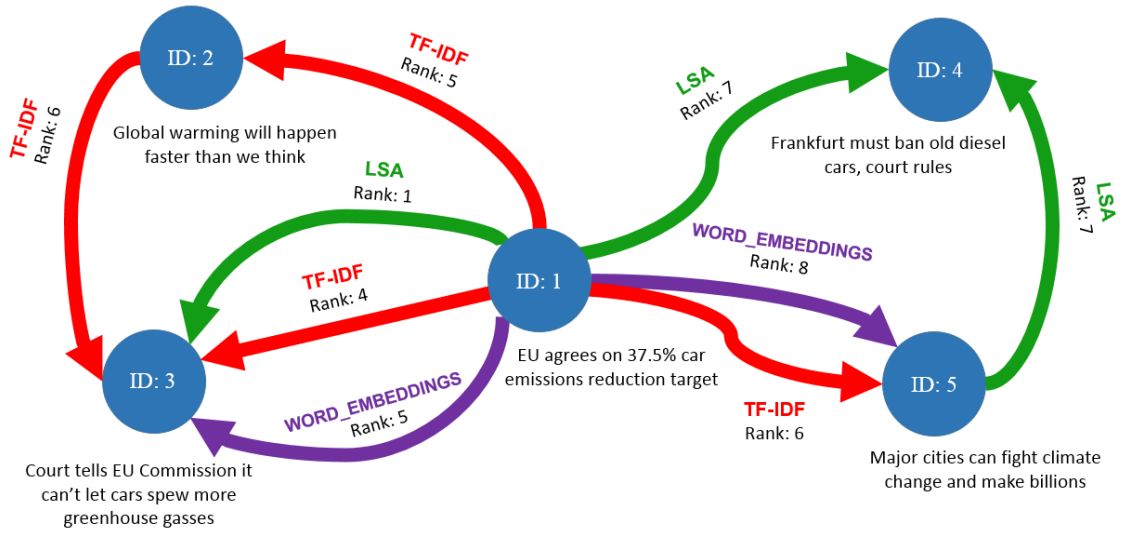

Figure 3.2: Example of documents in the knowledge graph with relationships from the algorithms and their rank attached.

The user may select one of the algorithms implemented to request recommendations for a document. However, in this section we want to study the result of using an ensemble method. An *ensemble method* is a combination of the outcomes of diverse algorithms. The ensemble equation used in this Master's thesis is the following:

$$ensemble\_weight_{i,j} = \frac{1}{m}\sum_{k=1}^{m}\frac{max(rank) - rank_{i,j}^{alg}}{max(rank) - 1}, \tag{3.1}$$

where m is the total amount of algorithms considered in the ensemble method and $rank_{i,j}^{alg}$ is the rank between the document with ID i and the document with ID j obtained through the algorithm alg. For instance, $rank_{1,2}^{TF-IDF}$ is the rank of the relation from document with ID 1 to the document with ID 2 obtained with *term frequency - inverse document frequency*. If the relation between document i and document j has not been predicted by algorithm alg, then $rank_{i,j}^{alg}$ is equal to max(rank). In the example of Fig. 3.2, $rank_{1,2}^{TF-IDF} = 5$ and $rank_{1,2}^{LSA} = max(rank)$. Max(rank) refers to the maximum rank considered in the ensemble method. Normally, only a few other documents are related for each document in

the dataset. Max(rank) is a form of limiting the recommendations in order to not allowing many false positive recommendations. We have selected max(rank) to be equal to 10 for this dataset. In the example showed in Fig. 3.2, the ensemble weight from the document 1 to document 3 would be:

$$ensemble\_weight_{1,3} = \frac{1}{3}\left(\frac{10-1}{10-1} + \frac{10-4}{10-1} + \frac{10-5}{10-1}\right) = 0.740 \tag{3.2}$$

This ensemble method provides a weight in the recommendations in the range [0,1]. The only algorithms that we have considered in the ensemble method are: TF-IDF, LSA and word embeddings ($m = 3$). This is due to the fact that LDA performed significantly worse than the rest of the algorithms in the evaluation results (see Chapter 4, Table 4.6). The results from the ensemble method are stored as relationships labeled as *RELATES_TO* where the weights are attached as properties of the relations (see Fig. 3.3).

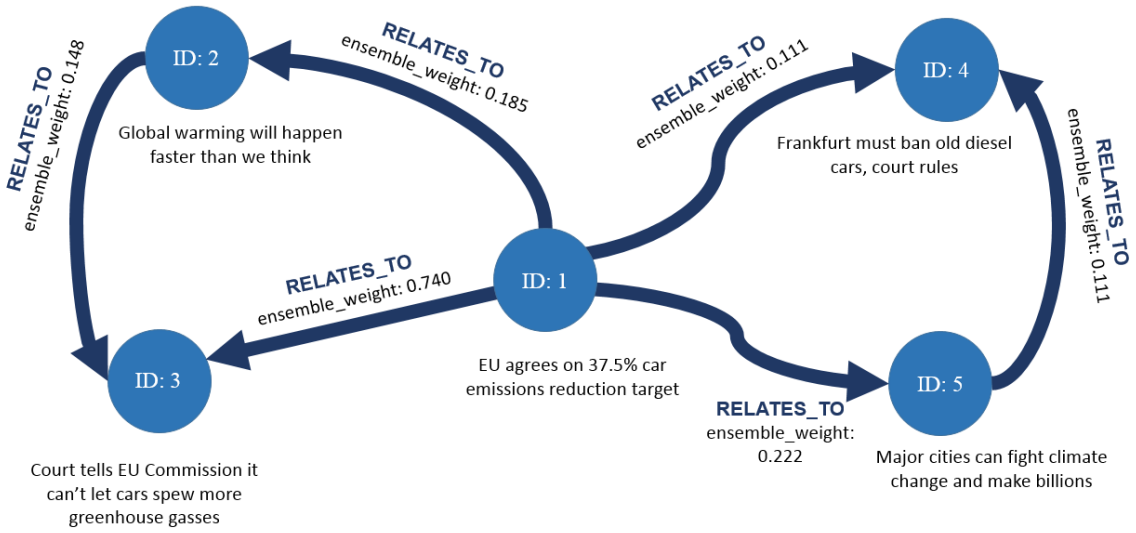

Figure 3.3: Example of documents in the knowledge graph with relationships from the ensemble method and their weight attached.

3.2.5 Community finding

The ensemble method provides with the knowledge of all the algorithms together a trustworthy solution to the recommendations. These recommendations among articles in the knowledge graph create a network. This network represents the map of articles in the database, with its internal connections and communities. A community is a group of nodes, normally interlinked, that have some properties in common and that can be related to a theme. This is useful to find more related articles that are not directly connected by a recommendation link.

Community finding in knowledge graphs is a type of clustering for discovering groups, or communities, in the network. The number of communities is not predefined, the process finds them through an iterative convergence process. There exist several community

finding methods available for knowledge graphs. The ones investigated here are derived from graph theory. The library of tools in Neo4j provides some of the most well known algorithms for community finding: label propagation, Louvain, triangle counting, strongly connected components and balanced triads. These methods have been investigated during the Master's thesis. The ones that have been implemented are label propagation and Louvain. Other community finding methods should be explored in a future work phase.

Label propagation is a semi-supervised algorithm that assigns random labels to datapoints. The algorithms propagates those labels. If more than one label exists that concurs to one datapoint, the label that has more links connected to that datapoint wins and its label gets assign to the node (Raghavan et al., 2007). Label propagation has strength in the running time and the amount of information required. Louvain is a method for community detection that maximizes a modularity score. This score evaluates how much densely connected the datapoints are in a community, compared to how they would be in a random network (Blondel et al., 2008). Louvain has what is called an "intuitive" community structure step in the algorithm. This feature provides the algorithm with the capability to find small communities, even in larger networks. These algorithms are explained in more detail in the book of Hodler and Needham (2019).

These graph algorithms are prone to perform well on complex networks, as they tend to have community structure. However, all algorithms have trouble with convergence to the global maximum as the database grows, due to increasing complexity of the network.

The run of the algorithms have been done with the *Graph Algorithms Playground* tool in Neo4j. The algorithms require a data-type of nodes which we want to find the communities for and the type of relationships to consider in the community finding (see Fig. 3.4). The nodes are of type *Article*, which is the format that we have selected to store the information of the documents (see Section 3.1.1). We have selected the relationships obtained from the ensemble method as the network to be considered in the community finding. The community found for each Article node is stored in a property of the node called *community*.

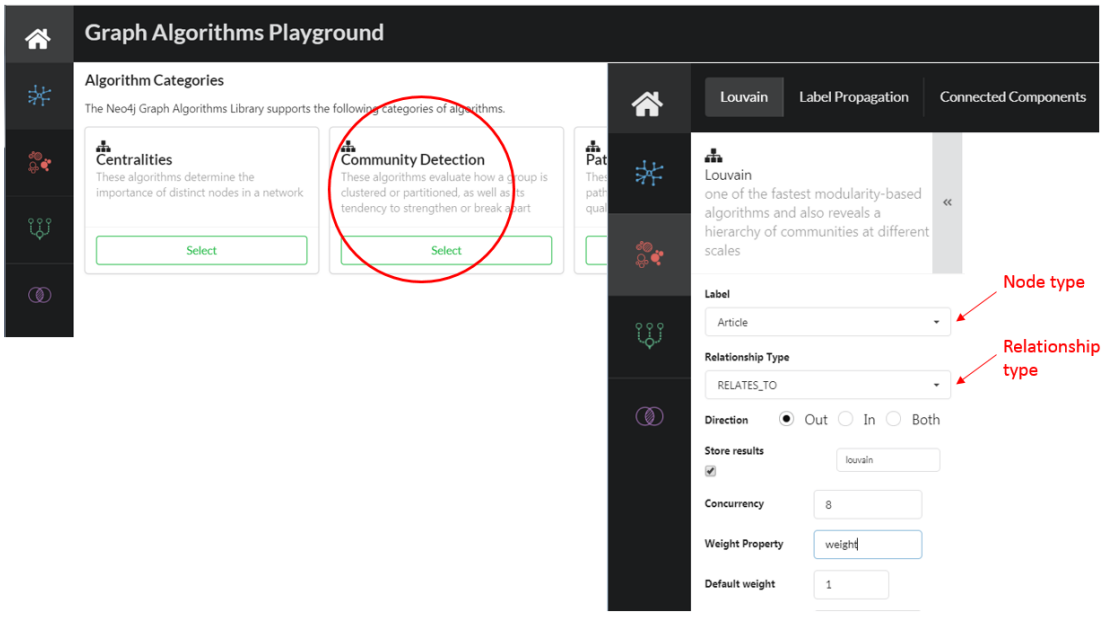

Figure 3.4: Graph Algorithms Playground menu, in Neo4j.

3.3 Graphical user interface

The problem and the approach towards solving it have been described. It is time to define how the user will interact with the system. This is what we call a *graphical user interface (GUI)*. A GUI is the application, the piece of software that the user will use to trigger the actions and visualize the information that the system provide. Through this application a user will be able to read documents, search for a specific one, either by title or by category, and obtain the most related documents for a selected document, plus some extra features in order to provide some more insights (listed afterwards) (Fig. 3.5).

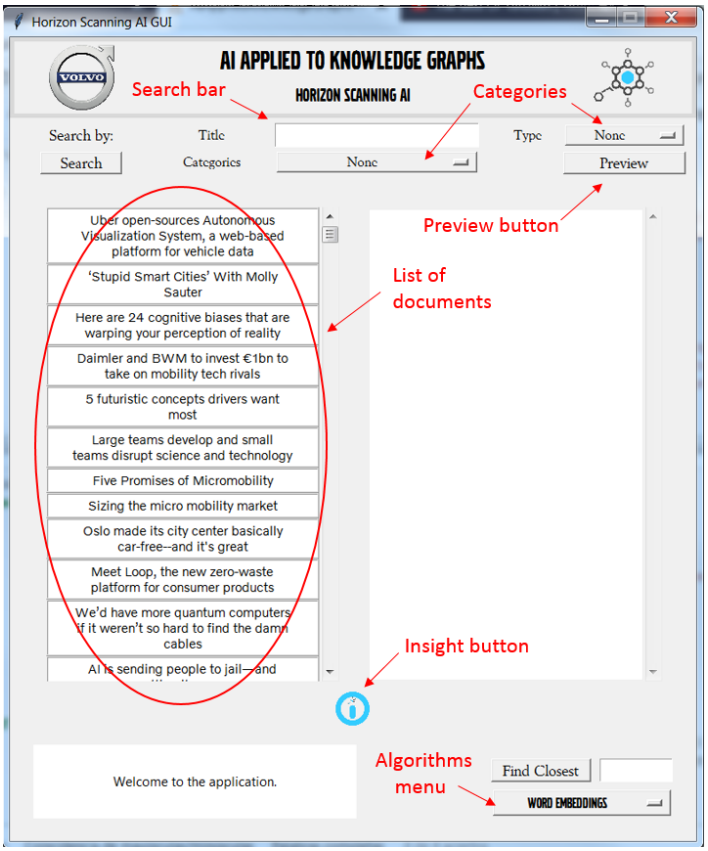

Figure 3.5: Graphical user interface designed with Tkinter.

The application has been designed with Tkinter, the standard Python GUI package. Tkinter is not the only module used for GUI design in Python. It is, however, the most commonly used. Tkinter works on top of Tcl/Tk (http://www.tcl.tk/), so it is required that this module is installed in the system. The application is event-sensible. This means that it is possible for the user to use some events in the keyboard to act on the GUI. For example: press Enter when typing the title of a document to start the search, use the wheel of the mouse to scroll the list of documents or let the application know which window has the user left its mouse on.

Also, the application lets you preview the articles with a button at the top-right corner. The system will help the user quickly find the principal ideas of the article by highlighting some words in the pre-visualization. These keywords are predicted by the TF-IDF algorithm, where the words with high weight are extracted (Fig. 3.6).

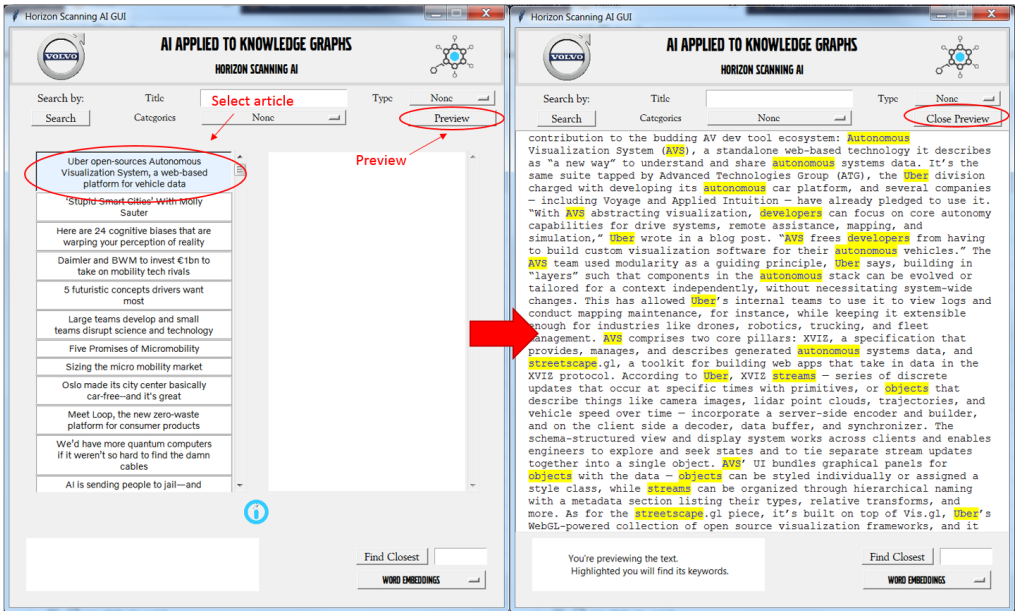

Figure 3.6: Graphical user interface using the preview mode to read a document where the keywords are highlighted in yellow.

The system can also provide some interesting features that the user can access to obtain some insights about either the documents or the predictions done by the algorithms. The user opens this menu by clicking in the icon in the middle, below the lists of documents (Fig. 3.7).

The list of features contains:

- **Read source material**: Opens a tab in the default web browser and read the article from the source material.
- **Idea of the document**: Creates a word cloud with the core ideas behind a document (see Fig. 3.8). This is done using the corpus of the article. The word cloud has been created using the python library (*https://github.com/amueller/word_cloud*), together with Pandas, Numpy and Matplotlib.
- **What else is in the topic?**: Uses the community finding analysis to search for the most commonly used words in the community of the selected article and constructs a word cloud out of it based on the general ideas in the community. This helps give an idea of what one can find in the same community of the article. The words in the community are selected with the support of TF-IDF to include the most relevant terms (keywords). The word cloud has been created using the python library (*https://github.com/amueller/word_cloud*), together with Pandas, Numpy and Matplotlib.
- **Why are they related?**: The system performs an analysis to extract a list of tokens that relate the two documents selected (one in the left column and one in the right column), if they exist. They are found by matching the keywords of the documents

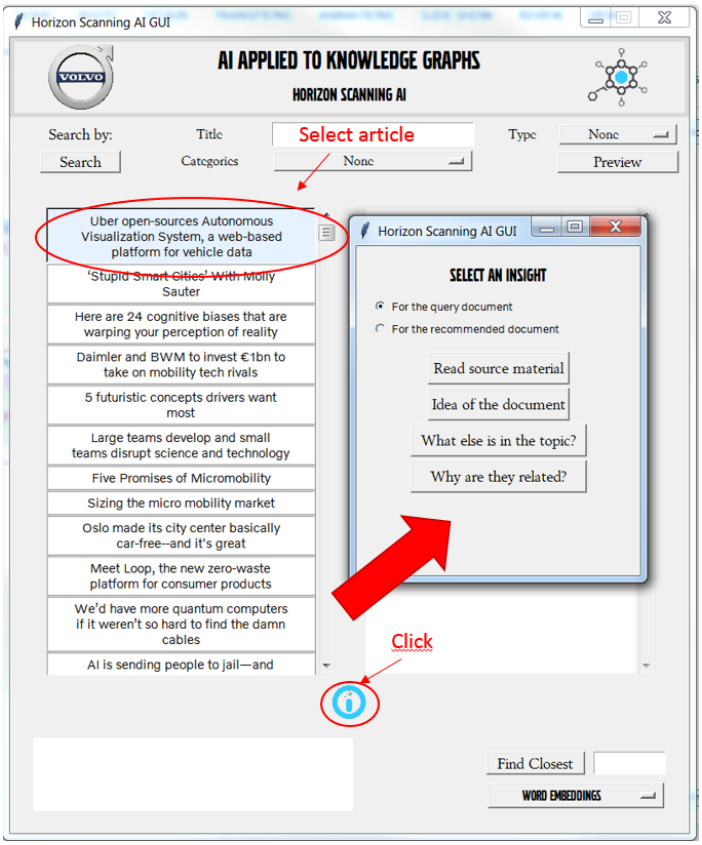

Figure 3.7: Insight menu in the GUI.

that were extracted using TF-IDF. The ones that are in common are mentioned as the reasons why these two documents are related (see Fig. 3.9).

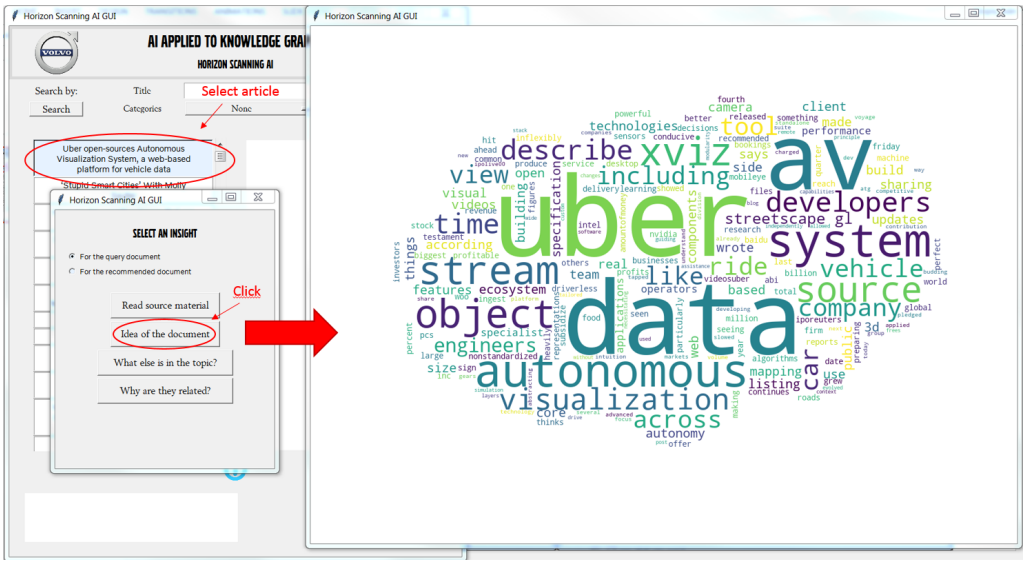

Figure 3.8: Summary of the idea of an article by means of a word cloud in the GUI.

At the bottom-right corner of the application there is the menu of algorithms. The user may select which algorithm they want to use to obtain recommendations. One may also select how many recommended algorithms they want to retrieve by writing a number in the field next to Find Closest. The recommendations are not affected by the categories selected in the search bar. Those are exclusively for searching articles in the database.

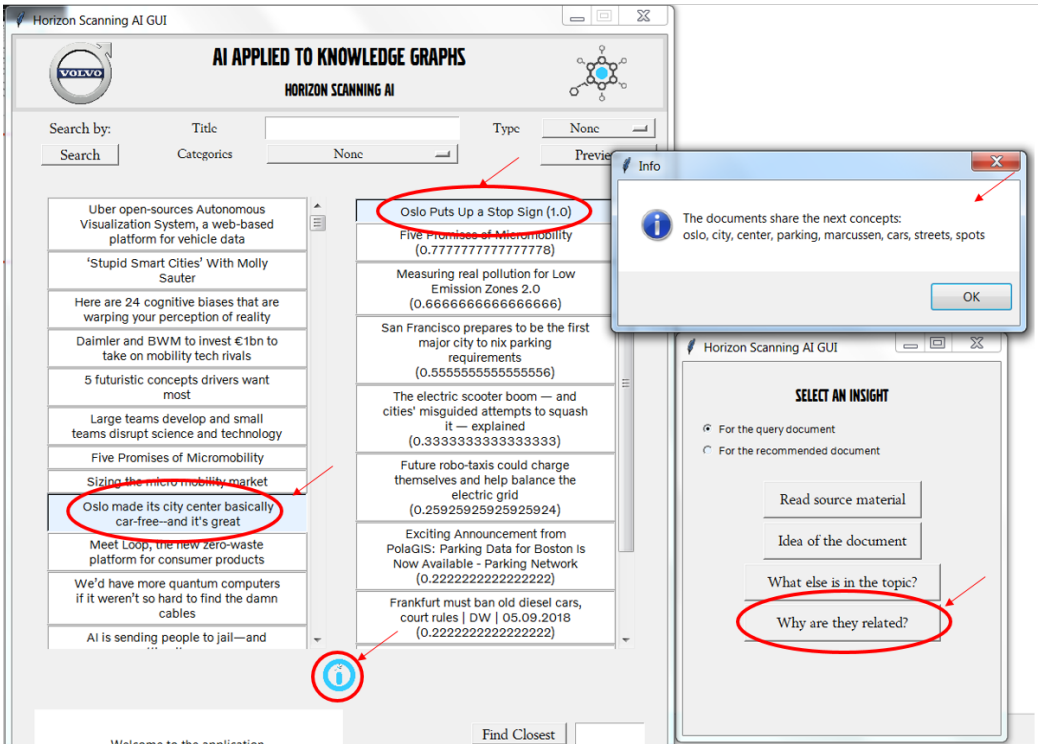

Figure 3.9: The application predicting the relation between two articles using the insight button.

Chapter 4
Evaluation

The first problem that most AI practitioners encounter is the lack of data available to use during training. This is because there is not a lot of high quality data, and if there is, it is not normally tagged. This problem arose in this Master's thesis, where the data available did not contain a training set of tagged relations. The way that this problem has been addressed is based on a similar approach as Losada et al. (2016). The method that we employed creates a test set by using the algorithms implemented in Section 3.2 to discover relations. We have considered the recommendations from these models and clustered the documents that were similar to create the test set.

The evaluation method analyzes the ability of the techniques at performing related recommendations. The metric used is a *recall* where the recommendations from the algorithms are compared against the documents appearing the clusters of the test set:

$$recall = \frac{1}{number\_of\_clusters} \sum_{i=1} \frac{number\_of\_relevant\_retrieved\_relations}{number\_of\_elements\_in\_cluster\ i}. \quad (4.1)$$

By using this score, we can tune the algorithms' parameters in order to obtain the models with the best results in the evaluation. However, this evaluation magnitude does not take into account the order of the recommendations as the recommendations are not ranked. For that, we will use a variant of a magnitude extracted from the paper of Losada et al. (2016). The coefficient, which is called *Ranked-biased precision (RBP)*, is calculated like this (Losada et al., 2016):

$$RBP_{losada} = (1 - p) \sum_{i=1} u_i \cdot p^{i-1}, \quad (4.2)$$

where p is the probability to switch algorithm ("*bandit*" in Losada et al. (2016)) for the next recommendation, i is the rank of the recommendation and u_i is the utility of a single recommendation at rank i. The modified version that we use follows the same concept of

Algorithm	Recall Score	RBP	RBP x Recall
TF-IDF	0.79	91.86	72.15
TF-IDF_NST	0.78	89.03	69.16

Table 4.1: Results of the evaluation on the TF-IDF method.

ranking score where the utility is modified with a decay factor δ to weight each recommendation based on the rank of the recommendation. The equation to calculate the RBP in this project takes the form:

$$RBP = \sum_{i=1} u_i \cdot \delta^{i-1}, \tag{4.3}$$

where the utility u_i for a document recommendation will be 1 if the recommended document is relevant (appears in the cluster defined in the test-set), 0 otherwise. The decay factor is to be decided by the user. We have decided to use 0.85 as the decay factor. The last metric, RBP x Recall, is a parameter that uses multiplication to entail the features from both the recall score and RBP: accuracy and rank order. The results obtained by performing this evaluation method are showcased in Section 4.1.

4.1 Algorithm Results

In Chapter 3 we went through the implementation process for the Volvo Cars use-case of document similarity. It is time to evaluate the performance of the algorithms. The comparison of topic modelling techniques (TF-IDF, LSA, LDA) will be centered around the number of topics as well as whether a stemmed version of the documents makes an improvement in the results. The core of the comparison for word embeddings will be the pre-trained models as well as whether the usage of cosine similarity or Euclidean distance as the metric for assessment of document relation is significant in the evaluation results.

4.1.1 Term Frequency - Inverse Document Frequency

The term frequency - inverse document frequency algorithm allows a few different methods to be implemented. We can compute the term frequency (*tf*) as binary, raw count, term frequency or log normalization. And for the inverse document frequency (idf), we use the unary version, the inverse document frequency or the inverse document frequency smooth. The optimization and testing of all the versions of the algorithms falls out of the scope of this project.

The version that we used for the tf-idf model is the recommended one for documents that vary in size, which uses normalization over their length. In the list mentioned above, this is called *term frequency* for the *tf*, and *inverse document frequency* for the *idf*. The evaluation results can be checked in Table 4.1 where we show a comparison between the non-stemmed version of the words (NST) versus the stemmed version. The stemmed model has approximately 1.2% increase in the recall score and 4.3 % in the RBP x Recall score. This result is not very significant and could vary with a dataset of a different size, although the indication here is to use the stemmed model of the TF-IDF.

4.1.2 Latent Semantic Analysis (LSA)

The latent semantic analysis (LSA) performs a reduction over the complexity of the tf-idf that is defined by the number of topics preset. It is obvious that this number will be decisive when training the models. In this section, we study the impact of the number of latent dimensions in the implementation of LSA models. In order to obtain meaningful models, we use two different heuristics to find numbers of topics to train our LSA models: *coherence model* and *maximum recall score*.

Coherence model

This is a technique used for evaluation of topic models. The pipeline is divided in four phases: segmentation, probability estimation, confirmation measure and aggregation. Gensim implements this technique as part of the models section in its library. Gensim's coherence model implementation is based on the pipeline from the paper Röder et al. (2015).

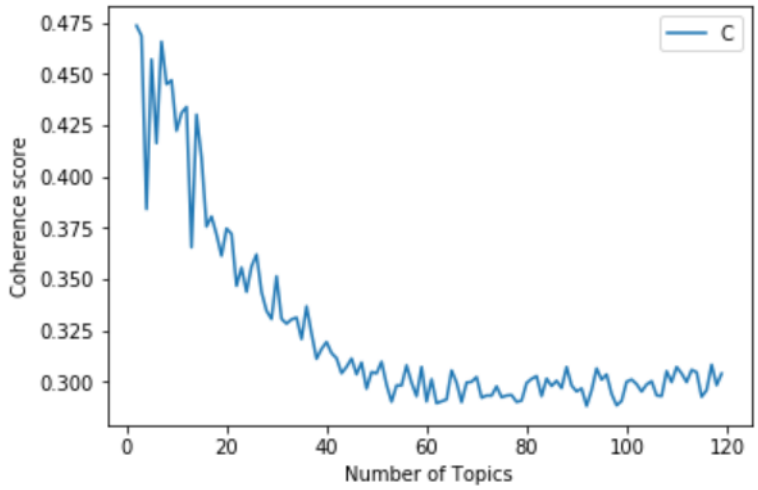

Figure 4.1: Coherence model analysis regarding number of topics for Latent Semantic Analysis (stemmed version).

The coherence model outputs a measure of coherence of the topic model over the dataset, which in a sense is an estimation of a magnitude that establishes some parallelism with the inferred suitability of the topic model. This has been used as a tool to measure the coherence of the model correlated with the number of topics of the LSA model.

The analysis displays the topic coherence of the model for each choice of number of topics (Fig. 4.1). The coherence will increase in front of (or around) a more suitable number of topics for the topic model. As can be seen, the topic coherence have two peaks around 15 and 20, with a maximum coherence for 5 topics. The evaluation results are displayed for the LSA model tuned at these three number of topics (Table 4.2).

Maximum recall score

This heuristic uses the recall score to tune the model. The evaluation in this case is done by using a wide range of topic models tuned at different number of topics to maximize the

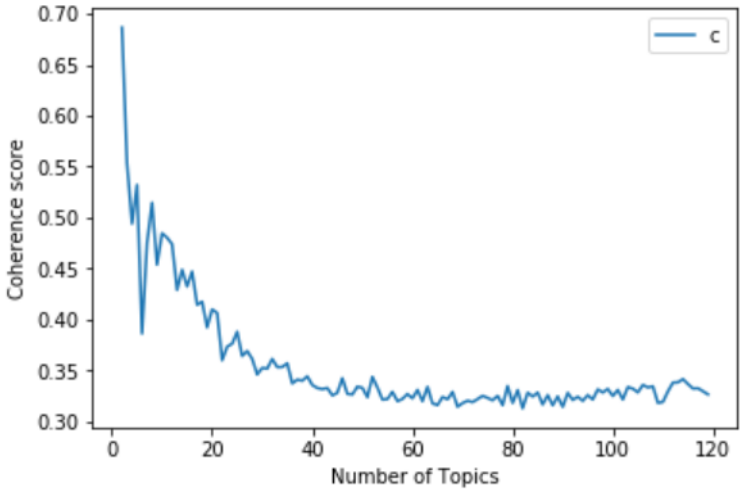

Figure 4.2: Coherence model analysis regarding number of topics for Latent Semantic Analysis (non-stemmed version).

recall evaluation score.

This is obviously the model that will give the best evaluation score as it uses that measure to tune the model. However, an interesting point of view would be to compare the recommendations of this model to the ones resulting from other models on a different corpus.

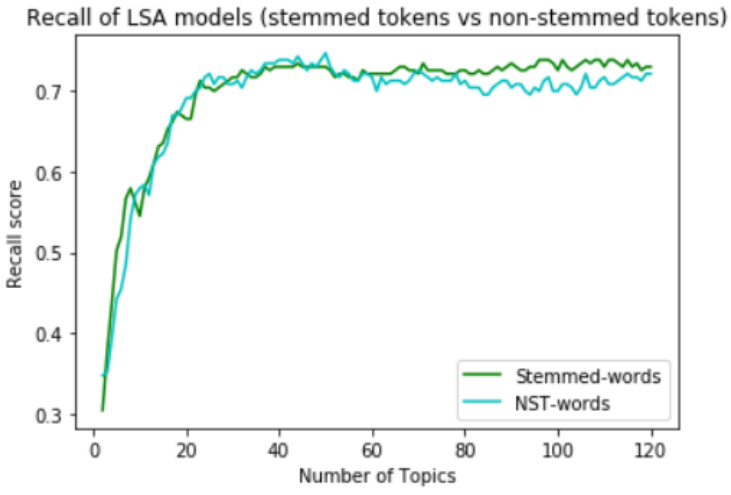

Figure 4.3: Recall score regarding number of topics for latent semantic analysis using stemmed tokens in the model (green) versus non-stemmed tokens (cyan).

Results summary for LSA

This part summarizes the results from the LSA models tuned with the different heuristics explained above. The evaluation is made through the metrics explained at the beginning

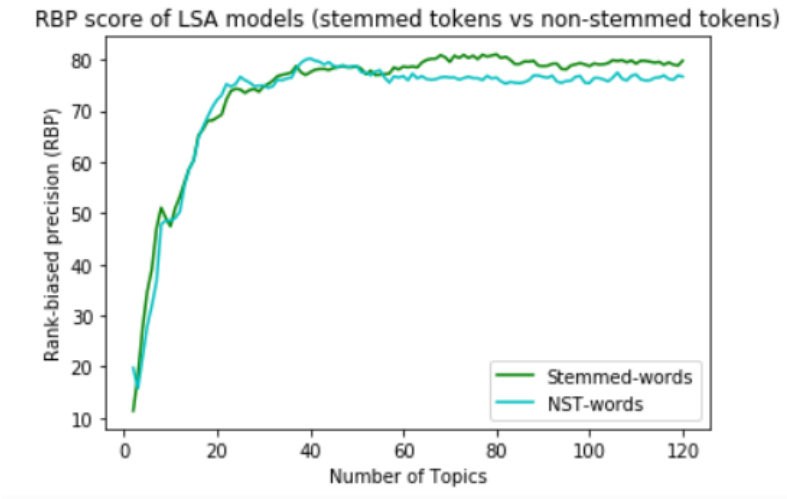

Figure 4.4: RBP score regarding number of topics for latent semantic analysis using stemmed tokens in the model (green) versus non-stemmed tokens (cyan).

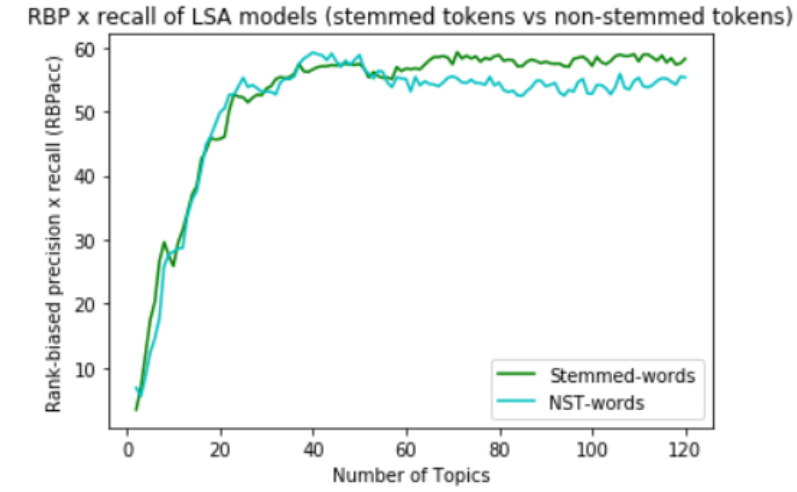

Figure 4.5: RBP x Recall score regarding number of topics for latent semantic analysis using stemmed tokens in the model (green) versus non-stemmed tokens (cyan).

of the chapter and the results appear in Table 4.2 and Table 4.3. The nomenclature used at the table stands for:

(x) = number of topics used for the model
chm = obtained through the coherence model
rc = obtained using the maximum recall score

The best model occurs for the model obtained through maximum recall analysis, as the maximum values in all recall, RBP and RBP x recall are achieved for this model. For the non-stemmed version it corresponds to 40 topics and for the stemmed version to 37 topics

Algorithm	Recall Score	RBP	RBP x Recall
LSA_chm (5)	0.50	34.67	17.41
LSA_chm (15)	0.64	60.33	38.32
LSA_chm (20)	0.67	68.63	45.65
LSA_rc (37)	0.73	78.91	57.58

Table 4.2: Results of the evaluation of latent semantic analysis of stemmed tokens, where the numbers between brackets correspond to the number of topics used to train the model.

Algorithm	Recall Score	RBP	RBP x Recall
LSA_chm_nst (5)	0.44	27.92	12.34
LSA_chm_nst (15)	0.63	60.51	37.92
LSA_chm_nst (20)	0.70	72.39	50.33
LSA_rc_nst (40)	0.73	80.08	58.77

Table 4.3: Results of the evaluation of latent semantic analysis of non-stemmed tokens, where the numbers between brackets correspond to the number of topics used to train the model.

using the Gensim library. One thing that can be observed is that there is not a big impact in the results from using the stemmed version over the non-stemmed version of the corpus until we start increasing the number of topics above 60 (Fig. 4.3 - Fig. 4.5).

4.1.3 Latent Dirichlet Allocation (LDA)

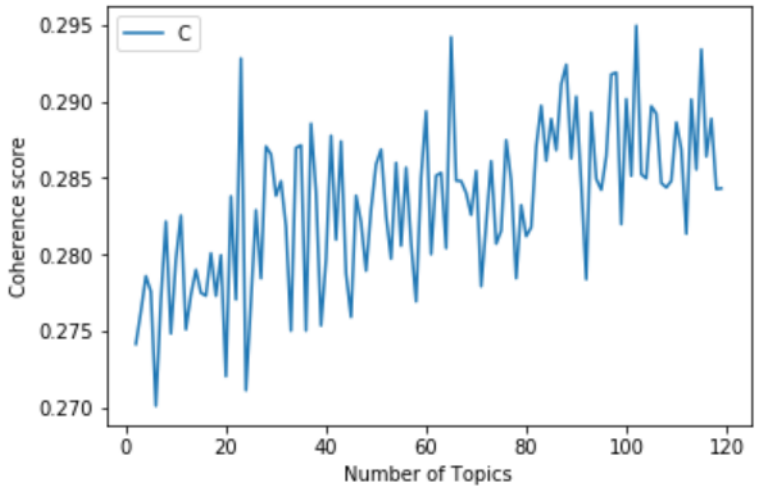

Figure 4.6: Coherence model analysis regarding number of topics for latent Dirichlet allocation (stemmed version).

Coherence model

Latent Dirichlet allocation is a variation of latent semantic analysis (see Chapter 2 for more information). Their internal structure work in a similar way: both require a number of "topics" or "concepts" to work and transform the corpora into a vectorization of those concepts. The coherence model applied can also be applied to LDA for this reason (Röder et al., 2015). The results of the coherence model can be checked in Fig. 4.6. The results are very noisy, whilst the results are significantly lower in the evaluation scores than the rest of the algorithms.

Maximum recall score

This method attempts to find the model which gives the maximum recall score. This does not transform this model into the best model in the other metrics (RBP, RBP x Recall). The peak of maximum recall score can be found with a model of around 8 topics. This goes according to the intuition that the LDA method finds *concepts* (general) instead of *topics*, like LSA. However, as we can see, the signals of the analysis are noisy, just like all the other metrics we have evaluated for this algorithm (Fig. 4.7 - Fig. 4.9). The global results of the algorithm can be checked in Table 4.4.

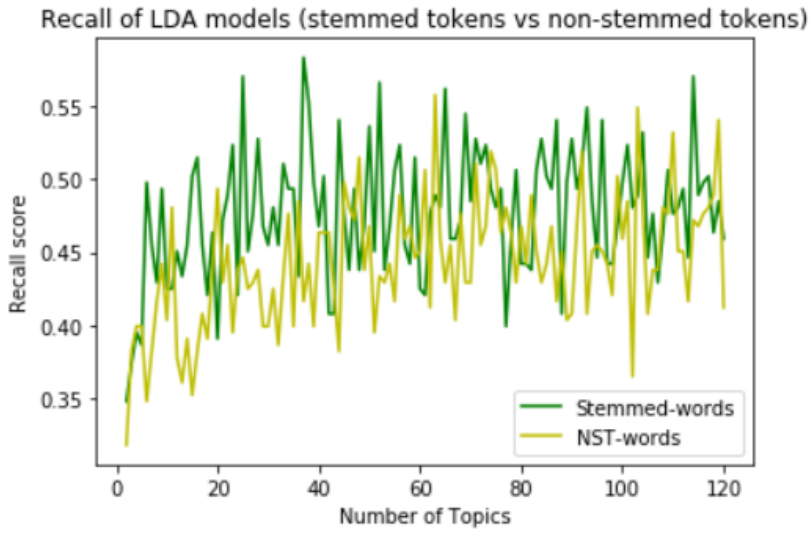

Figure 4.7: Recall score regarding number of topics for latent Dirichlet allocation using stemmed tokens in the model (green) versus non-stemmed tokens (yellow ochre).

Results summary for LDA

This part summarizes the results from the LDA models tuned with the different heuristics explained above. The results have been restricted to the stemmed version as the results showed by the graphs of the maximum recall score did not show an improvement over the stemmed model (Fig. 4.7 - Fig. 4.9) and in order to save time for more studies. The nomenclature used in the table stands for:

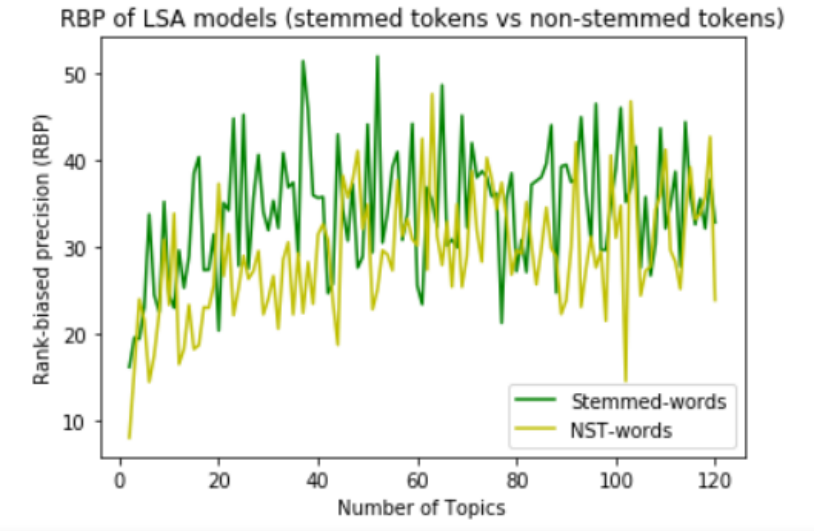

Figure 4.8: RBP score regarding number of topics for latent Dirichlet allocation using stemmed tokens in the model (green) versus non-stemmed tokens (yellow ochre).

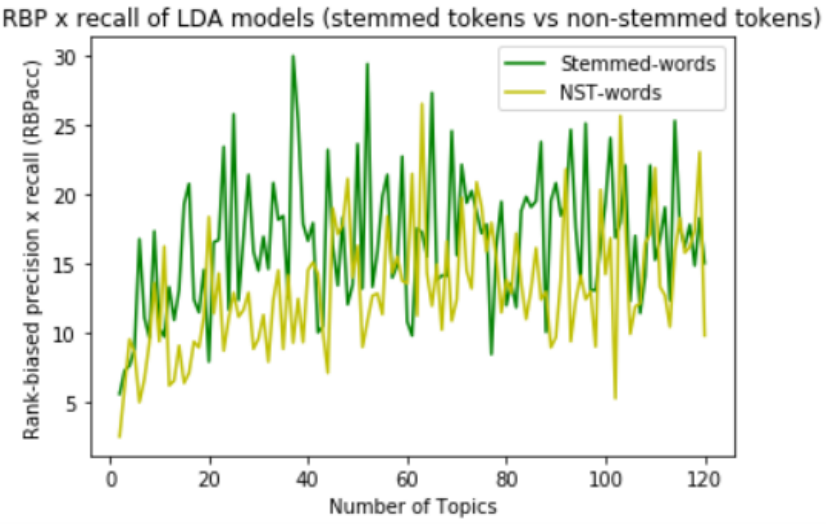

Figure 4.9: RBP x Recall score regarding number of topics for latent Dirichlet allocation using stemmed tokens in the model (green) versus non-stemmed tokens (yellow ochre).

(x) = number of topics used for the model
chm = obtained through the coherence model
rc = obtained using the maximum recall score

The best model occurs for the model obtained through maximum recall analysis, as the maximum values in all recall, RBP and RBP x recall are achieved for this model. That corresponds to 8 concepts using Gensim. One thing that can be observed is that there is not a big impact in the results from using the stemmed version over the non-stemmed version of the corpus due to the abundant noise existing in the results (Fig. 4.7 - Fig. 4.9).

Algorithm	Recall Score	RBP	RBP x Recall
LDA_chm (5)	0.45	25.71	11.57
LDA_chm (10)	0.43	23.34	10.03
LDA_chm (15)	0.41	20.54	8.42
LDA_rc (8)	0.60	35.89	21.53

Table 4.4: Results of the evaluation of latent Dirichlet allocation of stemmed tokens, where the numbers between brackets correspond to the number of topics used to train the model.

4.1.4 Word Embeddings

This section will focus on the exposition of results for the models described in Section 3.2.3. The models have been based on three main types of embeddings (BERT, GloVe and Flair). The metrics used have been either cosine similarity (Section 2.3.2) or Euclidean distance (Section 2.3.1) based on the average of the embeddings of one document, with the exception of one model that has been trained with the method described in Kusner et al. (2015). This method is called Relaxed Word Moving Distance (RWMD), term referred to from now on.

Results summary for Word Embeddings

Several models have been explored by trying different combinations of such systems (see Table 4.5). The metrics are the same ones that we have been using for the rest of the algorithms: recall score, RBP and RBP x Recall. The nomenclature for the models follow the syntax below.

Metric
cs = the metric used in the recommendations is Cosine Similarity
eu = the metric used in the recommendations is Euclidean Distance

Embeddings model
glove = the model has been implemented using pre-trained embedding representations from *Global Vectors for Word Representation (GloVe)*

flair = the model has been implemented using pre-trained embedding with *on-the-fly* dynamic training using representations from *Flair embeddings*

bert = the model has been implemented using pre-trained embedding with *on-the-fly* dynamic training using representations from *BERT embeddings*

Source
stn = the GloVe embeddings are obtained from the website of NLP Stanford Edu *(https://nlp.stanford.edu/projects/glove/,Wikipedia 2014 + Gigaword 5, 300d)*

multi = the Flair embeddings are obtained from Zalando Research using pre-trained

Algorithm	Recall Score	RBP	RBP x Recall
Word-embeddings_eu_glove_stn	0.65	69.22	44.86
Word-embeddings_cs_glove_stn	0.65	69.93	45.62
Word-embeddings_eu_glove_stn_title	0.61	58.63	35.99
Word-embeddings_cs_glove_stn_title	0.62	59.75	36.92
Word-embeddings_eu_flair	0.52	41.59	21.42
Word-embeddings_cs_flair	0.51	42.00	21.27
Word-embeddings_eu_flair_glove_news	0.57	52.60	29.80
Word-embeddings_cs_flair_glove_news	0.58	53.74	30.91
Word-embeddings_eu_flair_glove_multi	0.57	52.60	29.80
Word-embeddings_cs_flair_glove_multi	0.58	54.90	31.58
Word-embeddings_eu_bert_glove_stn	0.68	74.75	51.01
Word-embeddings_cs_bert_glove_stn	0.69	75.83	52.07
Word-embeddings_rwmd_stn	0.72	77.00	55.19

Table 4.5: Results of the evaluation of word embeddings models.

embedding representations extracted from Multilanguage 'Mix of corpora' source material

news = the Flair embeddings are obtained from Zalando Research using pre-trained embedding representations extracted from news articles source material

Note: If not specified by one of these terms, the representations have been extracted from the pre-trained embeddings existing in the library of Zalando Research *(https://github.com/zalandoresearch/flair)*

Other terms

rwmd = the metric used in the recommendations is Euclidean Distance calculated with the Relaxed Word Moving Distance (RWMD) method (instead of the average of the embeddings in the corpus)

title = the input to calculate the recommendations has been the embeddings of the keywords appearing in the title of the document instead of the average of the embeddings in the corpus. The keywords have been predicted using the TF-IDF algorithm.

The values obtained from evaluation of the models show a few insights:

- The best model is obtained using the Relaxed Word Moving Distance (RWMD) method. This method exposed in Kusner et al. (2015) shows the best results in all three metrics with an improvement of 10.7% in the Recall score and 21% in the Recall x RBP score, compared to using only GloVe. However, it is by far the most time consuming computationally (around x4 the time of the second best model).
- The Cosine similarity metric does not show any significant advantage over Euclidean distance, at least in the observations extracted from the models explored in this Master's thesis.

- Flair embeddings does not seem to improve the results in any of the models. However, BERT has improved the results compared to using only GloVe embeddings by 6% in the Recall score and by 14% in the Recall x RBP score.
- Using the full length of the corpus seems to have better impact in performance than using a smaller part of the corpus, such as, for example, the keywords in the title.
- Using a different source material in the pre-trained embeddings does not seem to have a significant impact in performance either, given a sufficiently large size of source material.

4.1.5 Ensembled Method

The ensemble method attempts to portray a better solution than the individual algorithms separately by ensembling the results. In this way, the method avoids being biased towards only one aspect of topic modelling. There are many ways in which the algorithms could have been combined. The objective was to have a weight between 0 and 1 and to emphasize overlapping of recommendations. The result for this method can be seen in Table 4.6.

One insight that can be retrieved from this ensemble method is that the result obtained from the evaluation on the test set is under the ones obtained with *tf-idf*. This is a combination of several algorithms, hence the result gets slightly dragged down by the results from LSA and word embeddings (LDA was not included in the ensemble). In terms of RBP and RBP x Recall, the results from the ensemble method are parallel to those of tf-idf. The expectation is that this method could surpass the individual models with a bigger dataset, improvements on the ensemble equation or by running over a test set that has not been seen by the algorithms during training. This is the real power of ensemble methods.

4.1.6 Community Finding

The results of the communities found with *label propagation* were too specific. Each community was composed of one or two documents, below the size of the sub-communities observed when composing the test set for evaluation. Hence, the results found with label propagation have been considered not reliable as for the communities expected for this use case. The communities found with the Louvain algorithm are more promising. The amount of communities that were discovered is fifteen. This goes in line of the more general themes that we were expecting to find in the graph. All the relationships perform a role in the final result. Hence, false positive recommendations can negatively affect the communities found by the algorithm. To reduce the amount of these, we have filtered out the relationships of the ensemble method with weight below 0.35 (see Fig. 4.10).

Louvain is a type of clustering method; it does not provide recommendations like the other methods provided. We have addressed this problem by using the ensemble method as the recommendation system in order to evaluate the RBP. We use the communities found to calculate the recall score. The results can be checked in Table 4.6.

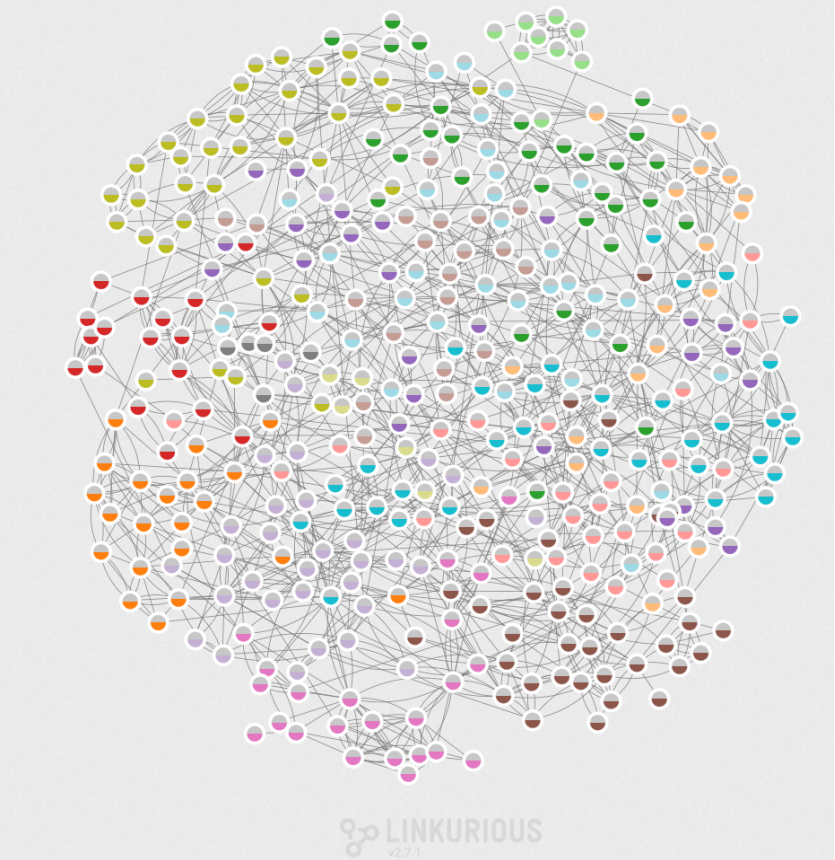

Figure 4.10: Communities found with the Louvain algorithm after filtering out relationships with weight below 0.35. Each community is represented with a different color. Visualization done in Linkurious Enterprise.

Technique name	Recall Score	RBP	RBP x Recall
TF-IDF	0.79	91.86	72.15
LSA	0.73	80.08	58.77
LDA	0.60	35.89	21.53
Word embeddings	0.72	77.00	55.19
Ensemble method	0.78	92.06	71.91
Community finding	0.88	92.06	80.75

Table 4.6: Comparison of the evaluation results of the best model from each algorithm.

Chapter 5
Conclusions

This chapter focuses on the insights gained from the study performed in this Master's thesis. Some conclusions to the specific algorithms have already been explained in the previous chapter, however, here we will focus on some of the conclusions and their consequences.

We will also have a look at some matters that were not seen in Chapter 4, as they were not considered as restrictions or design parameters for the discussed models. A future perspective of the system, specially if put in production, has to take these aspects into account.

5.1 Discussions

This section will talk about the limitations of the algorithms that have been explained and help establish some conclusions about them.

5.1.1 Algorithm Results

One of the first things that have been observed is that the fact of using cosine similarity or Euclidean distance has not had a defining impact in the results. The same behaviour occurs with using stemmed versus non-stemmed words, although in some cases it might indicate something for a different scenario.

TF-IDF This algorithm has proved very interesting, being useful not only as a topic modelling algorithm, but also as a tool to provide other parts of the system with text parsing capabilities: keywords, relevance, insights or even enhancement to other topic modelling algorithms. There has not been observed a significant improvement from using stemmed words versus non-stemmed words.

LSA In the evaluation plots of Latent Semantic Analysis there has not been observed a big impact in the results from using the non-stemmed version over the non-stemmed

version of the corpus until we start increasing the number of topics above certain threshold. This might be an indication that using a stemmed version of LSA might be useful over using a non-stemmed model for larger datasets. Perhaps this topic modelling technique might capture some topics that other algorithms cannot with more information.

LDA This algorithm has fallen behind in the evaluation results compared to other algorithms. One thing that has been observed during testing is that most of the time, the articles were assigned to the same topic ("concept"), leading to cosine similarity of one. What this means is that, either there is not enough data as for finding other concepts that distinguish the articles in the dataset or that the dataset was entirely overfocused on one subject.

Word Embeddings The mechanism behind word embeddings is very interesting as seen in Chapter 2. The possibilities of obtaining more information from the semantic analysis can be enhanced with such properties of word embeddings. The best model has been found using a classical model of pre-trained embeddings with a state-of-the-art optimization algorithm to transform word embeddings into document distances. There is still a lot of space to research new methods and investigate new ways to calculate document distances from word embeddings to push the results even further. BERT has proven to improve the results compared to other models when using the average of the embeddings to calculate document representations; another improvement would be to combine quality embeddings, such as BERT, with combinatorial optimization word embeddings to document distances algorithms.

5.1.2 Run time

This is the time needed to run some part of the system. A large latency in the execution can be stressful or annoying for the users, hence the execution time of the application during usage should be between certain boundaries of acceptability for the user. The range of admissible latency is defined by the company depending on the frequency of execution of that part of code and the alternatives available. In this section, we are not going to compute response times, however we will mention some problems that should be taken into consideration.

Reducing the amount of computations *on the fly* to the minimum is, of course, one way to reduce latency. There are ways to store intermediate results, models and information that has to be used repeatedly once it is computed. The training of models is also a part that usually takes a lot of computation time. The model that used optimization for word embeddings needed x4 more time than the second best model of that algorithm. That was around 90 hours for a single computer with 8GB of RAM. The number of computations required for a new document grows fast ((n-1) computations for a dataset with n elements for each new document). The optimization is costly time-wise; some actions should be taken as the dataset grows. Some recommendations are: using a relaxed version of the optimization algorithm (for example, Relaxed Word Moving Distance (RWMD) was used in this Master's thesis for this exact reason), improving the power of the machine, either by using a machine with more computational power or by using distributed computing, using CUDA to take advantage of GPU computing or using a different method that is cheaper computationally.

5.1.3 Scalability

The scalability is another issue to take into account when putting a system into production stage. Normally, a system will grow over time, including more data, performing more tasks or adding more features that require larger systems. Everything could end up not fitting into memory; the usage of generators may useful.

In terms of algorithms, dictionaries often include thousands of terms. The ones used here included more than 6,000 tokens in the stemmed model and around 10,000 in the non-stemmed version. The vector representations extracted from tf-idf have that same length. The algorithm might break when the size of the dictionary is too big as for handling the document vectors created. Latent semantic analysis and latent Dirichlet allocation might improve their performance with a larger corpus with the possibility to find more meaningful topics or concepts. The size of the dictionary might still cause some trouble if it is too large nonetheless. Other techniques have to be researched that could take advantage of big data, such as graph algorithms, deep learning or knowledge-base solutions.

5.2 Improvements and future work

There is a lot of space for improvement, algorithmic-wise, data management-wise and capability-wise. The community finding algorithms explained in Section 3.2.5 do not allow for community overlap. This might be an issue in our use-case, where we might be interested in finding overlapping themes or sub-themes inside a theme. Other methods to be investigated in the future can solve this issue. There are other features that a user might find useful, more information about relations and other methods to compute more relations, while adapting to the aspects mentioned in Section 5.1. There are a couple of algorithms that have been researched, but that could not be implemented in the scope of this Master's thesis. These algorithms are briefly explained in the next sections.

5.2.1 Doc2Vec

The vectorization of a document (Doc2vec) is the extension of the idea explained in Section 2.2.4 to a full text. This technique was exposed by Mikolov in Le and Mikolov (2014) only one year after creating the word representation Word2vec in 2013.

The idea is to train a vector representation of a document. This is a technique that followed Word2Vec. The vectorization of an entire document, the doc2Vec, is achieved by adding a new vector alongside the vectors representing the words during training; this time representing the full document itself (Fig. 5.1).

The result will be representations for documents that will keep the context of the words included, as they will be bound to the n-grams containing those words. The representation will be closer to similar documents in the vector space and it can be used for topic modelling, such as document matching or document similarity.

These vector representations depend highly on the corpus that they have been trained upon. The form that those will take will depend on the word embeddings that at the same time are based on the n-grams surrounding them, giving context to the document vectors.

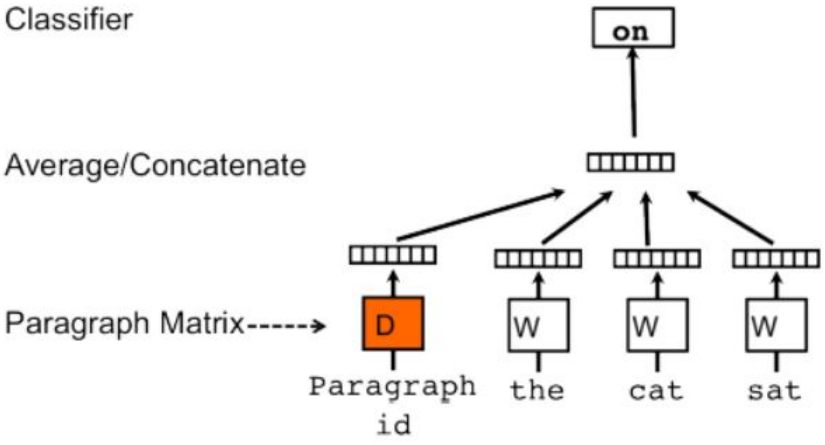

Figure 5.1: Training schema for Doc2Vec embeddings as continuous bag of words (PV-DM model).
From: https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

5.2.2 Knowledge-base algorithm

This class of algorithm uses the power of a knowledge-base to extract information about the relations between two pieces of corpora. The terms extracted during pre-processing are put in a graph and related through the knowledge-base. This type of architecture provides a system with information that would be difficult to extract otherwise. How to use this information and combine it to compute document similarity evaluations would be a different aspect of the problem.

There are some attempts at solving this problem, such as the research performed by Schuhmacher and Ponzetto (2014), where they create a method to compute document modeling with a knowledge-base solution. DBpedia is an example of a knowledge-base with information extracted from Wikipedia. This is the product of a collaboration of Leipzig University, Open Link Software and the Free University of Berlin in 2007. An important aspect is to keep the database updated, so new relevant information may contribute to obtain information from the corpora.

Bibliography

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. *27th International Conference on Computational Linguistics*, pages 1638–1649.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3:993–1022.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

Deerwester, S., Dumais, S., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ehrlinger, L. and Wöß, W. (2016). Towards a definition of knowledge graphs. *SEMANTiCS Posters, Demos, SuCCESS*.

Gabrilovich, E. and Markovitch, S. (2007). Computing semantic relatedness using wikipedia-based explicit semantic analysis. *International Joint Conference on Artificial Intelligence*.

Gatys, L. A., Ecker, A. S., and Bethge, M. (2016). Image style transfer using convolutional neural networks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423.

Harris, Z. (1954). Distributional structure. *WORD*, 10(2-3):146–162.

Hodler, A. E. and Needham, M. (2019). *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 1 edition.

Kusner, M. J., Sun, Y., Kolkin, N. I., and Weinberger, K. Q. (2015). From word embeddings to document distances. *International Conference on Machine Learning*.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. *International conference on machine learning*.

Lin, D. (1998). An information-theoretic definition of similarity. *Icml*.

Losada, D. E., Parapar, J., and Álvaro Barreiro (2016). Feeling lucky? multi-armed bandits for ordering judgements in pooling-based evaluation. *Proceedings of the 31st annual ACM*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26:1–9.

Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1–135.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E 25th Anniversary Milestones*.

Ramos, J. (2003). Using tf-idf to determine word relevance in document queries. *Proceedings of the Conference on Research and Development in Information Retrieval*.

Rehurek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.

Resnik, P. (1995). Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1–6.

Röder, M., Both, A., and Hinneburg, A. (2015). Exploring the space of topic coherence measures. *Conference on Web search and data 2015*.

Schuhmacher, M. and Ponzetto, S. P. (2014). Knowledge-based graph document modeling. *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 543–552.

Wu, Z. and Palmer, M. (1994). Verbs semantics and lexical selection. *Proceeding ACL '94 Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138.