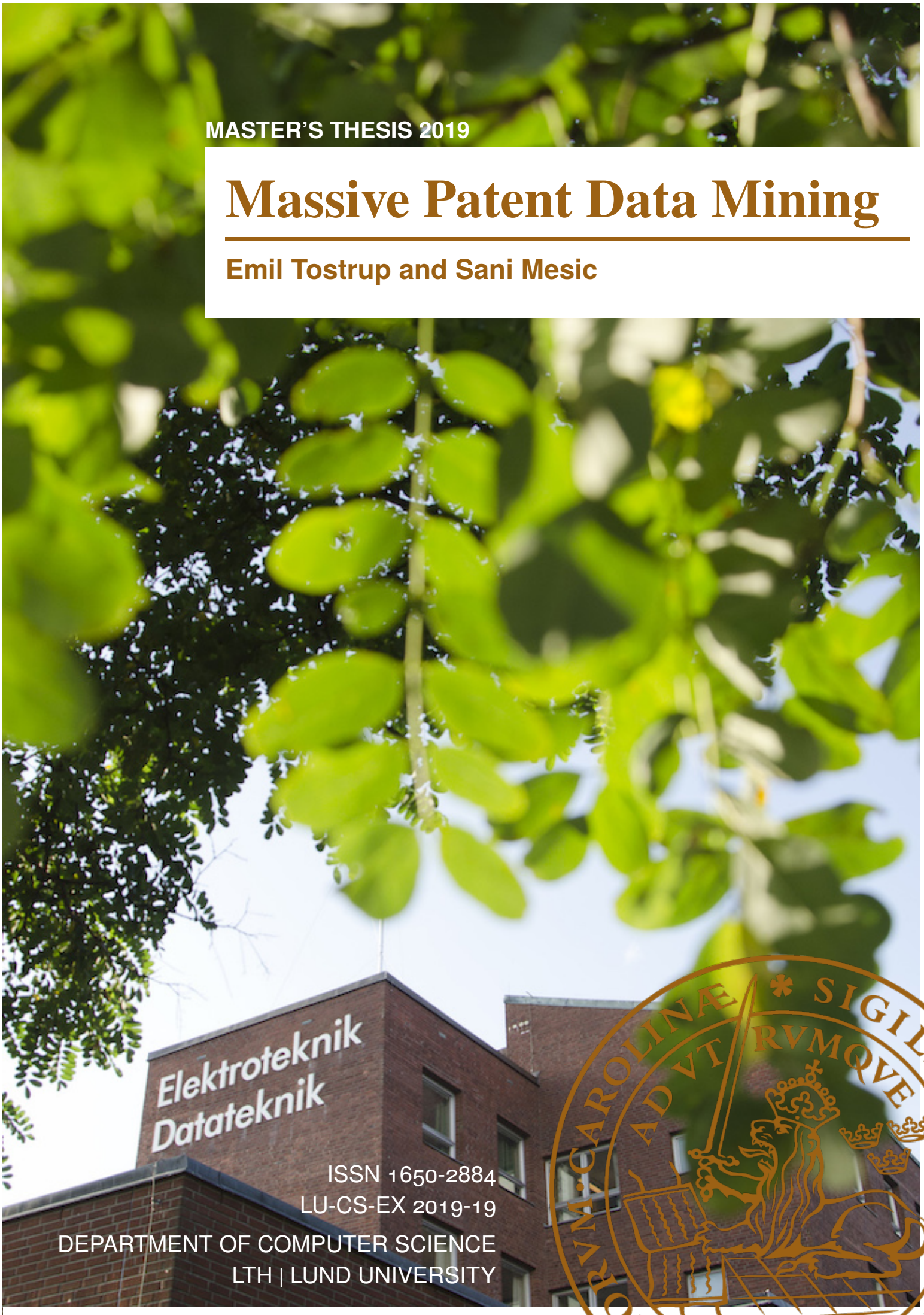

MASTER'S THESIS 2019

Massive Patent Data Mining

Emil Tostrup and Sani Mesic

ISSN 1650-2884
LU-CS-EX 2019-19

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-19

Massive Patent Data Mining

Emil Tostrup, Sani Mesic

Massive Patent Data Mining

(A semantically intelligent search engine for patents)

Emil Tostrup
dat14eto@student.lu.se

Sani Mesic
dat14sme@student.lu.se

July 5, 2019

Master's thesis work carried out at Ström & Gulliksson and Qlik.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Filip Skarp, filip.skarp@sg.se
Victor Olsson Fekadu, victor.olsson.fekadu@sg.se
José Díaz López, jose.diazlopez@qlik.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

This Master's project presents a system to carry out semantically intelligent searches in patent databases. Prior art study is a crucial step in the process of drafting a patent application for an invention. The current state of the art search engines such as Google Patents, Orbit or Patsnap are based on keyword searches, which is a problem because important information can easily be missed. The semantics of a patent are essential, and the difference between inventions often exist in the smallest of details. With an exponential growth in patent data worldwide, finding the correct invention is becoming increasingly difficult.

This report describes a cloud-based system with the capacity of handling a massive amount of patent data. Three different topic models have been implemented and compared to one another, with the focus of finding relevant search results efficiently. The models are based on transforming word vectors obtained from patents to feature vectors, using different methodologies. When the user inputs a query to the system, the query is transformed to feature vectors and inferred to the model which returns the most similar patents to the query.

The thesis suggests that the word embedding based topic model *Doc2Vec* provided the best results for the problem, in regards to scalability, efficiency, and accuracy.

Keywords: patent text mining, information retrieval, natural language processing, big data

Acknowledgments

This Master's project has been a product of close collaboration with several persons to whom we would like to show special gratitude.

We would like to thank José Díaz López at Qlik for invaluable guidance and support during the development of the system architecture.

We would also like to thank Filip Skarp and Victor Olsson Fekadu at Ström & Gulliksson for developing the problem formulation and their knowledge of the patent industry, thus giving us this unique Master's project opportunity.

We would also like to thank Pierre Nugues at LTH for constant feedback regarding the development of the algorithms and natural language processing.

Contents

Chapter 1
Introduction

Intellectual property in general and patents, in particular, are becoming increasingly important to companies around the world. Statistics presented by the World Intellectual Property Organization, WIPO, shows that the number of international patent applications increased by more than double the amount between the years of 2004 and 2017. The trend is likely to continue (WIPO, 2019). With this significant number of patent applications, there is a need to sort, categorize, and filter the information it contains to make use of it.

This Master's project was developed in close collaboration with Qlik (2019) and Ström & Gulliksson (2019). Qlik is a company that provides end-to-end data management and analytics platforms built to transform business. In this Master's project, Qlik provided the necessary system and software knowledge, together with means to set up a functional cloud architecture. Ström & Gulliksson is a European patent firm with vast experience and substantial knowledge of Swedish, European and international patent law, and in this Master's project, Ström & Gulliksson are the final users of the system. The problem formulation together with the system requirements was developed by them.

This introduction chapter aims to provide an overview of the problem. We will define the main objective, previous related research done in this field, limitations the project may have, its scientific contributions to disciplinary fields, and finally a report outline.

1.1 Problem formulation

Due to the current massive quantities of patent data, obtaining relevant correlations concerning similar patents has become increasingly difficult. Patenting an innovation requires searching through already existing technology and attempting to figure out if this innovation may coincide with previously patented technology. An International Patent Classification, or IPC, is a hierarchical classification system used to classify patent documents according to the technical fields they pertain. In addition to patent related fields such as the IPC, assignee, date of filing, date of priority and title, the relevant differences may often

be found in the nitty-gritty details of the patent's body text. The task is daunting, and by providing a program which may effectively filter and categorize patents utilizing machine learning approaches, and finally visualizing them in a comprehensive manner, the process can be vastly simplified.

As an inventor with an idea, it can sometimes be difficult to exactly define new technology, but you still want to be able to patent the idea to protect it. In order to confirm that someone else hasn't patented a similar idea, the examiner examining the patent application will have to find keywords which define the technology. The keywords are being entered in search tools to find any previous usage of these words. As it currently works, context is not taken into consideration, and as a consequence, may not be detected by the keywords search. This may result in involuntary patent infringements.

1.2 Objective

The principle of a program like this is based on collecting large quantities of data from existing patents. The collected data will be parsed and stored in a word corpus, containing all important sections of all the analyzed patents. The goal is to use the system by inputting a query, which is to be compared with the patent corpus using different methodologies. The output of the system will contain a list of the n most similar patents of the collected patent data set to the query. A concise breakdown of the objectives is given in the list below.

1. Download a large patent corpus which will be the base of a locally configured repository. We have obtained the patents from the United States Patent and Trademark Office (USPTO, 2019) data bank, which are stored either locally or on a cloud storage.

2. Investigate different types of machine learning algorithms related to text mining with the goal of finding the best algorithm with regards to efficiency and accuracy. The objective will be to research how information retrieval techniques can be applied for patent text data mining to produce a desired output.

3. Create a web application with a user friendly design so that the implemented functions can be executed remotely.

Additionally, a better way to handle the context of whole sentences will be investigated and implemented, as a way to facilitate an improved clustering of documents, where words and their context are taken into consideration.

1.3 Related research

In early studies conducted during the 1960s regarding massive technological development, Schmookler (1966) examines whether inventions are driven by market-specific demands or if they are self-generating. Schmookler's approach is the first in this area utilizing statistical analysis to study the correlation between technological and economic issues.

Beside Schmookler's pioneering work in statistical patent analysis, Hehenberger and Coupet (1998) were the first to apply text mining techniques to patent analysis. Since then, several research studies have been conducted, as well as practical implementations of business models which may provide a standard for accurate patent searching. However, no standard is currently widely used across the industry. Tseng et al. (2007) analyze text mining techniques for patent analysis in their study. They claim that automatic tools for assisting patent agents, or other individuals making decisions in the patent industry, are in great demand. Tseng et al. provide techniques for using topic maps in order to cluster related patents to optimally suggest patterns, relations, and trends to the data. However, Tseng et al. assert that more patent-related mining issues can be studied and that the current methodologies for analyzing patents deserve more attention.

Lucheng et al. (2010) investigated the general idea of combining data mining techniques and patent information analysis. This study describes how the process of implementing machine learning algorithms such as K-means and neural networks can classify the distribution of the main IPC to produce some visual charts. Lucheng et al. analyze patent information with the goal of providing enterprises with information related to, for example, less studied technical fields, to save time. Ultimately, this information is valuable to an enterprise for enhancing technical innovation and competitiveness.

To the best of our knowledge, the method we have designed in this Master's project had not been investigated before. Text mining algorithms have been applied to collections of patent data previously, as shown by e.g Hehenberger and Coupet (1998); Tseng et al. (2007) and Lucheng et al. (2010). However, by evaluating the difference between the bag-of-words and word embeddings methods and how the respective methods perform on large sets of data to spot similarities, we extend a practical use to the current research. Our implementation also provides a good method using existing software libraries and parallelization to handle large collections of data as smoothly as possible.

1.4 Limitations

We will have to make certain work limitations to ensure that the Master's project is completed in the allotted time period with the given resources.

Data management. Data management is a big concern for this Master's project. The ideal system would be to create a database containing all of the world's patents in a single database, but for testing and efficiency purposes a reduced dataset will be used. During development, the created system will serve its purpose as a prototype, tested and improved upon thoroughly, before handling much more data using a scalable cloud architecture.

Hardware. We implemented the system in two steps. In the first step, we had limited computing capacity. In the second one, we scaled up the processing power with the help of a cluster of nodes in the cloud. With that being said, until the system has a scalable cloud architecture, many problems are going to arise due to hardware limitations. The system will be implemented on local machines with 8 GB RAM, 2 core processors and an approximate hard drive storage of 50 GB. This was bound to be problematic during many processes of the system, which enhanced the necessity of a cloud architecture.

Once a scalable product has been delivered, hardware limitations are less of an issue. The required cloud resources will be provided by Qlik as needed. There may however be further issues in the future in case of massive system load, but it is not relevant for the Master's project.

Scope. We implemented a system with features necessary to cover the objectives explained in Section 1.2. The objectives served as a base for the Master's project's limitations, and any work not addressed as an objective can be regarded as out of scope. The system we created is potentially a subject for further development, exceeding the total amount of necessary work required to successfully carry out the Master's project. This essentially means that there are potential improvements that can be done to the system, even after the scope of the project has been completed. Therefore, limitations have to be defined to obtain a workload appropriate to the thesis.

1.5 Contributions

Our Master's project is expected to contribute to knowledge regarding patent text mining by finding a way to build on the existing data mining and information retrieval techniques. We implemented and compared several topic models and analyzed the semantics of patents, with the help of the latest research in machine learning algorithms. Additionally, we compared our best results with the current state of the art patent search engines, to propose a potential improvement. Furthermore, the finished prototype may potentially be a subject of utilization in other fields, where semantics are equally important.

1.6 Report outline

The report consists of four chapters: Approach, Development, Results, and Discussion, each covering unique areas of the Master's project. In addition to these four chapters, the conclusion will highlight the key findings of the work.

The Approach chapter covers the fundamental theory behind the software and algorithms used during implementation, ranging from intellectual property, to natural language processing, data storage, and evaluation techniques. Moving on to the Development chapter, a concise explanation of the implemented system is covered. The approach used is thoroughly explained, together with all key design decisions chosen along the course of the project. The results obtained after evaluation of the system are described in the Results chapter. Finally, a couple of important discussion topics are covered in the final Discussion chapter.

Chapter 2
Approach

This chapter presents the underlying and fundamental approach behind the practical work conducted in this Master's project. In order to get a full understanding of the problem, we explain the theory from the patent perspective as well as the data science perspective.

Firstly, this chapter outlines the background concepts of intellectual property related to this Master's project, together with a brief explanation of the patent parts that we analyzed.

Secondly, we present the approach behind the implementation process, starting with thorough coverage of the natural language processing fields of data preprocessing, corpora, and topic modeling. Additionally, we will explain data mining and how we used learning algorithms to solve our problem. A section regarding data storage is also included, and lastly how the implemented models will be evaluated using a similarity measure.

2.1 Intellectual property

Intellectual property (IP) is a tangible representation of intellectual ideas. IP can be realized as patents, copyrights, and trademarks. Patent attorneys and patent engineers working at corporate offices are constantly dealing with intellectual property issues with existing technologies and inventions. A big problem in finding relevant patents and patent applications exists due to the sheer amount of data in patent databases such as the European Patent Office (EPO) or the United States Patent and Trademark Office (USPTO). It can sometimes be extremely difficult to pinpoint the exact technology and purpose of a new invention, thus resulting in patent engineers being unable to draft new patent applications. Frequently, one specific type of invention's area of technology overlaps with another. The patent of the invention needs to be newsworthy in order to ensure patentability.

Consider a specific type of engine which is a mechanical construction with a certain unique design decision. If the design shares essential features with a second engine's design it can potentially infringe on a patent of the second engine, even if the construction of the engine itself is different. This can be problematic in two ways. Firstly, it is not

entirely clear exactly what type of correlating fields an invention infringes on, due to cross disciplinary features of the invention. Secondly, highly skilled patent engineers have the ability to practically write patents on anything by presenting the technology with different semantics and descriptions, although the words essentially relate to the same technology as another. This is why it is crucial to discover the true underlying meaning behind the words, to find out how the technology actually works.

2.1.1 Patent structure

A patent contains useful information to make sure that the invention prohibits infringements on a similar invention. In this Master's project, we analyze three different parts of the patent: abstract, description and claims:

- The first part is the patent abstract. This section will clearly and concisely break down the core components of the invention.
- The second part is a thorough description of the invention.
- The third part contains claims. The claims are crucial in regards to warning what others must not do if they are to avoid infringement liability.

Moreover, patents contain several other pieces of information like for example the IPC class, assignee, date of filing, date of priority, and title. However, all these parts of information can be found on the first page of the patent. Although the current state of the art in patent search engines provide keyword-based searches, they lack the ability to handle the context of words. This means that if a paragraph is semantically explained in a totally different fashion than what is being searched for, context is not taken into consideration and the search results become inaccurate.

2.2 Natural language processing

Before the creation of the patent text corpus, the retrieved patent texts are expected to go through a pipeline of text processing modules, as a way to prepare the data to be used by the data mining algorithm. The purpose of the text processing phase is to reduce the sample data set to make the learning algorithm more accurate and efficient. Text processing can be performed using different approaches, based on which problem is being solved.

2.2.1 Data preprocessing

Data preprocessing is the art of formatting data before analyzing it, by removing noise, inconsistencies, and missing data. Since low-quality data will lead to low-quality data mining, the preprocessing step is very important for the mining algorithm to provide desirable output.

Data cleaning is related to removing noise and correct inconsistencies in data (Han et al., 2011). An example of correcting an inconsistency can be to convert all capitalized letters to lower-case.

Data integration aims to merge data from different sources to create a more coherent data collection which can find further context from individual words (Han et al., 2011).

To reduce data size from the raw patent text, *data reduction* is applied. The reduction will eliminate certain words which are not considered to be relevant for the mining algorithm to function as supposed (Han et al., 2011). When analyzing patent text, there are some irrelevant data which can be removed. Cardinal numbers do not give any meaningful information to the technology, and can, therefore, be discarded. The same reasoning applies to words that occur very rarely (exists only one time in one patent), and words that occur very commonly (exist in more than set percentage value of the observed patents), and they can also be removed. A significant amount of data in the natural language are so-called stop words, e.g. *it*, *the*, *a*, *an*, *over*, and *they* generally do not provide any useful information and they can, therefore, be removed too.

2.2.2 Unsupervised learning

Unsupervised learning is defined as a learning algorithm, which is used by the model to learn relationships between data that is unlabeled from the beginning, by looking for hidden structures behind the data, and group the raw data together using e.g. *cluster analysis* (Aggarwal and Zhai, 2012). As said by Hastie et al., unsupervised learning is *the act of learning without a teacher* (Hastie et al., 2009, p. 486). Hastie et al. consider a case with a set of N observations $(x_1, x_2, ..., x_N)$, where the goal is to find coherent properties without a supervisor or teacher providing correct answers or degree-of-error for each observation. The properties which are to be found and correlated to each other are defined differently depending on what problem is being solved. In patent text analysis, the relevant properties are essentially weighted values for each word that is being processed. The learning algorithm uses three different sets of data in order to provide the best possible output: the *training set*, the *validation set*, and finally, the *test set* (Gutierrez-Osuna, 2005):

1. Training the algorithm is done on the training set. This data set usually comprises about 50-60% of the total data samples. The training set is used to construct the prediction model to decide how to link inputs with outputs.

2. The validation set is a set of data samples which attempts to determine the performance of how well the prediction algorithm managed to predict the training data. The quantity of the validation set is normally between 20-25% of the total data samples.

3. The final data set is the test set, and this is normally the last 20-25% of the remaining data samples. The test set will determine how well the chosen prediction algorithm is performing on previously unseen data.

Consider the same probability density $Pr(X)$ mentioned by Hastie et al. again. Cluster analysis has the objective of trying to discover multiple convex regions of the X-space that contains modes of $Pr(X)$ (Hastie et al., 2009). Performing cluster analysis on the entire data set, training set, validation set and test set, will answer the question if the probability density $Pr(X)$ can be divided into smaller subsets with the content of a more similar set of data compared to the data set before the cluster analysis. The patent text cluster analysis is

based on clustering patents by having similar values in the same cluster, so that correlations between patents can be observed and analyzed.

2.2.3 Supervised learning

Supervised learning is defined as constructing a model using training data, where for each input there is a desired/correct output value predefined. Compared to unsupervised learning, it can be seen as having a supervisor providing the correct answers. A *neural network* can be used as a learning method to classify inputs and form patterns based on a large amount of training data. In this Master's project, we used a neural network as a supervised learning method. The neural network will create meaning to the input by converting it to feature vectors for each word or pixel. Consider the following image:

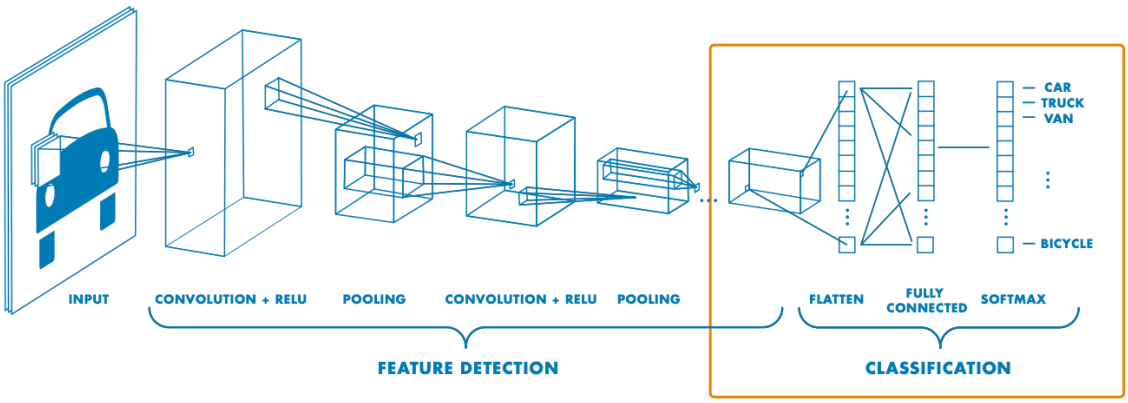

Figure 2.1: Image classification example. Image retrieved from MathWorks (MathWorks, 2018).

The neural network will attempt to classify the image by analyzing it pixel by pixel, and the same methodology is applied to word classification. Once the model has received a substantial amount of training examples during the feature learning process, or training, it will know what type of feature vectors correlate with the new input. Upon receiving a new input, the model will analyze it based on previous data received by concatenating the pixels and finally, in this case, classifying it as a car as the most likely content.

A neural network will always consist of one input layer, one output layer, and N hidden layers. During training, each word or pixel will travel between all layers for the number of epochs specified. For each epoch, the parameters of the model will be tuned for more accurate classifications. However, it is important that the model does not overfit and cause a decrease in accuracy, by iterating too many times in the network. A substantial amount of research and different strategies have been applied in order to prevent a model from overfitting. For example, randomly dropping units during training can be used as a means to prevent overfitting, as explained by Srivastava et al. (2014).

2.2.4 Topic model

Topic modeling is an unsupervised or supervised text mining technique used to measure similarities between words, sentences, or documents. Different topic models use differ-

ent approaches to process significant quantities of data, e.g from large websites such as Wikipedia, Twitter, or Google, with billions of terabytes of data. The general approach is based on vectorizing the words into feature vectors, and subsequently extracting these features and comparing it with other vectors to determine how similar they are, by for example calculating distances between the vectors. We analyzed and implemented three topic models in this Master's project: TFIDF, GloVe, and Doc2Vec.

TFIDF

A *term-document matrix* is a matrix that contains the number of word occurrences for every document, by storing it in a matrix, where each row has a document, and each column has a unique word. We used the *one-hot encoding* to form the term-document matrix as a preparation for the first topic model implemented. One-hot encoding is required when no ordinal relationship exists between the words in the data. A binary representation will be added for each vector in the term-document matrix, where a 1 means that the word exists, and a 0 means that the word does not exist. An example of how a one-hot encoding may look like is shown in Table 2.1.

	cat	*dog*	*fish*	*table*	*chair*	*scarf*	*hat*
Doc 1	1	1	1	0	0	0	0
Doc 2	0	0	0	1	1	0	0
Doc 3	0	0	0	0	0	1	1

Table 2.1: A potential one-hot encoding. This specific example contain three documents: 1, 2, and 3. In this case, the first document is related to animals, the second document is related to furniture, and the third document is related to clothes.

TFIDF, or *term frequency–inverse document frequency*, is used to calculate exactly how important a word is, by checking how many times a word occurs in a document, and offset the value by the total number of documents in the corpus that has this word (Mishra and Vishwakarma, 2015). The approach that TFIDF uses is called a bag-of-words approach, and there are two important statistical values that have to be calculated to form the tf-idf product: the *tf* (term frequency) weight, and the *idf* (inverse document frequency) weight. The feature vectors for the TFIDF-model is calculated by the formula:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}, \tag{2.1}$$

where $tf(t, d)$, the term frequency denoting the number of times a term t occurs in a document d, is multiplied with the idf component of the term t. In this Master's project, we used a raw count weighting scheme to calculate the tf weights as $f_{t,d}$. The idf weights are computed as:

$$\text{idf}(t) = log\frac{1 + n_d}{1 + \text{df}(d, t)} + 1, \tag{2.2}$$

where n_d is the total number of documents and $df(d, t)$ is the number of documents containing term t. Finally, the resulting TFIDF arrays are then normalized by the Euclidean

norm as:

$$v_{norm} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_n}^2}} \tag{2.3}$$

The resulting vectors are given as a matrix with the calculated TFIDF values.

GloVe

GloVe (Global Vectors for Word Representation) is a word embedding based topic model developed by Pennington et al. (2014a). Similar to TFIDF's one-hot encoding approach, word embedding is another method of representing words as vectors. When using a word embedding representation, each word is mapped with a vector in a predefined vector space. Word embeddings are used in situations when the context of the words is important to discover, and not just word count across the corpus. According to Pennington et al., GloVe is a global log-bilinear regression model that combines and utilizes the advantages of two different families in topic modeling, namely the global matrix factorization and local context window methods. The TFIDF-model uses a global matrix factorization to represent its feature vectors, which effectively takes advantage of learning the global corpus statistics. The downside of this is that it does not take the context into account, i.e the local surroundings of the analyzed word. In the local context window case, the most popular methods used in industry are the Skip-gram model and Continuous Bag of Words (CBOW) developed by Mikolov et al. (2013). These models are used to capture syntactic and semantic word similarities and relationships between distributed vector representations of words.

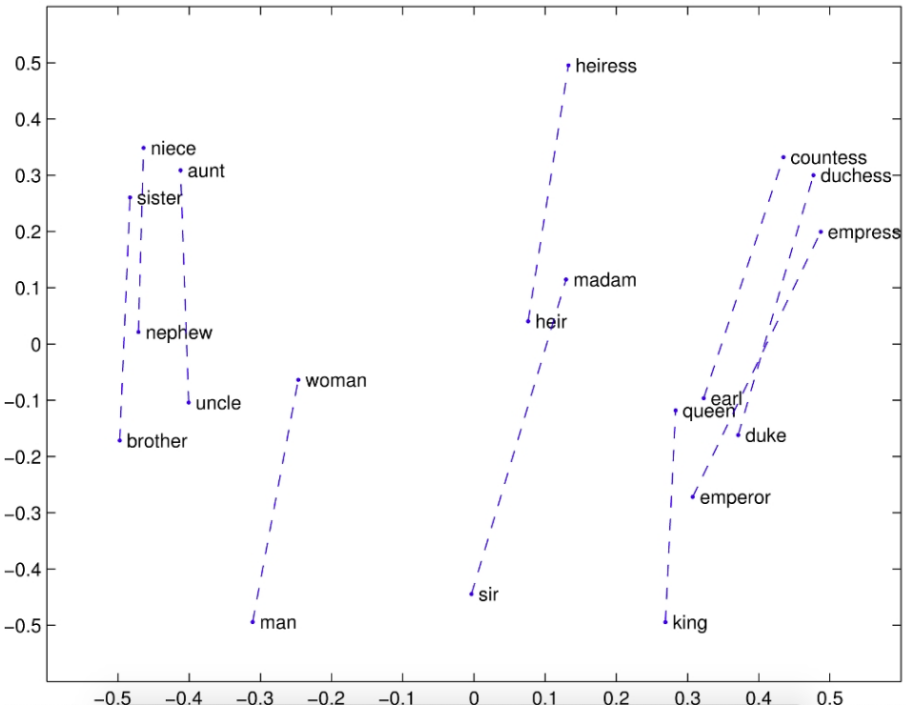

Figure 2.2: Image showing how words have been correlated to each other during GloVe feature vector training. Image source: GloVe official webpage (Pennington et al., 2014b).

As previously stated, GloVe effectively takes the best parts of the two families: large scale corpus patterns and locally retrieved sentence or word patterns. The core theory behind the GloVe model is to, instead of learning the raw co-occurrence probabilities of all of the words in the corpus, learning the ratios of all the co-occurrence probabilities in relation to each other. This will create a co-occurrence ratio vector for each of the analyzed words in the corpus, and effectively reduce the max space required to store all of the non-required zeros which are present in the sparse matrix of e.g the TFIDF-model.

Figure 2.2 illustrates an example of how certain words have been correlated with each other in two dimensions: X and Y. Consider the top three words for example: *niece*, *aunt*, and *sister*. These words tend to appear frequently together in both dimensions, as the meaning of the words are similar and used in similar semantical contexts. Looking at the three words approximately 0.4 below in the Y dimension, we see the words *nephew*, *uncle*, and *brother*. These three words, similarly like the ones before, appear frequently together. Now we can see that they appear very close in the X dimension to the group of three words above, but further away in the Y dimension, which reveals that they may be the opposite word in a semantical context.

Doc2Vec

Doc2Vec, also called Paragraph Vector, utilizes a word embeddings based topic model for discovering word contexts in a global perspective of a text corpus. Doc2Vec came to life after Mikolov et al. analyzed the weaknesses of the one-hot encoding based topic models, where words having equal distance from each other despite the fact that the semantics were entirely unassociated. Concretely, much like GloVe, each document was instead represented by a dense vector. With Doc2Vec however, the model is trained with a neural network to predict words in the document.

Mikolov et al. developed two different models based on paragraph vectors: A *distributed memory model (PV-DM)* and a *distributed bag-of-words model without word ordering (DBOW)*. The framework used by Mikolov et al. for learning the word vectors in both of the models are shown in Figure 2.3.

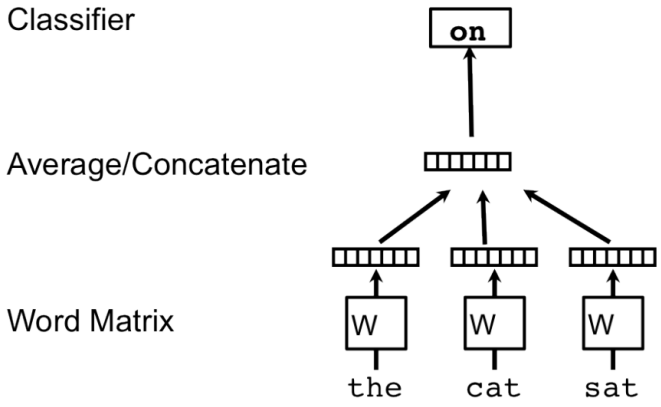

Figure 2.3: The framework used for learning word vectors. Image source: Mikolov et al. (2013).

The three words presented in the figure: *the*, *cat*, and *sat*, together predict the word *on* by being mapped as columns of the matrix W. Considering the context of the sentence, the words following *on* might have been *the* and *mat*, which led to the model predicting *on* as the classifier. The full sentence would then be *the cat sat on the mat*. Each word is mapped as a feature vector, which subsequently gets concatenated to an average feature vector of the words. A sequence of r words will opt to maximize the average log probability as

$$\frac{1}{T}\sum_{t=k}^{T-k}\log p(w_t \mid w_{t-k}, ..., w_{t+k}) \tag{2.4}$$

In order to predict the word, the classifier is using a softmax function denoted as

$$p(w_t \mid w_{t-k}, ..., w_{t+k}) = \frac{e^{y_{wt}}}{\sum_i e^{y_t}} \tag{2.5}$$

where each y_i is an un-normalized log probability log-probability for each output word i computed as

$$y = b + Uh(w_{t-k}, ..., w_{t+k}; W) \tag{2.6}$$

where U and b are the softmax parameters and h is the concatenated string from the respective word vector extracted from W.

As previously stated, both of Mikolov et al.'s models utilizes the framework presented as Figure 2.3. In this Master's project, both the *PV-DM* model and the *PV-DBOW* model are used, and further explanation is therefore required. The key difference between the models is that they form the classifier using different methodologies:

- PV-DM predicts 1 word from N inputs, whereas
- PV-DBOW works contrarily by predicting N words from 1 input.

The extension made which creates the PV-DM model is shown in Figure 2.4. The difference compared to learning word vectors as shown in Figure 2.3 can be denoted as the paragraph matrix D, with a specific id per paragraph. Each paragraph vector in the matrix D serves as a memory of what is missing from the current topic of the paragraph being analyzed, hence the name distributed memory.

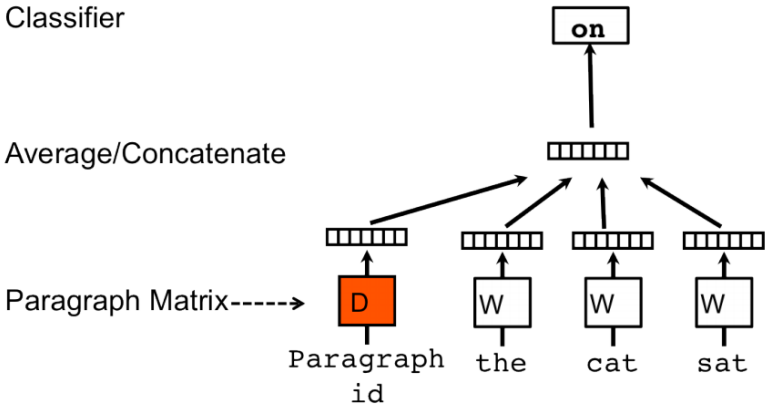

Figure 2.4: PV-DM extension to the word vector representation model. The paragraph matrix D denotes the memory obtained for each training step. Image source: Mikolov et al. (2013).

PV-DM consists of two key stages: training, and what Mikolov et al. calls the *inference stage*. The vectors in the paragraph matrix D, are trained using stochastic gradient. For each step of the gradient descent in the network, an error gradient can be calculated. This error is used to tune the parameters in the model for the next iteration in the network. The parameters tuned during training are the paragraph matrix D, the word vectors W, and the softmax weights U and b for the hierarchical classification. The purpose of the inference stage is to predict new paragraph vectors by adding more columns in D while keeping the trained parameters W, U, B fixed. This is done once the model has finished training, e.g to predict similarities among word vectors.

While instead considering the PV-DBOW model, the context words of the input are now ignored, and instead forced to predict randomly sampled words from the paragraph in the output. After each training step, a text window is sampled together with a random word from the text window. These random words create a classifier, shown in Figure 2.5.

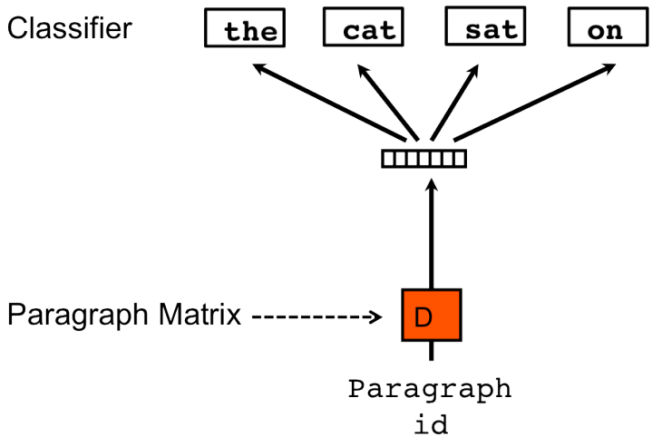

Figure 2.5: PV-DBOW extension to the word vector representation model. The classifier is formed from the small text window after each training step. Image source: Mikolov et al. (2013).

As opposed to PV-DM, the PV-DBOW model doesn't need to store as much data. Only the softmax weights are required, and not the word vectors in matrix D.

2.3 Data storage

We will develop a system which is heavily reliant on storing and analyzing massive data. The data storage medium used will serve a big role in order to assure consistency for system maintenance.

2.3.1 Non-relational database

Non-relational databases provide, as the name implies, a technique for collecting and storing data in non-tabular relationships. This can be beneficial in some types of data management. A big advantage is the simplicity of design a non-relational database provides. One item can be stored with many attributes, which is absolutely essential when storing patents with different important attributes as keys. In case of a relational database, the document instead would have had to be stored in multiple tables, with a join operator to get all the necessary data. It would be extremely inefficient to join tables of the sizes required in the system. The simplicity of design in non-relational databases will, therefore, imply that operations are faster and less costly.

Non-relational databases also provide better horizontal scaling to clusters of machines. The database will dynamically and concurrently spread data across servers when the user load is heavy. This is beneficial in case of a server shutdown to quickly restore data without the users of the system noticing.

Non-relational databases will also be able to capture both unstructured and semi unstructured data. Consider the scenario that the structure of a patent gets changed, and several new important fields get added to the content. A non-relational database will easily have the opportunity to freely add a key value to a JSON document without the necessity of defining changes. A document does not need to have all the key values present either. In certain situations, this could be beneficial. For example, design patents tend to have few words explaining the invention, but instead have multiple images. So for design documents, we would not want as many key values as in for example a chemistry patent.

2.3.2 Matrix model representation

Different types of matrices are generally used by the model for both storage and computations, meaning that a big part of the efficiency of the model is directly dependent on the representation of the matrix. There are two types of matrices that we will mention in this section: *sparse matrices* and *dense matrices*.

Sparse matrix. A *sparse* matrix is characterized as a matrix where most of the elements are zeros. This is associated with a complexity problem in both time and space. Matrix operations on two matrices containing many zeros will be slow because many zero elements will be added or multiplied with one another. Moreover, storing zero values

doesn't contribute to the richness of the data which means that it is a waste of memory resources. The one-hot encoding described in Table 2.1 is an example of a sparse matrix.

Dense matrix. A *dense* matrix is characterized as a matrix where most of the elements are *non-zero*, i.e. where the sparsity of the matrix is low, with a very low amount of zero elements. *Dense* matrices are much less memory-dependent compared to *sparse* matrices, as most of the data stored is valuable. Word embedding based models used for the GloVe or Doc2Vec algorithm uses a dense matrix representation.

2.4 Model evaluation

Evaluating a model is an important part to determine how well a certain implementation performs on a test set. The two major evaluation terms are called *precision* and *recall*. The formulas for calculating precision and recall are the following:

$$Precision = \frac{TP}{TP + FP} \tag{2.7}$$

$$Recall = \frac{TP}{TP + FN}, \tag{2.8}$$

where TP = True Positive, FP = False Positive and FN = False Negative. Precision is defined as the percentage of the results which are relevant, whereas recall refers to the percentage of total relevant results which have been correctly classified by the algorithm. The mean average precision (MAP) calculation measures how precise the model is on average for a set of queries (Cormack and Lynam, 2006). MAP is the calculated mean of the AP values. The formula for AP is defined as:

$$AP = \sum_{k=1}^{|S|} rel(S_k) \cdot \frac{P@k}{R}, \tag{2.9}$$

where $P@k$ is the precision at k returned documents:

$$P@k = \sum_{i=1}^{k} \frac{rel(S_i)}{k}, \tag{2.10}$$

and R and $rel(d)$ are defined as:

$$R = \sum_{d_i \in D} rel(d) \tag{2.11}$$

$$rel(d) = \begin{cases} 1, & \text{if document d is relevant to the test set} \\ 0, & \text{otherwise.} \end{cases} \tag{2.12}$$

2.5 Similarity measure

Cosine similarity is used to measure the similarity between two non-zero vectors and return a value between 0 and 1, where the closer the value is to 1, the more similar the two vectors are. The formula for calculating the cosine similarity is as follows:

$$similarity = cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{2.13}$$

where A and B are the two vectors to be compared.

Chapter 3

Development

The methodology chapter presents how we implemented the system. We implemented the system in Python, using several libraries for text processing and topic modeling, as well as distributed computing clusters for computational parallelization. We saved relevant system data using efficient binary serialization formats and subsequently stored in a non-relational database. A total of 3 models: TFIDF, GloVe, and Doc2Vec, were created and evaluated against each other by us, using measures well defined for patent information retrieval. For the final prototype system though, only Doc2Vec was developed and integrated with the resulting architecture.

The general idea of how the system is intended to work in practice is as follows: The system's corpus data will be created once with the desired amount of patents as a base. Subsequently, the user of the system will be able to input a query to the system which returns a set of k documents with the highest similarity score with the input query.

3.1 Preprocessed data format

The bulk dataset containing US patents was retrieved at USPTO (PatentsView, 2019), where patents can directly be downloaded in XML format. The patent files are released every week, and one file contains an average of 6,000 patents. These downloaded files would have been parsed and preprocessed, where the general structure would look like Figure 3.1.

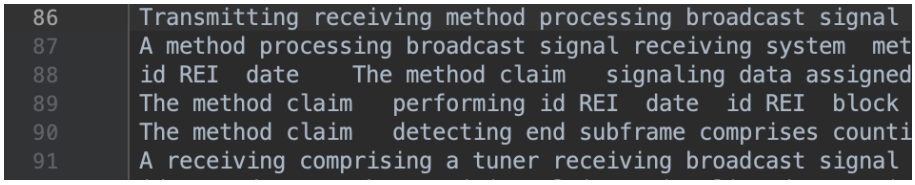

Figure 3.1: Figure showing the layout of the file containing the preprocessed data.

where each line corresponds to one paragraph, resulting in a file with the layout shown in Figure 3.1. Each line in the file was labeled with its corresponding file line number. Simultaneously while this file is being written, the document id and corresponding paragraph index of the paragraph are noted and written to another file, with the following general structure:

<doc-id>!<paragraph-index> (3.1)

which can be shown in Figure 3.2, which are the labels for the data in Format 3.1.

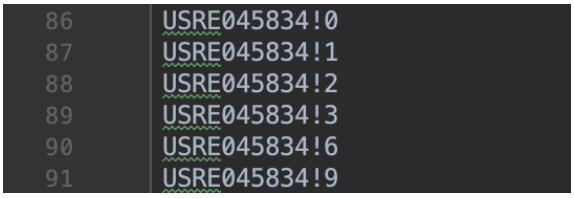

Figure 3.2: Figure showing the layout of the file containing the labels.

3.2 Gensim

The open source Python library Gensim is a collection of scalable statistical semantic tools, plaintext analyzers, and semantical similarity extractors, and it has been used for many parts during the implementation process of this Master's project (Řehůřek, 2019). Gensim both has bag-of-words based topic models and word embedding models in its library. They serve the purpose well of evaluating and comparing different models on the same test set.

Moreover, Gensim has some important features related to memory handling which is valuable when handling massive amounts of data: Gensim employs both memory independence and memory sharing. Memory independence means that the corpus may be loaded document by document instead of loading the entire corpus into the RAM at once. Memory sharing is related to loading and unloading trained models to the disk to effectively allow processes to share the same data. By passing different parameters to the Doc2Vec model, it can effectively change implementation between Mikolov et al.'s different Paragraph Vector models explained in Section 2.2.4.

3.3 Model implementations

The three models covered in the theory section were implemented using Gensim, with the purpose of being tested on how well they performed.

3.3.1 TFIDF

The first model that was implemented was a bag-of-words TFIDF-model. This served as a baseline model for the system, and further developments could effectively be compared to the baseline model in order to determine how well they performed. A hierarchical overview of the first model is shown in Figure 3.3.

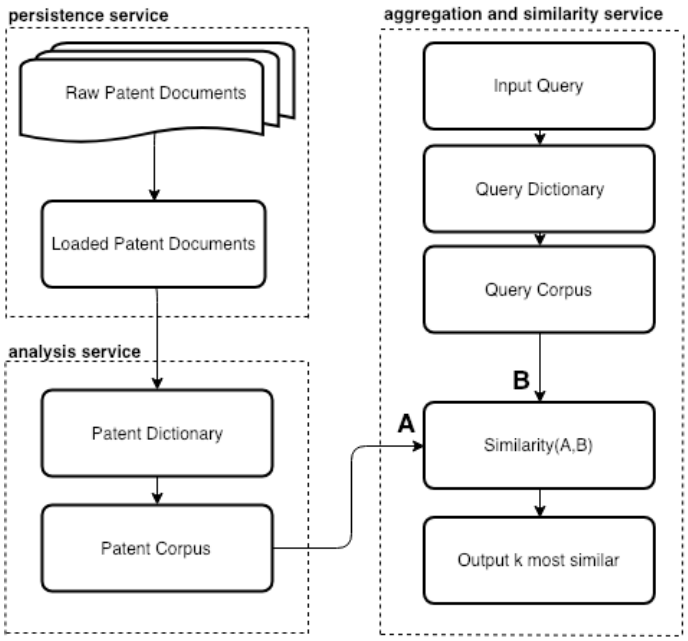

Figure 3.3: Hierarchical structure of the implemented baseline model.

In order to utilize the system more efficiently, the creation of the TFIDF corpus may be executed and saved to a MessagePack-file before using the system for its intended usage; namely to find relevant patents based on the input query. MessagePack is an efficient serialization formatter, which is deemed far more efficient when it comes to speed and memory efficiency compared to other serialization formats (Furuhashi, 2019), which was also confirmed during the implementation process by us. The essential features of the baseline model can be broken down to three different services as shown in Figure 3.3.

1. The first thing we did was to download the bulk patent data set described in Section 3.1. In Figure 3.3 this part is called the *persistence service*. We loaded the patents by linearly iterating over all of the documents, saving them document by document in a texts array containing each document per element in the array.

2. The second service is called the *analysis service*, and in this service, the texts array was subsequently fitted to a Gensim Dictionary object. The dictionary contained all of the words denoted as tokens. In order to keep the dictionary as small as possible, duplicates of the words were not kept. The words were, therefore, added uniquely from text arrays. The dictionary served as a base for creating a corpus, but before it could be fitted to a corpus, extreme values had to be filtered to effectively reduce the total space. This was done in accordance with removing noise and inconsistencies to the data as explained in Section 2.2.1. A Dictionary object can be filtered by providing filter values for a bottom and top threshold of the frequency of the words. The thresholds denote what words will be filtered out and thus removed from the dictionary before creating the TFIDF vectors. The feature weights extracted from the TFIDF calculations were based on Gensim's TFIDF-model. The TFIDF-model was fitted by passing the corpus as a parameter, which returned a list of the TFIDF vectors and it was finally stored as a MessagePack-file.

3. Once the corpus TFIDF had been stored, the system was ready to be used for querying in order to find relevant documents. This service is called the *aggregation and similarity service*. The input query was to be processed similarly as for the bulk patent dataset, which finally created a TFIDF vector of the input query. The vectors were then to be compared using the cosine similarity metric explained in Section 2.5. The k documents with the highest similarity score would be sorted out using a partial heap, which for small number k will be linear in complexity.

3.3.2 GloVe

The second model that was implemented was based on a pretrained GloVe model, explained in Section 2.2.4. The figure below shows the hierarchical structure design of the implemented model:

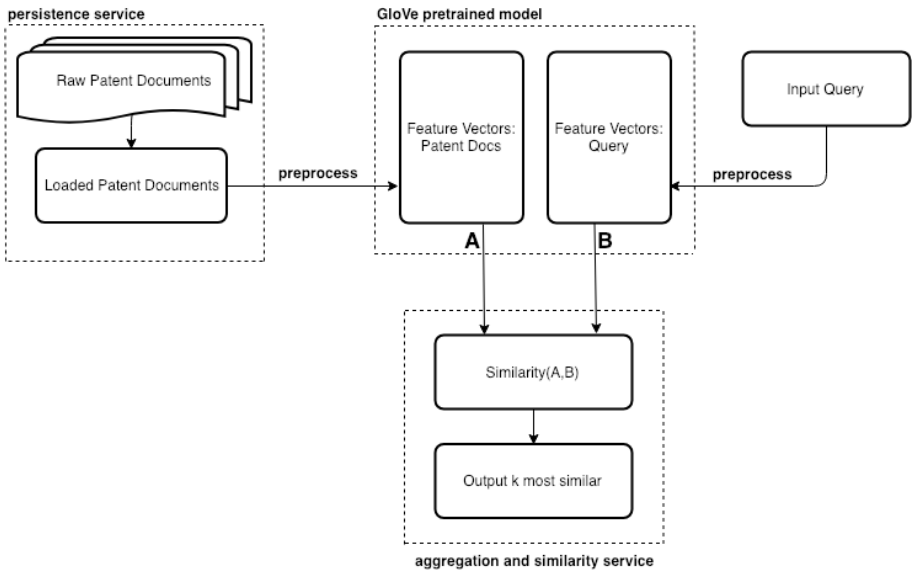

Figure 3.4: Hierarchical structure of the implemented GloVe architecture.

Following Figure 3.4, the first part of the model is a *persistence service*, which is more or less the same as the one from the baseline model, as shown in Figure 3.3. The only difference is that the texts are preprocessed here, by removing noise and inconsistencies to the data as explained in Section 2.2.1. After removing the stop words of the patents, the texts are turned into feature vectors using the pretrained model retrieved from GloVe.

The next step was to load the pretrained models into memory. These were retrieved from GloVe's website, and the models had been trained on Wikipedia text. The GloVe pretrained model is based on word embeddings as explained in Section 2.2.4. The Wikipedia text contains 6 billion tokens, 400 vocabularies, which subsequently created 300-dimension feature vectors. Whenever an input query is being sent into the system, the model which has been loaded would then create a feature vector for the query and for each document in the system, before going into an *aggregation and similarity service*. In this service, the cosine similarity between the vectors would be calculated, as explained in Section 2.5. Lastly, the returned similarity vector is sorted so that the k documents with the highest calculated similarities would be returned by the application.

Each document in the corpus was loaded, file by file, for each query, leading to efficient use of memory. There is never more than one document loaded into the memory at any time during the execution of the query.

3.3.3 Doc2Vec

The third model implemented was based on directly training a Doc2Vec model using the corpus containing patents. The figure below shows the hierarchical structure design of the implemented model.

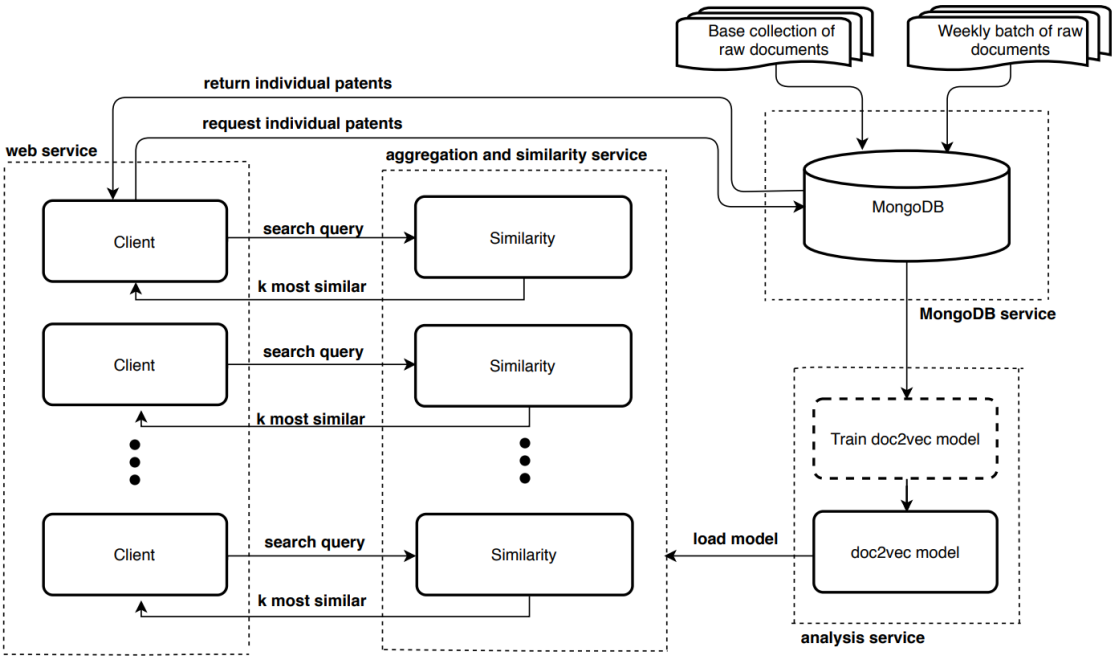

Figure 3.5: Hierarchical structure of the implemented Doc2Vec model.

Following Figure 3.5, there are four different services in total:

- *Web service* - Contains the web application and front-end part of the system. The web service is the one shown to clients or users of the system to operate the system. The users can provide a search query to an aggregation and similarity service which collects the k most similar patents based on the search query. Once the results have been retrieved, the users can then directly perform GET-requests to the MongoDB database, by clicking on a document, to obtain further information regarding the selected patent obtained from the aggregation and similarity service.

- *Aggregation and similarity service* - Accepts search queries from the client, and performs a similarity check with the client's input and return the output to the client.

- *Analysis service* - Used to train the Doc2Vec model, as explained in Section 2.2.4. The Doc2Vec model will be initially trained on a fixed amount of patents, which will serve as a base model. USPTO provides weekly batches of raw patents which can effectively be added to the database and model for further training. This would enable keeping the model updated on a weekly basis if needed. The *MongoDB* service takes care of the database, which is used to provide the corpus for the training process, while also providing separate individual requests from the web service.

- *MongoDB service* - Handles the database containing all of the patents.

The Doc2Vec model was implemented based on two different approaches:

1. In the first approach, Gensim's tagger was used to tag 3 lines for each document. A tag is defined as an entity which when used in a single sentence, it captures the meaning of all the words in this sentence. The lines were analyzed as abstract and title concatenated together, claims, and finally description. The purpose of this separation was simply because of the difference in semantics of the three different sections. The title and abstract are very concise and shortened to just a few hundred words maximum, whereas the description is a detailed description of the invention that may very well be more than 50 pages of text. The claims are totally different since they more or less constitute an enumerated list of features or specifications the invention has to follow or comprise. The fundamental semantic differences between the three sections warrant a separation, with the purpose of having separate querying towards the three different models.

2. The second approach implemented was to move away from the idea with three separate models and instead treat the document as one entity with multiple paragraphs, tagging the paragraphs individually, across every section. The idea was to handle the three semantically divergent sections (abstract, claims, description), but still be able to search across an entire document. When implementing this approach, the visualization and similarity results were depicted slightly differently, since one document can have several matches on different paragraphs across the document. Relevant results were now filtered in two different ways, namely on the number of matches per document, combined with the best matching paragraph per document.

3.4 Test set

In order to evaluate the performance of the different models in relation to their MAP score (see section 2.4), we defined a test set. We used the same test set on TFIDF and GloVe to ensure correct, realistic, and as unbiased a result as possible. Key values of the test set are shown in the table below:

	TFIDF	GloVe	Doc2Vec
Test set patents	100K	100K	~1M
Number of queries	50	50	8
Assessments per query	10	10	10

Table 3.1: Table presenting the test set used for TFIDF, GloVe and Doc2Vec to evaluate its mean average precision

As shown in Table 3.1, the test set for TFIDF and GloVe contained the same 100,000 patents from the year of 1980. Note that the test set only contained the patent abstracts. The test set could not contain more than 100,000 abstracts because of hardware limitations which arose during querying. We used 50 queries to evaluate the MAP score. For each query, 10 manual assessments were made to determine if the results obtained were relevant or not. It is important to note that these tests were evaluated by ourselves, with little to none expertise in the field of patents.

There was, for several reasons, a problem using the same test set with the Doc2Vec model as the two others. First and foremost, the Doc2Vec model requires a lot more data in order to work properly. By only having 100,000 abstracts in the test set corpus with the Doc2Vec, the similarity results obtained were subpar, thus giving the other two models an unfair advantage. The test set used for the Doc2Vec model instead consisted of all granted patents released by USPTO from the year 2016 through 2018, which corresponds to approximately 1M patents.

Doc2Vec was tested slightly differently compared to TFIDF and GloVe. We instead let two experts in the patent industry perform the manual assessment of the queries, using a total of 8 previously unknown queries with 10 assessments per query. We then let the same experts convert the queries to keyword based queries compatible with one of the industry standard patent search engine: Orbit Intelligence (Questel, 2019), for a system comparison.

3.5 System architecture

The following section will cover the architecture of the final Doc2Vec based system, which will build on the usage of various microservices, which would be used to create a highly scalable and efficient system. A microservice architecture is defined as an architectural style that structures an application as a collection of services. The architecture allows a large and complex system to be separated into an arbitrary number of subroutines to handle different areas of responsibility. The final production model is based on Doc2Vec, specified in Section 3.3.3.

The system now handles communication between microservices in an optimal manner so that each service has one specific responsibility. The division between microservices has also been made to ensure that I/O delays are kept to a minimum. Finally, the division of labor will also be very beneficial when creating isolated containers for a cloud solution, so that certain services do not need libraries that are not used.

3.5.1 Microservices

Figure 3.6 shows in detail how the microservices of our system are communicating with each other and which data is being sent between them.

In total, the number of microservices present in the system is four. A detailed explanation of how the user uses the system and how the microservices are communicating is given in this section.

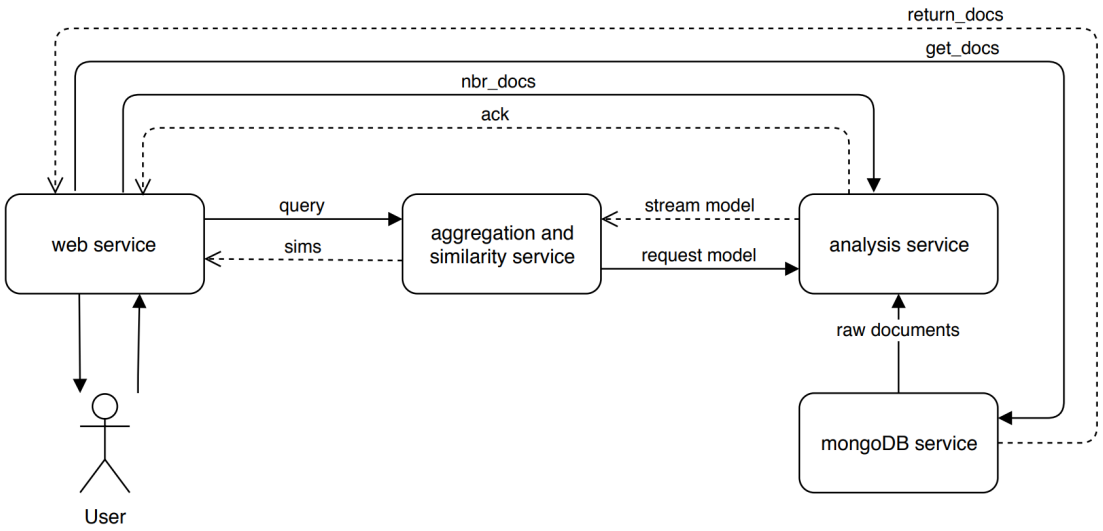

Figure 3.6: Microservice communication.

The user interacts with the web service through a Python Flask app, which is a lightweight web framework that doesn't require particular tools or libraries. In order to use the system to find similar patents, the model first has to be trained. The user specifies how many documents they want to have in the model when querying, and send this number from a web service to an analysis service. The analysis service responds with an acknowledgment, to notify the web service that the specified number of documents have been provided, and subsequently request this number of documents from a MongoDB service. The analysis service will now begin its training to create the Doc2Vec model. Once the training is finished, the model will be saved in the analysis service, and wait for the user to request a similarity score from a query from the web service.

When a query is initialized from the web service to an aggregation and similarity service, the aggregation and similarity service will send a request to the analysis service to let it know that it wants the Doc2Vec model. The analysis service will begin streaming the model, and once it has finished, the aggregation and similarity service will perform its operation to return similarity scores to the web service based on the query received.

Lastly, the user may request additional information from the simulation scores by clicking on a specific document name. The web service will then request the patent directly from the MongoDB service, and the MongoDB service will send back the entire document with all interesting parts of the patent (title, abstract, claims and description).

3.5.2 gRPC

Communication between services was accomplished using gRPC, an open-source universal remote procedure call framework implemented by Google, which would be used for both simple requests, but also streaming of data. A remote procedure call, or RPC, is a signal sent from one subroutine of a system or a network to another, which either a request or a response. The RPC is defined globally, and the different subroutines do not need to understand the details of each other's networks.

The first step is to define the gRPC services and the method request and response type, using *protocol buffers*. This is done in a *.proto* file. The protocol buffers provide a way to serialize structured data. gRPC has four different options to define services, where two of them were defined in this project:

- A *simple RPC* which works similar to a normal function call. The client sends a request to a server using a stub and awaits a response. Simple RPCs were used for requesting a document from the MongoDB by the web service, querying for similarity score from the aggregation and similarity service by the web service, and initializing the training in the analysis from the web. The only communication where Simple RPCs does not work is when streaming a model. In this case, a *response streaming RPC* was used instead.
- With a *response streaming RPC*, the client sends a request to the server and then waits for the server to return a stream of messages.

When the services had been defined, the gRPC server and client interfaces were generated from the *.proto* service definition. These interfaces were used by the microservices presented in Figure 3.6. Traditionally, a front end service would act as a client and a back end service as a server. However, out of the services presented in Figure 3.6, only the web service is actually a front end module. Hence, we created a custom definition of which services would act as servers and which services would act as clients:

- Web service: **client** for both the MongoDB service, the analysis service, and the aggregation and similarity service.
- Aggregation and similarity service: **server** for the web service, and **client** for the analysis service.
- Analysis service: **server** for both the web, the aggregation and similarity service, and the MongoDB service.

As the MongoDB service serves as a separate entity, it does not use the same gRPC microservice communication as the other services do. Instead, the MongoDB will receive requests from its two clients web service and analysis service.

3.5.3 MongoDB

We used MongoDB (2019), a cross-platform document-oriented database, to store the data from the USPTO bulk data. MongoDB is a non-relational database, which in our case has several advantages over a relational database such as design simplicity, availability control, easy capture of new data, and efficiency. The MongoDB was integrated with the Flask web application, where simple JSON-requests would be used to retrieve specific documents.

3.5.4 Docker

Docker (Docker, 2019) was used to deploy the application in a container, with all of the required dependencies. The isolated environment that was created with the help of Docker, makes it much easier to enhance the application into a more scalable approach, with the same predefined environment, as stated by a Dockerfile. A Docker image is a file which contains multiple layers, used to execute a program in a Docker container. Once the services had been created, their respective Docker image was then pushed to a Docker Hub repository.

3.5.5 CircleCI

CircleCI (2019) was used as a continuous integration tool during development, which would build all services for every commit to the repository and automatically push the images to Docker Hub, after each successful build. CircleCI provides fast builds, build isolation where the builds are run in clean containers, extensive support, and coverage for other tools such as Docker, while also being GitHub friendly.

A CircleCI config file was used to define the CircleCI builds, which included setup of a remote Docker engine, a docker build process being started for each of the three services, and then the resulting build image for each service being published to Docker Hub.

3.5.6 Amazon Web Services

Amazon Web Services (Amazon, 2019), or AWS, was used as an on-demand cloud computing platform, which provided a virtual cluster of computers, where the services would be deployed for execution. The system architecture used the services S3, EC2, EFS, and ELB.

- S3 (Simple Storage Service) was used as a long-term object storage service, where the data containing the patents in JSON-format would be stored and loaded into MongoDB during setup.
- EC2 (Elastic Compute Cloud), was used to provide computing capacity in the cloud in the form of hardware.
- EFS (Elastic File System) was used for storage of temporary data files that would be required both during and after the training process, where e.g. files such as the preprocessed corpus, labels, and the finished models would be stored and easily accessible by all parts of the system.

- Lastly, ELB (Elastic Load Balancing) was used by the web service, which would handle incoming connections to the web application, while also providing an external IP-address which could be used to connect to the application.

3.5.7 Kubernetes

Support for distributed computing was implemented for the application, as a way to tackle the efficiency problem related to processing such big amounts of data. Docker was used as the first and foremost base building block, where containers would be deployed to remote computers using services such as AWS, where the computations would be done over the cloud. Means for improving the parallelization during the execution were also taken into consideration during the implementation, as a way to improve the efficiency even further.

Kubernetes is an *open-source system for automating deployment, scaling, and management of containerized applications* (Kubernetes, 2019). It acts as an orchestration system which organizes and schedules an application over an arbitrary number of machines and can be seen as a high-level operating system for the containers deployed by Docker. Kubernetes has many advantages which makes it a state of the art open source system for managing Docker containers. Firstly, it is light, simple and easily accessible. Secondly, Kubernetes can be deployed in many different states (public, private, cloud) which makes it portable and thus suitable for our application. Finally, it is also possible to add extensions to the Kubernetes system in case future iterations require such.

3.5.8 Cloud architecture

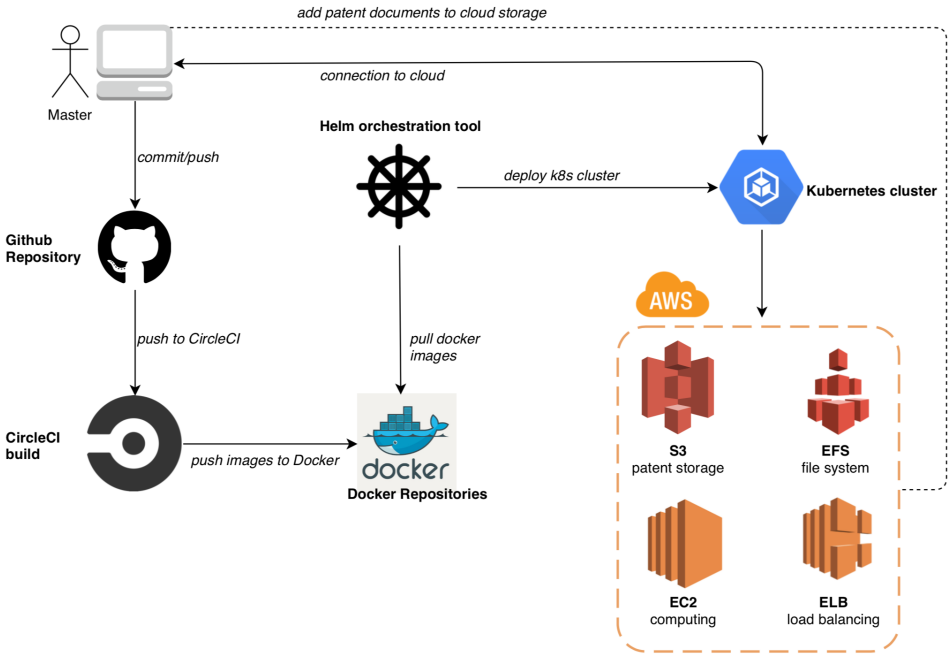

Figure 3.7: The complete cloud architecture of the finished system.

The cloud architecture was created by setting up the different components explained in Sections 3.5.2, 3.5.3, 3.5.4, 3.5.5, 3.5.6, and 3.5.7. Figure 3.7 illustrates how the components were coupled.

A Github repository was kept for code and version control. Once a push to the repository was initiated, the changes were automatically pushed further to CircleCI for a continuous build of the system. The final part of the CircleCI build was to push images of the services to the Docker repositories to prepare for deployment. *Helm* was used as an orchestration tool used to set up the Kubernetes clusters, and the system was subsequently deployed to the AWS cloud with an S3 bucket for cloud storage. Once the system was deployed to the cloud, a master user could connect to it from a local machine.

One important design decision that was made upon deploying the system to the cloud, was to provide it with EFS-provisioner to ensure persistence. The EFS system consists of a container that has access to an AWS EFS resource. This decision greatly simplified the communication between services covered in Section 3.5.1, in the regards of models not being required to be streamed from the analysis service to the aggregation and similarity service. The persistent volume provided by the EFS-provisioner will instead contain the saved models and model files created when training the model during analysis. Since the aggregation and similarity service will have direct access to that volume, the files are no longer required to be streamed, as opposed to the schema shown in Figure 3.6. The aggregation and similarity service can simply load the model once it exists in the persistent volume.

Chapter 4

Results

We created the system with the purpose of handling as much data as possible, without hindering the ability for the users to find appropriate results. During the implementation, we obtained important results which led to certain design choices regarding further development.

This chapter will present all of the results obtained from testing the different models. Before launching the system on the cloud with Kubernetes, some local experiments were made on a test machine. The purpose of these tests was to evaluate base models: If and how well they could be used to scale up the system. We made manual assessments to evaluate the models on a test set for the local system before deployment. The reasoning behind this is that it shows that some of the models cannot be operated to its full potential due to restrictions that emerge when handling massive data. Finally, the results of the finished and deployed system are presented, together with the most important parameters for additional tuning of the system.

While evaluating the models, a couple of different aspects were taken into consideration:

- The most obvious one is how well the model works with regards to accuracy. Specifically, what MAP score a query obtains after manually evaluating how precise the model returns its results.

- Another crucial aspect is related to how much data the model can handle, both in terms of memory usage and computational speed. As the goal is to train a model on as much data as possible so that ideally every relevant patent document can be found, the system is bound to be computationally dependent. Hardware requirements and algorithm optimization are therefore consistently taken into account when analyzing the results.

4.1 TFIDF

The TFIDF model described in Section 3.3.1 was tested on the test set defined in Section 3.4. A manual assessment of how many of the returned results were deemed relevant was done, yielding an approximate total MAP score of 57%. The implementation based on the TFIDF model resulted in memory errors when scaling up by more than 100,000 patent abstracts.

4.2 GloVe

The GloVe model described in Section 3.3.2 was tested with the same test set as the TFIDF model, and only on the abstract part of the patents. The manual assessment for the same queries and the same test machine that was used while evaluating the TFIDF model returned an approximate total MAP score of 52%. Unlike the TFIDF model, the memory usage was handled well by the GloVe model. However, our implementation of the GloVe model resulted in slow querying when scaling up by more than 100,000 patent abstracts.

4.3 Doc2Vec

The user interface of the final prototype we developed is shown in Figures 4.1 and 4.2. Figure 4.1 shows the design of the system. There are in total three pages on the website: Dashboard, Query, and Admin. The Dashboard page presents itself when a user navigates to the website and gives a short introduction to the contents of the system. The Admin page allows for logging in and retraining of the model, but this will not be available for the ordinary user since it is solely used for development and testing purposes. Upon navigating to the Query page, the user is presented with the interface in Figure 4.1. The user is here prompted to enter a search query, which infers a word vector to the Doc2Vec model.

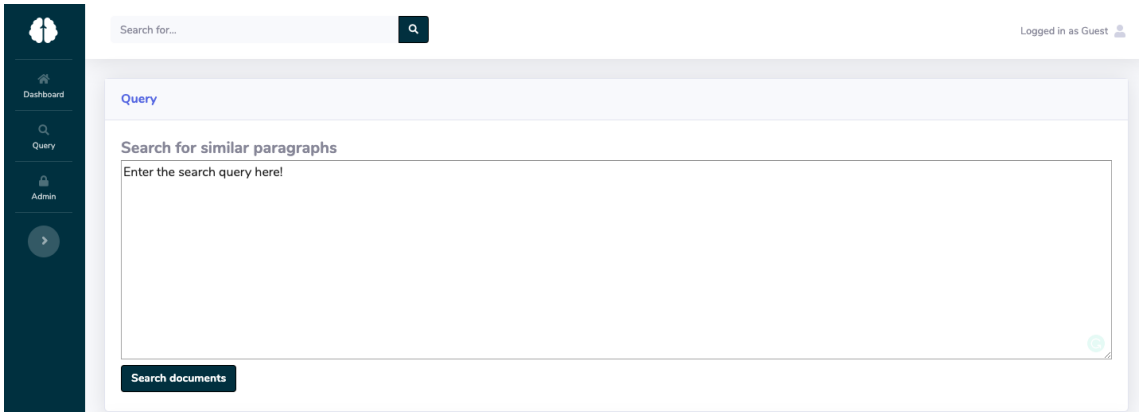

Figure 4.1: Image showing the query page of the final prototype.

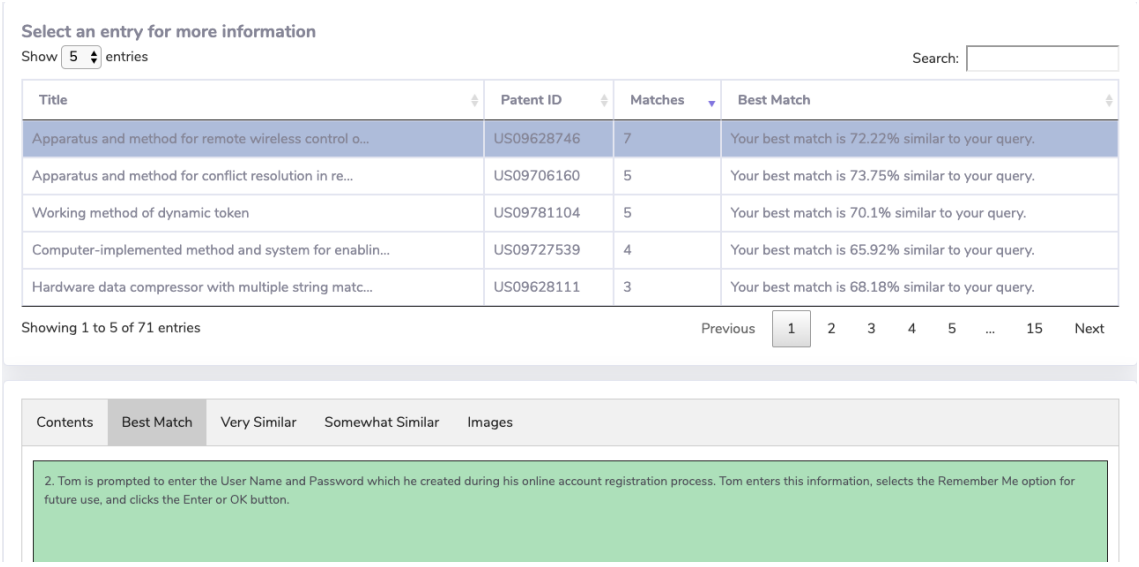

Figure 4.2: Image showing the results obtained after executing a query to the system.

Figure 4.2 shows what happens once the user has entered a query. A list will present itself, showing the 100 best matching paragraphs. The user can then order and filter the results based on the title, the patent ID, the number of matches per document, or the best matching query. Once the user clicks on one list entry, the document will be collected from the database, showing its full content. The user may also look up all the individually matching paragraphs in each document.

The final prototype shown in Figures 4.1 and 4.2 contained approximately 1 million patents. A query to the system was executed in approximately 3 seconds. When testing our system on the test set in Section 3.4, it returned a MAP score of 40%.

Chapter 5
Discussion

This chapter discusses the implications of the results obtained during the development of the system. Given the implemented system, we give answers to the following questions:

- How well does our system actually facilitate patent engineers in performing their prior art study when receiving a new project?
- How well does the system stand up to the current industry standards with regards to efficiency and accuracy?
- What are the potential drawbacks of the current system, and what future research can be done in order to discover potential improvements to the system?
- Why is massive patent data a problem and how is our system developed to handle this?

5.1 Model comparison

5.1.1 TFIDF

The first model, the TFIDF model, worked sufficiently well when it came to smaller amounts of data. The TFIDF model takes into account specific words only, with no context of words included, which still worked relatively well. One big drawback with the TFIDF model is its sparse matrix representation, which is fast, but very memory expensive to scale up. Also, one of the key requirements and goals of this new system would be that it would be able to handle the context of words and their semantic representations, which wasn't taken into account here at all, only the frequency of terms in relation to the occurrence in all of the documents.

It is important to note that the querying was only done on abstracts, because of the simple reason that testing on claims and descriptions too would take a very long time, perhaps even be impossible due to memory errors, with the test machine. By querying only the abstracts of the patent, a substantial amount of essential data is missing from the corpus, and even if the precision appears to be high in some cases, it is very likely that it gives a false sense of accuracy when taking the entire patent into consideration. The model was tested on claims and description too, however, after processing for over a minute the memory requirement had exceeded well over the total amount of RAM on the test machine, which resulted in memory errors and no relevant output.

We could have provided our TFIDF implementation with an indexer, such as Lucene, to improve the performance (Lucene, 2019). Instead of searching for the entire text, each word is instead mapped with an index, and Lucene will perform a fast lookup on the specific index which subsequently returns the text. However, since TFIDF still is lacking the context based searches, we decided that the effort of implementing an indexer would not be worth it, and we instead decided to implement a second model.

5.1.2 GloVe

We tested the second model, the pretrained GloVe model on a Wikipedia corpus, as an alternative to the TFIDF model. This model tackled two problems which TFIDF did not: It countered memory problems caused by the sparse matrix representation from TFIDF, and it also used semantic representations of the words with the use of word vectors.

The GloVe model suffered from a different issue compared to TFIDF when scaling up. The pretrained model loaded without issues no matter the size of the corpus. However, it suffered from not being able to handle querying fast enough. Even though the model was loaded, and the query didn't require much memory (less than 1 GB), it took a very long time. When querying with abstracts, claims, and description in the model with 100.000 documents, one query took 9 minutes to process. Since the goal of the system is to handle ideally more than 100 million patents, it would be practically impossible for the GloVe model to process that much data. Even with perfect parallelization of the workload with computing clusters, it would cost a substantial amount of money to maintain all these machines.

The explanation behind this poor performance is due to the fact that with each query, feature vectors are constructed using the pretrained model for both the query and all of the patents in the system, where the similarity is then calculated with the constructed feature vectors, and then sorted so that the n most similar documents are retrieved. A possible optimization would be to compute the feature vectors for each patent document in the database initially and store them somewhere for later use, which would probably make the queries much quicker. A better system for our use case would be to perhaps train our own GloVe model on patent data, instead of using a pretrained model which has been trained on other types of text data that doesn't cover the representation of patents particularly well.

5.1.3 Doc2Vec

The third and last model, which also is the one we decided to incorporate into our final prototype, is our own trained model using Doc2Vec. The first approach we took was to treat

the whole text of a patent separately as one input to the Doc2Vec trainer. So in this case, each document would be trained only once. During testing of the model, we discovered various problems with treating the entire patent as one input to the system, as presented in Section 3.3.3. As the general structure of a patent consists of an abstract, a list of claims, and a description, the variance in semantic representation between these three sections tend to be high. This makes it harder for the model to learn a general representation of the document as a whole. This is why we decided to analyze and tag the patent paragraph by paragraph instead, and this is what the final prototype, shown in Figures 4.1 and 4.2, is built on.

5.2 Comparison with an industry standard search tool

The system we developed had the goal of facilitating the prior art study done during the patent drafting process. Since the purpose of our system is to find semantically similar patent paragraphs, it is very interesting to see how well it performs in relation to the current state of the art patent search engines. As explained in Section 3.4, we benchmarked our system with that of Orbit Intelligence. The comparison between our system and Orbit Intelligence was done on the exact same dataset, containing patents released by the USPTO between the year 2016 and 2018. As shown in Table 3.1, this corresponds to approximately 1 million patents.

The Doc2Vec testing was conducted by patent experts performing new queries, so the MAP score on the systems compared to the MAP score retrieved when testing the TFIDF and GloVe based systems is no longer relevant. The interesting comparison in this case instead is between the MAP scores retrieved when querying our system versus the keyword-based system Orbit. Additionally, how the contents of the patents retrieved differ from each other may give some suggestions about cases where one system performs better than the other, without regarding the MAP score. We could let the patent experts evaluate the TFIDF and GloVe models too, with the initial test set. However, the result would most likely still be difficult to evaluate, since the test set only contained 100,000 patent abstracts. Since an abstract doesn't contain sufficient information for an expert to thoroughly evaluate its relevance, we instead decided to focus on the 1 million patent test set for Doc2Vec.

Considering the MAP score, Orbit gave us 46%, which compared to our system's 40% only is a slight difference. It is however remarkable that none of the search results that were returned by Orbit overlapped with our system's results. This means that Orbit was able to find several relevant patents which our system did not manage to find. And conversely, our system managed to find several documents which Orbit did not manage to find. This observation substantiates the claim that the prototype we developed can serve a very important role in the prior art study. In order to deduce for what type of queries and patents our system works best on, further research has to be done. It is, however, clear that our system can be used as a complement to the keyword-based search engines, similar to Orbit, used today.

An additional comparison can be done in regards to the efficiency of the two systems. An advantage with our system is that the keyword searches do not have to be defined as

they do for Orbit searches. The keyword definition is of the essence when searching for a patent, and the current keyword creation scheme of Orbit is based on different Boolean algebra combinations together with advanced filtering. Our system instead only requires a description, a claim, or a technical specification to be able to find appropriate results. To exemplify this, the keyword-based search for the invention *Video doorbell* is shown in Format 5.1.

((Video door?bell) AND (access control) AND (fac+ w recogn+) AND (facebook or social media))

(5.1)

As seen in Format 5.1, Orbit takes a selection of cleverly defined keywords as input. These types of searches take some effort to produce whereas an input to our system may be fast if there exists a technical specification or similar. Moreover, the inputs to Orbit have a wide variety of combinations that you may try combined for bigger coverage, whereas our system only takes the input once. The input compatible with our system, corresponding to Format 5.1, is shown in Format 5.2.

A video enabled doorbell connected to a security system for access control to a domestic residence. The doorbell uses facial recognition to identify visitors and matches the face of the visitor with a database of known faces. The database of known faces are friends on Facebook.

(5.2)

The reason why we did not have any overlapping search results when comparing our system with Orbit can be a combination of different aspects, where the following two are the most feasible:

- Since the systems works fundamentally different when it comes to input queries, one explanation may be that both of the systems would have required multiple inputs describing the same invention. To increase the search space, Orbit could take a different boolean combination of keywords, whereas our system could be given e.g a claim or an alternate description. However, our system would most likely show similar result, since it is based on semantical context.
- The nature of a word embeddings based topic model compared to a keyword based model will cause the systems to give different results. Since the inputs were previously unseen, we did not know whether an invention like this existed or not in the database.

5.3 Massive data in relation to computational speed and memory

A source of complexity in the system is the introduction of additional data. If, for example, the prototype would have contained 100,000 abstracts from patents, as the initial test set did, the development of the system would require much less effort. On the other hand, it would not serve its purpose very well in practice, because by only examining a fraction of

the total patent space, as well as a fraction of each individual patent, a substantial amount of data would be disregarded. This data is very important to analyze when deciding whether the found patent is relevant or not. The introduction of massive data is evidently also the reason why a cloud architecture was necessary, together with the hypothetical scenario where the system would contain multiple simultaneous users.

The primary bottleneck of the final system is definitely the memory usage. There are two major reasons why the system requires a large amount of RAM to function.

1. Firstly, before training the model, the system needs to have quick access to the patents in the database. Hence, the MongoDB database will reserve a big amount of memory. One of the advantages with MongoDB database however, which is mentioned in Section 3.5.3, is that it can easily be scalable to multiple machines if necessary.

2. Secondly, the memory requirement is also large during training. Although the training has been implemented to use as little memory as possible, with several iterators and streaming, the files saved to the disk from Gensim's Doc2Vec still requires a substantial amount of memory. The aggregation and similarity service needs to have direct access to the model files to be able to infer a new feature vector from the obtained query to return a similarity score.

When considering computational speed, a fundamental requirement is a high-end processor during the creation of the model. By having a higher clock frequency, the time spent preprocessing could be reduced, and by having more processing cores, the worker thread count during training could also be increased, thus improving parallelization of the process.

Given the system hardware requirements, an interesting question to be posed is: How well does the system scale with more data? The MongoDB service containing the database can be parallelized over many nodes, whereas that possibility doesn't exist when it comes to the analysis and aggregation and similarity services. There are several files being formed during the creation of the model, all which takes up a bit of memory each. The min_count threshold mentioned below defines the minimum frequency of words required for a specific word to be included in the vocabulary. For the test set containing one million patents, these are the following values:

- **One million unique words in vocabulary with min_count threshold set to 10:** As there exist a finite number of unique words, this file will be expected to have a sublinear complexity notation as more and more patents are added to the vocabulary. This essentially means that the number of unique words will not contribute to the memory bottleneck of the system when adding more data.

- **70 million paragraph labels:** The number of paragraph labels, on the other hand, will consistently increase linearly with every patent, if the assumption is made that a patent will continue to consist of the same amount of paragraphs per document. However, with the current test set of approximately one million patents, this file only takes up just over 3 GB of memory.

As of June 2018, the USPTO had published over 10 million patents in total (USPTO, 2018). The interesting thing to note is that USPTO published its first five million patents between the years of 1790 and 1991, and the last five million in the following 27 years. This trend is likely to continue, and statistics from WIPO establishes that this trend is followed by other large patent organizations. Considering that the number of issued patents will be expected to grow in the future, there may not be a finite resource requirement set. The reason why we want to upscale the system is twofold:

1. The first and most apparent reason is that we ideally want to cover the entire space of patents, to be able to find every relevant document that exists. The system has so far only been developed and tested on the USPTO bulk data, but extensions to other organizations only require a way to obtain the bulk data, via e.g adjusting the XML parser. The other parts of the system work identical without considering what language is being analyzed.

2. The second reason is that more data may increase accuracy. By providing the system with more training examples, the system will continuously update itself with new semantic formulations and additional unique words, which finally enables it to better predict the content of the new, previously unseen, queries.

5.4 Further development and improvements

During development, several issues and reflections have risen regarding a future development approach. Although it is outside of the scope of the Master's project, it is clear that our current prototype can be improved further.

5.4.1 Parameter tuning

The general way of improving the precision and overall performance of the trained model consists of experimenting with various parameters, as there isn't really a straightforward way of figuring out the optimal parameter values for a set of data beforehand.

In Section 2.2.4, we presented two different approaches for learning the word vectors of Doc2Vec. Distributed bag of words (PV-DBOW) was only used out of the two options. Further experimenting is required to evaluate whether training word-vectors, as the distributed memory (PV-DM) approach suggests, would give any significant precision boost to justify the much longer time needed to train the model, as opposed to only training the doc-vectors.

The dimensionality of the feature vectors can potentially be experimented further with. The final prototype had a fixed vector size of 100, but a higher dimension would be able to catch a more detailed context and potentially improving the precision in that regard. The problem with increasing this specific parameter is that it would require much more memory and storage, which hurts the scalability of the system. This means that there are two other parameters that may be tuned before increasing the vector size, which does not affect the scalability. The first one being *epochs*, which modifies the number of times word travel in the neural network. This parameter only affects training time, which is less

of an issue. Finally, we could filter out more or fewer words by adjusting the minimum threshold for how many times a word may occur in the corpus.

5.4.2 Patent releases

Patent organizations handle update distributions for new patents differently. USPTO provide weekly releases of newly granted patents, and they should be able to be inserted into the database and trained, to keep the model up-to-date at all times. One issue though, regarding further training on an existing model, is that it is currently not possible to directly update/build the vocabulary without having to retrain the whole model.

A potential approach to this problem is to update the vocabulary less frequently (e.g. once per month). The reasoning behind this approach and why it could work well is due to the fact that if the model already has a sufficiently big and diverse vocabulary, one weekly update of 5000 to 7000 patents would usually not contain many (or perhaps any) new words that are not in the vocabulary already. It is also unlikely that one weekly update will contain new formulations or previously unseen sentence combinations. However, the model still requires retraining after an arbitrary number of months. This is because the model will require to keep pace with innovation and the filing of new patent applications.

5.4.3 Paragraph concatenation

The suggested Doc2Vec implementation we have taken in this thesis is to train the model paragraph by paragraph. This is because a poor performance was discovered when analyzing the patents by abstract, claims, and description. After labeling and training these large chunks of texts as three separate training examples, the model would yield results from a global perspective, rather than a detailed section of a text. This is a problem, because the differences may often exist in small details. The similarities provided were all almost exactly the same, and there was no clear way of making sense of the results. This is why our approach performed better, by summarizing how many individual paragraphs matches one document contained.

In some cases you may want to combine several paragraphs, to instead be able to get a more global perspective of the patent text. The approach would be that two or more consecutive paragraphs in a patent may be of greater importance combined, compared to what they would have been if they were tagged and trained separately. The idea behind this is rather abstract, and it could definitely be something to research further.

Consider Figure 5.1 as an example. In this specific example, two patents are being analyzed.

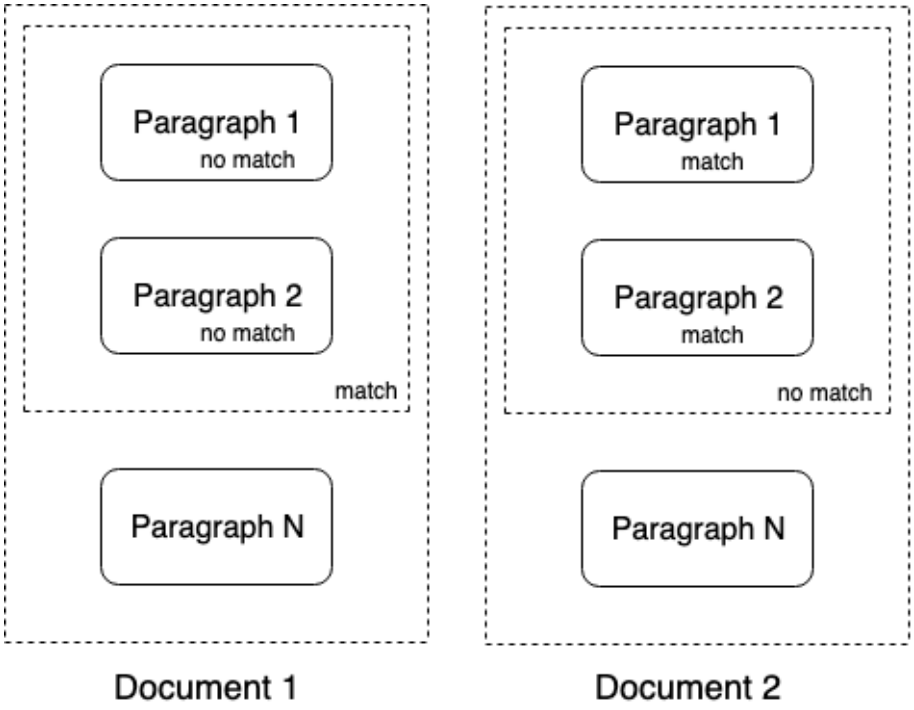

Figure 5.1: Paragraph concatenation example with two patents.

Our system will, with the current prototype implementation, return zero matches for Document 1 and two matches for Document 2. However, if we were to train the model with labels representing a concatenation of Paragraphs 1 and 2, Document 1 would return one match, whereas Document 2 now would return zero. If the paragraph match from Document 1 is potentially more relevant than the two separate matches from Document 2, this could now be found with the new trained pairs of paragraphs. With a combination of pairs, the time complexity would increase exponentially, which would result in poor scalability.

However, this is a subject of further research as certain modifications can be made to what paragraphs are to be concatenated with each other. For example, a certain concatenation can serve a greater semantic meaning in a patent describing a mechanical construction as opposed to a chemistry-related patent.

Chapter 6
Conclusion

Statistics show that there is a consistent increase in patent filings across the world. The core differences between innovations are often found in small details of the patent description, which asserts the need for an efficient, robust, and accurate search engine. The current state of the art search engines are insufficient; performing a prior art study today can take up to 20 hours for an expert in the patent industry, depending on what type of invention is being examined. This is because the keyword-based search engines of today lack intelligence. They require combinations of cleverly crafted search queries, and the results retrieved still may not be of relevance. An intelligent search engine that can discover the context of a technical specification or comparable document is in great demand (Tseng et al., 2007).

As shown in Section 1.3, suggestions for intelligent tools in the patent industry have been investigated. The system which we have developed during this Master's project offers a practical utilization accordingly. The development process presented in Chapter 3 showed some flaws with using TFIDF and GloVe as topic models for the specific problem of the thesis. Suggestions to intelligent tools in the patent industry have been investigated, as shown in Section 1.3.

- The one-hot encoding which TFIDF is built on required heavy memory to store its vector representations for the words, because of the sparse matrix representation. TFIDF also disregarded context and would, therefore, be inappropriate for the problem.

- GloVe was not suitable during the development because of performance issues when inferring queries as vectors, as the feature vectors for the model had to be rebuilt.

The cloud architecture built around Doc2Vec was shown to work best for the problem in regards to scalability, efficiency, and accuracy. By using a model based on word embeddings like Doc2Vec, to discover word and paragraph context, two problems can be resolved.

1. The first finding is that word embeddings discover context which the current state of the art patent search engines do not find. The system comparison presented in Section 5.2 showed that relevant search results could be retrieved by our system which the Orbit tool did not find. With the current prototype, the converse was also true. Therefore, the suggestion of using our word embeddings tool as a complement to the current state of the art patent search engines was made.

2. The second finding presented in Section 5.2 showed that our system can be used with less effort in producing the search queries. This can potentially be very beneficial in terms of hours saved when performing prior art studies.

Numerous potential improvement areas are suggested. The improvements are related to different approaches in training the model, handling new releases, and feature extensions. The model creation approach we have implemented is to train the model paragraph by paragraph, although further potential improvements can be done to this.

The purpose of our Master's project was to contribute to knowledge regarding information retrieval techniques in patent text mining and address limitations of the current state of the art patent search engines. The implemented system can be seen as a sufficiently good first prototype, which can be further researched and built upon in future work.

Bibliography

Aggarwal, C. C. and Zhai, C. (2012). *Mining text data*. Springer Science & Business Media.

Amazon (2019). Cloud computing services. `https://aws.amazon.com`. Accessed: 2019-03-06.

CircleCI (2019). Continuous integration and delivery. `https://circleci.com/`. Accessed: 2019-03-05.

Cormack, G. V. and Lynam, T. R. (2006). Statistical precision of information retrieval evaluation. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 533–540, New York, NY, USA. ACM.

Docker (2019). Enterprise container platform. `https://www.docker.com/`. Accessed: 2019-03-05.

Furuhashi, S. (2019). Messagepack. `https://msgpack.org/index.html`. Accessed: 2019-01-29.

Gutierrez-Osuna, R. (2005). Introduction to pattern analysis. *Texas: Texas A&M University*.

Han, J., Pei, J., and Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). Unsupervised learning. In *The elements of statistical learning*, pages 485–585. Springer.

Hehenberger, M. and Coupet, P. (1998). Text mining applied to patent analysis. In *Annual Meeting of American Intellectual Property Law Association (AIPLA)*, pages 15–17.

Kubernetes (2019). Production-grade container orchestration. `https://kubernetes.io/`. Accessed: 2019-03-05.

Lucene (2019). Ultra-fast search library and server. https://lucene.apache.org/. Accessed: 2019-05-20.

Lucheng, H., Yanhua, Y., and Zhihua, Z. (2010). A study on the application of data mining in the patent information analysis for company. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 1, pages 618–622.

MathWorks (2018). Introducing deep learning with MATLAB. https://www.mathworks.com/content/dam/mathworks/tag-team/Objects/d/80879v00_Deep_Learning_ebook.pdf. Accessed: 2019-05-13.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mishra, A. and Vishwakarma, S. (2015). Analysis of tf-idf model and its variant for document retrieval. In *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 772–776.

MongoDB (2019). The most popular database for modern apps. https://www.mongodb.com/. Accessed: 2019-03-05.

PatentsView (2019). Patent data exploration platform. http://www.patentsview.org/download/. Accessed: 2019-01-29.

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

Pennington, J., Socher, R., and Manning, C. (2014b). Glove: Global vectors for word representation. https://nlp.stanford.edu/projects/glove/. Accessed: 2019-06-12.

Qlik (2019). Data analytics for modern business intelligence. https://www.qlik.com/us/. Accessed: 2019-05-13.

Questel (2019). Orbit: Intellectual property software and services. https://www.questel.com/ip-business-intelligence-software/orbit-intelligence/. Accessed: 2019-05-16.

Schmookler, J. (1966). *Invention and Economic Growth*. Harvard University Press.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Ström & Gulliksson (2019). A guarantee for strong and solid patent rights in Europe. http://www.sg.se/. Accessed: 2019-05-13.

Tseng, Y.-H., Lin, C.-J., and Lin, Y.-I. (2007). Text mining techniques for patent analysis. *Information Processing & Management*, 43:1216–1247.

USPTO (2018). Patents through history. https://10millionpatents.uspto.gov/. Accessed: 2019-05-02.

USPTO (2019). United States Patent and Trademark Office. https://www.uspto.gov/. Accessed: 2019-02-06.

WIPO (2019). World Intellectual Property Organization. https://www3.wipo.int/ipstats/keysearch.htm?keyId=221. Accessed: 2019-01-10.

Řehůřek, R. (2019). Gensim: Topic modeling for humans. https://radimrehurek.com/gensim/. Accessed: 2019-01-29.

EXAMENSARBETE Massive Patent Data Mining
STUDENTER Emil Tostrup, Sani Mesic
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Semantiskt intelligent sökmotor för massiv patentdata

POPULÄRVETENSKAPLIG SAMMANFATTNING **Emil Tostrup, Sani Mesic**

En exponentiell ökning av patent i världen tillsammans med varierande semantiska teknikbeskrivningar försvårar möjligheterna att hitta sammanfallande teknik. Detta arbete presenterar ett system som, till skillnad från nuvarande industristandarder, använder smarta textbehandlingsalgoritmer för att hitta relevant sökresultat.

Det krävs en djup teknisk förståelse för att analysera om ny teknik går att patentera. För att kunna avgöra om en uppfinning är unik eller uppfyller ett unikt syfte behöver en genomgående studie utföras, där redan känd teknik undersöks. Skillnaderna ligger inte sällan i detaljerna. Två tekniska uppfinningar kan potentiellt sätt beskrivas på hur många sätt som helst, trots att de behandlar exakt samma område. Detta innebär att det inte räcker att studera nyckelorden som beskriver tekniken, utan det krävs en förståelse på detaljnivå. Ett stort problem idag bland nuvarande state of the art patentsökmotorer, är att de tar beskrivande nyckelord som inparametrar, och jämför dessa specifika orden med tidigare förekomster i patentdatabaser. Genom att enbart analysera de beskrivande orden försvinner mycket semantisk mening bakom dem. Semantisk mening i detta fallet talar exempelvis om hur ord förekommer i meningar, hur de används i förhållande till närliggande ord, eller hur storskaliga synonyma formuleringar kan vara av relevans. Detta är information som kan visa sig vara väldigt nödvändig då man genomför en nyhetsundersökning för att ta reda på relevant närliggande känd teknik.

Vi har utvecklat ett system som avser förenkla den inledande delen av processen som krävs för att få patent på sin uppfinning. För att producera sökresultat med semantisk förståelse används en modell som tränats på en enorm mängd rå patentdata från den stora amerikanska patentorganisationen, USPTO. Med hjälp av denna enorma mängd data, har modellen blivit så pass intelligent att den kan skatta den semantiska innebörden av nya, tidigare osedda, meningsformuleringar. Modellen baserar sina förutsägningar på i vilken kontext orden tidigare har använts i andra patentdokument. Det har visat sig att systemet fungerar som ett utmärkt komplement till dagens nyckelordbaserade sökmotorer för att hitta relevant sökresultat.

Hur använder man då systemet? Enkelt! Det är bara att navigera till webbadressen på din dator eller mobila enhet. Väl där, letar du dig in på sidan för sökningar och sen är det bara att direkt börja använda systemet, likt den valfria sökmotor du är van vid! Träning av modellen sker offline utan att användaren behöver tänka på det. Sökresultaten kommer uppenbara sig i en lista inom loppet av några sekunder. I listan kan du sen på ett begripligt sätt sortera sökresultaten baserat på antalet träffar i enskilt dokument, bäst hittade enskild sökparagraf i ett dokument, eller varför inte på patenttitel?