

MASTER'S THESIS 2020

Using Natural Language Processing to Identify Similar Patent Documents

Hannes Jansson, Jakob Navrozidis

Elektroteknik
Datateknik

ISSN 1650-2884

LU-CS-EX: 2020-05

DEPARTMENT OF COMPUTER SCIENCE

LTH | LUND UNIVERSITY



EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-05

**Using Natural Language Processing to
Identify Similar Patent Documents**

Hannes Jansson, Jakob Navrozidis

Using Natural Language Processing to Identify Similar Patent Documents

Hannes Jansson
e.hannes.jansson@gmail.com

Jakob Navrozidis
jakob@navrozidis.com

February 10, 2020

Master's thesis work carried out at AWA Sweden AB & Mindified AB.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Anders Fredriksson, anders.fredriksson@awa.com
Henrik Benckert, henrik@mindified.com

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

Abstract

The search for prior art documents is an important, but time-consuming task for the patent attorney. Today, these searches are carried out using keywords, which is problematic since inventions often are described using abstract and general terms in the patent applications. In addition, synonyms must be taken into account and formulated manually. This means a risk of relevant documents being overlooked.

In this Master's thesis, we investigated the use of *natural language processing* (NLP) on a huge database of patent applications. The aim was to create a tool that can find similar documents by comparing the title and abstract of a provided document with existing documents in the database, thus removing the need to manually extract keywords.

We investigated several machine learning models that transform text into numerical representations, and applied them to the documents in the database. These models include a number of recent, pre-trained, word embeddings and sentence embeddings. We also developed a web application, which allows the user to perform a search using patent application number or a short text describing an invention. Cosine similarity was used to compare the numerical representations of documents. We also investigated the use of clustering as a way to limit the search domain and speed up the process.

Patent associates helped us to evaluate the different models on a set of test queries. Among the models, Sentence-BERT (SBERT) outperformed the others, reaching a mean average precision (MAP) of 0.7655 at finding relevant or very relevant documents.

Keywords: natural language processing, patent search, document similarity, word embeddings, sentence embeddings, machine learning

Acknowledgements

There are a couple of persons we want to acknowledge for their contributions to this Master's thesis. First, we want to thank our supervisors at AWA and Mindified, Anders Fredriksson and Henrik Benckert, both for giving us the opportunity, and for their support throughout the project. A special thank you goes to Cathrin Johansson at AWA, for all the energy and encouragement she has given us.

Then, we want to thank Pierre Nugues, our supervisor at the Faculty of Engineering at Lund University. He has guided us through the process and provided us with feedback and lots of ideas.

Also, a big thank you to the trainees at AWA for taking the time to evaluate the models. Without them it would not have been possible to complete the project.

Lastly, we want to thank everyone else who in one way or another has been involved in this project. Among those are the the people at Mindified, Henning Petzka at the Department of Mathematics and our families and friends.

Glossary

Here we summarize the most commonly used abbreviations.

<i>bi-gram</i>	A sequence of two tokens, for example two characters or two words
<i>biLM</i>	Bidirectional language model
<i>DAN</i>	Deep averaging network
<i>EPO</i>	European Patent Office
<i>LM</i>	Language model
<i>LSTM</i>	Long short term memory, a neural network architecture
<i>MAP</i>	Mean average precision
<i>n-gram</i>	A sequence of <i>n</i> tokens
<i>NLP</i>	Natural language processing
<i>RNN</i>	Recurrent neural network
<i>STS</i>	Semantic textual similarity. A collection of datasets used to benchmark NLP models on sentence similarity
<i>SQL</i>	Structured query language. A language used to access or manipulate data in a relational database

Contents

1	Introduction	9
1.1	The Problem	9
1.2	Previous Works	10
1.3	Objectives	10
1.4	Structure of a Patent	10
1.5	Resources	12
1.6	Limitations	12
2	Theory	13
2.1	Numerical Representation of Documents	13
2.1.1	TF-IDF	14
2.1.2	Word embeddings	14
2.1.3	fastText	16
2.1.4	Flair	17
2.1.5	ELMo	18
2.1.6	Sentence embeddings	18
2.1.7	BERT-based models	19
2.1.8	Universal sentence encoder	21
2.2	Document Similarity	22
2.2.1	Cosine similarity	22
2.3	Clustering	23
2.3.1	K-means clustering	23
2.4	Evaluation Method	24
3	Approach	27
3.1	Pipeline	27
3.2	Preprocessing the Data	28
3.2.1	Preprocessing of the database	28
3.2.2	Preprocessing of the data	29

3.3	Implementation Details	29
3.4	Graphical User Interface	31
3.5	Evaluation Procedure	31
4	Evaluation	35
4.1	Results	35
4.2	Discussion	37
4.2.1	Embedding models	37
4.2.2	Clustering	39
4.2.3	Sources of errors	39
4.2.4	Improvements and future work	40
5	Conclusions	41
	References	43
	Appendix A Evaluation Instructions	47
	Appendix B List of Evaluation Documents	51

Chapter 1

Introduction

A hard and time-consuming part of the patent attorney's work is to do manual patent searches. We have investigated if this is an area where the use of artificial intelligence (AI), specifically natural language processing (NLP), can be of use. In this chapter we present the problem in more detail together with our objectives. It also includes what resources we had available and what limitations we had to make.

1.1 The Problem

Conducting patent searches is a significant part of a patent attorneys work. The search for *prior art*, i.e. evidence of an invention already being publicly known, is relevant both when deciding if something is patentable, or if a party is alleged for patent infringement. In the latter case, a common strategy is to lodge a revocation action against the patent in question to have it invalidated. This can be achieved by finding prior art to deprive the patent of its novelty.

Manually searching for these prior art documents is a time-consuming project, while the risk of relevant documents being overlooked is substantial. The reason is primarily that the search today is limited to the use of keywords, which is a problem since patents often use abstract and general terms to describe inventions. In addition, synonyms must be taken into account and formulated manually. Therefore, improving the search process, for example with the help of AI so that it can be carried out on entire pieces of text, would be desirable.

At our disposal we have access to a large database of world wide patent applications. This amounts to about 55 million documents, written in English. Given the large size of the

database, the problem to be solved is to find a machine learning algorithm that can identify similar documents in reasonable time.

1.2 Previous Works

Natural language processing has previously been used on patent datasets to some extent. Some systems using full text patent documents and others only abstracts, but as far as we know, not to the extent of using a large world wide database covering all technical areas.

In a report by Helmers et al. (2019), full text patent documents are compared in order to find inventions that are similar to each other. Good results were achieved by using so called word embeddings (further described in Section 2.1.2) on a limited dataset of documents in the patent class *Medical or veterinary science and hygiene*. As a future work, the authors suggest using a larger dataset with documents from multiple patent classes.

A similar work was made by Andrabi (2015) in his Master's thesis. The objective was to create an intelligent search engine for internal documents describing inventions at a large company. These documents contained a thorough description of the inventions, and are therefore similar to patents. Already available software was used to index the documents and compute similarities, but yet again the amount of data was limited.

1.3 Objectives

The aim of this thesis is to explore the possibility of using machine intelligence to speed up and improve the search for prior art documents, and can be seen as a “proof of concept”. The intention is to investigate and implement different NLP models in order to be able to find similar patents, and to compare how well the different models perform. We also hope to conclude whether abstracts and titles provide sufficient information for this purpose.

1.4 Structure of a Patent

To understand more about the problem to be solved, it is important to have some basic knowledge about the structure of a patent application. There are many parts of a patent application, of which the most obvious are *title*, *inventor* and *filing date*. There is usually an *abstract* consisting of a few lines, describing the invention in short. Sometimes it is just a summary of the first few claims.

The *claims* are the part of the application which describe the scope of the protection. A patent can have several claims, where each one describes a certain part or feature of the invention. They are structured so that the later claims go more and more into detail and refer to earlier claims. Therefore, the first few claims usually give a general description of the invention.

Furthermore, the patent application contains a detailed *description* and *background* which describes the problem to be solved and how the invention does it. The application usually has several *figures* of the invention. An example of the first page of a European patent application is shown in Figure 1.1.



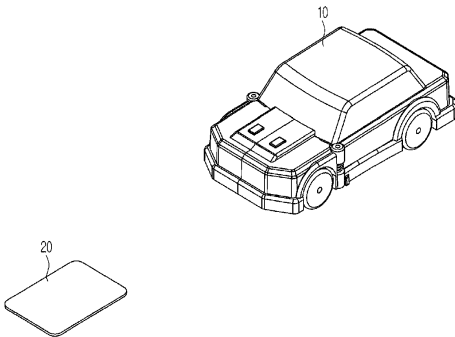
(19)  Europäisches Patentamt European Patent Office Office européen des brevets	
(12)	(11) EP 3 409 333 A1
EUROPEAN PATENT APPLICATION	
(43) Date of publication: 05.12.2018 Bulletin 2018/49	(51) Int Cl.: A63F 1/00 (2006.01) A63H 33/00 (2006.01) A63H 17/02 (2006.01)
(21) Application number: 18183676.8	
(22) Date of filing: 20.02.2013	
(84) Designated Contracting States: AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR	(72) Inventor: Choi, Shin-Kyu Seoul 158-050 (KR) (74) Representative: Thun, Clemens Mitscherlich PartmbB Patent- und Rechtsanwälte Sonnenstraße 33 80331 München (DE)
(30) Priority: 24.02.2012 KR 20120019210 26.03.2012 KR 20120030798	<u>Remarks:</u> This application was filed on 16-07-2018 as a divisional application to the application mentioned under INID code 62.
(62) Document number(s) of the earlier application(s) in accordance with Art. 76 EPC: 13752327.0 / 2 818 217	
(71) Applicant: Choi, Shin-Kyu Seoul 158-050 (KR)	
(54) TRANSFORMABLE TOY CAR	
(57) The present invention relates to a transformer toy car that automatically transforms the shape to turn a card over to provide the information on the card, if the card is attached to the transformer toy car. The transformer toy car includes: a separable toy car body; and cards 20 adapted to be attached to the underside of the toy car body 10, wherein if one of the cards 20 is attached	to the underside of the toy car body 10, a portion of the separable toy car body 10 is separated, and a portion of the separated toy car body 10 pressurizes a floor surface, thereby making the toy car body 10 stand up or turn over and thus allowing the underside surface of the card 20 attached to the underside of the toy car body 10 to be exposed to the outside.
【FIG. 1】	
EP 3 409 333 A1	
Printed by Jouve, 75001 PARIS (FR)	

Figure 1.1: The first page of a European patent application. Bibliographic information like filing date, application number, inventor, etc. are presented at the top of the page. This is followed by the title of the invention, and a short abstract describing it. A figure of the invention can be seen at the bottom of the application.

Table 1.1: Statistics of the EPO DOCDB database that we use as corpus. Only documents with an English abstract are considered.

Number of documents	55 893 936
Average abstract length	143 words
Number of unique words	378 571

1.5 Resources

As corpus, we use a database from the European Patent Office (EPO) called *EPO Worldwide Bibliographic database* or EPO DOCDB for short.¹ It contains bibliographic data (filing date, inventors etc.) as well as abstracts, titles and citations, but no figures or full text such as claims, description or background. Some statistics of the corpus are presented in Table 1.1.

The large amount of data called for large computational resources. For this reason Mindified AB provided us with a computer specially built for the purpose of machine learning. It had a 24 core AMD Ryzen Threadripper CPU, an EVGA GeForce RTX 2080 Ti 11GB GPU and 64GB of RAM. This was used to calculate the embeddings as well as hosting the web application.

In hope of speeding up the process even more we requested access to Aurora, a computing cluster at LUNARC, the center for scientific and technical computing at Lund University. This was used for some computations but due to high demands of the computing nodes only to a limited amount.

1.6 Limitations

Only the titles and abstracts of the patent applications are considered in this thesis. This is due to a limitation in the EPO DOCDB database, which does not contain claims, background or description. Also, only applications that have an abstract written in English are considered.

For the model evaluations, we used a limited dataset because of computational reasons. This dataset consists of all the European (EP) patent applications available in the database, amounting to about 1.8 million documents. Although this limits the amount of documents, the spread among technical areas should be similar.

Since the aim was to develop a proof of concept, we limited ourselves to only investigate pre-trained models. Training our own model would require large computational resources, a lot of time and a training set which needs to be constructed by hand.

¹<https://www.epo.org/searching-for-patents/data/bulk-data-sets/docdb.html>

Chapter 2

Theory

In this chapter, we present the underlying theory behind numerical representation of text and the different models that we investigated for this purpose. We also go into detail of how the similarity between vectors can be calculated, and introduce an evaluation metric that can be used to compare the results between the models. We also present some theory about clustering.

2.1 Numerical Representation of Documents

A way to compare documents is to first encode the text into numerical vector representations. These vectors can then be compared using mathematical methods. An important part of this thesis is to find the best way to make this vectorization.

The most simple way to vectorize text is *one hot encoding*. This is a binary representation, where each index corresponds to a word in the vocabulary. A vector representing a document has the value 1 at the indices of the words that are present and 0 for those that are not. There are however more advanced methods. In the following sections, we will explain a number of different ways to do the encoding, ranging from models that simply calculate the frequency of words, to more advanced models that can capture semantic relationships and are context aware.

2.1.1 TF-IDF

The *Term Frequency* (TF) representation generates a numerical vector for every document. Each word w in the vocabulary of the entire corpus is given an unique index in the vector, much like the one-hot encoding. The value at each index of the vector is then assigned the term frequency according to

$$TF(w) = \frac{\text{occurrences of } w \text{ in document}}{\text{number of words in the document}} \quad (2.1)$$

The problem with using TF for document comparison is that some words are very likely to occur in many of the documents. The word *invention* for example, occurs in most of the patent abstracts, and does not provide a lot of information about the actual content of the document. A way to solve this is by introducing the *inverse document frequency*, IDF. The inverse document frequency of a word w is defined as

$$IDF(w) = \log \frac{N}{n_w}, \quad (2.2)$$

where N is the total number of documents in the corpus and n_w is the number of documents containing the word w . This means that the IDF is low for words that have a high frequency throughout the corpus, and even zero for words that occur in all documents. Words that only occur in a few documents have a high IDF.

By combining TF and IDF, the *Term Frequency-Inverse Document Frequency* (TF-IDF) is obtained. The mathematical formula for TF-IDF is

$$TF-IDF(w) = TF(w) \cdot IDF(w). \quad (2.3)$$

This means that TF-IDF outputs a vector where words that should be significant to the document, i.e. having a large frequency in the document but a low frequency in the corpus, get large values. Words that are common in the corpus get small values, while words that are missing from the document or occur in all documents are set to zero.

2.1.2 Word embeddings

Word embeddings are a type of models that represent individual words as real-valued vectors in a pre-defined vector space. They are designed to quantify and categorize semantic relations between similar words. For example, it finds that the relation between the word embeddings of man and woman, is very similar as for king and queen.

Word embeddings provide a dense vector representation of words of a fixed length. The output dimension is usually in the region of a few hundred. This can be compared to the hundreds of thousands or more dimensions required by TF-IDF, depending of the corpus.

The simplest word embeddings can be seen as just a look up table of words in a vocabulary, where all words have been assigned a pre-determined vector representation. An example of this is the *GloVe* embeddings proposed by Pennington et al. (2014). The model was built by the use of a word-to-word co-occurrence matrix, containing probabilities of how likely it is for two words to occur in the same context. It works with the non-zero entries of this matrix and concludes whether or not two words have some linguistic or semantic similarity based on the statistics.

Since the word embedding models output a vector for every word, they need to be combined in some way, in order to get a representation for an entire document. One simple way of doing this is by averaging over the embeddings of all the words in a document.

A problem in NLP, and for anyone who learns a new language for that matter, is that a word can have completely different meaning depending on the context. For instance, the word *book* could mean the thing you read or the thing you do when you have decided to take that trip to Mallorca. This is called polysemy of words, and is something some models are able to capture.

There are other ways of building word embedding models than to use the co-occurrences statistics, for instance by making use of neural networks. And there are many different pre-trained word embedding models available. They can be grouped into two types of models. The first are the static models, that work as look up tables, as mentioned above. Each word gets the same embedding, regardless of where in a sentence it appears. Some examples of static embedding models are *GloVe* and *fastText*. These word embeddings have a reasonably small dimension and are fast to compute. The second type of word embeddings are more advanced, and produce different embeddings for a word depending on the other words that surround it in a sentence. This is called context awareness, and should make the models able to capture polysemy. Some examples of such models are *Flair* and *ELMo*. This type of model is however computationally heavier.

Table 2.1 shows a schema over the considered embedding models with their properties. For comparison, TF-IDF is also included, although it is not technically an embedding model. Two sentence embedding models that are investigated below are also included in the table. All embedding models will be presented thoroughly in the following sections, and later implemented to evaluate their performance on the patent similarity task.

Table 2.1: Properties of different embedding models.

Model	Based on	Output dimension	Word similarity	Unseen words	Context sensitive	Type of embedding
TF-IDF	Word	"∞"	No	N/A	No	N/A
<i>GloVe</i>	Word	25–300	Yes	No	No	Word
<i>fastText</i>	Sub-word	300	Yes	Yes	No	Word
<i>Flair</i>	Character	2148	Yes	Yes	Yes	Word
<i>ELMo</i>	Character	1536	Yes	Yes	Yes	Word
USE	Sub-word	512	Yes	Yes	No	Sentence
SBERT	Sub-word	768	Yes	Yes	Yes	Sentence

The column *Based on* in Table 2.1 shows how the models treat words. It could be as complete words, sub-words where the models splits the words into smaller parts, or on a character level. The *Output dimension* is the length of the vector that the model outputs. For TF-IDF we put “ ∞ ” because it is the same as the number of unique words, which could be however large as possible. GloVe can have different dimensions depending on the chosen model. The *Word similarity* means whether or not the model returns similar vectors for similar words. For instance, the embeddings for the words *apple* and *pear* would be more similar to each other than the words *apple* and *car*. By *Unseen words* we mean if the model can calculate embeddings from words it has not seen during training. This is not applicable for TF-IDF since it is not actually trained, but just applied on the whole corpus. The last two columns are simply if the model is context sensitive or not, and if it’s a word or sentence embedding.

2.1.3 fastText

FastText is a word embedding proposed by Bojanowski et al. (2016). As the name suggest it was created with fast text classification in mind. In their paper, they recognize the fact that models based on deep neural networks have become increasingly popular, because of their high performance on a variety of NLP tasks. However, they tend to be very heavy in terms of both memory and computational resources, as well as taking long time to train and evaluate. When dealing with small data sets, this might not be a problem, but since we are dealing with a huge data set of over 55 million documents, this is certainly a factor to take into account.

The proposed model, fastText, is a subword-based linear model. In contrast the models mentioned in the following sections, fastText contains predetermined embeddings for a set of words. However it is not limited to these. It is based on a skipgram model proposed by Mikolov et al. (2013) which is built on a vocabulary of words. It uses a log-linear classifier to maximize the classification of surrounding words in a sentence. What this method fails to do is capture the internal structures of words. What Bojanowski et al. (2016) propose is a modified scoring function which takes this into account. It relies on the use of character n-grams in the representation of words. What this means is that to get the representation of a word, the models splits the word into so called character n-grams. N-grams within language modeling is a contiguous sequence of n long items from a single sample. As an example, a tri-gram of the word *patent* as Bojanowski describes would be

$$< pa, pat, ate, ten, ent, nt >,$$

where the padding characters $<$ and $>$ are added to mark the beginning and end of the words. This is the reason why the model is said to be subword-based. The representation of this word would then be constructed as the sum over the vector representations of its n-grams, as well as the word itself.

The use of n-grams also allows for the method to output representations of words not already in the vocabulary, since new words can be constructed of multiple n-grams. All vectors from the fastText model have a length of 300.

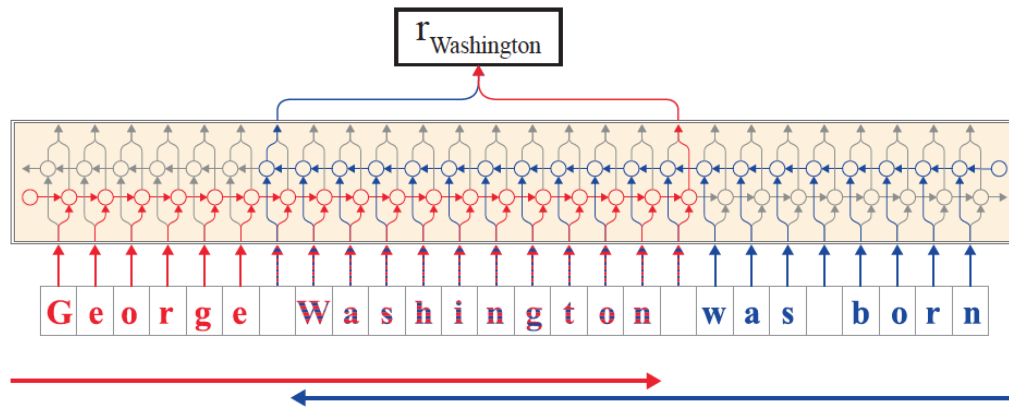


Figure 2.1: Visualization of the extraction of the word Washington.
Image from Akbik et al. (2018).

The reason for using fastText would mainly be the speed and relatively low memory requirement. Compared to other low-memory, low-computational power models such as GloVe (Pennington et al., 2014), fastText can be used on out-of-vocabulary words.

2.1.4 Flair

Another recent word embedding is Flair embeddings proposed by Akbik et al. (2018). It is of the type character based meaning it models words as a sequence of characters, as well as being context-sensitive, meaning it can capture the polysemy of words. At the time of publication the authors claim to have outperformed previous state-of-the-art works on several classic sequence labeling tasks, i.e. the task of labeling words in a text with linguistic tags.

The architecture consists of a bidirectional character-level neural language model. It uses a forward-backward LSTM recurrent neural network. The sentences are passed through this network as sequences of characters and for each word a contextual string embedding is retrieved. This embedding can then be used in any downstream NLP task (Akbik et al., 2018).

Figure 2.1 visualizes how the model extracts the representation of the word *Washington* from a sentence. The forward feature extraction is shown in red. The output from each hidden state is propagated forwards from the beginning of the sentence all the way up to the end of the word *Washington*. Similarly the backwards language model propagates its information from the hidden state at the end of the sentence up to the first hidden state of the word. The two outputs are then concatenated to receive the final embedding of the word.

Akbik et al. (2018) also suggest that combining different types of embeddings by concatenating them might prove to give even better results. Combining the Flair embeddings with GloVe embeddings, they managed to increase the F1 score on a *name entity recognition* (NER) task from 91.97 to 93.07. The reason behind this as Akbik et al. (2018) suggest, is that the classic word embeddings capture word-level semantics that complement the character-level features captured by their proposed Flair embeddings.

Following the recommendations of Akbik et al. (2018), the model we chose to implement is the proposed forward and backward Flair embeddings together with GloVe as the classical word-based embedding.

2.1.5 ELMo

Another method with similar characteristics as Flair is ELMo, proposed by Peters et al. (2018). Much like Flair it is a context sensitive character based model based on a bidirectional LSTM, i.e. both a forward *language model* (LM) and a backward LM. They compute the probability of tokens given the history and future respectively. However, unlike Flair and many other previous works ELMo does not only use the top LSTM layer. Instead it learns a linear combination of all the internal layers of the *bidirectional language model* (biLM).

The output vector of the final model presented by Peters et al. (2018) consists of three concatenated parts, one corresponding to a context insensitive token representation (i.e. a simple word representation), followed by two LSTM layers. Peters et al. (2018) also suggest that improvements can be made by experimenting with settings, such as adding a simple pre-trained word embedding like GloVe (Pennington et al., 2014) or adding dropout and/or layer normalization.

2.1.6 Sentence embeddings

So far, we have only presented models that create embeddings for words. Some have a static representation for each word, while others are context aware and create different embeddings for a word depending on the words surrounding it. In order to get a numerical representation for an entire document, we explained that an average of the word embeddings for all the words in the document could be used.

This approach can be problematic for static word embeddings, since information about the sentence structure is lost. Two sentences containing the same words, but in different order, will get the same numerical representation. For example, the sentences

I will get a cookie, but he will not
He will get a cookie, but I will not

have completely different meaning, but will get the same numerical representation when their word embeddings are averaged. Context-aware models do not suffer from this problem, but the question of which way to represent sentences or documents has been raised. This is still an active research area.

In the last few years, new ways of representing sentences or entire documents have been developed. For instance, several different *sentence embedding* models have been developed. Unlike word embeddings, they take entire sentences or paragraphs as input, process them through a neural network, and output one embedding for an entire sentence. For this thesis we consider two recent models, *Sentence-BERT* (SBERT) and *Universal Sentence Encoder* (USE).

2.1.7 BERT-based models

BERT

BERT stands for *Bidirectional Encoder Representations from Transformers* and was introduced by Devlin et al. (2019). It is a method to pre-train a single language model that later can be used for multiple different downstream tasks, a process called *transfer learning*. As the name suggests, BERT learns bidirectional representations of text, meaning that the method is context aware. The architecture behind BERT is the *Transformer*. This is an architecture that has become a popular alternative to RNN-based models such as the LSTM, because of its ability to process sequences in parallel. For further reading about the Transformer architecture, we refer to Vaswani et al. (2017).

BERT uses a trained WordPiece model to create embeddings for all the words in a sentence. The WordPiece model breaks down longer words into subwords, and is therefore also able to handle unseen words. This collection of word and subword embeddings – the token embeddings – are then used as input to the transformer network, but must first be combined with two other types of embeddings.

Unlike RNNs and LSTMs, the transformer architecture has a non-sequential structure. In order to capture information about sentence structure, a position embedding is needed for each token in the input. BERT is also able to take two sentences as input, which can be useful for some tasks. Therefore, it also has segment embeddings that identify to which sentence each token belongs. This ability was however not used by us.

The token embedding, position embedding and segment embedding are added for each token. This becomes the input to the transformer network. See Figure 2.2 for a visualization.

Sentence-BERT

Sentence-BERT (SBERT) by Reimers and Gurevych (2019) is a model based on BERT that produces sentence embeddings. It adds a pooling layer to the end of the BERT network in order to get sentence embeddings of fixed size. It is then fine-tuned to a specific task.

Several versions of the SBERT model exist, but the one selected for this thesis uses mean pooling, which means taking the mean of all output vectors from BERT. It is first trained on two *natural language inference* (NLI) datasets¹. The model is then fine-tuned on the *semantic textual similarity* (STS) benchmark², with the objective to minimize the mean-squared-error loss function of the cosine similarity between sentences. This should make the model particularly suitable for sentence comparison. The architecture of this SBERT model can be seen in Figure 2.3.

¹<https://github.com/UKPLab/sentence-transformers/blob/master/docs/pretrained-models/nli-models.md>

²<http://ixa2.si.ehu.es/stswiki/index.php/STSBenchmark>

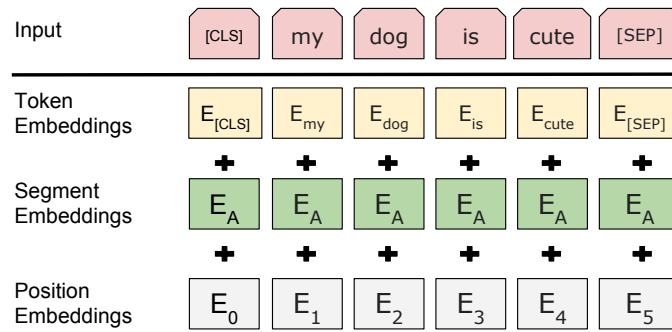


Figure 2.2: The input to the transformer network used in BERT. Token embeddings are created from a WordPiece model, and are words or subwords. Position embeddings capture information about the sentence structure. Segment embeddings determine which sentence each token belongs to, in case two sentences are used as input to the model. The token embeddings, sentence embeddings and position embeddings are added, and then become the input to the transformer network. Image from Devlin et al. (2019).

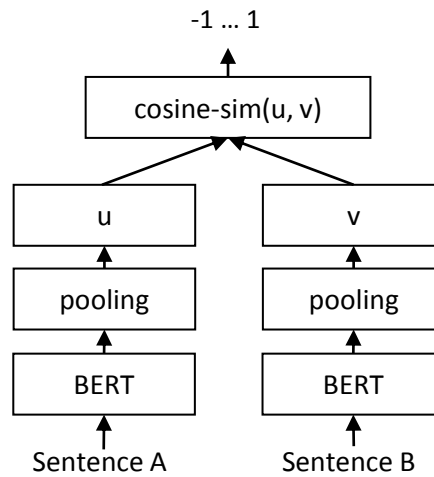


Figure 2.3: The SBERT architecture. The sentences *A* and *B* are sent through a BERT network followed by a pooling layer, and output the sentence embeddings *u* and *v* respectively. The embeddings can then be compared, for example using cosine similarity. Image from Reimers and Gurevych (2019).

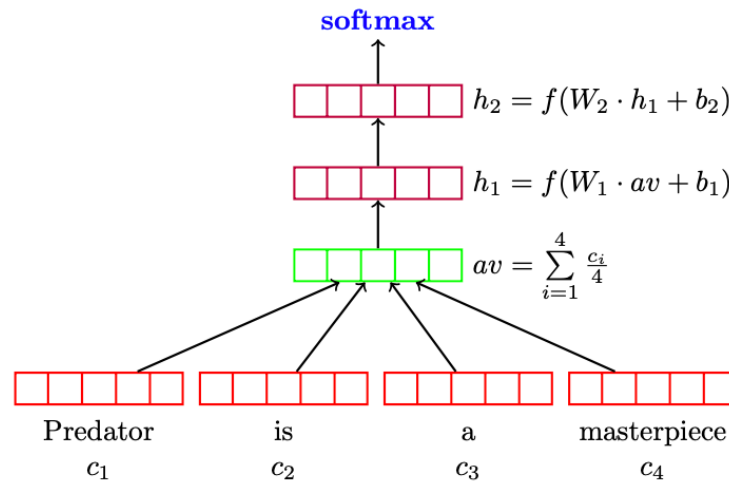


Figure 2.4: A two-layer deep averaging network. The input to the network is the word embeddings for the string *Predator is a masterpiece*. The word embeddings are averaged and then passed to the two hidden, nonlinear layers. Image from Iyyer et al. (2015).

2.1.8 Universal sentence encoder

The universal sentence encoder (USE) by Cer et al. (2018) is another method for producing sentence embeddings. It has been shown to perform well several transfer learning tasks, for example *semantic textual similarity* (STS).

USE is available in two different versions: one transformer-based and one using a *deep averaging network* (DAN). The authors claim that the DAN has “slightly reduced accuracy”, but requires noticeably less computing resources. The transformer-based version does in fact have complexity $O(n^2)$ in sentence length, while the DAN is $O(n)$. The memory usage is also $O(n^2)$ for the transformer model, but constant for the DAN. For these reasons, we choose to only consider the DAN.

The deep averaging network was proposed by Iyyer et al. (2015), and is an unordered composition function for combining word embeddings into sentence embeddings. Similarly to the above mentioned approach of averaging word embeddings to get sentence embeddings, the first layer of the DAN is an average of the individual word embeddings for all the words in the sentence. It is then followed by several non-linear layers, which are shown to be able to capture subtle differences, for example negations, better than just averaging. See Figure 2.4 for a visualization of the DAN architecture. Here, f is an arbitrary activation function. Although DAN does not take word order into account, it is able to compete with more advanced models that consider sentence structures.

The DAN used by USE averages embeddings for both words and bi-grams in the first layer of the neural network. The network then outputs a sentence embedding of 512 dimensions.

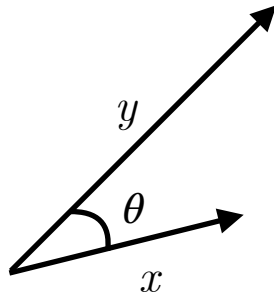


Figure 2.5: Two vectors x and y separated by an angle θ . The cosine similarity between the two vectors is given by $\cos \theta$.

2.2 Document Similarity

The models described above are able to create numerical representations of documents by using embeddings for words or sentences. In order to determine how similar two documents are, their corresponding embeddings need to be compared. There are several ways of doing this, for example by calculating the Euclidean distance or using cosine similarity. Cosine similarity is one of the most common ways to do document comparison for embeddings, and is the method that we consider in this thesis.

2.2.1 Cosine similarity

The *cosine similarity* is a way to measure the similarity between two n -dimensional vectors, by comparing their direction from the origin. It can be derived from the definition of the dot product between a vector x and a vector y according to

$$\cos \theta = \frac{x \cdot y}{|x||y|} \quad (2.4)$$

where θ is the angle between the two vectors. See Figure 2.5 for a visualization. The cosine similarity is always between -1 and 1 , where 1 means that the vectors point in exactly the same direction, -1 means that the vectors point in opposite directions, and 0 means that the vectors are perpendicular.

The *cosine distance* is defined as

$$1 - \cos \theta \quad (2.5)$$

and is simply a transformation of the cosine similarity to the range $[0, 2]$. Note that unlike cosine similarity, two vectors pointing in exactly the same direction will get a cosine distance of 0 . Just like for the Euclidean distance, a larger cosine distance means more dissimilarity between the vectors. The cosine distance is however upper limited to the value of 2 , while the Euclidean distance is unbounded.

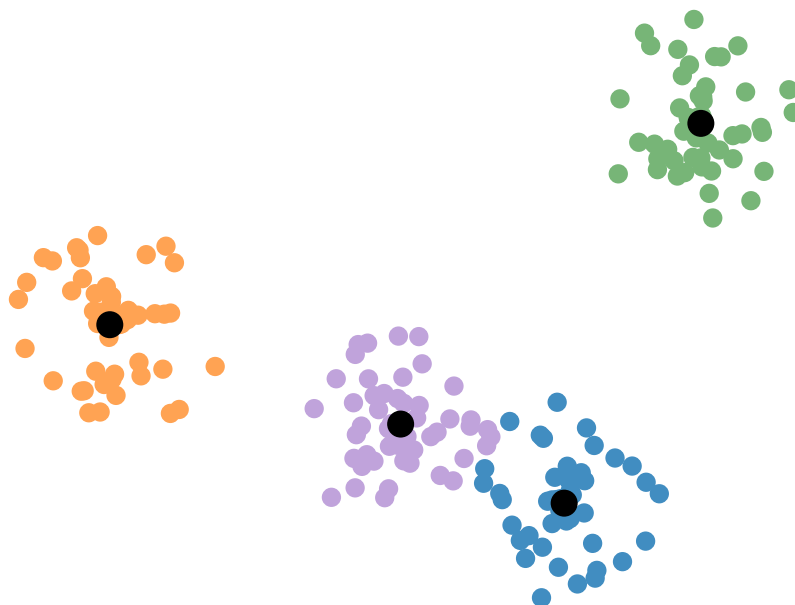


Figure 2.6: K-means clustering applied to a 2-dimensional data set, with $k = 4$. Two of the clusters have a slight overlap, but are still divided into two clusters because of the choice of k .

2.3 Clustering

Calculating the cosine similarity between two documents is a relatively efficient operation. However, when a huge number of documents, for example 55 million, are to be compared to each other, this can take a couple of minutes. A way to speed up the process is to *cluster* the document embeddings, and focus the search to only the closest clusters.

Clustering is an unsupervised learning method used to group data points into clusters. The goal is that the data points within each cluster should be more similar to each other than to points in other clusters.

2.3.1 K-means clustering

K-means clustering is a commonly used method to cluster data points into k different clusters, where k is a number decided before the algorithm starts. In K-means clustering, each cluster has a centroid $c \in C$. The positions of the centroids are first initialized by randomly selecting k points from the data. The method then consists of two steps that are repeated until convergence.

In the first step, each data point $x \in X$ is assigned to the cluster of the closest centroid. The second step then consists of updating the positions of the centroids. This is done by setting each centroid to the mean value of all data points assigned to their clusters. Figure 2.6 shows a 2-dimensional data set that has been clustered with K-means.

Mathematically, K-means clustering can be seen as the optimization problem of choosing the centroids c to minimize the objective function

$$\min_{c \in C} \sum_{x \in X} \|f(C, x) - x\|^2, \quad (2.6)$$

where $f(C, x)$ is the centroid with shortest Euclidean distance to x (Sculley, 2010).

Mini batch K-means is a modified version of K-means that uses mini-batches to speed up the clustering. It uses small randomly selected batches of data points so that they can be stored in memory. In each iteration, a new, random batch is chosen and used to update the cluster centroids. This is repeated until convergence, after which all data points are assigned to the final centroids to obtain clustered data the same way as the ordinary K-means. Mini batch K-means has been shown (Sculley, 2010) to be several orders of magnitude faster than regular K-means on a large data set, while only producing slightly worse results.

Another way to improve the performance of K-means is to use the *k-means++* initialization method, instead of randomly choosing some of the data points as initial centroids. *k-means++* still selects centroids randomly from the data points, but uses a weighted probability distribution to do this. It is defined as follows (Arthur and Vassilvitskii, 2007):

1. Select the first centroid c_1 randomly from the uniform distribution of the data points.
2. Let $D(x)$ be the shortest Euclidean distance from the point x to a centroid that already has been selected.
3. Now select centroid c_i randomly from the remaining data points $X - C$, with probability

$$P(C_i = x) = \frac{D(x)^2}{\sum_{x \in X - C} D(x)^2} \quad (2.7)$$

4. Repeat Step 3 until k initial centroids have been selected.

The creators of *k-means++* have shown that it is faster than regular K-means, and also less likely to converge to a local minimum.

2.4 Evaluation Method

In order to determine how good an embedding model is at finding similar documents, some kind of evaluation metric is needed. One way of doing the evaluation, is to take the best matches for a given document, ordered by cosine similarity, and manually label them as either *similar* or *not similar*. By doing this for a test set of documents, the labeled data can then be used to calculate a metric of how good the model is. In this section, we present two general evaluation metrics, and a specific metric used to evaluate ordered results in a top list.

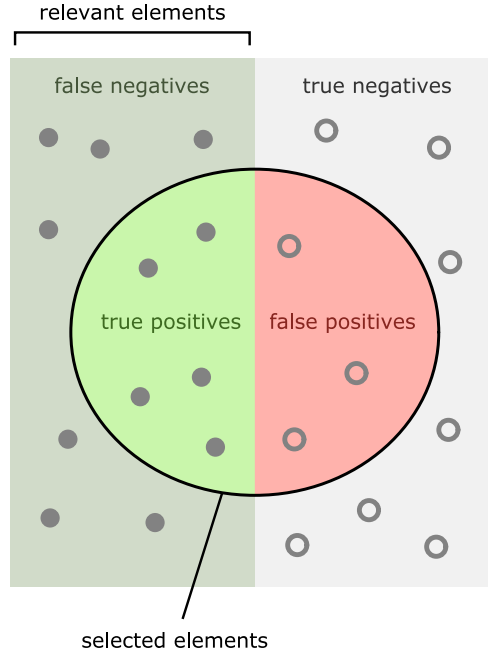


Figure 2.7: Visualization of the concepts *true positives*, *false positives*, *false negatives* and *true negatives*. Image from Wikimedia Commons (2014).

Precision and *recall* are two important evaluation metrics in machine learning. They are defined as

$$\text{precision} = \frac{tp}{tp + fp} \quad (2.8)$$

$$\text{recall} = \frac{tp}{tp + fn} \quad (2.9)$$

where tp is the number of true positives, fp is the number of false positives, and fn is the number of false negatives. These concepts are visualized in Figure 2.7. Both the precision and the recall are relevant when evaluating a model, and there is a trade-off between the two measures (Zhu, 2004).

A variant of the precision called *average precision* (AP) is often used in the area of information retrieval, where the order of the results also is important, and needs to be taken into account in the evaluation. The average precision penalizes models that return false positives in the top of the list. It can also be calculated for just the top k results in a list, meaning that all results do not need to be labeled. It is then called the *average precision at k* .

For a search query that presents k documents $(q_1, \dots, q_k) \in Q$ ordered by their relevance, the average precision at k is defined as

$$AP@k(Q) = \frac{\sum_{i=1}^k rel(q_i) \cdot P@i(Q)}{\sum_{i=1}^k rel(q_i)} \quad (2.10)$$

where $rel(q_i) = 1$ if document q_i is relevant, and 0 otherwise (Teufel, 2007). $P@i$ is the precision as defined by (2.8) if only the first i documents are considered.

After N such queries, the *mean average precision* (MAP) at k can be calculated as,

$$MAP = \frac{1}{N} \sum_{i=1}^N AP@k(Q_i). \quad (2.11)$$

If the same queries are run using the different models, the MAP can be used as a measure to evaluate the performance of the models. The higher the value, the better the model is at finding relevant documents.

Chapter 3

Approach

This chapter describes the implementation of the algorithms and the overall pipeline of the resulting web application. This includes preprocessing of the data, interaction with the database, development and features of the user interface and details of model implementations. The code for the implementation was written in Python, utilizing the large number of machine learning and NLP libraries available for that language.

We used the PostgreSQL relational database to store all the data needed for the project, including the original DOCDB files, extracted abstracts, titles and the calculated word embeddings.

3.1 Pipeline

The general pipeline is presented in Figure 3.1. The pipeline can be divided into two parts. A preparation part which is run only once and a runtime part which is run for each new query. The details are presented in the following sections.

The preparation part begun with preprocessing both of the database and the data. The data was then fed forward to calculate the embeddings of the five different models. The embeddings were then saved to the database, where they later could be collected and used to compare documents. To avoid comparing against all documents, the embeddings were clustered using the mini batch K-means described in Section 2.3.1.

The runtime part of the pipeline describes what was done for each query. First the embedding of a target document was either retrieved or computed. This was compared to the centroids to find the n closest clusters. The embeddings of all documents in the n clusters were retrieved

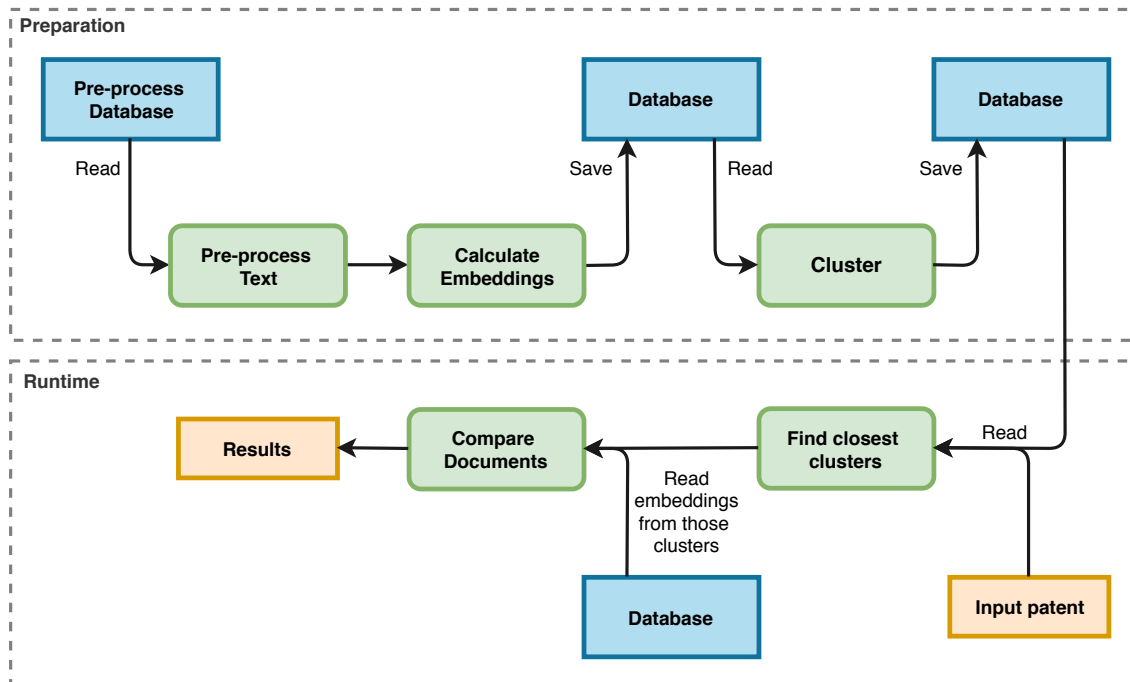


Figure 3.1: Illustration of the pipeline. Blue boxes represent the database, green computations and orange input and output.

from the database, and then compared to the target document using cosine similarity. Finally the results were presented in a list ordered by the cosine similarity score.

3.2 Preprocessing the Data

An important part in all machine learning tasks is the preprocessing of the data. For this project, preprocessing was done at two places: in the database, to extract needed data from the patent applications, and in the Python code just before calculating any embeddings.

3.2.1 Preprocessing of the database

The SQL database that was provided contained the original EPO DOCDB data as a table of XML-files. Some preprocessing had already been done to extract abstracts and titles into separate tables in the database. New tables were also created to store the resulting word embeddings, cluster centroids and cluster assignments.

In order to speed up lookup and data retrieval, SQL indexes were created for the most important tables (e.g. abstracts and titles). Non-English documents were removed from the tables.

3.2.2 Preprocessing of the data

Before calculating any embeddings, the data needs to be preprocessed to fit and make sense for the models. This can be divided into two parts.

The first part is related to cleaning the data, and is the same for every model. The available data consists of titles and abstracts from the EPO database. These texts contain a lot of figure references, HTML tags and other numbers which don't make sense to include when calculating the embeddings. They were therefore removed. All characters were also converted to lower case.

The second part is regarding the structure of the data. Each model has different requirements when it comes to the data it receives as input.

FastText is the only non context aware word embedding model that we used separately. For this model the sentences are broken down into words with the help of the *Natural Language Toolkit* (NLTK). Also, common words such as *the*, *a*, *is* and *are*, which are known as stop words, are removed, since they are present in almost all documents and therefore don't contribute to the uniqueness of them. We used the list of stop words as defined in the NLTK library.

The remaining models are all either context sensitive or sentence embeddings, meaning the sentence structure needs to be kept in some way. Because they are significant to the semantics, the stop words are not removed, as with fastText. The ELMo and Flair embeddings were both implemented through the Flair framework, which allows the input to be a string of the whole document. This is a bit different from USE and SBERT, which require the words to be grouped into lists representing the sentences.

3.3 Implementation Details

The implementation details for the models and the clustering are presented below. It includes the sources of the models and what libraries were used. An often used library is *Scikit-learn* (sklearn) which provides both a cosine similarity method and the mini-batch K-means algorithm.

As mentioned above one way of obtaining document embeddings from word embedding models is to average over the word embeddings of the document. This was the chosen method for all word embedding models that were used. For the sentence embedding models, the entire abstracts were input as one single paragraph. This meant no averaging was needed to obtain a document embedding.

TF-IDF

We used the sklearn library to implement TF-IDF. However, it quickly occurred to us that having more than 55 million documents would result in a huge amount of unique words, making TF-IDF a bad choice memory wise. TF-IDF was therefore discarded, and not used in the following evaluation.

fastText

For the implementation of fastText, the model was collected from the fastText website¹. This model has been trained on both the *Common Crawl* and *Wikipedia* datasets using CBOW with character n-grams of length 5. The output vectors are of size 300. (Grave et al., 2018)

The fastText implementation from the Gensim framework² was then used in the code. It is a widely used software framework for topic modeling with large corpora (Řehůřek and Sojka, 2010).

Flair

The Flair embeddings were implemented through the simple-to-use Flair framework.³ As mentioned in Section 2.1.4, the chosen model is a combination of both the forward and backward Flair embeddings as well as GloVe. This results in an output dimension of 2148; 1024 for each Flair embedding and 100 for the GloVe embedding. The model has been trained on a corpus of 1 billion words proposed by Chelba et al. (2013).

ELMo

The ELMo embeddings were also implemented through the Flair framework. The model used was the medium sized one with output dimension 1536, collected from the creators website.⁴ It has been trained on the same dataset as Flair.

USE

The USE embeddings were implemented using TensorFlow Hub.⁵ The chosen model is the one trained using the DAN architecture with output vectors of dimension 512.

SBERT

The SBERT model was implemented using UKPLabs *SentenceTransformer* library.⁶ The *bert-base-nli-stsb-mean-tokens* model was chosen because of its high score on STS tasks. It produces an output vector of dimension 768.

Clustering

As mentioned above, we clustered the embeddings of each model to avoid having to compare against all documents in every search query. For this, the mini batch k-means implementation of the sklearn library was used. The centroids were initialized using the *k-means++* method described in Section 2.3.1. The number of clusters was set to 37, which should give about 50 000 documents in each cluster.

¹<https://fasttext.cc/docs/en/crawl-vectors.html>

²<https://github.com/RaRe-Technologies/gensim>

³<https://github.com/zalandoresearch/flair>

⁴<https://allennlp.org/elmo>

⁵<https://tfhub.dev/google/universal-sentence-encoder/3>

⁶<https://github.com/UKPLab/sentence-transformers#pretrained-models>

The goal was not to create a perfect clustering, but to filter out documents that would be too dissimilar, and thereby speed up the search process. Therefore, documents in the 8 closest clusters of the target document were retrieved and compared using cosine similarity. For every search, about 1/5 of the documents in the database would then need to be fetched.

Other parameters used in the clustering is a maximum number of iterations of 500 and a batch size of 100 000.

3.4 Graphical User Interface

We also created a graphical user interface in the form of a web application, in order to make a patent search tool that was portable and easy to use. This was done in Python using the Flask⁷ framework for web applications, which made it easy to integrate with the machine learning code. We used HTML partials to produce the actual content to be displayed to the user, together with CSS for styling and some JavaScript for dynamic features.

A visualization of the web application can be seen in Figure 3.2. As shown in Figure 3.2a the user is able to perform a search for similar patent documents by entering either a patent application number or free text. The embedding model to be used can also be selected. The user also has the option of displaying the cited documents of the target patent, and limit the results to only contain patent applications with earlier filing dates, since newer ones could not act as prior art.

Figure 3.2b shows how the results are presented with the target document at the top. It is followed by a list of documents sorted by their cosine similarity with the target document, in descending order. Information like title, abstract, applicant and filing date are presented for all the documents. A link to the complete patent document on Espacenet is also provided.

3.5 Evaluation Procedure

The five models fastText, ELMo, Flair, USE and SBERT were all used to compute embeddings for a set of documents. The idea was then to compare the results to find out if they could be used for the intended purpose, and which model would be most suitable.

Due to limitation in time and computational power the number of documents used in the evaluation of the methods had to be limited. A subset of the complete database of patents filed with EPO was chosen for this reason. This subset contained about 1.8 million documents within a variety of subject areas. This was thought to be representative to using the complete dataset, in contrast to previous works where instead a certain patent class has been chosen to limit the data. Choosing only the *European patents* (EP) also removed the case of duplicate patents occurring because of patents being filed in multiple countries.

⁷<https://github.com/pallets/flask>

Search on patent number

Insert patent number

EP1785977A1

S-BERT

☐ Display citations

☐ Ignore documents after filing date

Search

Search using free text

Insert free text

FastText

Search

(a) The search form of the web application. The user can select one of the implemented models and perform a search using either a patent application number as in the upper search form, or by entering free text in the lower search form.

When a search is done on patent application number, it is possible to list the documents cited in the application. It is also possible to ignore any documents after the filing date of the application.

Search results

Method: **SBERT**

Plasma display apparatus
EP1785977A1
LG ELECTRONICS INC

TARGET
Application date: 2006-01-09
Priority date: 2005-11-14

A plasma display apparatus comprises a plasma display panel (400) including a scan electrode and a sustain electrode (102,103,Y,Z); and a controller (440) for applying a negative waveform and a positive waveform to the scan electrode between a reset pulse and a scan pulse having negative polarity, wherein the controller applies a sustain bias voltage to the sustain electrode when the negative waveform is applied to the scan electrode. The generation of unwanted discharges can thereby be suppressed.

[Find on Espacenet](#)

Sustain driver for a plasma display
EP1887545A2
LG ELECTRONICS INC

86.85%
Application date: 2007-08-10
Priority date: 2006-08-10

Plasma display apparatus and driving method thereof
EP1612760A2
LG ELECTRONICS INC

82.83%
Application date: 2005-06-24
Priority date: 2004-06-25

(b) The search results. The target document (entered patent application or free text) is displayed at the top. It is followed by a list of documents, sorted according to their cosine similarity to the target document.

Title, abstract, applicant, application number and priority date are presented for all documents.

The *compare* button displays a window where the most similar sentences between the target document and the selected document are highlighted. There is also a link to the document on *Espacenet*, where more data about the application can be found.

Figure 3.2: The web application created to evaluate the models.

As mentioned in Section 2.4, search results need to be labeled as either “relevant” or “not relevant” to be able to calculate the MAP. In the case of text similarity it is hard to accomplish this in any other way than to do it manually. But to evaluate search results on a large number of documents for five different methods is a very time consuming process for two people. In order to do this within the limited time frame of this thesis, we had the help of trainee associates at AWA.

The trainees were instructed to make searches on 3 different documents each, using all five models. They would then score the top five results on a scale from 1–5 according to Table 3.1, where 1 was not at all relevant and 5 was very relevant. We chose this ranking system since it made it easier for both us and the trainees to separate the different levels of relevance, compared to using a larger scale of e.g. 1–10. The opposite case of using a smaller scale, for instance a binary scale of just *relevant* and *not relevant* was also dismissed. The reason being that it could give more uncertain results for borderline cases.

To reduce the human factor of some finding a document relevant which others don’t, we assigned three trainees to the same three documents. Their results were then combined by taking the highest scores for each document. This was done because only one person finding a patent relevant should be sufficient reason to investigate that patent further in order to minimize the number of false negatives. The detailed instructions provided for the trainees can be found in Appendix A.

Table 3.1: The scoring system that was used by the trainees to evaluate the search results for the different models.

Score	Ranking	Explanation
1	Not at all relevant	<i>The patents are covering completely different subjects</i>
2	Not relevant	<i>I can see why the model thought it was similar, but wrong subject</i>
3	Could be relevant	<i>The patents seem to cover the same area but not really the same thing</i>
4	Relevant	<i>The patents cover pretty much the same area, could be useful if investigated further</i>
5	Very relevant	<i>The patents cover the same area and just from the abstracts are very similar</i>

In order to calculate the MAP according to Section 2.4, the scores must be converted to binary representations as *relevant* or *not relevant*. We did this in two different ways, producing two different MAP values. First, all documents with a score of 4 or 5 were considered relevant, while all other documents were considered irrelevant. This was used to calculate the $\text{MAP}_{4,5}$ result. Then, only documents with a score of 5 were considered relevant, leading to the MAP_5 value.

We also evaluated the clustering, to make sure it did not have a large negative impact on the search results. This was done separately for each model, and could be made automatically, without the need of any human input. As a test set, ten random documents were chosen

from each cluster. The embedding of each such document was then used to search for similar documents using cosine similarity. This was done in two different ways.

First, the embedding was compared to the embeddings of all other documents in the database. A list of the top n most similar documents, in descending order, was returned. Then, only documents in the 8 closest clusters were considered – just like in the search tool used by the trainees – and another top list was returned. In the ideal case, both these lists should consist of the same documents. A bad clustering would however cause documents to be missed. This evaluation was done for $n = 50$ and $n = 10$. The number of missed documents and the recall between the two lists could then be calculated.

Chapter 4

Evaluation

This chapter begins with a presentation of the results from the evaluation. This includes different MAP scores, individual rankings of the models and an evaluation of the clustering. This is followed by a discussion of the results as well as sources of errors. Finally, the chapter ends with some suggestions of future work.

4.1 Results

Out of the 17 trainees at AWA who received the task of evaluating the models, 8 people with at least one from every group replied with answers. The results are presented below.

Tables 4.1 and 4.2 present the mean average precision, as well as the number of true positives and false positives for the different models. For Table 4.1, documents were considered relevant if they were given a score of 4 or 5 by the trainees. For Table 4.2, only documents that received a score of 5 were considered relevant.

The highest mean average precision was reached by Sentence-BERT, with a $\text{MAP}_{4,5}$ score of 0.7655 and MAP_5 score of 0.5391. Sentence-BERT was also the model that managed to find the most true positives in both cases.

Table 4.1: The Mean Average Precision (MAP), number of true positives (TP) and number of false positives (FP) for the different embedding methods. Here, a document is considered relevant if it was given a score of 4 or 5.

Model	MAP _{4,5}	TP _{4,5}	FP _{4,5}
<i>fastText</i>	0.5996	32	58
<i>Flair</i>	0.3324	16	74
<i>ELMo</i>	0.5222	26	64
<i>USE</i>	0.6318	41	49
<i>SBERT</i>	0.7655	52	38

Table 4.2: The Mean Average Precision (MAP), number of true positives (TP) and number of false positives (FP) for the different embedding methods. Here, a document is considered relevant if it was given a score of 5.

Model	MAP ₅	TP ₅	FP ₅
<i>fastText</i>	0.3843	13	77
<i>Flair</i>	0.2685	10	80
<i>ELMo</i>	0.2907	11	79
<i>USE</i>	0.4250	18	72
<i>SBERT</i>	0.5391	28	62

In Table 4.3 the individual ranking results are presented. They are based on the average score each person has given to all documents for each model. In the case of equal average scores the number of 5's and 4's decide the ranking, where more is better. Persons *C* and *D* as well as *G* and *H* were assigned to the same groups of patent applications. All remaining persons represented their own groups.

The overall position is based on the average position from all persons. *Flair* and *ELMo* has the same number of 3rd, 4th and 5th places therefore they share the 4th place.

Table 4.3: Individual ranking of the models based on the average score.

	<i>fastText</i>	<i>Flair</i>	<i>ELMo</i>	<i>USE</i>	<i>SBERT</i>
<i>Person A</i>	3rd	5th	4th	1st	2nd
<i>Person B</i>	2nd	4th	5th	3rd	1st
<i>Person C</i>	1st	4th	5th	3rd	2nd
<i>Person D</i>	1st	3rd	5th	2nd	4th
<i>Person E</i>	2nd	3rd	5th	4th	1st
<i>Person F</i>	3rd	5th	4th	2nd	1st
<i>Person G</i>	4th	5th	3rd	1st	2nd
<i>Person H</i>	4th	5th	3rd	2nd	1st
Overall position	3rd	4th	4th	2nd	1st

The evaluation of the clustering for each model can be seen in Table 4.4. The result is presented as mean recall of the 370 (10 documents times 37 clusters) search queries, together with worst recall. This is done for both the top 10 documents and the top 50 documents, according to cosine similarity.

Table 4.4: The results of the clustering evaluation, both when the top 10 and the top 50 documents of each search query were considered. For each model, the mean recall of the 370 search queries is presented, together with the worst recall.

Model	Top 10		Top 50	
	Mean recall	Worst recall	Mean recall	Worst recall
<i>fastText</i>	0.9851	0.5000	0.9848	0.7000
<i>Flair</i>	0.9851	0.6000	0.9841	0.6200
<i>ELMo</i>	0.9973	0.6000	0.9974	0.8200
<i>USE</i>	0.9843	0.4000	0.9783	0.5200
<i>SBERT</i>	0.9805	0.6000	0.9874	0.6800

4.2 Discussion

A quick look at the results gives a hint of that machine intelligence can indeed be used for this application. Keep in mind that one relevant hit really is all it takes to deprive an invention of its novelty. So if just one out of five results are relevant it is still very good. This would mean that all methods could be used since they all found at least some documents of the score 5. That being said, of course the more relevant hits a model produces, the more likely it is to find just the right document.

Another thing to keep in mind is that a “bad” search result of very few relevant documents is not necessarily due to the model. It could very well be that the invention is the first of its kind and that it therefore could be patentable/not be revoked. This search result would then be an “good” result. The most dangerous part of a model used for this application would be to give too many false negatives, i.e. missing relevant documents. This is hard to identify but the risk can be reduced by presenting a large list of top results, at the cost of an increased amount of false positives.

4.2.1 Embedding models

Looking at the MAP of both Table 4.1 and Table 4.2 it is clear that the two sentence embedding models outperform the word embedding models when it comes to identifying similarities between patent documents. Not even the newer word embedding models like ELMo and Flair, that have reach state-of-the-art performance on several NLP tasks, were able to come close to the results of the sentence embeddings. In fact, they performed worse than the much simpler fastText model.

Table 4.3 shows the same trend. SBERT is ranked either as the best or second best model by almost every person, while Flair and ELMo was never better than third best. The table was created because of the reason that some people would consequently rank the documents relatively low, while others would rank them relatively high. By looking at the average scores given by each person you can rank the methods for each person individually. Worth to mention is that in the case of Person D Flair, USE and SBERT all had the same average score. So the ranking came down to which had the most 4's and 5's. Flair had the most, and SBERT the fewest, but it was very close. The 4th rank of SBERT is therefore not as strong outlier as it seems.

The method of simply averaging word embeddings into document embeddings is probably the main reason for the low results of the word embedding models. This approach does not take word order or sentence structure into account, and actually performs quite well considering these limitations. There are other alternatives to averaging, for instance using power-mean (Rücklé et al., 2018) or the so called *Smooth Inverse Frequency* (SIF) proposed by Arora et al. (2017) which takes weighted averages of word embeddings and modifying them via singular value decomposition. Another alternative is to use neural networks to combine the embeddings. There is an RNN based implementation of this in the Flair framework, but this requires fine-tuning of the model.

Considering that SBERT is a context aware model, that has been trained on STS to produce good sentence embeddings for cosine similarity, it is not really surprising that it achieved the best results in our evaluation. This is not the case for the other models, and it is possible that fine-tuning them on STS for cosine similarity usage could lead to better results.

USE was the second best model, even though the simpler version based on the DAN architecture was used. It is impressive how well this method performed, considering its speed — it was possible to calculate the embeddings for all 55 million documents in less than 3 hours, compared to several days for SBERT. The DAN version of USE is however not context aware, which could explain why it performed worse than SBERT. It would be interesting to do a comparison between SBERT and the larger, transformer based version of USE.

FastText is the oldest and simplest model that we evaluated. It is very efficient, both when it comes to computation and storage; the output dimension being just 300. Despite this, it showed descent results, close behind USE in both MAP evaluations. However, as with USE it lacks the feature of being context aware. It could however be that context awareness is not that important when working with short texts as in this case. A possible use case for fastText could be in combination with another model, like GloVe was used together with Flair. For instance combining fastText and USE would still result in a reasonable small dimension, close to that of SBERT, but far from the sizes of Flair and ELMo.

When it comes to ELMo and Flair, both models output high-dimensional embeddings compared to the other models. This can be a problem when word embeddings are averaged and compared using cosine similarity. This phenomenon is called the *curse of dimensionality*, and is common when dealing with high dimensional spaces in machine learning. A possible solution is to concatenate the forward and backward embeddings of these methods, which has been suggested on the GitHub page of the flair framework. However, there are no reports that proves if this helps or not.

It is also possible that ELMo and Flair produce word embeddings that are unsuitable for document comparison using cosine similarity. The embeddings have not been trained or benchmarked using a dataset like STS. Instead, they are often used as a first layer in a larger neural network. Building a sentence embedding network using these models, and fine-tuning it on a STS task, like mention above, could perhaps increase the performance of these models.

Another thing that the creators of ELMo suggests on their GitHub (AllenAI, 2019) is to experiment with some hyper-parameters. One could for instance add dropout layers, weight regularization or layer normalization. One could even add a classical pre-trained embedding model such as GloVe. However there is no way to know beforehand what works best for a certain task. The only way is to experiment and compare results. Since this is hard and time consuming for a text similarity task, it could not be done within the scope of this thesis. On the same GitHub page the creators also mention that due to the way ELMo was trained, it is not completely deterministic. Running the same text several times will give slightly different embeddings each time. Furthermore the first few batches after loading the model will be negatively impacted. They suggest to run a few batches before hand to warm up the model. These are all reasons to why ELMo did not perform that well.

4.2.2 Clustering

The evaluation of the clustering showed that most of the time, the clustering did not have a large impact on the returned search results. The mean recall for all of the models was at least 0.97, both when the top 10 and the top 50 results were considered. This means that almost the same documents were returned as for a complete database search, without the clustering. This also proves that our method of clustering the embeddings, and only using documents in the nearest clusters for cosine similarity, is successful in speeding up the search process. It is therefore possible to have a large database of patent applications, and still search for relevant documents in a relatively short time.

K-means clustering uses the Euclidean distance to group data points into clusters, see the optimization problem in equation (2.6). This may not be ideal for our use case, since we want points in the same cluster to have a high cosine similarity, not a short Euclidean distance. An alternative clustering method that could be used is *Spherical K-means clustering*, a modified version of K-means that uses the cosine distance instead of the Euclidean (Hornik et al., 2012). It is possible that Spherical K-means would provide even better clustering results with higher recall, but this would need to be investigated further.

4.2.3 Sources of errors

A limitation we had to do was to let the trainees only look at the top five results when evaluating the models. The reason to which was the time constraint. This limitation however entails a problem. The difference between the documents in the top five and the documents in the top ten is sometimes very small. Relevant documents could very well be found in the top 10, top 20 or even further down. Since they are missed there could be a case of inaccurate results.

Another source of error is the clustering. Limiting the set of which the target document is compared to of course entails a risk of missing relevant documents which might have ended up in a different cluster. To minimize this risk, the 8 closest clusters were included, but no matter how well one evaluates the clustering, it is a source of error which one can't eliminate.

4.2.4 Improvements and future work

The evaluated models have all been trained on different datasets and on different tasks. None of them has specifically been trained on patent texts or with this application in mind. Doing this could give better results. But to train the models from scratch is a very heavy task in almost every aspect. Instead of doing this one could fine-tune the models. This is something we would suggest as a future work.

Another thing that proved to be problematic is how to combine individual word embeddings to represent a document. In this thesis, a simple averaging was used, but there are other alternatives as discussed above, of which the so called power mean is one.

The data we had available were abstracts and titles. How well these describe an invention differs a lot from patent to patent. Also if one really wants to go into the scope of the protection of a patent one needs to look at the claims, as described in section 1.4. As a continuation of our work it would be interesting to see what having the full text documents available, could mean for the results.

Chapter 5

Conclusions

The aim of this thesis was to investigate the possibility of using artificial intelligence to improve the search for prior art within the patent industry. Also to find out what method performs the best and develop a working tool as a proof of concept.

We tested five different models, all of which proved to be able to find at least some relevant hits. This itself could be seen as proof of AI being useful for this use case. To evaluate the models further we let a group of people knowledgeable within the patent system perform search queries and rank the results. The results from this, presented in Section 4.1, indicated that the model SBERT with the highest MAP score of 0.7655 would be the most suitable for this application. The reason was thought to be a good balance between complexity and dimension, as well as it being a sentence embedding which proved to be suitable for document representation.

To develop this tool further, we presented numerous suggestions of future work. The main thing is to investigate what would be the results of using full text documents instead of just abstracts and titles. It would essentially be just longer documents containing more information which would require even more from the models to capture the meaning in a single vector representation. One could imagine that word embedding of low dimensions like fast-Text would struggle because of this, and the sentence embeddings USE and SBERT prove to be even better.

Another suggestion of future work is more related to the models. Firstly there is the choice of fine tuning the models on patent texts, to make them more specialized on this type of corpus. Secondly one could combine different models together in the hope of them being good at capturing different features.

In the end the aim of providing a proof of concept can be seen as successfully achieved.

References

- Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.
- AllenAI (2019). ELMo: Deep contextualized word representations. https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md. Accessed: 2020-01-06.
- Andrabi, L. H. (2015). Intelligent retrieval and clustering of inventions. Master’s thesis, KTH, School of Information and Communication Technology (ICT).
- Arora, S., Liang, Y., and Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. 5th International Conference on Learning Representations, ICLR 2017 ; Conference date: 24-04-2017 Through 26-04-2017.
- Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’07, pages 1027–1035, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.
- Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching word vectors with subword information. *CoRR*, abs/1607.04606.
- Cer, D., Yang, Y., Kong, S., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y., Strophe, B., and Kurzweil, R. (2018). Universal sentence encoder. *CoRR*, abs/1803.11175.
- Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., and Koehn, P. (2013). One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

- Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.
- Helmers, L., Horn, F., Biegler, F., Oppermann, T., and Müller, K. (2019). Automating the search for a patent’s prior art with a full text similarity search. *CoRR*, abs/1901.03136.
- Hornik, K., Feinerer, I., Kober, M., and Buchta, C. (2012). Spherical k-means clustering. *Journal of Statistical Software, Articles*, 50(10):1–22.
- Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.
- Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.
- Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.
- Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365.
- Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. <http://is.muni.cz/publication/884893/en>.
- Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *ArXiv*, abs/1908.10084.
- Rücklé, A., Eger, S., Peyrard, M., and Gurevych, I. (2018). Concatenated p-mean word embeddings as universal cross-lingual sentence representations. *CoRR*, abs/1803.01400.
- Sculley, D. (2010). Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web, (WWW ’10)*, page 1177–1178, New York, USA. Association for Computing Machinery.
- Teufel, S. (2007). An overview of evaluation methods in trec ad hoc information retrieval and trec question answering. In *Evaluation of Text and Speech Systems*, pages 163–186. Springer Netherlands, Dordrecht.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.
- Wikimedia Commons (2014). Precision and recall. <https://commons.wikimedia.org/wiki/File:Precisionrecall.svg>. File: Precisionrecall.svg.
- Zhu, M. (2004). Recall, precision and average precision. *Department of Statistics & Actuarial Science University of Waterloo*. Working Paper.

Appendices

Appendix A

Evaluation Instructions

Background

We have as our master thesis investigated the possibility of applying AI on patent data. The idea is to aid in the search for prior art by using natural language processing (NLP) to compare the abstracts of patent applications in order to find similar documents. This can hopefully be an alternative to difficult and time consuming keyword searches.

We have implemented a couple of different NLP methods, and we now need your help to evaluate how these methods perform on existing patent applications. This will be done through a web application we call PatHawk, where you enter an application number and get a list of the abstracts the method finds to be most similar to the entered application.

Due to limit in computational power, only European patent applications (EP) have been fully processed for all methods. Therefore all application numbers we provide to you are European, and the search results will only consist of European applications.¹

Usage

The search form is shown in Figure A.1 with some comments. One can search using either an application number or free text. You also have the choice of different methods and some display settings such as showing citations and ignoring documents with a later filing date.

¹The method "USE, complete database" can be selected to run the method USE on the entire database. Feel free to try it if you want, but it is not a part of the evaluation.

Search on patent number

Insert patent number

Display the citations of the target patent

Display only patents with filing dates before the target patent.

Search on patent number

Choose method

Run search

USE

☐ Display citations

☐ Ignore documents after registration date

Search

Search using free text

Insert free text

Search on free text

You will not use this in the evaluation

USE

Search

Figure A.1: Search form.

Method used

Patent you searched for

List of top 50 results in descending order

Search results
Method: FASTTEXT

Title	Application number	Abstract
Toy gun EP1416244A1	TARGET 2004-05-06	The toy gun has a gas flow control mechanism (50) movably arranged with respect to a movable portion (10) to form a first gas passage (51) for guiding gas into a loading chamber (4), and a second gas passage (52) for guiding gas to pressure receiving portions (14,22). The gas flow control mechanism makes preparations in supplying bullet (P) from a magazine (5) to the loading chamber. When in first state, the first gas passage is opened and the second gas passage is closed to supply the gas obtained in a gas outlet passage (34) to the loading chamber through the first gas passage. When in second state, the first gas passage is closed and the second gas passage is opened to apply the gas obtained in the gas outlet passage to the pressure receiving portions through the second gas passage.
TOY GUN CA2447419A1	98.98% 2004-04-30	To avoid the situation, in which a gas pressure to be used for shooting a bullet from a loading chamber is partially lost, a toy gun in which a paint bullet charged in the loading chamber may be reliably shot with gas under a relatively low pressure is provided. A gas flow control mechanism is arranged movably with respect to a movable member and forms a first gas passage for guiding the gas into the loading chamber and a second gas passage for guiding the gas to pressure receiving portions. For a period in which gas supply controllers take a gas supply state as the movable member advances, the gas flow control mechanism transfers from a state, in which the gas passage is opened whereas the gas passage is closed to supply the gas to the loading chamber through the gas passage, to a state, in which the gas passage is closed whereas the gas passage is opened to apply the gas to the pressure receiving portions through the gas passage, so that the movable member is moved back and forth thereby to make preparations for supplying the bullet P from a magazine to the loading chamber.

Compare against target

Go to application on Espacenet

Cited documents
Only citations in category X or Y are displayed.

Model gun with automatic bullet supplying mechanism.
EP0647625A1 1995-04-12: CATEGORY Y

Score

Filing date

List of X and Y citations

Pneumatic gun
US5063905A 1991-11-12 CATEGORY Y

Figure A.2: Search results.

For the evaluation you will use the methods *FastText*, *Flair*, *ELMo*, *USE*, and *S-BERT* on application number search.

The search results are presented like in Figure A.2. The 50 most similar documents are presented in descending order. Note that the scores are not comparable between the different methods, just between the different documents using the same method. So don't use this as an evaluation metric.

Procedure

What we would like your help with is to run three different searches each, using the five different methods. Then rate the search result based on how similar/relevant they are. We estimate it to take about 2 hours and you don't have to go deeper than to look at the abstracts to evaluate the similarity.

Please note that it is the underlying methods themselves we want you to evaluate, not the tool as such. But if you have any feedback on the user interface or other improvements we are happy to receive them.

Method

1. Log in to PatHawk and press **Search** in the menu.
2. Choose a method.
3. Search for an application number in the Excel list which your group has been assigned.
Note: It may take up to 2 minutes for the search to finish.
4. Rate the top five results on a scale of 1-5 using the provided Excel sheet. Feel free to look at the full documents in Espacenet if you want.
5. If a search result is an obvious duplicate, please skip it and rate the following document instead. Duplicates can for example occur because of different kind codes (A1, A2, etc).
6. Repeat for the remaining two numbers of your group, and the other methods.
7. When you are done, please send the Excel sheet to the email address below.

Good luck and thank you very much for your help!

Appendix B

List of Evaluation Documents

The patent applications used to evaluate the different embedding models are listed below.

Group	Patent application number
<i>Group 1</i>	EP2036750A2
Person B	EP2589282A1
	EP2117077A1
<i>Group 2</i>	EP1757721A1
Person C, D	EP2353677A2
	EP2182246A1
<i>Group 3</i>	EP2127577A1
Person F	EP0054730A1
	EP2239105A2
<i>Group 4</i>	EP1803857A2
Person E	EP1785977A1
	EP2942611A1
<i>Group 5</i>	EP1375913A1
Person A	EP3133150A2
	EP1332845A2
<i>Group 6</i>	EP0343381A1
Person G, H	EP1475172A1
	EP1942472A1

EXAMENSARBETE Using Natural Language Processing to Identify Similar Patent Documents**STUDENTER** Hannes Jansson, Jakob Navrozidis**HANDLEDARE** Pierre Nugues (LTH), Anders Fredriksson (AWA), Henrik Benckert (Mindified)**EXAMINATOR** Jacek Malec (LTH)

Smart patentsökning med hjälp av AI

POPULÄRVETENSKAPLIG SAMMANFATTNING Hannes Jansson, Jakob Navrozidis

Kan artificiell intelligens (AI) användas för att hitta liknande uppfinningar i en stor patentdatabas? Vi har utvecklat ett verktyg som lyckas med detta genom att representera dokument matematiskt med hjälp av AI. Detta kan vara till stor nytta för patentombudet, som ofta behöver göra krävande sökningar manuellt.

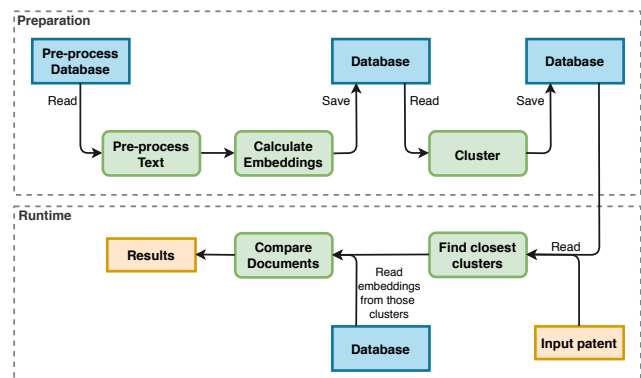
Sökandet efter *prior art*, d.v.s. bevis på att en uppfinning är ny eller inte, är en viktig men tidskrävande process för ett patentombud. I dagsläget görs dessa sökningar manuellt med hjälp av nyckelord, vilket är problematiskt eftersom uppfinningarna ofta beskrivs med abstrakta och generella termer. Dessutom behöver man ta hänsyn till synonymer. Detta innebär en risk för att relevanta dokument missas.

Vi har tagit fram en metod för att hitta liknande patentansökningar med hjälp av AI. Metoden har utvärderats av flera personer med kunskaper inom patent, och har visat sig ge lovande resultat. I de flesta fall lyckas metoden hitta ansökningar för liknande eller väldigt liknande uppfinningar.

Metoden vi använt oss av är baserad på att representera dokument matematiskt i form av vektorer. Det finns flera olika modeller för att göra detta. Ett vanligt sätt är att använda så kallade *word embeddings*, där en unik vektor tas fram för varje ord. För att representera ett helt dokument på detta sätt kan man ta medelvärdet av vektorerna för alla ord som förekommer. För att avgöra hur lika två dokument är kan man helt enkelt jämföra dessa vektorer. Vi undersökte även modeller som arbetar med hela meningar istället för ord, så kallade *sentence embeddings*.

För att snabba upp sökningen användes *klus-*

tring, ett sätt att gruppera dokument som liknar varandra. På så sätt kan man begränsa sökningen till närliggande kluster.



I figuren visas ett schema över vårt verktyg. Det består av en förberedelsedel som bara görs en gång för varje modell. Här beräknas och klustras embeddings innan de sparas till en databas. I den andra delen kan man sedan skicka in en patentansökan och söka upp liknande dokument bland de i databasen.

Till vårt förfogande hade vi en databas över ett stort antal patentansökningar. Dessa innehöll bland annat titel och sammanfattningar, vilka vi använde för att jämföra dokumenten.