INTERNATIONAL HELLENIC UNIVERSITY
School of Information Technologies

Giorgos Sideris 272020

RESEARCH AND IMPROVEMENT OF PATENT RETRIEVAL USING MACHINE LEARNING METHODS

Master's Thesis

Supervisor: Michalis Salampasis

Thessaloniki 2023

Γιώργος Σιδέρης 272020

Μελέτη και βελτίωση της ανάκτησης πατεντών χρησιμοποιώντας μεθόδους μηχανικής μάθησης.

Διπλωματική εργασία

Επιβλέπων καθηγητής: Μιχάλης Σαλαμπάσης

Θεσσαλονίκη 2023

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Giorgos Sideris

16.05.2023

Abstract

This dissertation researches and compares patent retrieval architectures using a combination of traditional first-stage retrieval algorithms and second-stage deep learning techniques. The author conducts experiments using various combinations of tools such as BERT models, the Pyserini indexing software, and the DeepCT software tools, and evaluates the effectiveness of each retrieval architecture. The datasets used, the field selection process, and the algorithm selection procedures are documented, along with the scripts and software developed for the experiments. The paper aims to determine the best techniques and methodologies for efficient and effective patent retrieval. The author concludes with their findings and proposes future research directions in this field. The study highlights the significance of information retrieval methods and their applications in everyday life in the 21st century. Furthermore, the paper provides an introduction to artificial intelligence, machine learning, and deep learning concepts, which are fundamental to understanding the thesis's technical aspects.

The thesis is written in English and is 57 pages long, including 13 chapters, 17 figures and 3 tables.

Σύνοψη

Αυτή η διπλωματική εργασία ερευνά και συγκρίνει αρχιτεκτονικές ανάκτησης διπλωμάτων ευρεσιτεχνίας χρησιμοποιώντας συνδυασμό παραδοσιακών αλγορίθμων ανάκτησης πρώτου σταδίου και τεχνικών βαθιάς μάθησης δεύτερου σταδίου. Ο συγγραφέας πραγματοποιεί πειράματα χρησιμοποιώντας διάφορα εργαλεία, όπως το μοντέλο λέξεων BERT, το λογισμικό ευρετηρίασης Pyserini και το εργαλείο με δυνατότητες χρήσης μεθόδων μηχανικής και βαθιάς μάθησης DeepCT, και αξιολογεί την αποτελεσματικότητα κάθε αρχιτεκτονικής ανάκτησης. Τα σύνολα δεδομένων που χρησιμοποιήθηκαν, η διαδικασία επιλογής πεδίων και οι διαδικασίες επιλογής αλγορίθμων καλύπτονται με λεπτομέρεια, μαζί με τα σενάρια και το λογισμικό που αναπτύχθηκε για τα πειράματα. Στόχος αυτής της διατριβής είναι να ερευνήσει τις καλύτερες τεχνικές και μεθοδολογίες για αποτελεσματική ανάκτηση διπλωμάτων ευρεσιτεχνίας. Ο συγγραφέας παρουσιάζει τα ευρήματά του και προτείνει περαιτέρω κατευθύνσεις έρευνας σε αυτό το πεδίο. Επιπρόσθετα, το παρόν εγγραφο περιέχει εισαγωγικές πληροφορίες για τις έννοιες Τεχνητή Νοημοσύνη, την Μηχανική Μάθηση και την Βαθιά Μάθηση, οι οποίες είναι σημαντικές για την κατανόηση των τεχνικών και τεχνολογικών πτυχών της παρούσας διπλωματικής.

Acknowledgements

The completion of this dissertation could not have been possible without the help from my supervisor, Mr. Michalis Salampasis and especially Mr. Vasilis Stamatis who gave me unwithering support in the first steps of my research and helped me establish a plan that I followed along this journey.

I am also very grateful to Ms. Rafaela Takou for her constant support throughout those years that I spent on this research and her words of encouragement and I also wish her good luck on her PhD program.

List of Abbreviations and Terms

IR	Information Retrieval
QF	Query Formulation
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
ANN	Artificial Neural Network
NLP	Natural Language Processing
TF-IDF	Term Frequency - Inverse Document Frequency

Table of Contents

List of Figures

List of Tables

List of code and command blocks

1. Introduction

The scope of this dissertation is to research and compare different patent retrieval architectures by focusing on Information Retrieval methods and by using a combination of traditional first stage retrieval algorithms and second stage Machine Learning techniques, more specifically, patent retrieval using Deep Learning algorithms. First stage retrieval algorithms, namely TF-IDF and BM25 are used for term weight predictions and these term weights are subsequently used for second stage retrieval models and comparisons will be executed between those retrieval architectures.

With the introduction of Deep Learning models, such as BERT, indexing software such as Pyserini and newly created algorithmic tools such as DeepCT, which leverages the capabilities of machine learning and deep learning methods, the author provides some context for his subsequent experimenting and test runs to further explore how to maximize efficiency of patent retrieval. The aforementioned tools will be presented in greater detail in the following chapters.

Moreover, the author presents a brief introduction to Information Retrieval and its methods, their application to our every-day life and its importance for reading through the noise of the overabundance of data that defines life in the 21st century. A preamble to artificial intelligence, machine learning and deep learning is also included to define some fundamental concepts and terms used in this thesis and in computer science in general. Additionally, the tools and algorithms that are used in this experimental analysis are introduced and explained in detail. BERT models and the DeepCT algorithm are included, as well as Information Retrieval software for indexing such as Pyserini and algorithms, mostly bag-of-words algorithms as mentioned above.

Furthermore, several chapters are focused on describing the process of the experiments that the author executed. The structure and origin of the datasets used is documented on a separate chapter as well as the field selection process that relates to the process of deciding which parts of the documents are the most suitable to be used in the test runs. Moreover, the algorithm selection procedure for defining the term weights used and the process in which the experiments were executed are all explained in detail in the following chapters. The scripts and software developed or used for the purpose of this thesis are briefly presented as well.

Finally, the different evaluations are compared side to side and observed to reach some final conclusions. This is also the final purpose of this paper, to find the best techniques and methodologies for efficient and effective patent retrieval. Lastly, after reaching some conclusion with the test results, further steps to progress the research are listed and discussed in this dissertation.

2. Introduction to Information Retrieval

Rapid development of the internet, computers and the increase of the capacity of storage devices in the last 50 years has led to an overabundance of information one can find on the web. Especially in the 21st century, the amount of information that is stored on the internet has been increasing almost exponentially. Meanwhile, people have been interested in finding information for thousands of years now, and lately more than ever they are keen to find relevant information quickly and effectively.

The rise of search engines, mainly Google, made users get used to rapidly finding information about what they are searching for between millions of websites and publications. More specifically, researchers around the globe are increasingly looking for scientific written knowledge online. Considering this information is mainly available in an unstructured way all over the web, one can realise how paramount intelligent and automatic ways to sift through information have been for humanity. The information found must be relevant to what the user is looking for and it needs to be useful and this can only be possible using automated methods and algorithms, considering the sheer amount of data available on the internet.

This is what prompted to the genesis of Information Retrieval - will also be referred to as IR in this document - and the corresponding field in science. IR was born around the 1950s out of necessity [1]. The field has matured significantly over the 21st century and late 20th century.

2.1 Definition of Information Retrieval

Information retrieval is the science of retrieving information in a document, in a collection of documents, or searching for the document itself. IR can also be described as the process of finding non-structural material (mainly documents) that meets the need for information from large collections [2]. IR is used extensively today by millions of users and some of the use cases include: web search, question retrieval, email search, scientific material search.

2.2 IR methodology

It would be utterly inefficient for an IR application to search through millions of documents and their corresponding text to find information relevant to a user's query. To overcome this challenge, IR systems use a representation of a document's content instead of the complete information of a document. IR systems do this by creating a representation of said content. [3]

One of the most important function of an IR systems is called indexing, which is the creation of a representation of a document, and its elements are often called keywords. Nevertheless, depending on requirements, phrases, parse trees, and semantic structures are employed as indexes. Relevant papers are those that provide users with the information they require. Ideally, only pertinent papers should be returned by a perfect IR system.

Four operations make up a typical IR system: indexing, query formulation, matching, and re-ranking. The flowchart schematic for the IR process is shown in Figure 1. IR systems may use a variety of retrieval models throughout the matching process in order to respond to a user query. [2]

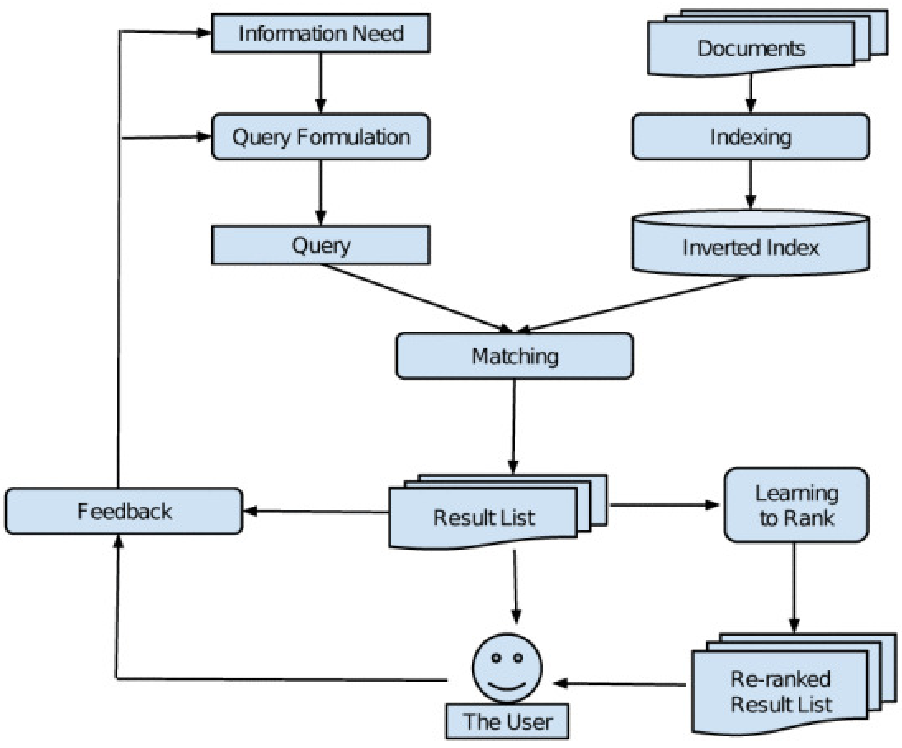


Figure 1. *Information Retrieval Process.*

2.2.1 Indexing

Firstly, indexing - as mentioned above - is the process of creating a representation of a document for easier retrieval and relevance exploration. Indexing, ideally, can occur after some pre-processing on the dataset about to be indexed. Pre-processing can include but is not limited to: stemming, tokenization and case folding, which all make indexing more efficient by categorizing words with the same or similar meaning.

Stemming is the process of grouping words by converting them to their root form, for example rapping, rap and rapper can be connected to the same root word "rap" and queries containing the word rap will probably be related to the document containing these words.

Tokenization is the process of dividing a stream of text information into tokens, which can be words, concepts, symbols, or other significant components. It is a crucial stage in the IR process. Exploring the words in a sentence is the goal of tokenization. Tokenization facilitates word-by-word breakdown of the textual content.

Case folding includes changing the case of all alphabetic tokens. Since people will use lowercase regardless of the 'proper' capitalization, it is frequently beneficial to use lowercase only and group lowercase and uppercase words together.

Furthermore, indexing documents may mean that all indexing terms that represent the meaning of a document have the same semantic weight, or term weights may differ based on calculations that estimate the importance of a term weight. The bigger the term weight, the more important a word is to a document and the more likely it is for the document to be matched to a query that includes the term weight or something similar.

Term weighting can be implemented with various methods, and in this paper the author explores a term weighting algorithm that utilises Deep Learning and Machine Learning to define the most useful and accurate term weights, namely DeepCT.

Finally, there are some widely used algorithms in computer science used for term weighting, such as TF-IDF, which is used on the experiments implemented by the author and are presented in subsequent chapters of this dissertation. TF-IDF is used to estimate how important a term is to a collection of documents, utilising information like how frequently a word occurs in a document or collection and the rarity of a word in the bespoke collection.

2.2.2 Query Formulation

The stage of the interactive information access process known as Query Formulation (QF) occurs when a user formulates a question from an information requirement and submits it to an information access system like a search engine (like Google) or a library database (like Factiva or PubMed). The system runs certain calculations to match the query with the papers most likely to be relevant to the inquiry and then provides the information seeker with a ranked list of those documents [4].

One of the difficulties of Query Formulation for IR systems to take into account is that QF usually occurs in the human mind, and that may change dramatically from person to person. Query formulation can be unpredictable and difficult to isolate in detail what the specific user is searching for.

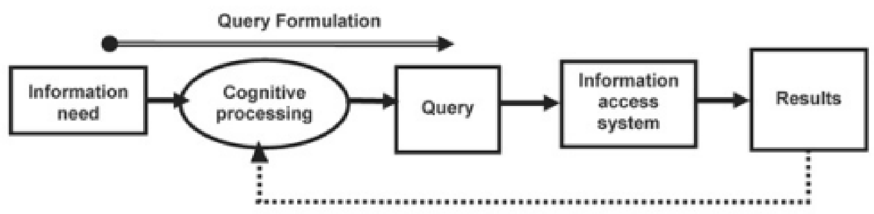


Figure 2. *Query formulation [4]*

2.2.3 Matching

Matching is the process that occurs after pre-processing, indexing and query formulation and includes matching the user query to some relevant documents, using some sophisticated - or simple - algorithm.

There are several types of algorithms IR systems use to match documents and terms to user queries. From simply looking for query words in indexing terms and matching them to more complex techniques that use the power of Artificial Intelligence and Machine Learning to more accurately extract meaning from the query to match more efficiently to documents of the indexed collection.

Subsequently, matching also includes ranking matches from most relevant to least relevant to provide value to the user's search and make sure they are retrieving the information they are looking for.

2.2.4 Re-ranking

Re-ranking can be described as the procedure of estimating some ranking and evaluating if this ranking was accurate for semantic similarity. Additionally, the initial ranking can be altered to a more efficient version.

This process can occur by running experiments manually or with some automated method that receives some hand picked ranking list defined by a human and tests it against the computed generated ranking list and adjusts ranking parameters accordingly.

2.3 IR in everyday life

Information retrieval is, as also stated above, used extensively by basic and advanced users all over the world. Almost any application that searches through large amounts of data collections is using some form of an Information Retrieval system. It becomes obvious when one thinks that the alternative would be searching every word of every element on the searchable collection to match with the user's query, whereas with IR and therefore some form of indexing the search for relevant results would be looking through key words and parts of collection items.

Common uses of information retrieval include but are not limited to:

- Search engines
  - Web search
  - Mobile search
  - Social search
  - Desktop search
- Digital libraries
- Recommendation systems
- Multimedia search
  - Music retrieval
  - Video retrieval
  - Speech retrieval
  - News retrieval

The list above contains only some of IR use cases in everyday life, and it is evident that it is an important field of study in computer science that affects everyone.

Subsequently, the importance of efficient Information retrieval and the numerous use cases led to research to develop more sophisticated systems that utilise modern technologies such as Machine Learning and Deep Learning. On the next chapter, the author provides a brief introduction to this field before presenting algorithms and methodologies that are related to both fields, Information Retrieval and Deep Learning.

3. Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence is the broader term and field of science that refers to "intelligence" exhibited by machines. This field has been rapidly developing in the last few decades, and is now present in numerous applications in software, mobile phones and many aspects of our life. There are several different technologies that fall under the same "umbrella" term that is Artificial Intelligence. Ranging from simple to very complex problem solving, the main - and most promising - characteristic of AI is that the possibilities are endless. In this chapter, the author provides a brief introduction to the reader of AI, Machine Learning, Deep Learning and defines some useful terms for better understanding of the experimental work undertaken for this dissertation.

Additionally, as it is stated below, Artificial Intelligence is a broader field that includes Machine Learning. Similarly Deep Learning is a subset of Machine Learning technologies. This can be visualised in the figure below.

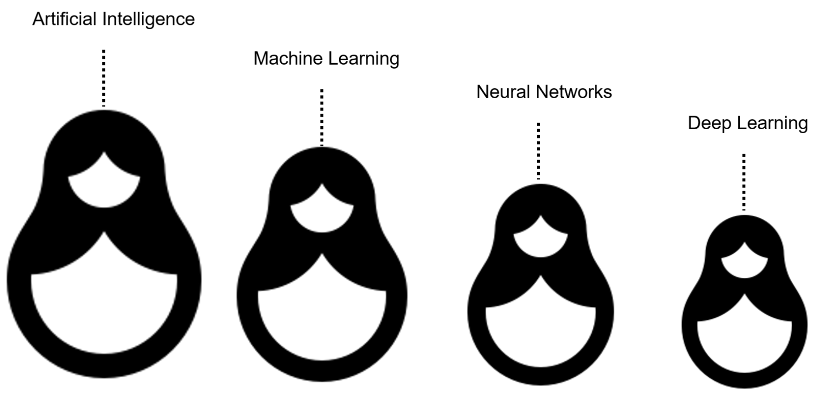


Figure 3. *AI, ML and DL visualised as Russian nesting dolls [5]*

3.1 Artificial Intelligence

The study of "intelligent agents," or any technology that senses its surroundings and acts in a way that maximizes its chances of success in achieving a goal, is what the area of artificial intelligence (AI) research in computer science refers to. When a machine imitates "cognitive" abilities that people typically associate with other human minds, such as "learning" and "problem solving," the phrase "artificial intelligence" is used. [6]

Popular applications of AI include but are not limited to speech recognition, self driving algorithms, image recognition and natural language processing.

3.2 Machine Learning

Frequently, artificial intelligence is attained through the use of machine learning. As one of the founders of machine learning, Arthur Samuel proposed in 1959 that machine learning is a field of study that allows computers to learn without being explicitly programmed. Machine Learning is an approach or subset of Artificial Intelligence that places more emphasis on "learning" than on computer programs. Without a human writing precise instructions into the machine's software, machines utilize complex algorithms to analyze vast volumes of data, find patterns in the data, and make predictions. A system can use machine learning to learn from its errors and enhance its pattern recognition. [7]

There are three main types of Machine Learning [6]:

- *Supervised Learning*, where the system is trained with training data examples where the input and desired output are provided. By using those examples, the system can be trained to provide outputs for provided inputs by using the intelligence acquired by training data. This type of ML is useful for identifying future outcomes such as for stocks or predicting customer actions

- *Unsupervised Learning* is applied to information that lacks historical labels. The "correct answer" is not provided to the system. The showing must be determined by the algorithm. The objective is to investigate the data and identify any internal structure. It can be useful for identifying hidden patterns between items and for providing suggestions.

- *Semi-supervised learning*

- *Reinforcement learning* Through trial and error, the algorithm learns through reinforcement learning which actions result in the biggest rewards. The learner or decision-maker is the agent in this sort of learning, together with the environment and actions, which are all things the learner or decision-maker interacts with (what the agent can do). The goal is for the agent to make decisions that maximize the anticipated benefit over a predetermined period of time. By adhering to a sound policy, the agent will attain the target much more quickly. The best policy is what reinforcement learning aims to learn.

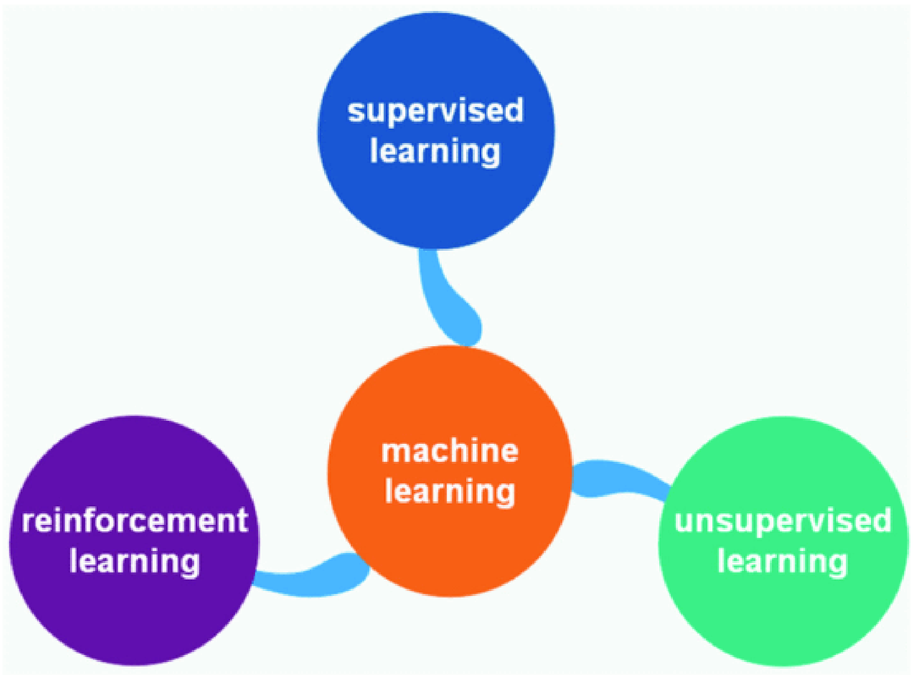


Figure 4. *The components of machine learning [7]*

3.2.1 Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their structure and nomenclature are modeled after the human brain, mirroring the communication between organic neurons. [8]

A node layer of an artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, is connected to others and has a weight and threshold that go along with it. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next tie.

3.3 Deep Learning

Deep learning refers to the use of neural networks and related machine learning algorithms that contain one or more hidden layers.

An observation (such as an image) can be represented in a variety of ways, including as a set of edges, areas of a certain form, or more abstractly as a vector of intensity values per pixel. Some representations (such face recognition or facial expression recognition) are more effective than others at making the learning job simpler. Deep learning holds the potential of replacing manual feature creation with effective unsupervised or semi-supervised feature learning and hierarchical feature extraction methods. [6]

3.4 Pre-trained deep models

In the next chapter, the author introduces the reader to pre-trained deep models, such as BERT, used for text contextualization and more efficient passage retrieval. Pre-trained models have been trained against very large corpora to better understand term importance in context.

Additionally, DeepCT-index which is utilising pre-trained BERT models is presented as it is the used on most of the author's experiments that are explained in this thesis.

4. Introducing BERT and deepCT

In recent years, there has been significant progress in passage understanding methods with the introduction of deep contextualized word representations such as BERT and ELMo [9]. These techniques use sizable text corpora to train a neural language model. Each token is given a representation that depends on both the token and other factors as well. These neural language models have been demonstrated to characterize a word's meaning and syntax, and more significantly, how they differ in multi-language settings.

4.1 BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It is a machine learning technique for Natural Language Processing pre-training. It was developed by Google engineers, more specifically it was developed and published in 2019 by Jacob Devlin and his colleagues [10]. BERT models can significantly help with term or passage importance identifying. They are already being actively used by Google search engine.

To understand how BERT works and how BERT models are created, one needs to understand some basic architecture concepts behind it. A transformer is an attention mechanism that learns the contextual relationships between words (or subwords) in a text and is used by BERT. A Transformer's basic design consists of two independent mechanisms: an encoder that reads the text input and a decoder that generates a task prediction. Only the encoder mechanism is required because BERT's aim is to produce a language model. More detail about the transformers inner workings can be found in Google's paper [10].

The Transformer encoder reads the entire sequence of words at once, in contrast to directional models, which read the text input sequentially (from right to left or left to right). Although it would be more accurate to describe it as non-directional, it is therefore thought of as bidirectional. This feature enables the model to understand the context of a word.

4.2 DeepCT

DeepCT is a **Deep C**ontextualized **T**erm Weighting framework and it is utilised for improved first-stage retrieval models using the contextualized word representations generated by BERT [9].

A BERT-based contextualized word model and a mapping function that calculates terms weights from representations are used to train DeepCT in a supervised manner. Context also affects a word's representation and subsequently its estimated importance. More information can be found on DeepCT's public code repository on Github as it is released with open-source licensing [11].

4.3 DeepCT hands-on and its importance in this thesis

In this dissertation, DeepCT is quite central to the experiments executed. The experiments are explained in great detail in the corresponding chapter below and in this section the reader can learn more about how the DeepCT command line executable works **practically**.

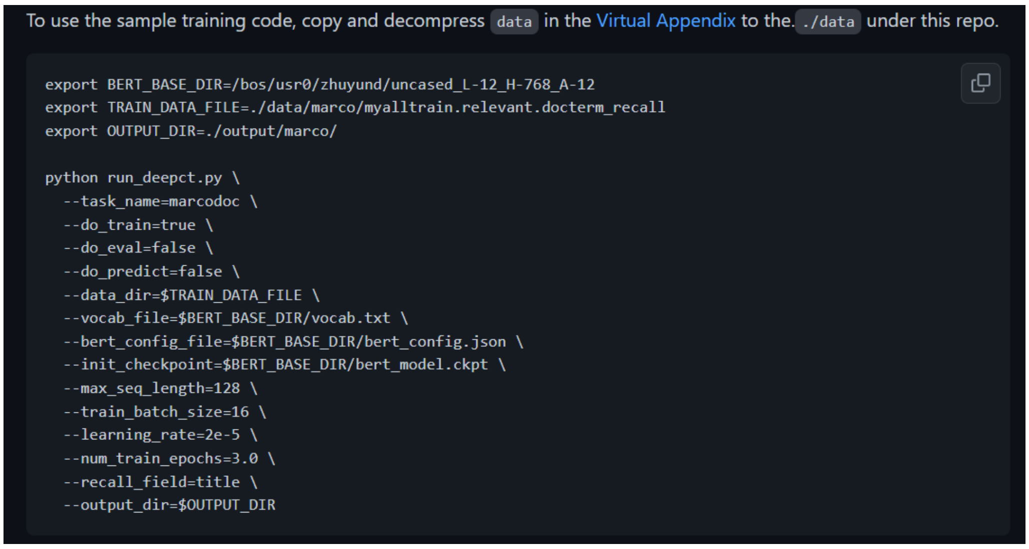
There are three tasks the DeepCT executable can perform and these are incremental procedures that lead to the main end product of this program that is contextually-accurate term weights for a document collection that can be used to create an index, so that this index then can be used for query/term retrieval. So these procedures are:

- *Step one: Training* - The program is fed with data from the document collection which helps it better understand the context of this collection and the terms contextual value to better estimate term weights. The program reads and trains itself by creating a word model better suited to this data collection. It is not done from scratch, a pre-trained BERT model is used as a starting point to make the process more efficient.

- *Step two: Prediction* - After the training is completed and the customized word model is built, the executable can be used to infer term weights for the document collection. The model from the previous step one is used and DeepCT also requires a file containing all the documents and their contents to run. The end result is a file containing floating-point term weights.

- *Step three: Index weights* - The executable converts the term weights calculated in the previous step (prediction) to a form better suited for them to be fed into software like Anserini, Pyserini to create an index. The term weights are converted to integers from their previous format that was floating-point numbers.

The steps above are followed - sometimes without training, listed in step 1 - in this thesis to reach the point where an index is generated through DeepCT. These indexes generated with DeepCT, are later compared to indexes created directly with Pyserini without the contribution of a Machine Learning powered tool like DeepCT.



To use the sample training code, copy and decompress `data` in the Virtual Appendix to the `./data` under this repo.

```
export BERT_BASE_DIR=/bos/usr0/zhuyund/uncased_L-12_H-768_A-12
export TRAIN_DATA_FILE=./data/marco/myalltrain.relevant.docterm_recall
export OUTPUT_DIR=./output/marco/

python run_deepct.py \
  --task_name=marcodoc \
  --do_train=true \
  --do_eval=false \
  --do_predict=false \
  --data_dir=$TRAIN_DATA_FILE \
  --vocab_file=$BERT_BASE_DIR/vocab.txt \
  --bert_config_file=$BERT_BASE_DIR/bert_config.json \
  --init_checkpoint=$BERT_BASE_DIR/bert_model.ckpt \
  --max_seq_length=128 \
  --train_batch_size=16 \
  --learning_rate=2e-5 \
  --num_train_epochs=3.0 \
  --recall_field=title \
  --output_dir=$OUTPUT_DIR
```

Figure 5. *A part of the documentation instructing on how to run the executable, found in DeepCT's github repo [11]*

It should also be noted here that as the author explains in the subsequent chapters, the process of creating the help files that the DeepCT executable needs to work has also been fine tuned to produce better end results.

5. Structure and background of the data set used

In this chapter, the data set used for subsequent runs and experiments is explained in detail by documenting the structure and the background of the data set documents.

5.1 CLEF-IP

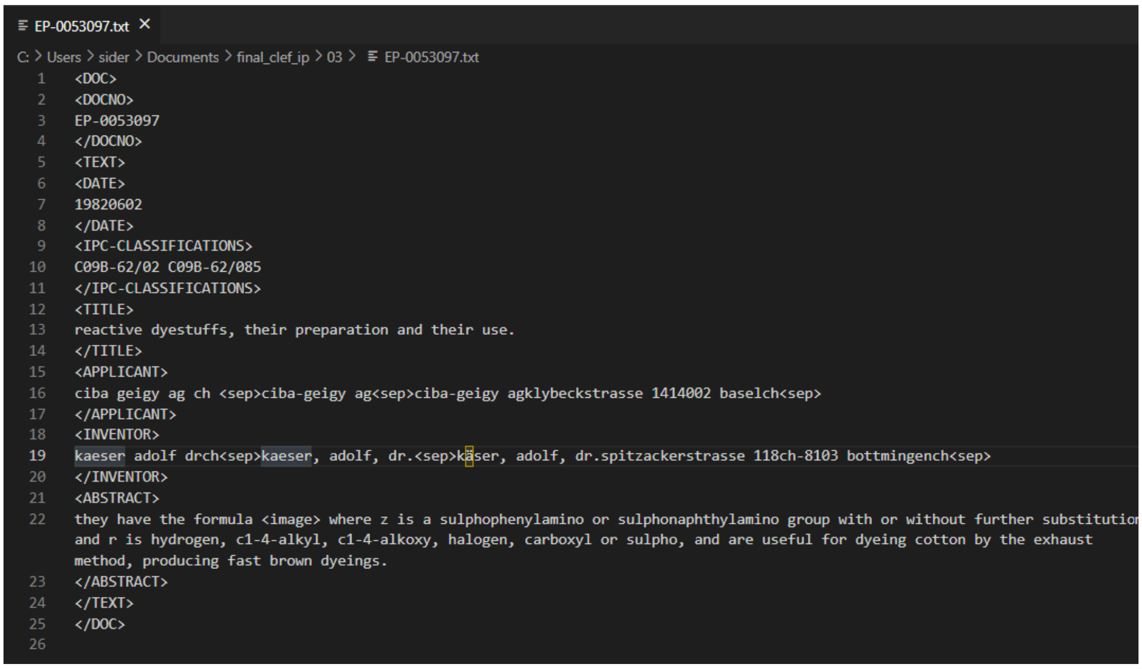
The dataset is provided by CLEF-IP which was an evaluation lab that was operated by Vienna University of Technology from 2009 to 2013. Its purpose was two-fold: to promote cross-language patent retrieval research and to make a big and clean patent-related data collection available in the three primary European languages for experimentation, with the languages being English, German and French [12]. The project was supported by the PROMISE Network of Excellence (co-funded by the 7th Framework Programme of the European Commission).

The single task that CLEF-IP organized in its first year was a text retrieval exercise that was modeled after the "Search for Prior Art" carried out by professionals at patent offices. The types of CLEF-IP tasks grew during the ensuing years to include (formal) structure recognition, patent text classification, and retrieval and classification of patent images. The test collection was increased to account for the new tasks.

CLEF-IP provided datasets for teams to come and compete in Information Retrieval techniques and methodologies to discover the most efficient ones and drive forward this specific field of IR. Each dataset included patent documents. The concept of a patent can be defined as follows: "A patent is a set of exclusive legal rights for the use and exploitation of an invention in exchange for its public disclosure". A patent is a set of legal documents by which a governing authority, through its patent office, grants a set of exclusive rights for exploitation [of the invention] for a limited number of years, usually 20 [13].

5.2 Data set figures

The CLEF-IP data set used in this dissertation contained patent documents in English, French and German language. It was the collection distributed by CLEF-IP lab in 2011. Its size is approximately 16gb (gigabytes), it contains a total of 1,768,641 files which are patent documents and these are divided into 100 folders. The patent documents follow TREC XML format, an example of this format can be seen in the figure below.


EP-0053097.txt ×
C: > Users > sider > Documents > final_clef_ip > 03 > EP-0053097.txt

```
<DOC>
<DOCNO>
EP-0053097
</DOCNO>
<TEXT>
<DATE>
19820602
</DATE>
<IPC-CLASSIFICATIONS>
C09B-62/02 C09B-62/085
</IPC-CLASSIFICATIONS>
<TITLE>
reactive dyestuffs, their preparation and their use.
</TITLE>
<APPLICANT>
ciba geigy ag ch <sep>ciba-geigy ag<sep>ciba-geigy agklybeckstrasse 1414002 baselch<sep>
</APPLICANT>
<INVENTOR>
kaeser adolf drch<sep>kaeser, adolf, dr.<sep>käser, adolf, dr.spitzackerstrasse 118ch-8103 bottmingench<sep>
</INVENTOR>
<ABSTRACT>
they have the formula <image> where z is a sulphophenylamino or sulphonaphthylamino group with or without further substitutio
and r is hydrogen, c1-4-alkyl, c1-4-alkoxy, halogen, carboxyl or sulpho, and are useful for dyeing cotton by the exhaust
method, producing fast brown dyeings.
</ABSTRACT>
</TEXT>
</DOC>
```

Figure 6. *Patent document example [7]*

In this format the beginning and ending of the document are denoted by `<DOC>` and `</DOC>` respectively. `DOCNO` opening and closing tags contain the document's unique identifier. `<TEXT>` and `</TEXT>` represent the start and end of documents text fields. The rest of fields are self-explanatory and represent different fields of the patent document like `ABSTRACT` tags contain the patent's abstract text.

Additionally, a set of 300 *topics*/queries was used to test against the indexes created. The results were then compared to a *QRELS* files to determine the effectiveness of the query results. A QRELS file includes the patent documents that *should* be returned when running the aforementioned queries/topics.

6. Experimental procedures

For the purposes of this dissertation, the author developed a large number of scripts and programs, that were executed numerous times for the experiments needed to determine the most efficient algorithmic method for patent retrieval.

6.1 Progressive experimenting

Firstly, the author analyzed DeepCT use and how it's predictions could be further improved. One stage that could be improved is the training of the word model. To work with DeepCT, it must be provided with a file containing all document data and important words of each document in a single file, with each line representing a single document in a format that can be seen below:

```
{"query": "channel supply degradingexcess zone pipe dissipation
   16 transfer located.the heat increased rate steam range.the
   vapor cross initially", "term_recall": {"channel": 1, "supply
   ": 1, "degradingexcess": 1, "zone": 1, "pipe": 1, "
   dissipation": 1, "16": 1, "transfer": 1, "located.the": 1, "
   heat": 1, "increased": 1, "rate": 1, "steam": 1, "range.the":
    1, "vapor": 1, "cross": 1, "initially": 1}, "doc": {"
   position": "1", "id": "EP-0000001","title": "The invention
   relates to a thermal heat pump consisting of a heat pipe (11)
    in which the steam passage (16) between the heat transfer
   zone for heat supply and the heat transfer zone for heat
   dissipation steam channel (16) initially changing over its
   length, the flow rate of the steam initially increasing and
   then degradingExcess cross-section and wherein in the region
   of the increased vapor rate, another heat transfer zone with
   heat supply or removal is located.The increased steam
   velocity can be in the column range or in the transition
   range.The cross-sectional change of the vapor channel (16) in
    the heat pipe (11) between the two outer heat transfer zones
    is advantageously effected by a displacement body (13) with
   a certain surface contour." }}
```

Listing 6.1. Code block: Docterm_recall helper file

We will from now on refer to this helper file as the **docterm_recall** file as per the example data file provided in DeepCT's repository.

At this stage, several observations and experiments were made. There were many questions on how to build the file for this patent document data collection. Questions such as what the *query* should be, what the *title* should be, which words can be included in the *term_recall* object and if their value needs to be "1" or if it can be weighted using a custom weighting procedure.

These questions where mainly answered by looking through the source code of DeepCT. The author deducted that the *title* field should contain text that briefly describes the patent and for this purpose, the description and the abstract were chosen as the most suitable fields. As the limit for the *title* field size was 128 characters, the first 128 characters of the abstract or description were selected for each document, unless the whole content fitted. Additionally, the *query* field was filled with the most important words in each patent document. The importance of each word was determined by the TF-IDF score of each word. After each word's TF-IDF score was calculated for each corresponding document, an average score was found and each word that had a score larger than the average was used to fill the *query* field for each document. Finally, the *term_recall* field was filled with the words used in the *query* and their TF-IDF scores as values.

A term weighting algorithm was selected for estimating the importance of each word in any patent document. The algorithm that was picked was TF-IDF (Term Frequency - Inverse Document Frequency). In the corresponding chapter of this dissertation, the author explains how TF-IDF was chosen over other algorithms.

As for the technical aspect of this process, to reach this point of using the TF-IDF scores of a specific field from every patent to create the *docterm_recall* file, starting from the XML patent document files explained above and visualised in Figure 6, a collection of scripts needed to be created and the process needed to be streamlined. Specifically, a Python script was developed, namely *create-db.py* to scan through all the documents, select the needed fields and create an SQLite database of all the patents for a more convenient usage afterwards. Subsequently, the author created a Python script (*pyserini_tfIdf.py*) that uses Pyserini functions and the database created earlier, to calculate the TF-IDF scores of each word for every document. These scores were then used to create the *docterm_recall* file using the words for each patent that had a score larger than the average, as explained briefly above. It should be noted that firstly the author create a Python script with a fully custom TF-IDF calculation, however the execution time was very large and the Pyserini TF-IDF functionalities were preferred instead.

Simultaneously, as the patent documents contained various fields such as *Abstract*, *Claims*, *Description*, *Applicant* the author set out to find the most efficient fields, or combination of fields, to use from the patent documents, keeping in mind two objectives: the most accurate representative of the patent text and substance should be picked and the selection must be smaller in size than the whole document text size. It is easily inferred that the most accurate representation of a patent's text is the whole text included in it, but this is highly ineffective in terms of computational power needed and it is impossible to use with some tools as there are limitations in the input size allowed.

After creating separate indexes with Pyserini, that were built by using different field input variations, these indexes were compared side by side by running identical queries and comparing the results. This process is further documented in the corresponding chapter of this thesis (see more in Chapter 7). The field that was selected as the most suitable for the experiments was the **Description** field.

The technical process of creating indexes using only Pyserini was the following. A python script named *pythonCreateInvertedIndexHelpFile.py* that uses Pyserini functions and the database created earlier with the *create-db.py* script to create a helper file that from now on will be referred to as the *jsonl file* was developed and used. This helper file's structure will be further explained in the next sub-chapter and it is needed by Pyserini to create the inverted index needed for these experiments. The author created indexes directly with Pyserini as well as *combined* with DeepCT preprocessing and Pyserini subsequently and all of those variations are used comparatively as it will be further explained earlier. The indexes were created by executing Pyserini command line tools with the usage of the *jsonl file*.

Nevertheless, for a more well rounded opinion and taking into account the fact that the *description* field seems to be missing from a number of patent documents - as well as the claims - the *abstract* field was also used to conduct experiments. These two experiments helped the author make more safe assumptions about the effectiveness of the methods and techniques used for effective Information Retrieval.

6.2 Helper files structure

There were several files created to help progress the experiments and use DeepCT and Pyserini effectively. These files will be referred to by the author as helper files moving forward. One file that was already mentioned is the *docterm_recall* file which is used for the training of the word model with DeepCT and an example is listed above in Listing 6.1.

Similarly, for the prediction stage of DeepCT, the DeepCT executable needs to be provided with a helper file containing lines for each document with each line carrying the IDs of each patent document and the text included in the selected field from previous steps. For example, the author used the **description** field as stated above, which was selected as the most effective and representative field for each patent document. The ID and text of each line were separated by a tab character and the extension of the document was *tsv*. This file will from now on be referred to as *edit.tsv* file. An example is listed below.

```
EP-0000001  The invention relates to a thermal heat pump
   consisting of a heat pipe (11) in which the steam passage (16)
    between the heat transfer zone for heat supply and the heat
   transfer zone for heat dissipation steam channel (16)
   initially changing over its length, the flow rate of the
   steam initially increasing and then degradingExcess cross-
   section and wherein in the region of the increased vapor rate,
    another heat transfer zone with heat supply or removal is
   located.The increased steam velocity can be in the column
   range or in the transition range.The cross-sectional change
   of the vapor channel (16) in the heat pipe (11) between the
   two outer heat transfer zones is advantageously effected by a
    displacement body (13) with a certain surface contour.
EP-0000009  1. a process for the oxidation of quinine to
   quininone and quinidinone by means of an oppenauer type
   oxidation reaction, characterized in that said reaction is
   carried out with a basic reagent known as "ketyl" resulting
   from the direct action, on a diphenylketone, of an alcaline
   metal in a solvent medium.
EP-0000010  1. a device for the temporary storage of coins of
   various values before their further conveying into the
   storage means of an automatic coin-freed machine for goods
   vending or services and comprising a coin slot, characterised
    in that a conveying screw (5) with a selector lever (19) is
   connected in series with the coin slot (12), the selector
   lever being selectively pivotal from a middle position in
   which the inserted coins (m) of a payment operation are
   retained at one end of the conveying screw (5), to either a
   return position in which the coins (m) are released for
   return, or an encashment position at which the coins (m) are
   transferred to the other end of the conveying screw (5), from
    which the coins (m) are delivered to down-stream storage
   stacks.
```

Listing 6.2. Code block: Prediction edit.tsv helper file

Another helper file that was mentioned briefly previously is the *jsonl file* needed by Pyserini to create the inverted index needed for the sake of our experiments. The format of this is file is JSON-Line which translates to a json line for every patent document and all the lines are included on the same file. That means that the file contains as many lines as the documents that are included on the patent document collection. A part of the file as an example is displayed below on Listing 6.3

Each line contains a JSON object containing the document identifier and the contents that are provided to Pyserini, in this example it is the abstract field of each patent, the description was also used for the experiments.

```
{"id": "EP-0000007", "contents":"The invention relates to a
   method for the polymerization of alphaolefins as well as the
   soli catalytic complexes of use for this polymerization and a
    method for their preparation. The solid catalytic complexes
   are prepared in fact react between: (1) at least one compound
    selected from organic oxygen compounds and the halogenated
   compounds of magnes SIUM; (2) at least one compound selected
   from organic oxygen compounds and the halogenated compounds
   of titanium; (3) At least one aluminum halide. The aluminum
   halide is selected from the organoaluminous chlorides of
   general formula A1RNC13-N wherein R is an alkyl radical
   comprising at least 4 carbon atoms and n is a number such as
   1 < N < 2. The polymerization method allows To obtain, with
   very high catalytic activities, polyolefins whose percentage
   of fine particles is reduced and whose average particle size
   is higher."}
{"id": "EP-0000008", "contents":"a lifting device for lifting
   and transferring a load, said device having a rigid carrier
   arm (1) supporting load receiving means (2), and transfer
   means (13, 12, 10, 9a, 9b, 8a, 8b, 7a, 7b, 6a, 6b, 5a, 5b)
   for lateral movement of said carrier arm. in order to achieve
    said movement of the carrier arm said transfer means
   includes two symmetrically disposed transfer assemblies each
   having a link member (5a; 5b) pivotably connected at a
   separate end of the carrier arm (1) and a lever (6a; 6b)
   pivotably connected at one end to the associated link member,
    said lever being pivotably supported at its other end by a
   pivot shaft (7a, 7b). said transfer means also includes drive
    means (13, 12) and transmission means (9a, 9b, 8a, 8b) for
   simultaneous pivoting the levers in opposite directions."}
```

```
{"id": "EP-0000009", "contents":"1. a process for the oxidation
   of quinine to quininone and quinidinone by means of an
   oppenauer type oxidation reaction, characterized in that said
    reaction is carried out with a basic reagent known as ketyl
   resulting from the direct action, on a diphenylketone, of an
   alcaline metal in a solvent medium."}
```

Listing 6.3. Code block: Pyserini jsonl helper file

6.3 Technologies and tools used

Regarding the technologies, the tools or the technology stack used in this project, the most significant were the ones needed for index creation and Information Retrieval, namely DeepCT and Pyserini that have already been explained in detail earlier in this dissertation. DeepCT also uses BERT models to function.

Additionally, for the custom scripts that were developed, the language of choice was Python and more specifically, Python 3. Pyserini and DeepCT are also powered by Python and have different packages and versioned dependencies that are needed for them to function. A difficulty that occurred with the different dependencies was that these pieces of software needed different versions of Tensorflow to work. These different versions of Tensorflow also required different versions of Python which made switching back and forth difficult. For this reason, Conda was used as a package and environment system, which allowed the author to create and store two separate environments with different Python and packages versions. These two environments can be easily installed on a new system and can be switched to and from more conveniently.

Furthermore, the script that creates a database storing all documents and their selected fields words, is utilising SQLite which is a simple library for databases based on the SQL engine that stores databases in the user's file system.

Finally, the experiments, code and environments are all uploaded to a public git repository stored in Github. This allows for easy versioning and work between different systems.

6.4 Limitations of the technologies

One important limitation that was already mentioned is that the size of the *title* field on the docterm_recall helper file should not exceed 128 characters that is needed for the DeepCT training step. The parameter *max_seq_length* in thee source code defines the maximum number of characters that can be be processed for every document by DeepCT.

As the patent documents contain various fields such as Abstract, Description, Claims, Applicant some of which contain duplicate informations, for example Title and Description and with the limitation of 128 the characters needed for representation of each document for DeepCT training, the most valuable field in terms of information density needs to be selected and extracted. For this purpose, the author executed some experiments to find the most suitable field for this process. These experiments are further explained and presented on the next chapter.

Furthermore, it must be noted that due to the sheer size of the document collection that was used, the computational capacity as well as the memory capacity needed to be significant. In order for a computer to be able to handle this large amount of data, an enormous amount of data needed to be stored on the RAM memory, especially for a personal computer as the one that was used for these experiments. Similarly, the amount of computational power needed is non negligible as calculations between large floating point numbers and matrices needed to be executed in the background.

The dataset contained approximately 1.7 million documents and for the training and prediction - more importantly for the training - executed by DeepCT the execution time needed was significantly large. For example training during the experiments for this dissertation lasted many days, sometimes more than a week. It is very probable that using a more powerful computer could have resulted in better execution times.

The biggest challenge in running these experiments was the computational power & simply the amount of time needed to complete deep learning reinforced training with DeepCT. Some of the test runs were run against a part of the collection, mainly on a one-fifth-sized part of the data set. This was done in order to save time and these experiments should be ran against the whole collection to further test the accuracy of the results and conclusion of this dissertation.

As an example on how large the execution times of the experiments that included DeepCT, a complete training of the word model with the following parameters: 3 epochs, 50,000 documents, 0.02 learning rate and a batch size of 16 took approximately 30 hours on the author's computer with the execution averaging 1.5 processed documents each second.

The computer that was used has the following specs: Intel Core i5-10400 2.9 GHz, 16GB RAM, Ubuntu on Windows WSL2.

It should also be noted here that while the dataset contained the amount of documents mentioned earlier, not all of them included every data field. On Chapter 7, "Fields selection", there is a single field that was selected to be used as the representative text of each patent document. This field was not contained on every document, but only on a part of it, which makes the participating documents count smaller.

7. Fields selection

In order to select the most valuable field, in terms of information density, the author executed some test runs. These included creating indexes containing only some fields with the use of Pyserini, then running queries on these indexes and evaluating the results with a Qrels file. The queries and the Qrels evaluation file are also used in subsquent experiments in the following chapters.

The evaluation produced by using trec_eval various metrics which can be seen below on Table 1.

Table 1. *Field selection experiment results*

#	**Field**	**GM_MAP**	**Num_rel_-ret**	**P_100**	**P_1000**	**iprec_at_-recall_1.00**
1	**all fields**	**0.0132**	**1206**	**0.0213**	**0.0040**	**0.0111**
2	abstract	0.0060	907	0.0148	0.0030	0.0055
3	claims	0.0049	961	0.0151	0.0032	0.0049
4	**description**	**0.0088**	**1076**	**0.0190**	**0.0036**	0**.0068**
5	title, abstract & applicant	0.0068	963	0.0157	0.0032	0.0050

The metrics above have the following meanings: GM_MAP: Average Precision. Geometric Mean, q_score=log(MAX(map,.00001))

Num rel ret: Total number of relevant documents retrieved over all queries

P100: Precision after 100 docs retrieved

P1000: Precision after 1000 docs retrieved

ircl_prn.1.00: Interpolated Recall - Precision Averages at 1.00 recall

The author inferred looking at the results, "all fields" and "description" produced the best metrics. This seemed like a reasonable outcome if it was considered that using "all fields", the index included every possible word contained in every document and this was naturally bound to produce the best results. But the earlier mentioned character limit needed for using DeepCT training and more importantly the fact that this wouldn't be reasonable as it was the whole content and not a represantation of the patent, made this option unsuitable for this thesis's experiments. On the other hand, "Description" was a more suitable field that was much more possible to contain less than 128 characters and produced better results than the rest of the fields, excluding "all fields".

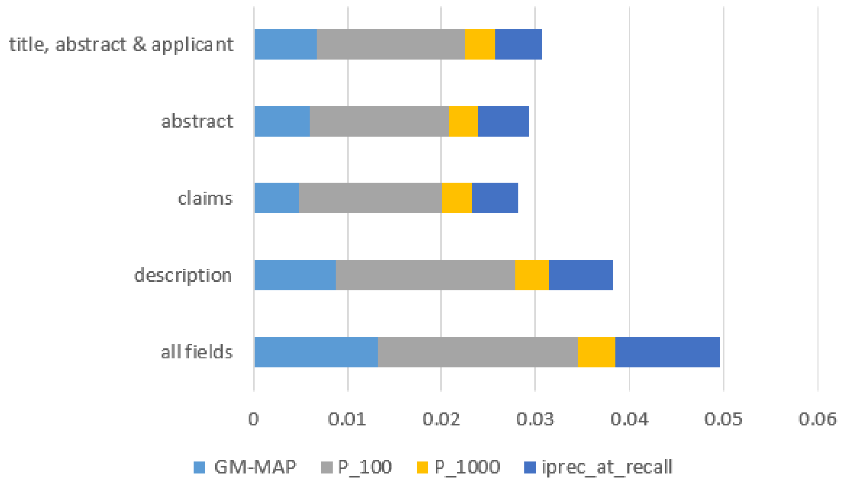


Figure 7. *Evaluation performance per field*

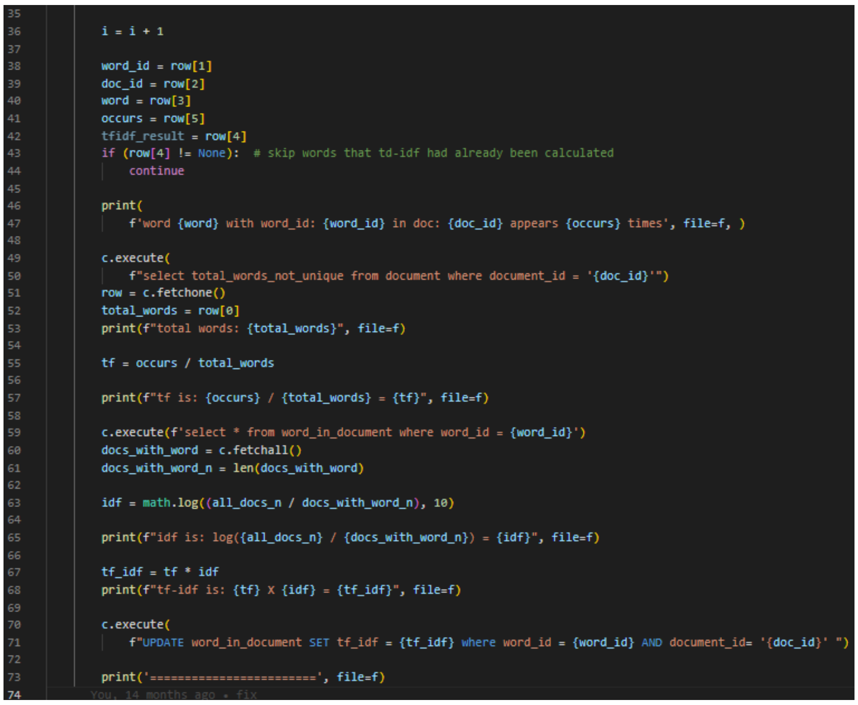
As a result, **description** was the better suited field to represent each document. However, the data set documents did not all contain this field and the final amount of documents that did was approximately 240,000.

8. Weights creation algorithm

As it was explained in previous chapters, before running DeepCT training, the author used a term weights creation algorithm. This was done in order to be used with deepCT training and to enhance deepCT's results with the term weight from this pre-processing procedure.

Initially, two algorithms were considered, namely TF-IDF and BM-25. TF-IDF, which is short for Term Frequency / Inverted Document Frequency is used for calculating term scores or weights within a document. Whereas, BM-25 is used mainly for document ranking *in relation* to a search query. It uses an algorithm similar to TF-IDF internally but it is more valuable in a different context than the one it is designed for.

Subsequently, considering that TF-IDF is a very well known and widely used algorithm in the field of Information Retrieval and Computer Science in general and that it is a simple and intuitive algorithm that is quite easy to implement on demand and finally there are various implementations available in the open source space, this was the algorithm that was selected for this purpose. It should be noted here that Pyserini already provides developers with APIs (Application Programming Interfaces) to calculate the TF-IDF score of documents.

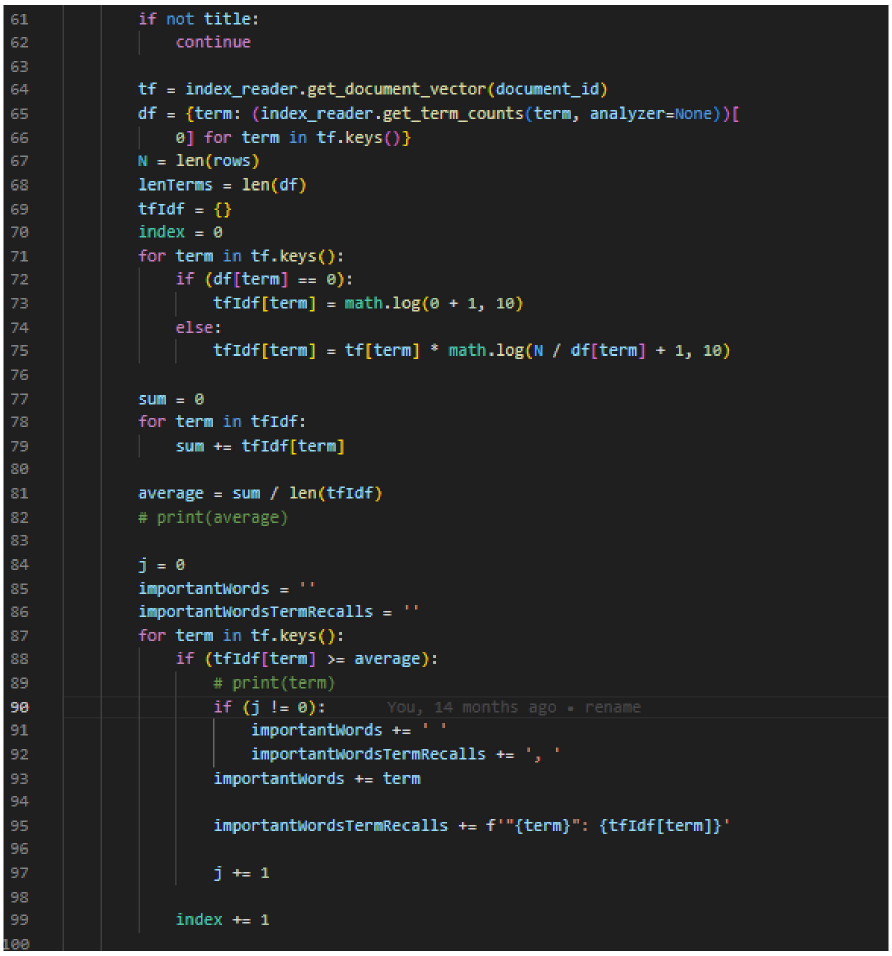


```python

        i = i + 1

        word_id = row[1]
        doc_id = row[2]
        word = row[3]
        occurs = row[5]
        tfidf_result = row[4]
        if (row[4] != None):  # skip words that td-idf had already been calculated
            continue

        print(
            f'word {word} with word_id: {word_id} in doc: {doc_id} appears {occurs} times', file=f, )

        c.execute(
            f"select total_words_not_unique from document where document_id = '{doc_id}'")
        row = c.fetchone()
        total_words = row[0]
        print(f"total words: {total_words}", file=f)

        tf = occurs / total_words

        print(f"tf is: {occurs} / {total_words} = {tf}", file=f)

        c.execute(f'select * from word_in_document where word_id = {word_id}')
        docs_with_word = c.fetchall()
        docs_with_word_n = len(docs_with_word)

        idf = math.log((all_docs_n / docs_with_word_n), 10)

        print(f"idf is: log({all_docs_n} / {docs_with_word_n}) = {idf}", file=f)

        tf_idf = tf * idf
        print(f"tf-idf is: {tf} X {idf} = {tf_idf}", file=f)

        c.execute(
            f"UPDATE word_in_document SET tf_idf = {tf_idf} where word_id = {word_id} AND document_id= '{doc_id}' ")

        print('======================', file=f)
      You, 14 months ago • fix
```

Figure 8. *Custom TF-IDF implementation*

At first, the author implemented TF-IDF in a custom Python script leveraging the use of an SQLite database, part of which can be seen in Figure 8. Although this script produced the intended results, the execution time needed for TF-IDF scores estimation in this large dataset, was immense and after comparison with Pyserini's TF-IDF implementation, it was decided that the latter was the most suitable tool for this task.

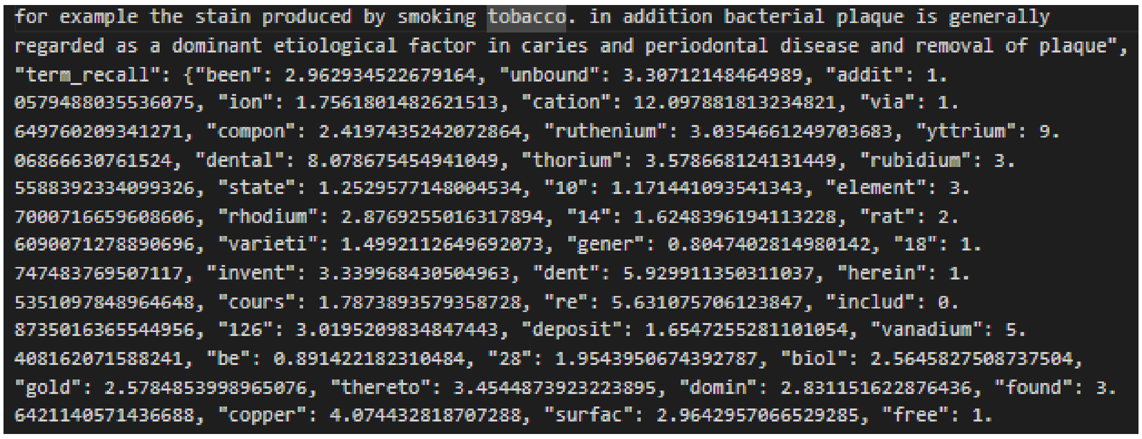
As a result, Pyserini's TF-IDF implementation was used. In addition to this calculation, to further enhance the final product and to make sure only important words were included - keeping in mind the 128 character limit of DeepCT - another distinction was made. It was decided that after calculating all words' TF-IDF score and the average score for each document word scores, only the words that had a TF-IDF score larger than the average were kept and the rest of the words were discarded. Part of this implementation can be seen in Figure 9. This process requires for a Pyserini Inverted Index to be created, to then be used to get a document vector and to estimate TF-IDF scores of each word in the context of the document it belongs to.



```
        if not title:
            continue

        tf = index_reader.get_document_vector(document_id)
        df = {term: (index_reader.get_term_counts(term, analyzer=None))[
            0] for term in tf.keys()}
        N = len(rows)
        lenTerms = len(df)
        tfIdf = {}
        index = 0
        for term in tf.keys():
            if (df[term] == 0):
                tfIdf[term] = math.log(0 + 1, 10)
            else:
                tfIdf[term] = tf[term] * math.log(N / df[term] + 1, 10)

        sum = 0
        for term in tfIdf:
            sum += tfIdf[term]

        average = sum / len(tfIdf)
        # print(average)

        j = 0
        importantWords = ''
        importantWordsTermRecalls = ''
        for term in tf.keys():
            if (tfIdf[term] >= average):
                # print(term)
                if (j != 0):       You, 14 months ago • rename
                    importantWords += ' '
                    importantWordsTermRecalls += ', '
                importantWords += term

                importantWordsTermRecalls += f'"{term}": {tfIdf[term]}'

                j += 1

            index += 1

```

Figure 9. *TF-IDF calculation & preservation*

Finally, following the calculation of the TF-IDF scores of each word in the context of each document, the selected words were used to create the aforementioned docterm_recall helper file, that is then used for DeepCT's training procedure. Additionally, the TF-IDF scores help reinforce DeepCT's deep learning algorithm and - in theory - produce more accurate results. This can be better visualised by looking at Figure 10, where the floating point numbers that can be seen correspond to the TF-IDF scores calculated earlier.



```
for example the stain produced by smoking tobacco. in addition bacterial plaque is generally
regarded as a dominant etiological factor in caries and periodontal disease and removal of plaque",
"term_recall": {"been": 2.962934522679164, "unbound": 3.30712148464989, "addit": 1.
0579488035536075, "ion": 1.7561801482621513, "cation": 12.097881813234821, "via": 1.
649760209341271, "compon": 2.4197435242072864, "ruthenium": 3.0354661249703683, "yttrium": 9.
06866630761524, "dental": 8.078675454941049, "thorium": 3.578668124131449, "rubidium": 3.
5588392334099326, "state": 1.2529577148004534, "10": 1.171441093541343, "element": 3.
7000716659608606, "rhodium": 2.8769255016317894, "14": 1.6248396194113228, "rat": 2.
6090071278890696, "varieti": 1.4992112649692073, "gener": 0.8047402814980142, "18": 1.
747483769507117, "invent": 3.339968430504963, "dent": 5.929911350311037, "herein": 1.
5351097848964648, "cours": 1.7873893579358728, "re": 5.631075706123847, "includ": 0.
8735016365544956, "126": 3.0195209834847443, "deposit": 1.6547255281101054, "vanadium": 5.
408162071588241, "be": 0.891422182310484, "28": 1.9543950674392787, "biol": 2.5645827508737504,
"gold": 2.5784853998965076, "thereto": 3.4544873923223895, "domin": 2.831151622876436, "found": 3.
642114057143668, "copper": 4.074432818707288, "surfac": 2.9642957066529285, "free": 1.
```

Figure 10. *TF-IDF scores in docterm_recall helper file*

9. Creating Indices with Pyserini

The indices created for the purposes of this thesis are variable in nature and the procedures to create them differ in the tools used, preprocessing used and fields used but are all ultimately created with Pyserini. Independently of which of the two distinct approaches (which are explained thoroughly in Figure 14) was followed to create the index, the final product (index) was produced through the use of Pyserini. In order to create the indices the author created a jsonl helper file, which is presented with more detail in Chapter 6 and more specifically in Listing 6.3.

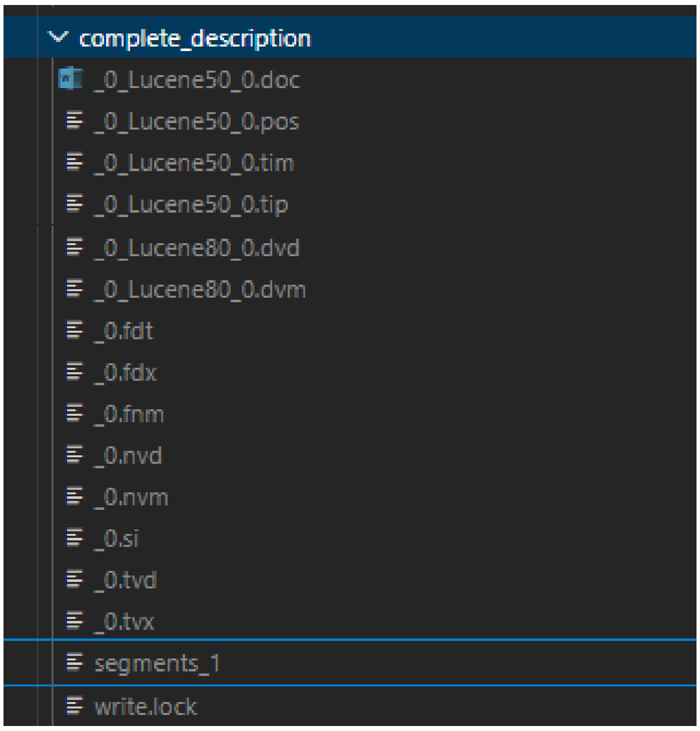
After the creation of the jsonl helper file ensued which includes a json object in every line, where each line contains the document identification number and the contents of the document. The contents of the document in this case are the selected fields of each patent document as described in previous chapters, particularly Chapter 7. This file can be used with a Pyserini command line tool to then build an index. The command line prompt that was used can be seen below.

```
python -m pyserini.index --input jsonl/<EXPERIMENT_NAME> --
   collection JsonCollection --generator
   DefaultLuceneDocumentGenerator --index indexes/<
   EXPERIMENT_NAME> --stemmer=none --threads 1 --storePositions
   --storeDocvectors --storeRaw
```

Listing 9.1. Command: Creating index with Pyserini

The final output of this process is a collection of files that represent the index which was created and the file structure can be seen below in Figure 11.

The Pyserini indexes that were created directly from the document collection - without any pre-processing - can be used the run queries and evaluate results. Similarly, with the experiments that involved DeepCT training, the document collection is processed with DeepCT, which is documented thoroughly in the next Chapter (Chapter 10). After this processing and Deep Learning procedures are run, the end product is a jsonl file that can be used with Pyserini to build indexes.



```
complete_description
    _0_Lucene50_0.doc
    _0_Lucene50_0.pos
    _0_Lucene50_0.tim
    _0_Lucene50_0.tip
    _0_Lucene80_0.dvd
    _0_Lucene80_0.dvm
    _0.fdt
    _0.fdx
    _0.fnm
    _0.nvd
    _0.nvm
    _0.si
    _0.tvd
    _0.tvx
    segments_1
    write.lock
```

Figure 11. *Index files*

Finally, the aforementioned indices that refer to the same data set can respond to search queries and allows us to compare and evaluate results from these different indices to reach a conclusion on which method is more effective.

10. DeepCT indexing

In this chapter, the process of leveraging DeepCT's capabilities to create more efficient and accurate indexes is explained. This was a multi-step process and some of those steps were explained briefly in the previous chapters.

It should be noted here that this process is documented in this dissertation as well as on the author's repository on Github that also contains all custom scripts developed for the purposes of this dissertation.

Firstly, the document collection was processed and a database containing the words and occurences of each word, along with the information of the documents these words were found. The created database was fed to a custom script (*pyserini_tfIdf.py*) to calculate each word's TF-IDF score, accordingly for each document of the collection. These scores were used to create the helper file needed for DeepCT training.

After creating the *docterm_recall* file with the usage of Pyserini and the custom scripts developed by the author, the training of the word model took place. The training was done by executing the following command:

```
python DeepCT-master/run_deepct.py --data_dir=output/<
   EXPERIMENT_NAME>/train.docterm_recall --vocab_file=bert-base-
   uncased/vocab.txt --bert_config_file=bert-base-uncased/
   bert_config.json --init_checkpoint=bert-base-uncased/
   bert_model.ckpt --output_dir=output/<EXPERIMENT_NAME>/train --
   do_train=true --task_name=marcodoc --num_train_epochs=3.0 --
   train_batch_size=16
```

Listing 10.1. Command: Training with DeepCT

The placeholder <*EXPERIMENT_NAME*> corresponds to each experiment, - corresponding to different fields chosen to create the index with - done by the author, as these were grouped in separate folders with the same structure inside. For the purposes of this dissertation, the experiments were named in the following format: <*LIMIT OF DOCUMENTS*>-<*PATENT FIELDS USED*>. The most important experiments include: inf-abstract and inf-description which are the ones that the author basically focuses on.

As for the parameters defined in this command, *data_dir* defines the path of the *docterm_*-

recall helper file that contains the contents of the collection as explained before, *vocab_file* and *bert_config_file* are provided to the pre-trained bert model that is used as a starting point for this training. Additionally, the parameter *init_checkpoint* defines the word model that DeepCT uses to train on, instead of starting from a blank basis, it leverages the knowledge generated on this pre-trained word model. The output directory is defined with the *output_dir* parameter, the *do_train* parameter simply declares that training is the intended action in this context, the *task_name* defines the type of the documents processed with DeepCT to create the pre-trained model. Marcodoc refers to the MS MARCO collection of datasets focused on deep learning in search [14] . Lastly, *num_train_epochs* describes the number of epochs or runs the deep learning model should do when being trained and the *train_batch_size* defines the count of documents that are used in each batch that is fed to the algorithm for the training. These last two parameters were decided after experimentation with the same or smaller datasets, and with guidance from university staff. Epoch means a set of complete available input dataset. Weights are analyzed to create a model after each period. The weights are modified and tested again against the following simulation cycle of the same dataset (called next epoch) [15].

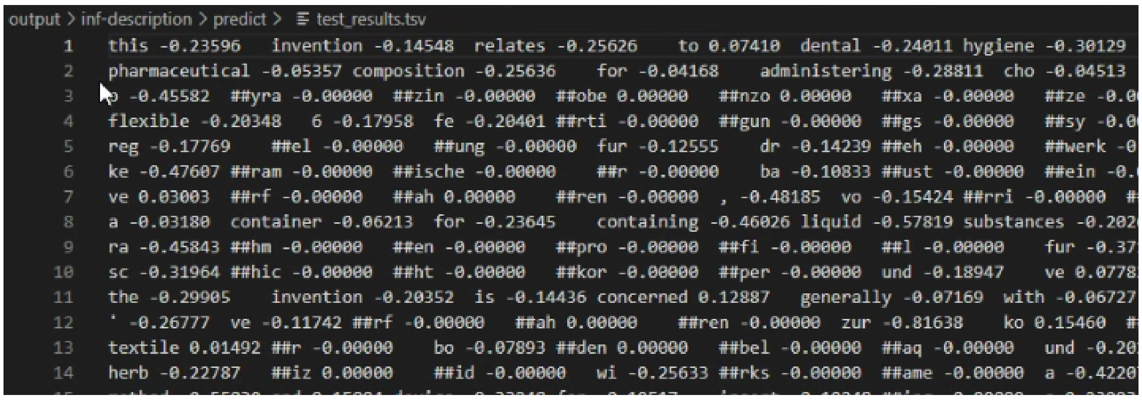
Subsequently, the prediction stage that was based on the trained word model from the previous step was executed using a command line command. Another helper file that is needed by DeepCT, the *edit.tsv* file, was created by using a custom Python script developed by the author as referred to before. The command used for prediction of the term weights can be seen in the block below:

```
python DeepCT-master/run_deepct.py --task_name=marcotsvdoc --
    do_train=false --do_eval=false --do_predict=true --data_dir=
    output/<EXPERIMENT_NAME>/edit.tsv --vocab_file=bert-base-
    uncased/vocab.txt --bert_config_file=bert-base-uncased/
    bert_config.json --init_checkpoint=output/<EXPERIMENT_NAME>/
    train/model.ckpt-0 --max_seq_length=128 --train_batch_size=16
     --learning_rate=2e-5 --num_train_epochs=3.0 --output_dir=
    output/<EXPERIMENT_NAME>/predict
```

Listing 10.2. Command: Predicting term weights with DeepCT

Looking into the parameters used in this command, some of them are already described above, such as *task_name*, *data_dir*, *vocab_file*, *bert_config_file*. It should be noted that the *init_checkpoint* is referring to the checkpoint generated by the training of the word model previously, with the *batch_size* and the number of epochs being identical to the training stage. Finally, some novel technical parameters seen here are *max_seq_length* which defines the maximum number of characters that can be be processed for every document by DeepCT and is set here to the maximum allowed number and *learning_rate* refers to the pace at which the model *learns* new weights for the neural network during training. Large Learning Rate produces a quick learning model with fewer epochs, but it also produces weights that are not ideal. On the other side, a low Learning Rate causes the model to learn slowly, but it also necessitates more training epochs and produces optimal weights [16].

The final product of this process was the calculation of floating-point term weights for every word and document accordingly, it can be seen on Figure 12 below.

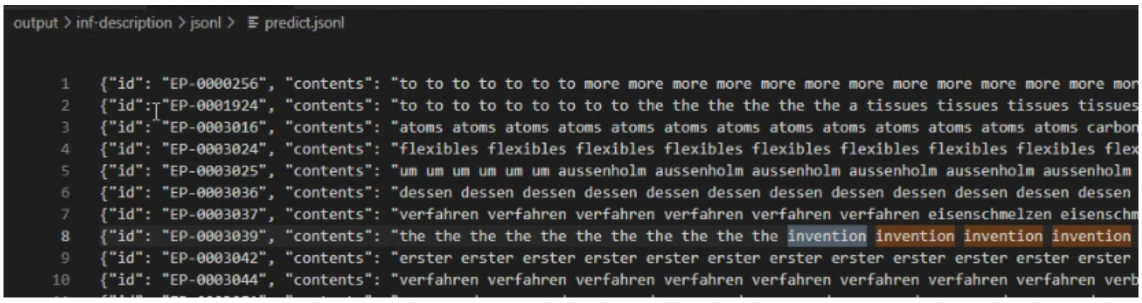


output > inf-description > predict > test_results.tsv

```
this -0.23596   invention -0.14548  relates -0.25626    to 0.07410  dental -0.24011 hygiene -0.30129
pharmaceutical -0.05357 composition -0.25636    for -0.04168    administering -0.28811  cho -0.04513
o -0.45582  ##yra -0.00000  ##zin -0.00000  ##obe 0.00000   ##nzo 0.00000   ##xa -0.00000   ##ze -0.0
flexible -0.20348   6 -0.17958  fe -0.20401 ##rti -0.00000  ##gun -0.00000  ##gs -0.00000   ##sy -0.0
reg -0.17769    ##el -0.00000   ##ung -0.00000  fur -0.12555    dr -0.14239 ##eh -0.00000   ##werk -0
ke -0.47607 ##ram -0.00000  ##ische -0.00000    ##r -0.00000    ba -0.10833 ##ust -0.00000  ##ein -0.
ve 0.03003  ##rf -0.00000   ##ah 0.00000    ##ren -0.00000  , -0.48185  vo -0.15424 ##rri -0.00000  #
a -0.03180  container -0.06213  for -0.23645    containing -0.46026 liquid -0.57819 substances -0.202
ra -0.45843 ##hm -0.00000   ##en -0.00000   ##pro -0.00000  ##fi -0.00000   ##l -0.00000    fur -0.37
sc -0.31964 ##hic -0.00000  ##ht -0.00000   ##kor -0.00000  ##per -0.00000  und -0.18947    ve 0.0778
the -0.29905    invention -0.20352  is -0.14436 concerned 0.12887   generally -0.07169  with -0.06727
' -0.26777  ve -0.11742 ##rf -0.00000   ##ah 0.00000    ##ren -0.00000  zur -0.81638    ko 0.15460  #
textile 0.01492 ##r -0.00000    bo -0.07893 ##den 0.00000   ##bel -0.00000  ##aq -0.00000   und -0.20
herb -0.22787   ##iz 0.00000    ##id -0.00000   wi -0.25633 ##rks -0.00000  ##ame -0.00000  a -0.4220
```

Figure 12. *Floating point term weights*

Thereafter, these weights estimation of the previous step needed to be converted to tf-like index weights. Another command line tool provided by DeepCT was used for this (*bert_-term_sample_to_json.py*). By configuring the script to create an output in json format, a jsonl file was created that was ready for use with Pyserini as explained in the previous chapters, to create an inverted index for testing queries on.



output > inf-description > jsonl > predict.jsonl

```
{"id": "EP-0000256", "contents": "to to to to to to to more more more more more more more more more more more more mor
{"id": "EP-0001924", "contents": "to to to to to to to to to the the the the the the a tissues tissues tissues tissues
{"id": "EP-0003016", "contents": "atoms atoms atoms atoms atoms atoms atoms atoms atoms atoms atoms atoms atoms carbon
{"id": "EP-0003024", "contents": "flexibles flexibles flexibles flexibles flexibles flexibles flexibles flexibles flex
{"id": "EP-0003025", "contents": "um um um um um um aussenholm aussenholm aussenholm aussenholm aussenholm aussenholm
{"id": "EP-0003036", "contents": "dessen dessen dessen dessen dessen dessen dessen dessen dessen dessen dessen dessen
{"id": "EP-0003037", "contents": "verfahren verfahren verfahren verfahren verfahren verfahren eisenschmelzen eisenschm
{"id": "EP-0003039", "contents": "the the the the the the the the the the the invention invention invention invention
{"id": "EP-0003042", "contents": "erster erster erster erster erster erster erster erster erster erster erster erster
{"id": "EP-0003044", "contents": "verfahren verfahren verfahren verfahren verfahren verfahren verfahren verfahren verb
```

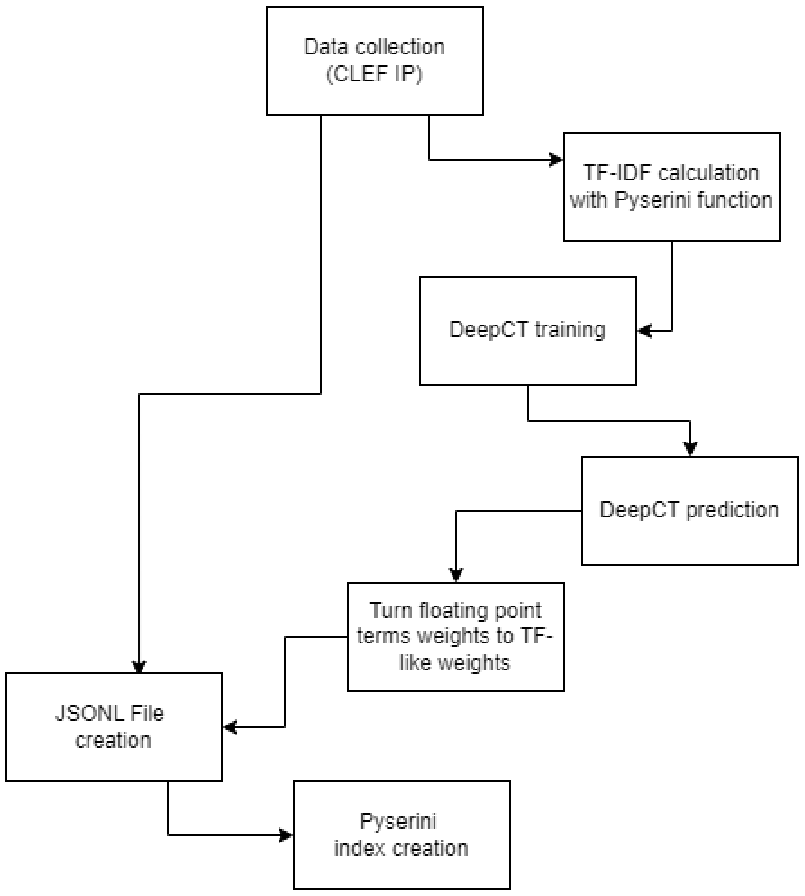
Figure 13. *Term weighting ready to use with Pyserini*

As one can see on Figure 13 above, the jsonl file contains a json object for each line of the patent document collection, while marking each important word by replicating it accordinly to the word's importance or term weight. This is estimated by the deep learning algorithm's prediction capabilities.

Finally, the jsonl file created previously was used with Pyserini command line tools to create the index. This index then can be used to run queries on and subsequently these queries can be tested for their accuracy to find the most effective indexing method. This process is further explained on the next chapters. The Pyserini command needed to create the index is listed in the Listing on the previous chapter.

11. Comparing indices

This chapter is focused on the comparison of the different indices that were created for this dissertation. Some of these indices were created simply by using Pyserini on the data collection that is mentioned throughout this documents.



Data collection (CLEF IP)
TF-IDF calculation with Pyserini function
DeepCT training
DeepCT prediction
Turn floating point terms weights to TF-like weights
JSONL File creation
Pyserini index creation

Figure 14. *Different approaches used*

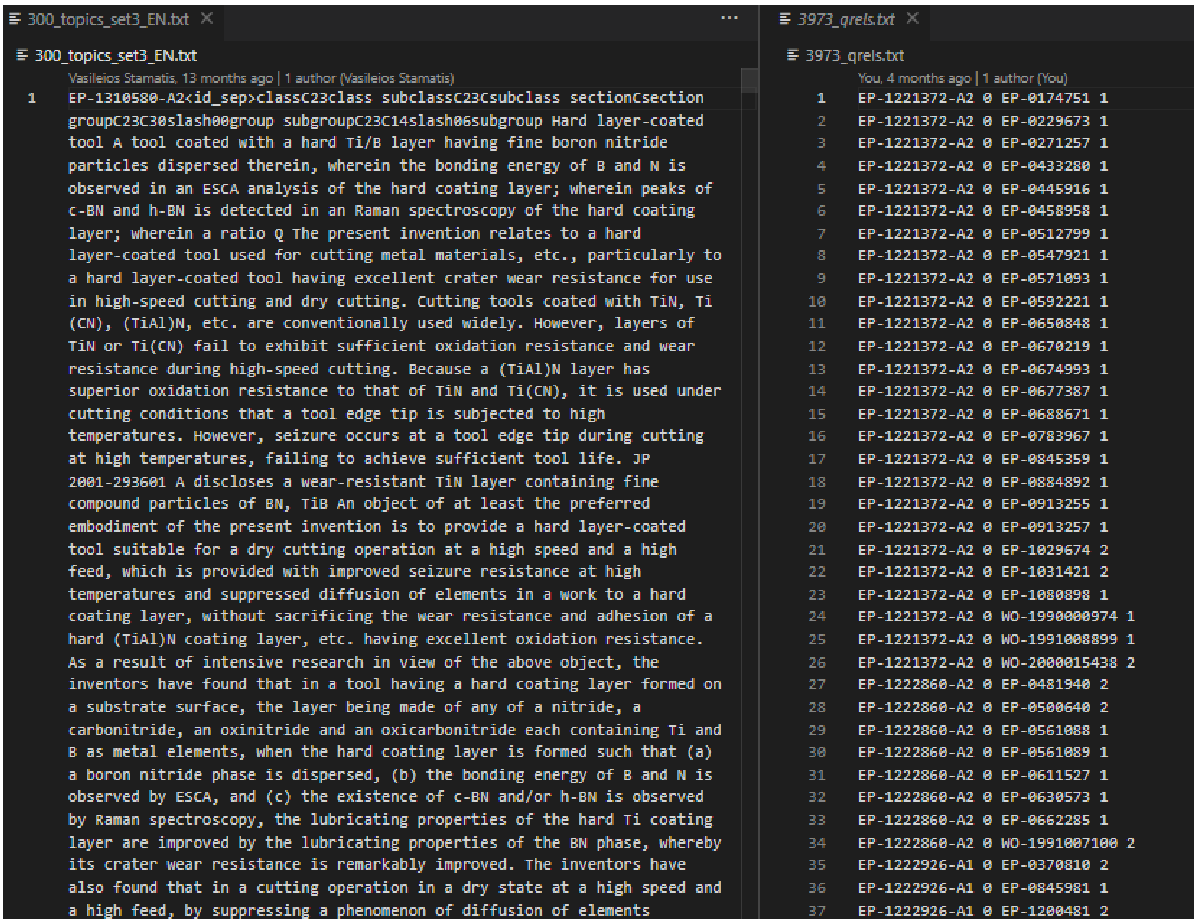
On the other hand, most of them were created by running a multi-step process that includes:

- *Term weighting* - powered by TF-IDF calculation.
- *Preprocessing* - it was done to include certain fields of the document
- *Word model training and prediction* - these processes were executed with DeepCT.
- *Term weights conversion* - with the use of another DeepCT function, term weights were converted to be suitable to be fed into the Pyserini Index creation.

- *Index creation* - the indices were created with the use of Pyserini

This complex process can be better visualised on Figure 14 above.

Folowing the creation of the indices, a set of 300 queries/topics were ran on them. Afterwards, the results of the query searches for each index are evaluated using a predetermined set of *expected* results, which are included in the *Qrels* helper file, mentioned in previous chapters.


300_topics_set3_EN.txt
Vasileios Stamatis, 13 months ago | 1 author (Vasileios Stamatis)
3973_qrels.txt
You, 4 months ago | 1 author (You)

```
EP-1310580-A2<id_sep>classC23class subclassC23Csubclass sectionCsection
groupC23C30slash00group subgroupC23C14slash06subgroup Hard layer-coated
tool A tool coated with a hard Ti/B layer having fine boron nitride
particles dispersed therein, wherein the bonding energy of B and N is
observed in an ESCA analysis of the hard coating layer; wherein peaks of
c-BN and h-BN is detected in an Raman spectroscopy of the hard coating
layer; wherein a ratio Q The present invention relates to a hard
layer-coated tool used for cutting metal materials, etc., particularly to
a hard layer-coated tool having excellent crater wear resistance for use
in high-speed cutting and dry cutting. Cutting tools coated with TiN, Ti
(CN), (TiAl)N, etc. are conventionally used widely. However, layers of
TiN or Ti(CN) fail to exhibit sufficient oxidation resistance and wear
resistance during high-speed cutting. Because a (TiAl)N layer has
superior oxidation resistance to that of TiN and Ti(CN), it is used under
cutting conditions that a tool edge tip is subjected to high
temperatures. However, seizure occurs at a tool edge tip during cutting
at high temperatures, failing to achieve sufficient tool life. JP
2001-293601 A discloses a wear-resistant TiN layer containing fine
compound particles of BN, TiB An object of at least the preferred
embodiment of the present invention is to provide a hard layer-coated
tool suitable for a dry cutting operation at a high speed and a high
feed, which is provided with improved seizure resistance at high
temperatures and suppressed diffusion of elements in a work to a hard
coating layer, without sacrificing the wear resistance and adhesion of a
hard (TiAl)N coating layer, etc. having excellent oxidation resistance.
As a result of intensive research in view of the above object, the
inventors have found that in a tool having a hard coating layer formed on
a substrate surface, the layer being made of any of a nitride, a
carbonitride, an oxinitride and an oxicarbonitride each containing Ti and
B as metal elements, when the hard coating layer is formed such that (a)
a boron nitride phase is dispersed, (b) the bonding energy of B and N is
observed by ESCA, and (c) the existence of c-BN and/or h-BN is observed
by Raman spectroscopy, the lubricating properties of the hard Ti coating
layer are improved by the lubricating properties of the BN phase, whereby
its crater wear resistance is remarkably improved. The inventors have
also found that in a cutting operation in a dry state at a high speed and
a high feed, by suppressing a phenomenon of diffusion of elements
```

```
EP-1221372-A2 0 EP-0174751 1
EP-1221372-A2 0 EP-0229673 1
EP-1221372-A2 0 EP-0271257 1
EP-1221372-A2 0 EP-0433280 1
EP-1221372-A2 0 EP-0445916 1
EP-1221372-A2 0 EP-0458958 1
EP-1221372-A2 0 EP-0512799 1
EP-1221372-A2 0 EP-0547921 1
EP-1221372-A2 0 EP-0571093 1
EP-1221372-A2 0 EP-0592221 1
EP-1221372-A2 0 EP-0650848 1
EP-1221372-A2 0 EP-0670219 1
EP-1221372-A2 0 EP-0674993 1
EP-1221372-A2 0 EP-0677387 1
EP-1221372-A2 0 EP-0688671 1
EP-1221372-A2 0 EP-0783967 1
EP-1221372-A2 0 EP-0845359 1
EP-1221372-A2 0 EP-0884892 1
EP-1221372-A2 0 EP-0913255 1
EP-1221372-A2 0 EP-0913257 1
EP-1221372-A2 0 EP-1029674 2
EP-1221372-A2 0 EP-1031421 2
EP-1221372-A2 0 EP-1080898 1
EP-1221372-A2 0 WO-1990000974 1
EP-1221372-A2 0 WO-1991008899 1
EP-1221372-A2 0 WO-2000015438 2
EP-1222860-A2 0 EP-0481940 2
EP-1222860-A2 0 EP-0500640 2
EP-1222860-A2 0 EP-0561088 1
EP-1222860-A2 0 EP-0561089 1
EP-1222860-A2 0 EP-0611527 1
EP-1222860-A2 0 EP-0630573 1
EP-1222860-A2 0 EP-0662285 1
EP-1222860-A2 0 WO-1991007100 2
EP-1222926-A1 0 EP-0370810 2
EP-1222926-A1 0 EP-0845981 1
EP-1222926-A1 0 EP-1200481 2
```

Figure 15. *Queries file & Qrels file*

Several observations can be made by examining the results that are presented on Table 2 below. Indices that were passed through DeepCT produce better results on Mean Average Precision (MAP), Precision@100 (Precision after 100 documents), Precision@1000 (similary, precision after 1000 documents) and Recall@1000. Generally, after some test runs the number of epochs used for training and prediction seems to produce ideal results with parameters set to the parameter values as suggested by DeepCT documentation. Surely, the machine learning process is beneficial for every metric and the more appropriate the parameters that word model is trained with, the better the predictions and results, generally speaking.

Concerning the different learning rates that were used, and keeping in mind that DeepCT examples used a learning rate of 2e-5(0.00002) or 5e-5(0.00005), we see that a much larger learning rate - for example 2e-2(0.02), produces much worse results. By increasing the learning rate, the model is taught to make bigger, rapid changes instead of small adjustments and thus a smaller amount of epochs is required. It can be observed that defining epochs in the range of 3 to 5 as our examples, the learning rate of 2e-5 (0.00002) seems to be well suited for better results. Another test run with a learning rate smaller that 2e-5 and more epochs might be in the right direction moving the experiments forward.

Table 2. *Querying evaluation of indices with common metrics*

#	Index	MAP	P100	P1000	Recall1000
1	**DeepCT, 3 epochs, 16 batch size, learning rate 2e-5**	**0.0016**	**0.0014**	**0.0004**	**0.0504**
2	DeepCT, 5 epochs, 16 batch size, learning rate 2e-5	0.0002	0.0001	0	0.0033
3	DeepCT, 3 epochs, 16 batch size, learning rate 2e-2	0.0002	0.0002	0	0.0036
4	DeepCT, 5 epochs, 16 batch size, learning rate 2e-2	0.0004	0.0001	0	0.0045
5	Simple Pyserini	0.0014	0.0002	0	0.0036

It is safe to say that DeepCT pre-processing largely improved every index's performance, with a few exceptions that had not used ideal parameters.

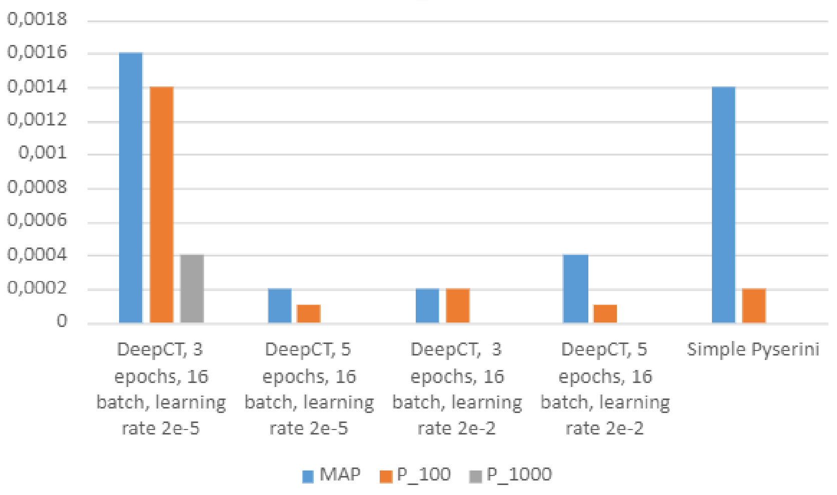


Figure 16. *Evaluation performance*

As one can see on Table 2 and Table 3 Recall after 100 & 1000 documents as well as, naturally, the absolute number of retrieved relevant documents on indices that involve DeepCT processing is significantly higher than their counterparts. This can be further visualised on Figure 16

Table 3. *Querying evaluation of indices with common metrics*

#	Index	Queries	Retrieved relevant documents	Recall100
1	**DeepCT, 3 epochs, 16 batch size, learning rate 2e-5**	300	**119**	**0.0181**
2	DeepCT, 5 epochs, 16 batch size, learning rate 2e-5	300	5	0.0025
3	DeepCT, 3 epochs, 16 batch size, learning rate 2e-2	300	5	0.0036
4	DeepCT, 5 epochs, 16 batch size, learning rate 2e-2	300	6	0.0030
5	Simple Pyserini	300	5	0.0036

Regarding these three metrics, looking at Recall@100 & Recall@1000, the difference between simple indices and DeepCT-processed indices seem to be significantly larger and the difference on the number of Retrieved relevant documents, which is 119 versus 5, appears to be very significant.

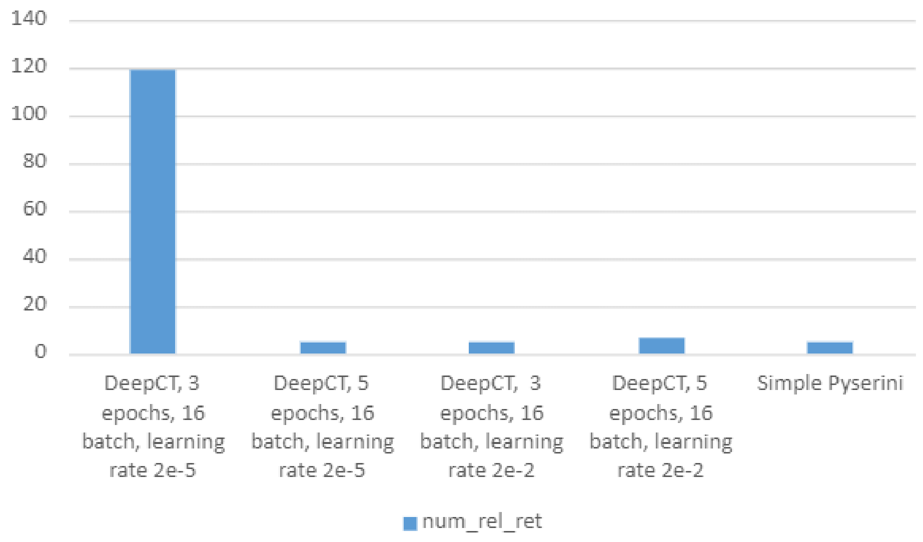


Figure 17. *Evaluation performance*

One could argue that this could make a large impact on patent retrieval which can largely influence businesses looking for patents conflicting with theirs. Another thing to note, is that "simple Pyserini" index appeared to perform well on one metric, Mean Average Precision (MAP).

Naturally, more extensive experiments shall be executed to make safer assumptions and conclusions.

12. Conclusion

This dissertation's objective was to research determine whether improvement of patent retrieval was possible using machine learning methods combined with traditionally used indexing software such as Pyserini.

More particularly, the machine learning methods used were the ones that are involved in a new software tool, using machine learning, DeepCT. DeepCT is based on Google's BERT word model and uses deep learning methods to make better sense of input text.

Some very important concepts are introduced and summarized for the reader to grasp the context of this research piece. Concepts like Information Retrieval, Artificial Intelligence, Machine Learning and Deep Learning, trained deep models are presented among others. Additionally, detailed information is provided by the author about the data, procedular details of the experiments and the parts of the data set that were used as well as exploration of weight creating algorithm were documented.

In this case, a data set containing hundreds of thousands of patent documents was used, namely the CLEF-IP dataset. After commencing running several experiments that included index creation with the use of machine learning methods, as well as simple index creation without machine learning enhancement and many test runs that were focused on finding the right parameters for running DeepCT procedures. These experiments produced as a final result several indices that each one of them originated of and provided representation of the same document collection.

Subsequently, it became apparent that DeepCT processing reinforced with TF-IDF term weight calculation provides more value on created indices, as opposed to indices that were created without any pre-processing, with only Pyserini index build. These observations were made on various different indices that the same queries/topics were ran on them as a baseline. Additionally, the query results of all the different indices were evaluated using a Qrels file with "expected" results, to further define which index performed better or not.

Finally, detailed experiment results are documented against the same baselines that provide the research with a way forward after careful interpretation. It is clear that Deep Learning can enhance term weighting and more importantly, Information Retrieval and in this case study, Patent Retrieval. Further and more thorough experiments and test runs shall be executed for researchers to make conclusions to a greater extend and crucially optimize the use of various tools like the ones discussed in this dissertation for the greater advancement of Information Retrieval.

13. Future research and work

In regards to how the presented research should be enhanced and expanded, there is one key aspect that can be improved and can provide us with safer and more accurate results. These test runs need to be run on the full dataset which requires for the use of a higher capacity computer with larger computational power and size of memory.

Several of the experiments were run on a part of the document collection, a part that contained approximately one fifth of the total data set. Being able to run the same experiments on the whole document collection would provide this research piece with much more confidence on the results and the observations that were made based on those results.

Moreover, TF-IDF was used in these test runs that involved DeepCT, as an extra way to further enhance the results. Continuing with this research, more term weighting algorithms should be considered and tested as alternatives to the TF-IDF algorithm, to determine if this part of the process can become more effective.

References

[1] Ehsan Nowroozi. "Introduction to new methodologies and applications in information retrieval indexing". In: *2010 2nd International Conference on Mechanical and Electronics Engineering* (2010). DOI: 10.1109/icmee.2010.5558505.

[2] Ahmet Alkilinc and Ahmet Arslan. "A comparison of recent information retrieval term-weighting models using ancient datasets". In: *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)* (2018). DOI: 10.1109/idap.2018.8620857.

[3] K. R. Chowdhary and V. S. Bansal. "Information retrieval using probability and belief theory". In: *2011 International Conference on Emerging Trends in Networks and Computer Communications (ETNCC)* (2011). DOI: 10.1109/etncc.2011.5958513.

[4] Nina Wacholder. "Interactive query formulation". In: *Annual Review of Information Science and Technology* 45.1 (2011), pp. 157–196. DOI: 10.1002/aris.2011.1440450111.

[5] *AI vs. Machine Learning vs. Deep Learning vs. neural networks: What's the difference?* URL: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

[6] Pariwat Ongsulee. "Artificial Intelligence, Machine Learning and deep learning". In: *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)* (2017). DOI: 10.1109/ictke.2017.8259629.

[7] Peng Shengguang. "Overview of meta-reinforcement learning research". In: *2020 2nd International Conference on Information Technology and Computer Application (ITCA)* (2020). DOI: 10.1109/itca52113.2020.00019.

[8] By: IBM Cloud Education. *What are neural networks?* URL: https://www.ibm.com/cloud/learn/neural-networks.

[9] Zhuyun Dai and Jamie Callan. "Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval". In: . *In Proceedings of ACM Conference* (2019). DOI: 1910.10687.

[10] Jacob Devlin et al. *Bert: Pre-training of deep bidirectional Transformers for language understanding*. 2019. URL: https://arxiv.org/abs/1810.04805.

[11] Dai Zhuyun. *Adedzy/DeepCT: Deepct and HDCT uses Bert to generate novel, context-aware bag-of-words term weights for documents and queries.* Jan. 2020. URL: `https://github.com/AdeDZY/DeepCT`.

[12] Florina Piroi and Allan Hanbury. "Multilingual patent text retrieval evaluation: CLEF–IP". In: *Information Retrieval Evaluation in a Changing World* (2019), pp. 365–387. DOI: `10.1007/978-3-030-22948-1_15`.

[13] *Information management and preservation.* URL: `http://www.ifs.tuwien.ac.at/~clef-ip/index.html`.

[14] *MS marco.* URL: `https://microsoft.github.io/msmarco/`.

[15] Ochin Sharma. "Deep challenges associated with Deep Learning". In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)* (2019). DOI: `10.1109/comitcon.2019.8862453`.

[16] Anusha Chamarty. "Fine-tuning of learning rate for improvement of object detection accuracy". In: *2020 IEEE India Council International Subsections Conference (INDISCON)* (2020). DOI: `10.1109/indiscon50162.2020.00038`.