T.C.

MARMARA UNIVERSITY

FACULTY of ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

CSE4197 Engineering Project I Project Specification Document

Title of the Project

SEARCH ENGINE IN TURKISH LAW DOMAIN

Date

01.12.2022

Group Members

Fatih Satı 150119625

Merve Hazal Özalp 150120827

Mehmet Selman Baysan 150120841

Supervised By

Assoc. Prof. Dr. Murat Can Ganiz

Table of Contents

1. Problem Statement

In this project, a semantic search engine specific to the law domain will be developed. In this search engine, unlike the classical search engine methods, by using Machine Learning and Natural Language Processing methods, it is aimed to show the end user not only the documents containing the searched word or sentence, but also the documents containing the words or sentences that are semantically close to this query, as a result, via an interface.

2. Problem Description and Motivation

We live in the information age, and by means of development of technology, most of this information is available on the internet in a digital way and people use this information to facilitate their work and daily lives, but the presence of too much digital data causes some problems. The most important of these problems is the difficulty in reaching the right information that people are looking for, so companies like Google have developed search engine algorithms and software to help people reach the information they are looking for in the fastest and most accurate way.

Large companies such as Google develop general purpose search engines, but some businesses require domain specific search engines since general purpose search engines may not work as effectively as specialized search engines for domain specific queries. The law domain is also one of the fields that needs a domain specific search engine because people working in the law domain use this domain specific search engines to find precedents and relevant law articles in the legislation to refer to when writing their legal petitions.

These search engines may significantly reduce the time that lawyers spend on researching. Although these search engines are used very often, they share a common problem which is that search engines do a word-based search in the law domain, so they only return law texts containing the searched words or phrases, therefore, users may not find the information they are looking for in the first place. So that lawyers and clerks may have to go to great lengths to find relevant precedents or legislation in

search results that are truly relevant to their cases. Also, legislation search and leading case search serve as separate search engines, that situation doubling the time spent.

Our solution to that problem is to combine these two search engines into one, to get the final search result. Our project is to produce a solution to this problem with the help of natural language processing techniques, text mining techniques, big data solutions and machine learning algorithms.

3. Main Goal and Objectives

Since the current law-specific search engines [1] perform word-based searches instead of semantic searches, results that are not semantically close to the searched query may also return among the returned results. In order for the user to reach the desired information from these results, it is necessary to examine all the incoming texts, which leads to a waste of time and resources. Our goal is to develop a semantic search engine and show results related to the desired information, thus shortening the time it takes to reach this information.

Our objectives as follows:

i. Collecting of documents specific to the law domain. These documents can be divided into sub-branches as Supreme Court decisions, constitutional court decisions and legislation.
ii. Cleaning and parsing the collected data so that it can be used in machine learning models and transferred to the search engine database.
iii. Developing machine learning models that extract the semantic properties of words and their numerical equivalent vectors using the data we have. In this part, different machine learning and deep learning models will be tested.
iv. Extracting vectors that represent the semantic properties of the data we have in order to perform semantic search.
v. Indexing the documents we have and the extracted vectors in the search engine.
vi. Implementation of algorithms required for semantic search and creation of semantic search pipeline for the entered query
vii. Make the search engine available to the end user via web based user interface.

4. Related Work

- **Bert and fasttext embeddings for automatic detection of toxic speech [2]**

  In this study, different word embedding models such as fasttext and BERT are compared for hate speech detection. In the scope of the project, BERT model is fine-tuned and compared with pre-trained models. As a conclusion of this study, both fasttext and BERT model's word representation perform very close. Also, fine-tuned BERT model for the study performed a little better than the pre-trained models.

- **A Word Embedding Model for Medical and Health Applications in the Arabic Language [3]**

  The study focuses on building a word embedding model for Arabic language in the medical domain. Three commonly used models which are fastText, GloVe and Word2Vec are pre-trained as a part of the study with the dataset which contains around 1.5 million documents. To evaluate the performance of the models, different approaches were applied. As a result of the study, fastText performed better than other word embedding models.

- **Comparing apache solr and elasticsearch search servers [4]**

  The study includes a comparison of different search engines which are Solr and Elasticsearch. The Importance of the search engines and document indexing are mentioned in the study. Since the platforms offer different advantages, using one of these platforms for document indexing is a reasonable decision. As a conclusion, although both platforms perform very well, it concluded that Elasticsearch has better advantages.

- **Comparative study of semantic and keyword based search engines [5]**

  In this article, semantic and keyword search engines were compared and the important differences between them were examined. Then, semantic search engines names, the technologies used and the pros and cons of these technologies are mentioned. The study stated that while keyword search engines are unable to locate precise and

accurate information for user queries because it depends on the syntax of the keywords and page rank algorithms, semantic search engines are able to resolve this issue by looking at the meaning of the keywords.

- **A vector space search engine for web services [6]**

  This study focuses on utilizing a search mechanism known as The Vector Space Model which is mainly used for search engines, based on natural language. The size of the keywords in the vector space and the position of these vectors to other vectors and their similarity to each other were examined.

5. Scope

Our scope is developing an NLP based semantic search engine on the scope of Turkish Supreme Court Jurisprudence and legislation documents. As law decisions and judgements may vary from country to country, only Turkish Law will be covered, hence the search engine is only for queries in Turkish language. In search engines, typo and spelling are frequently seen. The search engine to be developed within the scope of the project will return results according to the incorrectly entered query even if there is a typo error in the entered query. No improvement will be made to detect and correct this error, so it is the user's responsibility to enter the correct query. In order to test the functionality of the search engine, a certain number of jurisprudence and legislation will be downloaded for use. Afterwards, the search engine will be developed with this data and no new data will be added to the database. The performance metric in the search engine will mainly be accuracy, not speed. After the models are trained, an optimal number will be determined for the search results, and this number of results will be returned in the search queries. then these results will be sorted by their cosine similarity, so there will be no options such as the number of results returned and the sort option. Also, there will not be any saving or printing option for results.

A. Limitations

- Since understanding the law language and being able to comment on the law language requires a bachelor’s degree, it is difficult for non-law professionals to determine whether the search results are relevant to the entered query.

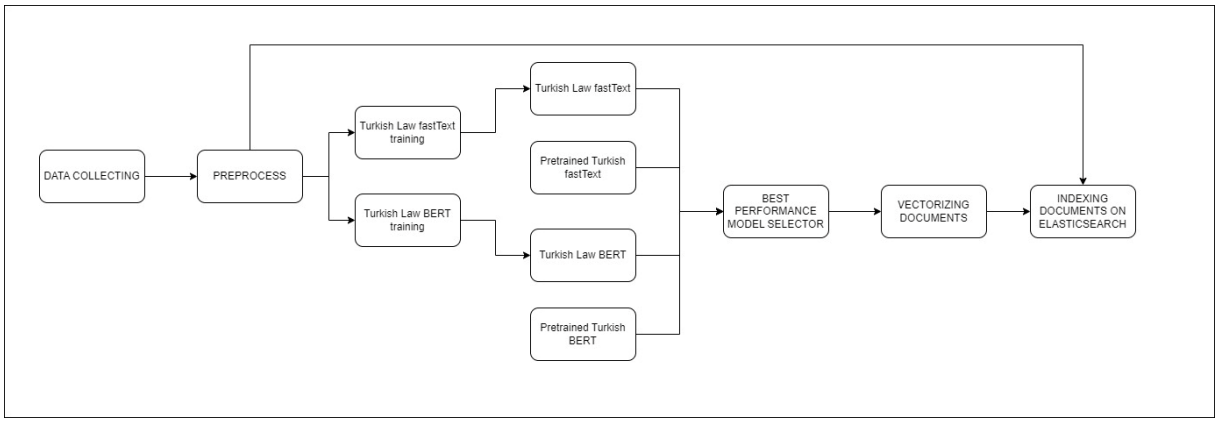
- Within the scope of the project, it may be necessary to consult law domain experts, but it may not be easy to reach these people whenever needed.
- Due to memory limitations, around 100000 of jurisprudence will be used out of more than 7 million jurisprudence shared by the Supreme Court.
- Training large language models requires specific and sufficient hardware, so models will be trained by fine-tune or minor pretraining.

B. Assumptions

- It will be assumed that the data will be crawled from the Supreme Court's website.
- It will be assumed that people who will use the search engine know the Turkish language.

6. Methodology and Technical Approach

In this Project we are going to develop a semantic search engine based on Turkish jurisprudence of the supreme court data and legislation data. This project gathers technologies from different fields under one roof so that it consists of several steps to accomplish. The methodology of this project can be summarized with the figure below, these steps will be explained in detail in order



6.1 Data Collection

In this step we are going to crawl Turkish jurisprudence of the supreme court and Turkish legislation documents from the official search engine of Turkish Supreme Court [1] and official Turkish regulatory database [7] with the help of Python's request [8], beautifulsoup [9], selenium [10] libraries. Selenium is a open-source umbrella Project

that enables browser automation with Python, with the help of selenium first we open supreme court website and we will pass “dava” keyword as query because after a little glance we make sure that “dava” keyword occurs in every document, then we filter chamber number so that we can make sure that we will collect enough data from each chamber of the supreme court to balance our dataset. After clicking the search button, with the request library we will get the raw html text of the result page and parse it to collect all document data with beautifulsoup. This search engine has a result limit which is 1000 documents found in 100 pages so we will move these result pages with selenium and repeat this process until collecting all documents. We will follow a similar path, using the same libraries and methodologies to download regulatory data.

6.2 Data Preprocessing

Our jurisprudence documents consist of subsections such as chamber type, document number, date etc. In this project, Elasticsearch [11] planned to be used as a database and it accepts documents in JavaScript Object Notation (JSON) [12] format. In this subtask, documents will be converted into JSON format with the appropriate subsections. To extract these fields from raw text documents, Regular Expressions [13] is planned to be used because these fields follow some patterns and it can be extracted using python’s re [14] library. laws in legislation contain multiple clauses, these clauses also will be splitted and converted into JSON format using the same methodology.

6.3 Elasticsearch Indexing

Elasticsearch is a distributed architecture open source project which is written entirely in Java [15] language. Elasticsearch is based on the search software Apache Lucene project that aims to shorten the search process on long unstructured text documents. Elasticsearch performs search operations over structured texts which is JSON format. The most important feature of Elasticsearch is that it returns query results quickly because it indexes which document a word is in while saving data to its database. This saving mechanism is called inverted index; it can be easily understandable with Figure [1] below.

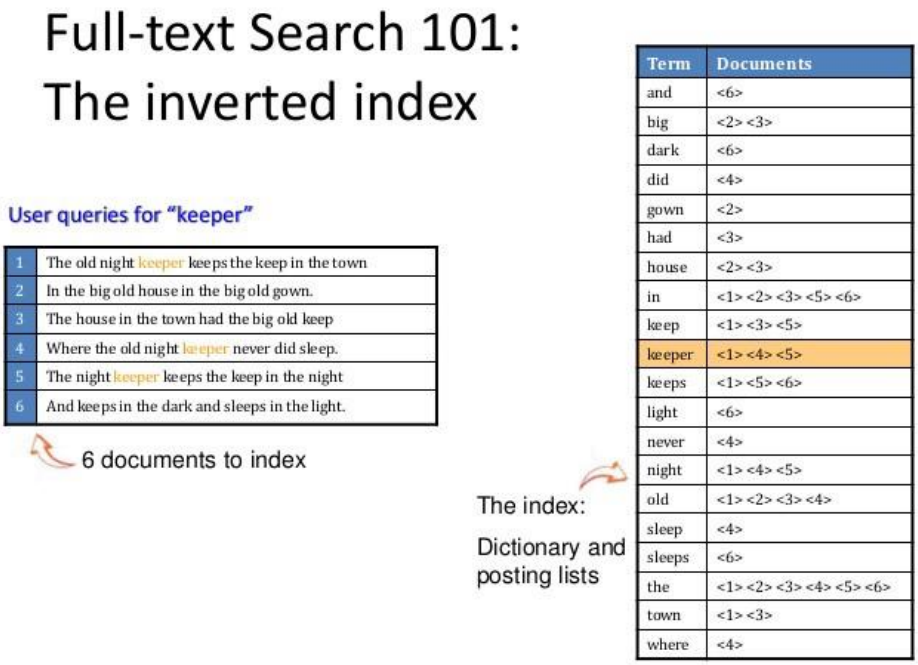


Figure [1] : A illustration of inverted index on sample data, figure taken from [16]

Elasticsearch will be used in this project because of its capabilities in the searching process. To use it, JSON documents that are created after data preprocessing will be indexed to Elasticsearch search engine database. A group of related documents is referred to as an Elasticsearch index, in the scope of this Project there will be two indices which are jurisprudence and legislation indices. In each document there are some fields *"keys in JSON"* and corresponding values for these values. In order to save each document to related Elasticsearch index mappings needed to be done correctly. Mapping is defining which field accepts what type of data. In Elasticsearch there are strings, numbers, Booleans, dates, arrays of values, geolocations, or other types of data, in this project although, string data type mainly used, date will be also used in mappings. After setting all these mappings, all of jurisprudence and legislation documents will be indexed accordingly with the Elasticsearch Python client [17].

6.4 TF-IDF

TF-IDF is a way of representing mathematical relationships between words and documents which is used in a variety of tasks. Elasticsearch uses this TF-IDF algorithm

for searching a given query into an index. In this project the TF-IDF method will not be used directly as a search algorithm but in the developing process it will be a good guide to compare with semantic search results. Also for our project the tf-idf algorithm can be used on filtering documents according to the user's desires. For example if a user wanted to get semantically relevant documents that do not contain specific words, this filtering will be applied on documents using tf-idf so that semantic search will be limited to these filtered documents. [18]

TF in the TF-IDF stands for “Term Frequency” and it is calculated as counting the appearance of words in a document. And its formula as below:

$$tf\,(t,\,d) = log\,(1 + freq\,(t,\,d))$$

(1)

IDF stands for “Inverse Document Frequency” and it is a calculation of how frequent or uncommon a word is over the full corpus of documents. And its formula below:

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

(2)

TF-IDF is multiplication of these two TF and IDF values, if this value close to 0 it does not valuable for searching documents but if this value close to 1 that has a valuable meaning which is high term frequency (in the supplied document) and a low document frequency of the term in the entire collection of documents. Elasticsearch makes these calculations while indexing documents. And TF-IDF formula as below:

$$tf\,idf\,(t,\,d,\,D) = tf\,(t,\,d)\,.\,idf\,(t,\,D)$$

(3)

6.5 Word Embedding

Main goal of this project is developing a semantic search engine, to develop such a system cosine similarity plays an important role. Cosine similarity will be explained in more detail later on but for now, to understand the importance of word embedding, it is briefly called as the dot product of two vectors divided by the norms of these vectors. [19] As it can be understood from the definition, vectors are needed to

calculate the similarity score. for this reason researchers in the NLP domain tried to find a way to represent words or sentences in n-dimensional vector for their tasks because if they can extract the meaning of a word with numbers they can apply mathematical methods to measure how similar is given two words so that they come up with a Word Embeddings. Word Embedding is a vectorial representation of the word that is extracted from unsupervised machine learning models. [20] These unsupervised models take a large corpus as an input and analyze the probability of a word being found among certain words by iterating through a dataset. There are two unsupervised machine learning models that we will use in this project for the word embeddings which are fastText and BERT.

6.6 Corpus Creation

To train unsupervised vector extraction models such as fastText and general language models such as BERT, a large corpus containing GB's of text data needed. For example, dbmdz/bert-base-turkish-cased which is general purpose language model can be found on huggingface's model hub was trained on 35GB of turkish text corpus [21] and pretrained Turkish fastText model trained with the all website content in turkish language [22] which is also very large corpus. In order to train these unsupervised machine learning models, Turkish Law corpus that contains turkish law related text documents needed to be collected. To accomplish this step we are planning to collect academic resources in Turkish Law domain such as master's thesis, doctoral thesis, journal publications, books. At the same time European Court of Human Rights judgements, constitutional court judgements and lawyers' legal blogs will be collected using python's requests, beautifulSoup and selenium libraries via using the same methodology mentioned in the data collecting part.

6.7 Pre-training Fasttext Model

fastText is a library developed by Facebook's AI Research (FAIR) team for word embedding and text classification learning. [23] fastText has Python binding, in this project fastText will be used with Python programming language. fastText has a

train_unsupervised function for training an unsupervised machine learning model that will be used for word embeddings. This function takes corpus as an input and returns model as an output. Although whole machine learning complexity abstracted from user this function has 17 parameters that can affect models performance. So that in this part, in addition to academic studies and similar works, we will try to create the most suitable model by playing with the parameters. These parameters are listed below. [24]

train_unsupervised parameters	
input	# training file path (required)
model	# unsupervised fasttext model {cbow, skipgram} [skipgram]
lr	# learning rate [0.05]
dim	# size of the word vectors [100]
ws	# size of context windows [5]
epoch	# number of epochs [5]
minCount	# minimal number of word occurences [5]
minn	# min length of char ngram [3]
maxn	# max length of char ngram [6]
neg	# number of negatives sampled [5]
wordNgrams	# max length of word ngram [1]
loss	# loss function {ns, hs, softmax, ova} [ns]
bucket	# number of buckets [2000000]
thread	# number of threads [number of cpus]
lrUpdateRate	# change the rate of updates for the learning rate [100]
t	# sampling threshold [0.0001]
verbose	# verbose [2]

Input parameter will be the corpus that mentioned in part 6.6. There are two model parameter options for computing word representations which are skipgram and **C**ontinous **B**ag **o**f **W**ords (cbow). By using a word that is close by, the skipgram model can learn to anticipate a target word. The cbow model, on the other hand, makes predictions about the target word based on the context. The words in the target word's fixed-size window that make up the context are shown as a bag. [20] This difference can be illustrated by the Figure [2] below.

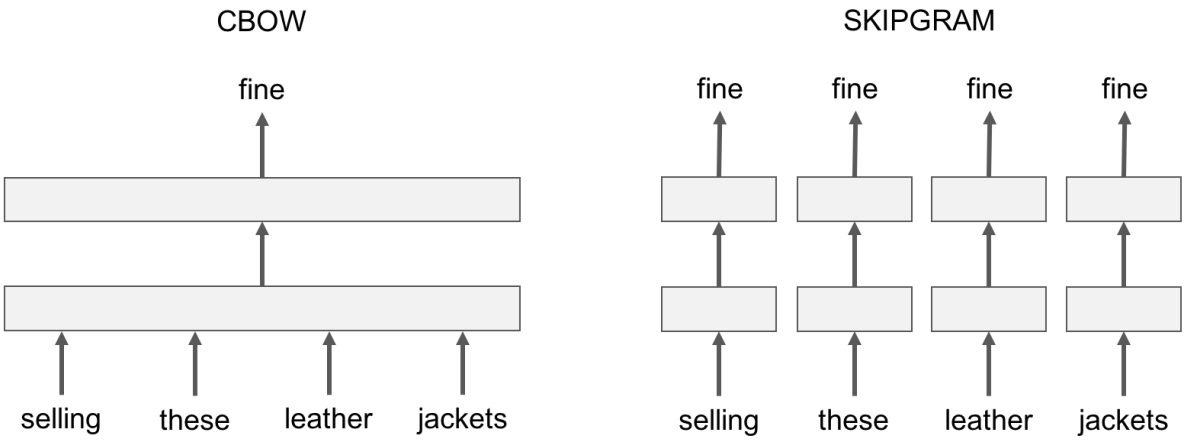


Figure [2]: Illustration of difference between CBoW and Skipgram. Figure taken from [25]

Default values will be used on the other parameters initially by playing with the values, the most suitable parameters will be found later on.

6.8 Pre-training BERT Model

Bidirectional Encoder Representations from Transformers, shortly BERT, is a brand-new language representation paradigm that is frequently used in Natural Language Processing tasks in recent years. By concurrently conditioning on both left and right context in all layers, BERT is intended to pretrain deep bidirectional representations from unlabeled text without making significant task-specific architecture alterations, the pre-trained BERT model may be improved with just one additional output layer to produce cutting-edge models for a variety of tasks, including question answering and

language inference. [26] Since this BERT architecture conditioning full context, unlike fastText which is word-based architecture, in this project domain specific pretrained BERT model performance on word embeddings and semantic search will be compared to fastText. There is no available pretrained BERT model on Turkish Law domain in Hugging Face model hub [27] so in this part BERT model on this domain will be trained. To train this model the same corpus created for the fasttext model will be used. Hugging Face is open source AI by offering a single location for materials such as models, datasets, ML demos and libraries. [28] One of their libraries is Transformers which help end users to pretrain, fine-tune and use big language models such as BERT. In this project, to train Turkish Law domain specific BERT model, Transformers library [29] will be used. run_mlm.py Python file handle training with the parameters below.

python run_mlm.py
model name or path roberta base
train file path to train file
validation file path to validation file
per device train batch size [8]
per device eval batch size [8]
do train
do eval
output dir /tmp/test mlm

Model name parameters will be bert-base-cased, Corpus will be given as train_file and other parameters will be adjusted according to hardware limitations.

6.9 Comparison of Model Performances

In this part, comparison of four different vector extraction models will be handled. These four models can be listed as below.

- fastText pretrained Turkish Common Crawl model.

- Turkish Law fastText model which will be trained in the scope of this project.
- dbmdz/bert-base-turkish-cased Pretrained general purpose BERT model for Turkish Language.
- Turkish Law BERT model which will be trained in the scope of this project.

First pretrained Turkish fastText model and Turkish Law fastText comparison, then pretrained Turkish BERT and Turkish Law Bert model comparison will be handled. Then the winner of each comparison will be compared to each other, so that final model will be used to extract word embeddings of documents.

6.9.1 Comparison of Pretrained Turkish Fasttext Model and Turkish Law Fasttext Model

In this part the test set will be prepared for comparison. In this dataset there will be word pairs related to Turkish law in a way that each pair words should be semantically close, although they are not very similar in character to each other. In this part, help from domain experts might be necessary. After the test set is ready we will compare the cosine similarity of each pair and take the average of cosine similarities of the whole set. Model which has a higher average cosine similarity score will be chosen for the final comparison.

6.9.2 Comparison of Pretrained Turkish BERT Model and Turkish Law BERT Model

In this part the test set will be different than the previous part because since BERT takes inputs as tokens and processes them as a whole, vectors will be extracted from this whole context. This time instead of word pairs there will be paragraph pairs that are semantically close with few common words will be created with the help of domain experts. Once the test set is prepared, we will compare each pair's cosine similarity and take the average of the cosine similarities across the board. The model used in the final comparison will be the one with the highest average cosine similarity score.

6.9.3 Comparison of fastText Model and BERT Model

In this part comparison of winner models of part 6.9.1 and 6.9.2 will be handled. In this part paragraph pairs will be used as a test set and this time while extracting vectors with fastText we will use fastText's get_sent_vector function which extracts each word's vector in the sentence and takes the average of them. On the result of that comparison, the model that will be used in the semantic search is determined with the highest average cosine similarity score.

6.10 Vector Extraction

In this part by using the model that was determined after the comparison on the part 6.9, document vectors will be extracted from all of the jurisprudence of the supreme court documents that have been collected in the scope of this project. For this vector extraction part, depending on the type of model to be used, two different problems may be encountered. If fastText model will be used, since fastText extract vectors word by word, it will also extract the vectors of words that repeat too much in the text and do not add anything to the text semantically, but only connect the words such as ''and'', ''or'' etc. in English. These words may lead our document vectors to lose their semantic features so that stopwords list that contains repetitive and meaningless words in Turkish Law domain may be extracted and while extracting document vectors these words can be skipped. If BERT model will be used for vector extraction task, input tokens must be less than or equal to 512 to be able to give it to the BERT model but documents will be used in this project generally have much more than 512 token so that to handle this problem input document may divided into several 512 tokens batches and extract the vector of each batch separately and applying semantic search on these batches separately. After handling these problems, all document embeddings will be extracted and saved as a new field on json documents.

6.11 Indexing Vectors

After extracting vectors, they needed to be indexed to the Elasticsearch database to apply semantic search on documents. To index vectors, Elasticsearch index mappings have to be adjusted according to this as shown in the Figure[3].

```
PUT my-index-2
{
  "mappings": {
    "properties": {
      "my_vector": {
        "type": "dense_vector",
        "dims": 3,
        "index": true,
        "similarity": "dot_product"
      }
    }
  }
}
```

Figure [3]: Elasticsearch index mapping sample on vector indexing. Figure taken from [30]

For the vector field type should be dense_vector which is Elasticsearch data type for storing numeric values or vectors, dims parameter should be vector dimension that will be extracted. index parameters should be true for calculating similarity on it. Similarity parameter determines which method to use when calculating distance between two vectors. There are three different options that Elasticsearch allows users for calculating vector distances which are l2_norm, dot_product and cosine. [30]

6.12 Semantic Search

In this part semantic search pipeline will be applied. Pipeline will take a query as an input and preprocess that input then extract the query vector. And this query vector will be input of the Elasticsearch knn_search function. knn_search function searches for the k closest vectors to a query vector in the indexed document vectors, as calculated by a similarity metric that is determined on the Elasticsearch vector indexing

which is dot_product in the scope of this project. Sample K-nearest neighbors (kNN) search can be done with Elasticsearch API in the Figure [4].

```
POST image-index/_search
{
  "knn": {
    "field": "image-vector",
    "query_vector": [-5, 9, -12],
    "k": 10,
    "num_candidates": 100
  },
  "fields": [ "title", "file-type" ]
}
```

Figure [4]: Elasticsearch kNN search example. Figure taken from [31]

Field parameter determines which field on our index, vector search will be applied, query_vector parameter will be vectorized input query taken from user, k parameter determines how many semantically relevant document should return as an output and finally num_candidates determines number of potential closest neighbors on each shard. After performing this search, up to k documents will be returned to the user. [31] To perform this search two techniques will be used mainly which are Cosine Similarity and KNN. These techniques will be explained in the following parts in detail.

6.12.1 Cosine Similarity

Although dot_product will be given as a similarity parameter when indexing vectors to Elasticsearch database, it is an optimized version of Cosine Similarity so that Cosine Similarity and its difference between dot_product will be explained briefly in this part. Since word vectors are represented in n dimensional vector space, cosine similarity is a calculation used to determine how similar two vectors are by multiplying their cosine angles in this n dimensional vector space. If the two vectors are close to each other

their cosine similarity value will approach to 1 otherwise value will approach to 0. [32] This value can be calculated with the formula in the Figure [5].

$$Cos\ \alpha = \frac{A \times B}{|A| \times |B|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Figure 5: Cosine similarity formula. Figure taken from [32]

This value is obtained by the dot product of two vectors divided by length multiplications of these two vectors.

As can seen in the formula above, dot product of two vectors is already calculated and while indexing documents to Elasticsearch database, vectors should be converted to unit vectors to use dot_product similarity parameter. This conversion takes an n-dimensional vector into one floating number and that is a kind of summary of that vector so that the divisor part in the cosine similarity formula will not be used in this calculation and that makes the calculation process faster than cosine similarity.

6.12.2 kNN

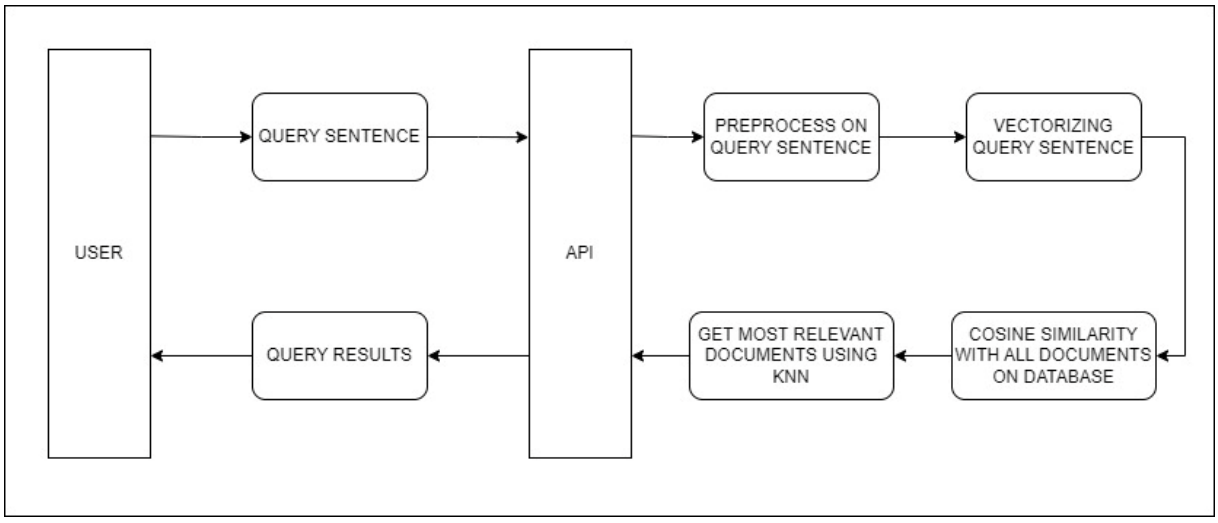
K-nearest neighbor (kNN) search is applied to a vector which finds the k nearest neighbors in terms of similarity. The kNN will be applied to the search query vector and the document vectors for finding the most relevant k documents. There are two methods for kNN search that Elasticsearch supports which are script_score and kNN which is Approximate kNN. Approximate kNN will be used for the law domain specific search engine. The kNN searches for closest vector points on the document space where the center of the space is the search query vector. The searching on the space stops when it reaches the k document, which means the speed of the process is related with the similarity of the search query vector and the k. [33] K is 10 as default on the Elasticsearch, for our project it will be determined after the measurement of the average related document number.

6.13 User Interface

Users need a system to interact with the website they are using. Usually this interaction is handled on the User Interface which includes display, mouse and keyboard actions. User interface is crucial for a website since all the interaction between system and user is handled here. A complex design may make usability of the website difficult where a user friendly design makes things easier. In the project, users will interact with the search engine using a user-interface. To implement the user-interface HTML and CSS will be used with the Flask API.

6.13.1 Flask API

Flask is a web framework which is used for creating Application Programming Interfaces (API) using Python. [34] APIs are used for the communication between the server and the web site. For the project, API will be used for reaching related documents on the search engine and it will be served to the website. Also, search queries will be sent to the Elasticsearch using Flask API. Flask API can be created in Python using Flask Library. Moreover, Flask can render HTML and CSS files which allows a connection between the website and the Restful API. The API will work as shown in the figure below



7. Professional Considerations

7.1 Methodological Considerations

Python: Python will be used as a programming language for almost every task during the development process of the project. For web crawling Python's request, beautifulSoup and Selenium libraries will be used. The data processing in all steps will be done using the "re" library. For building machine learning models, we will use a variety of Python libraries including NLTK [35], pytorch [36], scikit-learn [37] etc. For deploying our project, the necessary API will be written using Flask on Python.

Github: Version control is one of the most important aspects of programming projects. We are planning to use Github as the version control system. [38]

Elasticsearch: Elasticsearch will be used as a database but also its other beneficial features such as vector searching and TF-IDF searching will be used. It is decided to use Elasticsearch since it is fast and reliable [4].

HTML/CSS: Interface of the project will be done using HTML and CSS. We are planning to have a user-friendly interface that users can easily manage using and understanding of the project.

7.2 Realistic Constrains

7.2.1 Economic

We believe that our project is profitable since it has thousands of potential users. Since the law practitioners spend too much time on the research they would be willing to pay for a product that helps them to save lots of time.

There is a similar product in the market which has been published lately. De Jure which cost user 6.500₺/Annually. [39] Our product's price may differ accordingly to its performance and the maintenance costs.

There would be a cost of running a website beside developing a better search engine. Also, there would be the cost of advertisement, customer representative addition to the development costs.

7.2.2 Environmental

Since our project will be developed and served as online there will not be any environmental pollution or anything that discomforts any living creature.

7.2.3 Ethical

The project's target users are Law practitioners who are expected to search for precedent on it. The practitioner may be misled if the search engine returns irrelevant or incorrect documents, which is unethical.

7.2.4 Health and Safety

Since the project will be a web-based search engine it will not have a physical effect on the users. Also, project will consist only documents on the domain of law so there will not be any sexual or violent content.

Project is safe in terms of both physical and mental aspects.

7.2.5 Sustainability

The Law has been applied for thousands of years. It evolves, it changes but the existence of law will last forever. As long as law exists, law practitioners need to search for precedent. In this case our product may survive as long as it is preferable.

7.2.6 Social

Project does not have any side effects on anyone, instead it may have good effects on target users. Since it is a domain specific product, people outside of the domain will not be interested at all.

7.3 Legal Considerations

Since the search engine that is being developed will consist of thousands of documents, copyright may be considered as legal considerations on this project. The search engine will have jurisprudence and legislations as documents. Turkish Supreme Court publishes jurisprudences on its website [1] so all the documents that will be used on the search engine are public. Since all the documents are public, copyright cannot be issued on the project. Therefore, project does not have any legal considerations.

8. Management Plan

A. Description of All Tasks

1. Data Collecting:

I. The jurisprudence of the Supreme Court will be downloaded from the website of the Supreme Court [1]. In order to keep the jurisprudence of the Supreme Court up to date, approximately 200,000 documents will be downloaded from 2021 on.

II. All documents related to the legislation will be downloaded from the Turkish regulatory database[2].

2. Data Cleaning:

This task contains multiple subtasks as follows:

I. Parsing Supreme Court Jurisprudence: The Supreme Court Jurisprudence consists of fields such as base no, decision no, date of decision, court, type of crime, jurisprudence text, department number. It is aimed to extract these fields from the raw document with the help of regex and save them in JSON format.

II. Parsing Legislation: Legislation consists of laws, laws consist of clauses, and clauses consist of sub-clauses. It is aimed to separate them from each other with the help of regex.

III. Stop-Word Extraction: This subtask is for calculating frequency of each word in the documents. It is aimed to create a stop-word list specific to the law domain by identifying the words that are specific to the law domain but do not make any sense.

IV. Conversion of data into suitable format for model training: Besides indexing, this data will also be used for model training. For this reason, it is necessary to convert the data into a format suitable for model training.

3. Building Machine Learning Models for Vectorizing:

I. It is aimed to train Turkish Law domain specific fastText models.

II. It is aimed to train Fine-Tuning BERT sentence transformer models.

4. Data Vectorizing for Semantic Search

I. By using the Turkish fastText model, jurisprudence vectors will be extracted.

II. The jurisprudence vectors will be extracted using the fastText model that we will train.

III. By using the Turkish BERT model, jurisprudence vectors will be extracted.

IV. The jurisprudence vectors will be extracted using the BERT model that we will train.

V. The performances of the vectors extracted above will be compared on the semantic search and the final vector will be decided as a result of these comparisons.

5. Indexing Data on Search Engine

I. All of the fields specified in clause 2 will be indexed in ElasticSearch. In this way, it will be possible to filter the results to be returned in the search.

II. The vectors determined in sub clause 5 of clause 4 will be indexed to match each jurisprudence.

6. Interface

I. It will develop a user friendly Application Program Interface for abstracting all calculations and the complexity of the search operation from the user. API will accept input and output as JSON format.

II. The draft of the website will be created using an interface design program.

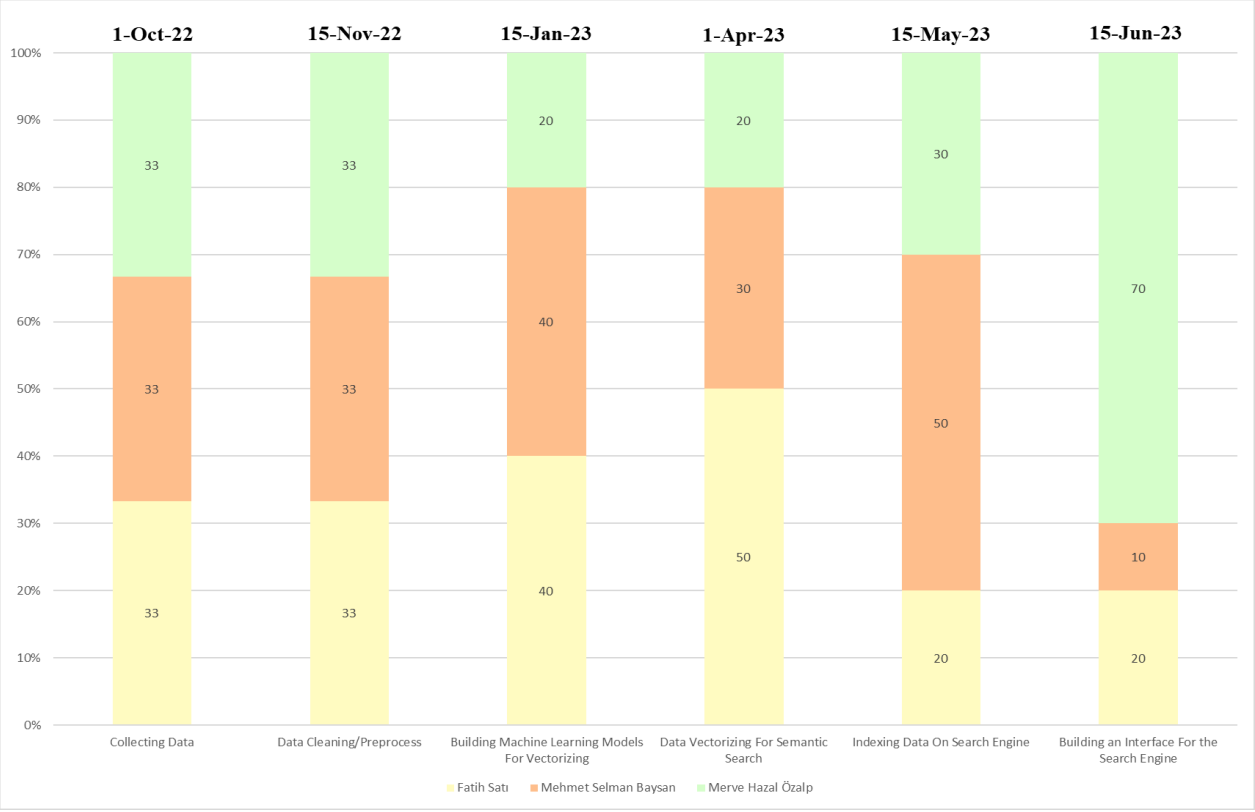
III. The website will be implemented according to the draft.

IV. An API connection will be linked to the website.

B. Division of Responsibilities

	1.1	1.2	2.1	2.2	2.3	2.4	3.1	3.2	4.1	4.2	4.3	4.4	4.5	5.1	5.2	6.1	6.2	6.3	6.4
Fatih Satı																			
Mehmet Selman Baysan																			
Merve Hazal Özalp																			

C. Timeline With Milestones



9. Success Factors and Risk Management

A. Measurability/Measuring Success:

i. Success Factor for Objective 1:

Approx. 200000 up-to-date Supreme Court jurisprudence, constitutional court decisions and downloading all legislation. The downloaded jurisprudence are composed of texts that contain the reasoned decisions of the Supreme Court.

ii. Success Factor for Objective 2:

The documents should not contain word errors and spelling mistakes, all documents should be in the same format, the jurisprudence should be parsed under some headings and the legislation should be indexed as clauses.

iii. Success Factor for Objective 3:

Since the Machine Learning models that are used for vectorizing are trained unsupervised, they do not have a performance metric as f1-score or accuracy. For this reason, performance metric of the machine learning models that will be trained in the scope of the project is decided as the higher cosine similarity of the two synonym words than the pretrained machine learning models such as fastText and BERT.

iv. Success Factor for Objective 4:

Extraction of both fastText and Bert vectors of jurisprudence texts and matching them with the relevant document.

v. Success Factor for Objective 5:

Transferring the supreme court jurisprudence with all its fields to the search engine as a table, transferring fastText and Bert vectors to the database as separate tables by matching the document id-vector. Transfer of all legislation parsed into clauses to the search engine as a separate table.

vi. Success Factor for Objective 6:

The first three results should be directly relevant in eight out of ten search queries entered under the guidance of a law expert.

vii. Success Factor for Objective 7:

The developed interface should be bug-free, user-friendly, have a filtering feature and it should work successfully.

B. Risk Management:

- Data Collecting – Supreme Court jurisprudences are publicly available at the Supreme Court's website which is planned to be used to reach the data. If the Supreme Court abandons publishing the jurisprudences, the data may be requested directly from the Supreme Court.
- Data Indexing – It is planned to use Elasticsearch as the search engine which is open-source and free of charge. If Elasticsearch charges for the usage of the application, the search engine for the project might be changed to Apache Solr.
- Model Training – Since the training of the language models requires huge amounts of data, it is possible that the dataset used for training might be deficient. In this case, the dataset can be enriched with other law documents.
- Search Engine – Results from the search queries may take longer to return than traditional search engines since semantic search requires more complex algorithms than traditional algorithms. To solve this problem, a few features of the search engine which are not critical for the results may be removed to speed up the process.

10. Benefits and Impact of the Project

Benefits/Implications:

Lawyers may spend too much time while researching the precedent that they will cite, which causes them to lose a lot of time. Our project is aiming to reduce the time that is spent on researching which will permit them to use their time more efficiently. Academics and students will be able to use our search engine as a guide while searching for practical applications of law.

1. Scientific Impact:

As vector extraction models for the search engine Turkish fastText and Turkish BERT models will be used initially. Afterwards, a law-specific corpus will be created and Turkish Law fastText and BERT models will be trained with this corpus, and their performance comparisons will be made. Depending on the performance of the domain-specific models we have trained in the Turkish field, these studies may be a base model for future studies in the field of law. At the same time, our project

can lead to studies in other domains for Turkish. As a result of the reasons mentioned above, academic publications may be issued.

2. Economic/Commercial/Social Impact:

The output of our project is expected to be a prototype. With this study, it is aimed to increase the life quality of law practitioners and to reduce their energy and time use. On the education side, students and academics can use this search engine to use their time more efficiently in their research.

3. Potential Impact on New Projects:

The semantic search engine that will be developed in the law domain can be pioneering the search engines that will be developed in other domains.

11. References

[1] https://karararama.yargitay.gov.tr/

[2] d'Sa, A. G., Illina, I., & Fohr, D. (2020, February). Bert and fasttext embeddings for automatic detection of toxic speech. In *2020 International Multi-Conference on:"Organization of Knowledge and Advanced Technologies"(OCTA)* (pp. 1-5). IEEE. https://ieeexplore.ieee.org/abstract/document/9151853?casa_token=ie_oKKOqRNsAAAAA:om-Bcw0PKzgHzV2NtbFgHh8LbOaZqwP6RUA9i-yWEqAMZ__fgPjv684_h9tAhsT_MM5WPl9JSQ

[3] Habib, M., Faris, M., Alomari, A., & Faris, H. (2021). AltibbiVec: A Word Embedding Model for Medical and Health Applications in the Arabic Language. *IEEE Access*, *9*, 133875-133888. https://ieeexplore.ieee.org/abstract/document/9548088

[4] Luburić, N., & Ivanović, D. (2016). Comparing apache solr and elasticsearch search servers. In *Proc. of the 6th International Conference on Information Society and*

Technology (ICIST 2016) (pp. 287-291). https://www.eventiotic.com/eventiotic/files/Papers/URL/icist2016_54.pdf

[5] Hussan, B. K. (2020). Comparative study of semantic and keyword based search engines. *Advances in Science, Technology and Engineering Systems Journal*, *5*(1), 106-111. https://www.astesj.com/publications/ASTESJ_050114.pdf

[6] Platzer, C., & Dustdar, S. (2005, November). A vector space search engine for web services. In *Third European Conference on Web Services (ECOWS'05)* (pp. 9-pp). IEEE. https://ieeexplore.ieee.org/abstract/document/1595717

[7] https://www.mevzuat.gov.tr/

[8] https://pypi.org/project/ requests/

[9] https://pypi.org/project/beautifulsoup4/

[10] https://pypi.org/project/selenium/

[11] https://www.elastic.co/

[12] https://www.json.org/json-en.html

[13] https://www.regular-expressions.info/

[14] https://docs.python.org/3/library/re.html

[15] https://www.java.com/

[16] https://medium.com/@kdrcandogan/elasticsearch-nedir-45d237c29b26

[17] https://elasticsearch-py.readthedocs.io/en/v8.5.2/

[18] Liu, C. Z., Sheng, Y. X., Wei, Z. Q., & Yang, Y. Q. (2018, August). Research of text classification based on improved TF-IDF algorithm. In 2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE) (pp. 218-222). IEEE.

[19] Rahutomo, F., Kitasuka, T., & Aritsugi, M. (2012, October). Semantic cosine similarity. In The 7th international student conference on advanced science and technology ICAST (Vol. 4, No. 1, p. 1).

[20] Almeida, F., & Xexéo, G. (2019). Word embeddings: A survey. arXiv preprint arXiv:1901.09069.

[21] https://huggingface.co/dbmdz/bert-base-turkish-cased

[22] https://fasttext.cc/docs/en/crawl-vectors.html

[23] https://fasttext.cc/

[24] https://pypi.org/project/fasttext/

[25] https://fasttext.cc/docs/en/unsupervised-tutorial.html

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

[27] https://huggingface.co/models

[28] https://huggingface.co/about

[29] https://huggingface.co/docs/transformers/main/en/index

[30]https://www.elastic.co/guide/en/elasticsearch/reference/current/dense-vector.html

[31]https://www.elastic.co/guide/en/elasticsearch/reference/current/knn-search.html

[32] Lahitani, A. R., Permanasari, A. E., & Setiawan, N. A. (2016, April). Cosine similarity to determine similarity measure: Study case in online essay assessment. In *2016 4th International Conference on Cyber and IT Service Management* (pp. 1-6). IEEE.

[33] Peterson, L. E. (2009). K-nearest neighbor. *Scholarpedia*, *4*(2), 1883.

[34] https://flask.palletsprojects.com/en/2.2.x/

[35] https://www.nltk.org/

[36] https://pytorch.org/

[37] https://scikit-learn.org/stable/

[38] https://github.com/about

[39] https://www.dejure.ai/aboneol