The Open University

The Department of Mathematics and Computer Science

advanced project in computer science

Using Word Embedding method to improve information retrieval engines

Presenter: Avigail Bronznick

Supervisor: Prof. Ehud Goodes

October 2021

A proposal for this advanced project was submitted as part of the requirements for obtaining a "M.Sc. degree in Computer Science" at the Open University

Table of contents

Overview

Current information retrieval (IR) approaches do not formally capture the explicit meaning of a keyword query but provide a comfortable way for the user to specify information needs on the basis of keywords. Currently, most web search engines are however based on purely statistical techniques. While they are not able to figure out the meaning of a query, they can provide answers by returning the statistically most appropriate answer to a user's query—based on some measures for computing similarity in vector space.

There are different sets of challenges when building an enterprise service, than internet search engines such as Google. The amount of content in the organizations is much smaller, so it's easier to index and operate on them, but they are also more narrow in the range making finer distinctions more important. As we mentioned before, keyword search does a reasonable job, but it requires that the user query words will exist in the corpus documents. Keyword search has no understanding of what the users are searching for (it can only match words). The search results can be greatly improved if the search engine will understand the user's queries intent and can use the context of terms within a document. This approach is called 'semantic search' and can give more relevant results since semantic search is based on the meanings of words rather than just the words themselves.

Word Embedding

Word embeddings are mathematical structures that represent a collection of words. This structure captures the context of a word or phrase plus its semantic and syntactic relation to other words. Each word is represented as a vector, so that a computer can do calculations with it. A vector is a mathematical object that has both magnitude and direction. Not only the word is represented as a vector but All the words across a big collection of documents are also represented as vectors, and they all exist within a vector space. The length (magnitude) and direction of the vector represent a particular word. Words that have similar meanings have similar vector representations and therefore end up close to each other in the vector space.

Word2Vec Embedding

Mikolov et al. introduced word2vec - a novel word-embedding procedure that learns a vector representation for each word using a (shallow) neural network language model. Specifically, they propose a neural network architecture (the skip-gram model) that consists of an input layer, a projection layer, and an output layer to predict nearby words. Each word vector is trained to maximize the log probability of neighboring words in a corpus, i.e., given a sequence of words $w_1, \ldots, w_t$

$$\frac{1}{T}\sum_{t-1}^{T}\sum_{j \in nb(t)}^{T} log\, p(w_j | w_t)$$

where nb(t) is the set of neighboring words of word wt and $p(w_i | w_t)$is the hierarchical softmax[1] of the associated word vectors $v_{wj}$and v_{wt}. [4]

Instead of using surrounding words to predict the center word, as with CBow Word2Vec, Skip-gram Word2Vec uses the central word to predict the surrounding words. The skip-gram objective function sums the log probabilities of the surrounding n words to the left and right of the target word wt

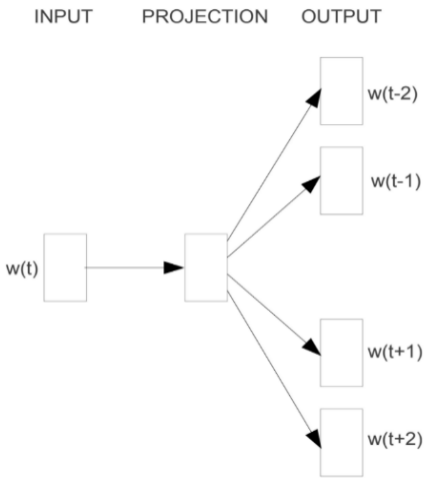


Figure 1: skip-gram

[1] The softmax func turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1

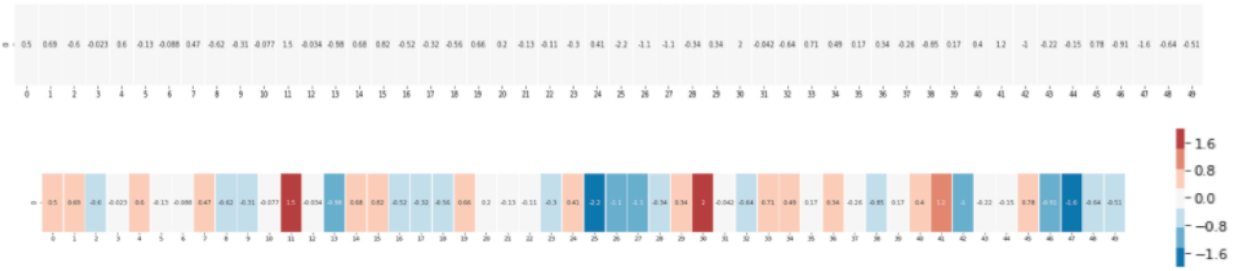
Due to its surprisingly simple architecture and the use of the hierarchical softmax, the skip-gram model can be trained on a single machine on billions of words per hour using a conventional desktop computer. The ability to train on very large data sets allows the model to learn complex word relationships such as: vec(Japan) - vec(sushi) + vec(Germany) ≈ vec(bratwurst) and vec(Einstein) -vec(scientist) + vec(Picasso) ≈ vec(painter). Mikolov et al. showed that proportional analogies (a is to b as c is to d) can be solved by finding the vector closest to the hypothetical vector calculated as c - a + b.

Word2vec input is a text corpus and its output is a set of vectors: feature vectors that represent words in that corpus - a vocabulary in which each item has a vector attached to it, which can be fed into a deep-learning net or simply queried to detect relationships between words.

For example - the word king vector is a list of 50 numbers:

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813
, 0.47377 , -0.61798 , -0.31012 , -0.076666, 1.493 , -0.034189, -0.98173 ,
0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 ,
-0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 ,
-0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 ,
0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 ,
-0.51042 ]
```

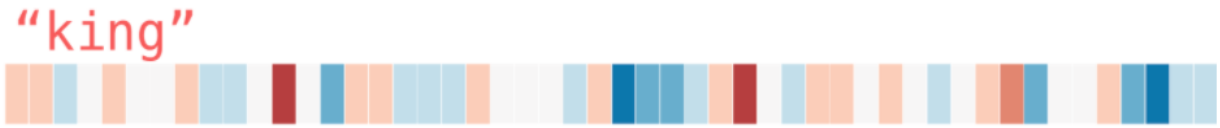
to visual it:



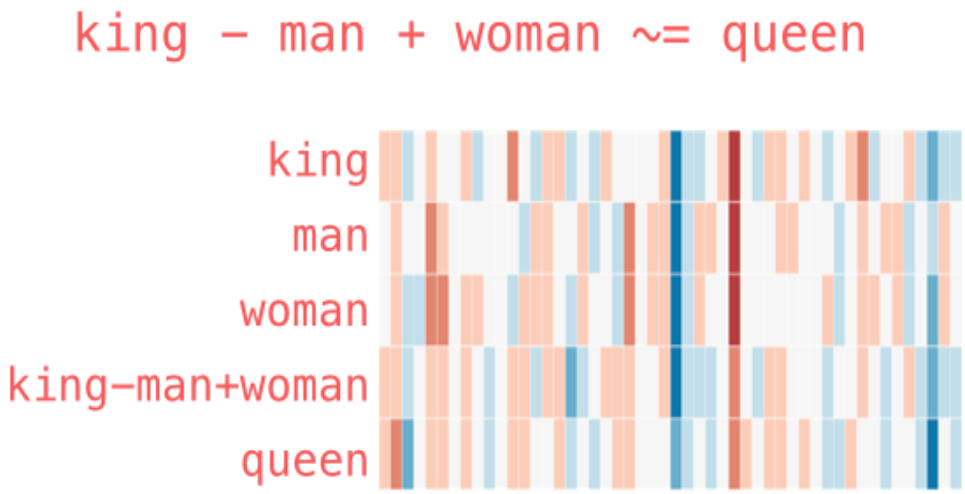
Where the color encodes the cells according to their values (red if they are close to 2, white if they are close to 0, blue if they are close to -2)

A good representation of a word via embedding is by capturing the semantic feature of the word beyond the text that represents the word, another feature that indicates about the quality of the embedding is the ability of capturing the relationships between pairs of words, hierarchies and analogies such as finding male to female ratio

“King” against other words



We can see that the words “Man” and “Woman” are much more similar to each other than either of them is to “king”.



We can see the proportional analogies (a is to b as c is to d) in this example by finding the vector closest to the hypothetical vector calculated as c - a + b.

In the above example the vectors having a dimension of 50, today it is customary to represent the vectors in a dimension of 100 to 3K.

Words of a Feather

Algorithms, in general, do not understand text, they aren't familiar with human language and therefore the first pragmatic step for using ML on text is to represent the text as numbers. The vector space is key to how semantic search works. Word embeddings can have dozens or hundreds of dimensions. It is not something we can visualize or even imagine easily, but machines can use multi-dimensional spaces to calculate things like word similarity. The vector representation is learned by considering all of the words across all of the documents to understand the context each word appears in. There are different approaches for projecting words into a vector space, but they all rely on the fact that in natural language, words with similar meanings tend to appear in similar contexts.

The purpose of the project

Often, for less common queries, we get a limited number of results by using classic methods of retrieving information, even if there are documents that can satisfy the query - but because there is no literal equality between the user query keywords and the document words. (Those documents will not be returned)

In order to increase the recall set (potential documents for return), and to provide the most relevant documents to the user as the first results position, we will create a textual search engine that incorporates the technique of Word Embedding.

In this project we will compare the text retrieval mechanism versus the semantic retrieval mechanism (word embedding as a method), and focus on improving the **relevance** of search results by using Word embedding techniques. We will also analyze the search response times as a result of using these methods.

System design

We will develop a search engine that includes information retrieval capabilities in textual relevance or by using a Word Embedding technique.

<u>The system includes two main phases:</u>

1. Data indexing

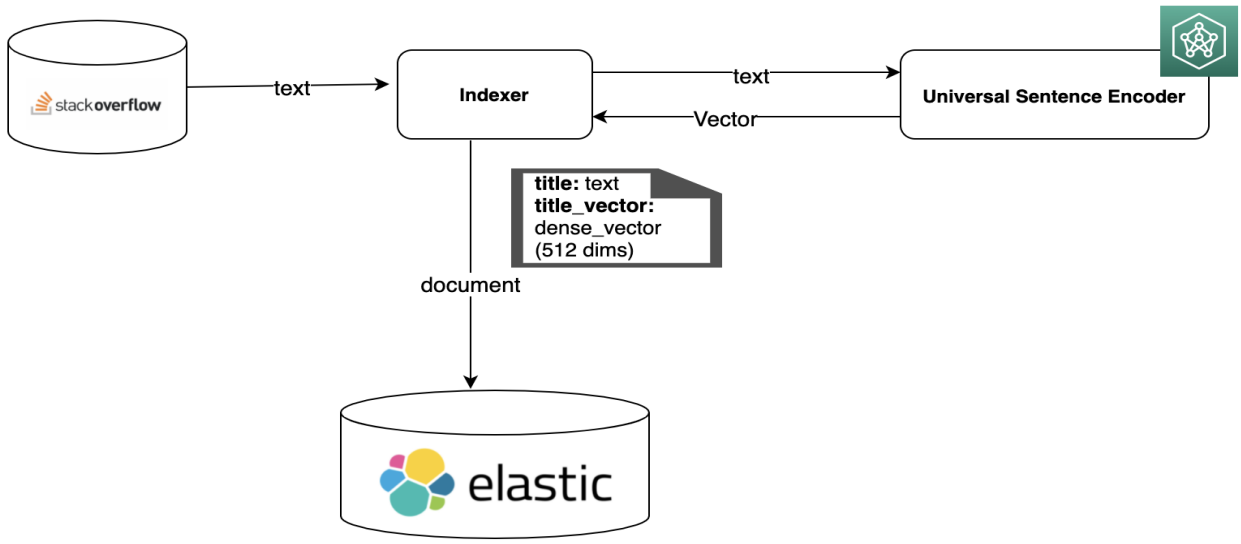


Figure 2: Indexer architecture

I chose to use Elasticsearch as a search engine which enables information retrieval capabilities according to both methods.
The way to achieve a particular method using elastic is in the way that the data is kept within the index and the specific type of the field (will be expanded at the implementation part)

The indexing phase of the system included - that for each document that was collected from Stackoverflow database in a text form, the “indexer” extracted the title of the document and send it to the “Universal Sentence Encoder” [3] model, in order to generate a numerical vector using the universal sentence encoder model. The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512 dimensional vector.
After obtaining the vector from the model, the "indexer" enriched the input document (that collected from stackoverflow) with a new field named “titel_vector” and pushed the document into elasticsearch (so that each document contains a title as text, and a title represented as a 512 dimensions vector)

2. Information retrieval capabilities

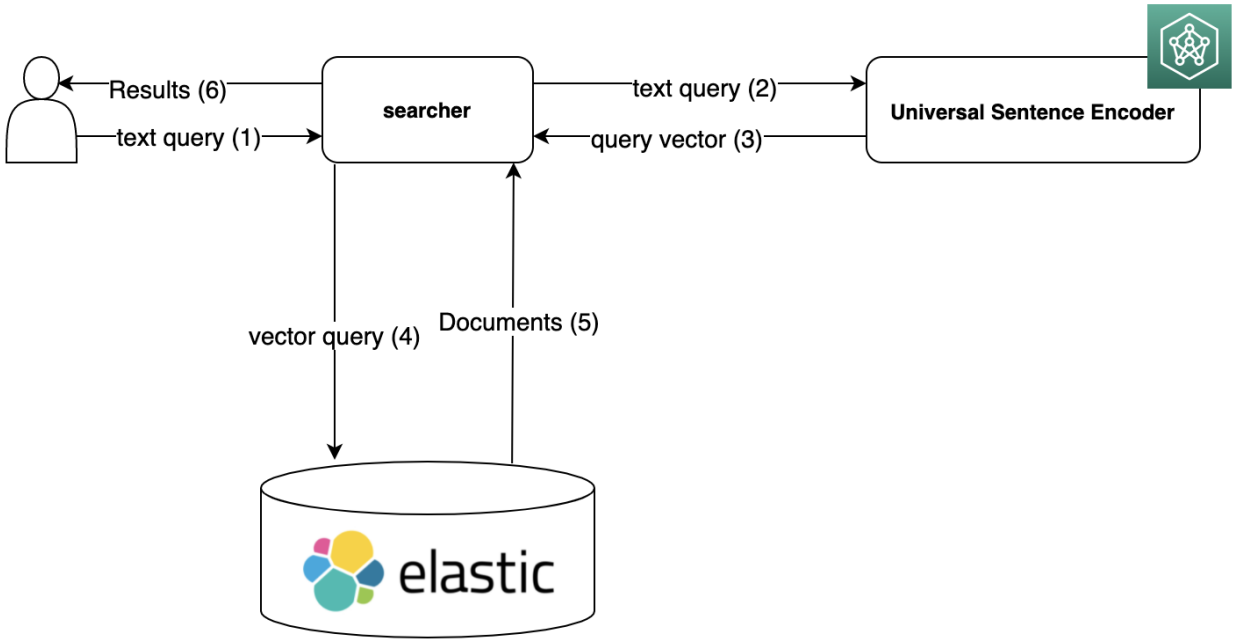


Figure 3: Searching architecture with word embedding technique

The flow of searching via word embedding technique includes:

1. Receiving a text query from the user
2. Converting the query representation from text to vector
3. Requesting the elasticsearch database, by performing cosine similarity between the user query vector (from phase 2) to the titles vectors documents stored in the elastic
4. Returning the relevant documents (in their textual representation) to the user

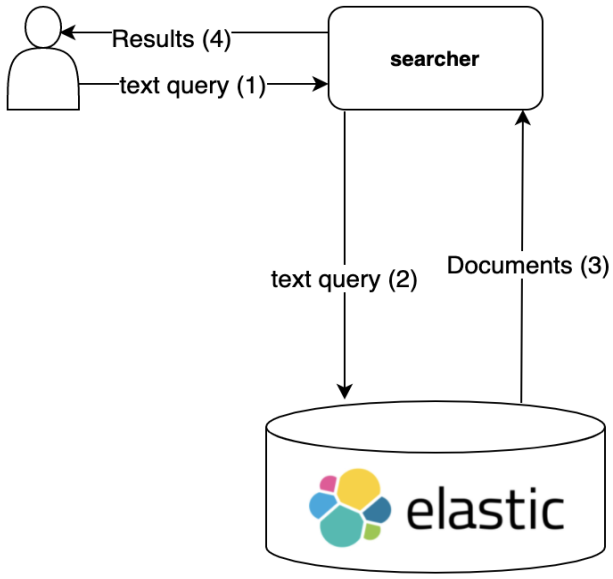


Figure 4: Searching architecture with textual retrieval technique

The flow of searching via textual technique includes:

5. Receiving a text query from the user
6. Requesting the elasticsearch database by performing a “match” query (based on the tf/ idf method)
7. Returning the relevant documents to the user

The project contains the following parts:

1. Collecting documents on which the information will be retrieved on
2. Creating a database which will contain the data (from step 1) and allow the desired functionality of information retrieval:
    a. Textual retrieval
    b. Semantic search (using word embedding method) retrieval
3. Indexing the data (from step 1) and inserting it to the database (from step 2) to allow information retrieval
4. Implementation of a search engine - that allows searching documents in the two different methods
5. creating an iteration that includes search retrieval results by the two difference methods, and manually tag the results
6. Calculating the relevance score of the iteration

Implementation

Data indexing

First I installed the elasticsearch database (version 7.13.2) [1] locally on my computer. Afterwards I created a large collection of questions documents (taken from the open database of StackoverFlow via kaggle [2]), were each document contains the fields:

1. Question title (text field)
2. Question body (text field)
3. Question **vector** title (vector of 512 dimensions - of real numbers)

I created an index in elastic with the mapping:

```
PUT posts
{
  "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
  },
  "mappings": {
    "dynamic": "true",
    "_source": {
      "enabled": "true"
    },
    "properties": {
      "user": {
        "type": "keyword"
      },
      "creationDate": {
        "type": "date"
      },
      "title": {
        "type": "text"
      },
      "title_vector": {
        "type": "dense_vector",
        "dims": 512
      },
      "questionId": {
        "type": "keyword"
      },
      "answerId": {
        "type": "keyword"
      },
      "acceptedAnswerId": {
        "type": "keyword"
      },
      "tags": {
        "type": "keyword"
      },
      "body": {
        "type": "text"
      },
      "type": {
        "type": "keyword"
      }
    }
  }
}
```

Figure 5: elastic index mapping

Configuration due to the fact that elastic is a distributed system:

- "number_of_shardes" - the numbers of nodes elastic database has
- "number_of_replicas" - the number of extra copies each documents has (2 copies overall)

Index mapping:

- The "Title" field type is **text**, which means the elastic analyzed this field for full-text search
- The "title_vector" field type is **dense_vector** (with 512 dimension), which means that this field support vector functions and vector access methods

The rest of the fields are part of the document in StackOverflow so they are indexed as well, we will use some of them to understand the meaning of the title when needed (e.g. using the "body" field)

Before indexing the question documents into the elastic index, I download the "Universal Sentence Encoder" [3] from "tenserFlow" and for each document I sent the question title to the model, in order to generate a numerical vector using the universal sentence encoder model (as mentioned before)

Indexing flow:

1. Receiving a question document from StackOverFlow
    a. output JSON doc - for example


```
{"user":"4481","tags":[".net","dependency-injection","inversion-of-control","castle-windsor","ioc-container"],"questionId":"148908","creationDate":"2008-09-29T14:28:52.800","title":"Which Dependency Injection Tool Should I Use?","acceptedAnswerId":"151921","type":"question","body":"I am thinking about using Microsoft Unity for my Dependency Injection tool in our User Interface. Our Middle Tier already uses Castle Windsor, but I am thinking I should stick with Microsoft. Does anyone have any thoughts about what the best Dependency Injection tool is? Autofac Castle MicroKernel/Windsor PicoContainer.NET Puzzle.NFactory Spring.NET StructureMap Ninject Unity Simple Injector NauckIT.MicroKernel WINTER4NET ObjectBuilder "}
```

2. Sending the title text to the sentence embedding model
    a. Input - text
    b. Output - embedding vector of 512 dimension - for example
    [0.06592217087745667, 0.017203116789460182 …
    -0.03768639266490936, ,0.053689248859882355]
    (What is shown here are the first and last two dimensions due to space limit)
3. Creating a JSON document, including the fields from step 1 + the “title_vector” from step 2 - and indexing the document to elastic field

For example - selecting the document from step 1

```
GET posts/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term": {
            "questionId": "148908"
          }
        }
      ]
    }
  }
}
```

Figure 6: elastic query - select document by id

The output (the “title_vector” field is collapsed due to space limit)


```
{
   _index: "posts",
   _type: "_doc",
   _id: "5sCUZnoBTAJax3aofp5Z",
   _version: 1,
   _seq_no: 8608,
   _primary_term: 1,
   found: true,
 - _source: {
      user: "4481",
    - tags: [
          ".net",
          "dependency-injection",
          "inversion-of-control",
          "castle-windsor",
          "ioc-container"
      ],
      questionId: "148908",
      creationDate: "2008-09-29T14:28:52.800",
      title: "Which Dependency Injection Tool Should I Use?",
      acceptedAnswerId: "151921",
      type: "question",
      body: "I am thinking about using Microsoft Unity for my Dependency Injection tool in our User Interface. Our Middle Tier already uses Castle Windsor, but I am thinking I
      should stick with Microsoft. Does anyone have any thoughts about what the best Dependency Injection tool is? Autofac Castle MicroKernel/Windsor PicoContainer.NET
      Puzzle.NFactory Spring.NET StructureMap Ninject Unity Simple Injector NauckIT.MicroKernel WINTER4NET ObjectBuilder ",
    + title_vector: […]
   }
}
```


Figure 7: elastic document output

Information retrieval capabilities

After indexing all the question documents, the elastic index contained 18, 848 documents
http://localhost:9200/posts/_count


```
{
    count: 18848,
  - _shards: {
        total: 2,
        successful: 2,
        skipped: 0,
        failed: 0
    }
}
```


Figure 8: elastic query count result

I Created two methods for retrieval the information based on a user query

1. "embeding_search" [GitHub link]
2. "textual_search" [GitHub link]

Searching via word embedding technique flow (figure 3)

1. User enters a search query into the "searcher" system (via a console window)
2. The "searcher" sends the user search query from (step 1) to the sentence embedding model and received an embedding vector with 512 dimension
3. The searcher created a SQL for elastic, using the vector value from step 2 and inserting the vector from step 2 into the "user_query_vector" placeholder in the below query (figure 9)
4. The user gets the first x results (describe below)

```
GET posts/_search
{
  "_source": {
    "includes": [
      "title",
      "body"
    ]
  },
  "min_score": 1.5,
  "track_total_hits": true,
  "query": {
    "should": [
      {
        "script_score": {
          "query": {
            "match_all": {}
          },
          "script": {
            "source": "cosineSimilarity(params.query_vector,
              doc['title_vector']) + 1.0",
            "params": {
              "query_vector": [user_query_vector]
            }
          }
        }
      }
    ]
  }
}
```

Figure 9: a template of an elastic query for performing cosine Similarity

Elastic query explanation:

- "**_source**" - for each retrieval documents, only the fields that inside the “includes” array will return
- “**track_total_hits**” - generally the total hit count can’t be computed accurately without visiting all matches, which is costly for queries that match lots of documents. The track_total_hits parameter allows control of how the total number of hits should be tracked, Because it is set to true, the search response will always track the number of matches that match the query exactly
- “**min_score**” - exclude documents which have a “_score” (elastic relevant score) less than the minimum specified in min_score (in our project min score is define to 1.5)
- **Query** - ranking the questions documents based on their similarity to the user’s query, using cosine Similarity function [4] elastic provides

The “**script_score**” query is designed to wrap a restrictive query, and modify the scores of the documents it returns. However, we’ve provided a “**match_all**” query, which means the script will be run over all documents in the index. This is a current limitation of vector similarity in elasticsearch, therefore I added the “**min_score**” parameter to provide some relevance threshold.

For example:

```
Enter:
            1) for querying the collection through word-embedding
similarity match
            2) for querying the collection through textual match
            3) for creating the evaluation data
            4) to exist: 1
Enter query: zipping up files

3190 total hits.
embedding time: 60.39 ms
search time: 479.87 ms
id: v8CTZnoBTAJax3ao-loH, score: 1.7865233
{'title': 'Compressing / Decompressing Folders & Files', 'body': 'Does
anyone know of a good way to compress or decompress files and folders in C#
quickly? Handling large files might be necessary. '}

id: osCUZnoBTAJax3aohKPK, score: 1.773902
```

```
{'title': 'Partial .csproj Files', 'body': 'Is it possible to split the
information in a .csproj across more than one file? A bit like a project
version of the partial class feature. '}

id: EcCUZnoBTAJax3aogqLz, score: 1.7686331
{'title': 'How to avoid .pyc files?', 'body': 'Can I run the python
interpreter without generating the compiled .pyc files? '}

id: I8CUZnoBTAJax3aoLHME, score: 1.7433983
{'title': 'Files on Windows and Contiguous Sectors', 'body': "Is there a
way to guarantee that a file on Windows (using the NTFS file system) will
use contiguous sectors on the hard disk? In other words, the first chunk of
the file will be stored in a certain sector, the second chunk of the file
will be stored in the next sector, and so on. I should add that I want to
be able to create this file programmatically, so I'd rather not just ask
the user to defrag their harddrive after creating this file. If there is a
way to programmatically defrag just the file that I create, then that would
be OK too. "}
```

Explaining the example

1. User choose option 1 (searching with word-embedding capabilities)
2. Entered a search query - “zipping up files”
3. The searcher returned to the user 4 documents that are closest to the query he was looking for
4. The response from the searcher is:
    a. Total hits (number of documents where there simulitarty score is higher than 1.5)
    b. Embedding time in ms - converting the user query to a word embedding vector
    c. Search time - performing cosine similarity between the user query vector to the documents in the corpus

    Metrics for the above example search query:

    ```
    3190 total hits.
    embedding time: 60.39 ms
    search time: 479.87 ms
    ```

d. for each document the searcher return
    i. the document elastic id and the score (elastic inner fields)
    ii. The json document (the fields that are mentioned inside the “includes” section)

    E.g. - first documents data:
    id: v8CTZnoBTAJax3ao-loH, score: 1.7865233
    {'title': 'Compressing / Decompressing Folders & Files', 'body': 'Does anyone know of a good way to compress or decompress files and folders in C# quickly? Handling large files might be necessary. '}

We can see from the above example that the first result is amazing in terms of information retrieval, but the rest of the results are on the verge of being irrelevant - we will expand on the above after cutting off the evolution phase.

Searching via textual technique flow (figure 4)

1. User enters a search query int the “searcher” system (via Console window)
2. The searcher created a SQL for elastic using the input value from step 1 and inserting it into the “user_query” placeholder in the below query (figure 10)
3. The user gets the first x results (describe below)

```
GET posts/_search
{
  "_source": {
    "includes": [
      "title",
      "body"
    ]
  },
  "track_total_hits": true,
  "query": {
    "should": [
      {
        "match": {
          "title" : user_query
        }

      }
    ]
  }
}
```

Figure 10: a template of an elastic query for performing textual search

- “match” - The match query is the standard query for performing a full-text search, including options for fuzzy matching.

For example:

```
Enter:
            1) for querying the collection through word-embedding
similarity match
            2) for querying the collection through textual match
            3) for creating the evaluation data
            4) to exist: 2
Enter query: zipping up files

531 total hits.
search time: 196.05 ms
id: LcCUZnoBTAJax3aoVoU_, score: 9.472768
{'title': 'CruiseControl + Starteam: not picking up all files', 'body':
"Our CruiseControl system checks out from starteam. I've noticed that it is
sometimes not checking out new versions of files, only added files. Does
anyone know why this is? "}

id: hMCUZnoBTAJax3aoQnrH, score: 6.5083327
{'title': 'Packaging up Tomcat', 'body': "In my job we have to deploy an
application on various environments. It's a standard WAR file which needs a
bit of configuration, deployed on Tomcat 6. Is there any way of creating a
'deployment package' with Tomcat so that you just extract it and it sets up
Tomcat as well as your application? I'm not sure that creating a .zip file
with the Tomcat folder would work! It certainly wouldn't install the
service. Suggestions welcome! I should note that - at the moment - all apps
are deployed on Windows servers. Thanks, Phill "}

id: lcCUZnoBTAJax3aoDGKH, score: 6.2230253
{'title': 'Cleaning up RTF text', 'body': "I'd like to take some RTF input
and clean it to remove all RTF formatting except \\ul \\b \\i to paste it
into Word with minor format information. The command used to paste into
Word will be something like:
oWord.ActiveDocument.ActiveWindow.Selection.PasteAndFormat(0) (with some
RTF text already in the Clipboard)
{\\rtf1\\ansi\\deff0{\\fonttbl{\\f0\\fnil\\fcharset0 Courier New;}}
{\\colortbl ;\\red255\\green255\\blue140;}
\\viewkind4\\uc1\\pard\\highlight1\\lang3084\\f0\\fs18 The company is a
```

```
global leader in responsible tourism and was \\ul the first major hotel
chain in North America\\ulnone to embrace environmental stewardship within
its daily operations\\highlight0\\par Do you have any idea on how I can
clean up the RTF safely with some regular expressions or something? I am
using VB.NET to do the processing but any .NET language sample will do. "}

id: CMCUZnoBTAJax3aoIW9Y, score: 6.2230253
{'title': 'Setting Up MySQL Triggers', 'body': "I've been hearing about
triggers, and I have a few questions. What are triggers? How do I set them
up? Are there any precautions, aside from typical SQL stuff, that should be
taken? "}
```

Explaining the example

1. User choose option 2
2. Entered a search query - “zipping up files”
3. The searcher returned to the user 4 documents that are closest to the query he was looking for
4. The response from the searcher is:
    a. Total hits (number of documents where there text match to the query)
    b. Search time - performing textual matching between the user query to the documents

       Metrics for the above example search query:

       ```
       531 total hits.
       search time: 196.05 ms
       ```

    c. for each document the searcher return
        i. the document elastic id and the score (elastic inner fields)
        ii. The json document (the fields that are mentioned inside the “includes” section)

       E.g - First documents data:

       id: LcCUZnoBTAJax3aoVoU_, score: 9.472768

       {'title': 'CruiseControl + Starteam: not picking up all files', 'body': "Our CruiseControl system checks out from starteam. I've noticed that it is sometimes not checking out new versions of files, only added files. Does anyone know why this is? "}

Example comparison between the two methods:

- Embedding search took much more time than the standard textual matching
    - 480 ms versus 200 ms
- Total hits number is higher at the first method (embedding) than the second method due to the limitation of the vector comparison capabilities in elastic, and the manual threshold addition made by me (a more accurate accuracy of the threshold is needed in a production system)
- The results in both methods are not amazing, probably due to the limited database we have (runs locally, with a **small** amount of documents), but, we can see the potential of the first method - which retrieved a document based on an understanding of the query and not just based on the visibility of the words themselves

Search Evaluation

Search engine evaluation is in a very basic sense, all about how well a search engine serves its users, or, from the other point of view, how satisfied users are with a search engine.

For any evaluation of search engine performance it is crucial to understand user behavior. This is true at the experimental design stage, when it is important to formulate a study object which corresponds to something the users do out there in the real world, as well as in the evaluation process itself, when one needs to know how to interpret the gathered data.

Due to the fact that the information about the user queries is not available to me, I randomly selected ~50 queries from the database itself and used the title question as the “user search query”.
The random selection was made via:

```
GET posts
{
  "size": 50,
  "_source": {
    "includes":
    [ "title",
      "questionId"
    ]
  },
  "query": {
    "function_score": {
      "functions": [
        {
          "random_score": {
            "seed": "2524657 4i543785o4"
          }
        }
      ]
    }
  }
}
```

Figure 11: an elastic query that select randomly 50 docs

Note:
The “**random_score**” generates scores that are uniformly distributed from 0 up to but not including 1. By default, it uses the internal Lucene doc ids as a source of randomness

Due to the fact that the queries were selected from the database itself - I added a filter section to the retrieval query of the document id itself from the returned results (so that the query document itself will not be returned)

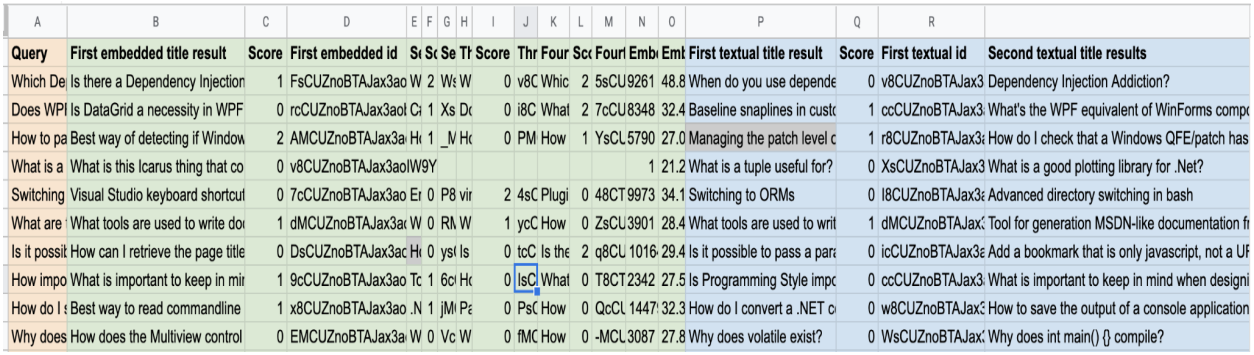
Building the iteration

- Creating a csv file:
    - File name: *"iteration_<curernt_date>.csv"*
    - Headers:
        - *Query*
        - *X embedded title result (x = first/ second/ third/ fourth)*
        - *X embedded id*
        - *X Score*
        - *Embedded total results*
        - *Embedded request time*
        - *X textual title result*
        - *X textual id*
        - *X score*
        - *Textual total results*
        - *Textual request time*
    - Notes:
        - In total 29 columns
        - The column scores were created with an empty value - and during the manual tagging phase the appropriate score was entered
- For each randomly selected query, both search methods were enabled, and the entries were entered in the appropriate places in the file
    - Notes:
        - The metrics were also inserted into the appropriate columns
        - Some searches returned less than 4 results in total and therefore, some of the columns were left blank

Tagging the iteration

In order to facilitate the tagging process, I added to the csv file (that was created in the previous step) colors, Which differentiate between the "query" and the returned results, (there is also a color difference between the results of the two different methods)

A peek at what it looks like:



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
Query	**First embedded title result**	**Score**	**First embedded id**	**Sc**	**Sc**	**Se**	**Tł**	**Score**	**Thr**	**Four**	**Sc**	**Four**	**Emb**	**Emb**	**First textual title result**	**Score**	**First textual id**	**Second textual title results**
Which De	Is there a Dependency Injection	1	FsCUZnoBTAJax3ao	W	2	Ws	W	0	v8C	Whic	2	5sCU	9261	48.8	When do you use depende	0	v8CUZnoBTAJax3	Dependency Injection Addiction?
Does WPI	Is DataGrid a necessity in WPF	0	rcCUZnoBTAJax3aol	Ci	1	Xs	Dc	0	i8C	What	2	7cCU	8348	32.4	Baseline snaplines in custo	1	ccCUZnoBTAJax3	What's the WPF equivalent of WinForms compo
How to pa	Best way of detecting if Window	2	AMCUZnoBTAJax3a	He	1	_M	Hc	0	PM	How	1	YsCL	5790	27.0	Managing the patch level c	1	r8CUZnoBTAJax3	How do I check that a Windows QFE/patch has
What is a	What is this Icarus thing that co	0	v8CUZnoBTAJax3aol	W	9Y								1	21.2	What is a tuple useful for?	0	XsCUZnoBTAJax3	What is a good plotting library for .Net?
Switching	Visual Studio keyboard shortcut	0	7cCUZnoBTAJax3ao	Er	0	P8	vir	2	4sC	Plugi	0	48CT	9973	34.1	Switching to ORMs	0	l8CUZnoBTAJax3	Advanced directory switching in bash
What are	What tools are used to write do	1	dMCUZnoBTAJax3a	W	0	RM	W	1	ycC	How	0	ZsCU	3901	28.4	What tools are used to writ	1	dMCUZnoBTAJax	Tool for generation MSDN-like documentation fr
Is it possit	How can I retrieve the page title	0	DsCUZnoBTAJax3ac	He	0	ysi	Is	0	tcC	Is the	2	q8CU	1016	29.4	Is it possible to pass a para	0	icCUZnoBTAJax3	Add a bookmark that is only javascript, not a UF
How impo	What is important to keep in mir	1	9cCUZnoBTAJax3ao	Tc	1	6c	Hc	0	IsC	What	0	T8CT	2342	27.5	Is Programming Style impo	0	ccCUZnoBTAJax3	What is important to keep in mind when designi
How do I s	Best way to read commandline	1	x8CUZnoBTAJax3ao	.N	1	jM	Pa	0	PsC	How	0	QcCL	1447	32.3	How do I convert a .NET c	0	w8CUZnoBTAJax	How to save the output of a console application
Why does	How does the Multiview control	0	EMCUZnoBTAJax3a	W	0	Vc	W	0	fMC	How	0	-MCL	3087	27.8	Why does volatile exist?	0	WsCUZnoBTAJax	Why does int main() {} compile?

Figure 12: iteration results snapshot

- The tagging was performed by me
- Each result got the value between 0-2
    - **0** - not relevant
    - **1** - Understandable why the result is back - but not entirely relevant or - the body of the question had to be read because the title was not enough (the identifying value of the document was returned in order to enable advanced comprehension capability)
    - **2** - relevant

Evaluation results

Discounted cumulative gain (DCG) [1] is a measure of ranking quality. In information retrieval, it is often used to measure effectiveness of web search engine algorithms or related applications. Using a graded relevance scale of documents in a search-engine result set, DCG measures the usefulness, or gain, of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom, with the gain of each result discounted at lower ranks.

Two assumptions are made in using DCG and its related measures.

1. Highly relevant documents are more useful when appearing earlier in a search engine result list (have higher ranks)
2. Highly relevant documents are more useful than marginally relevant documents, which are in turn more useful than non-relevant documents

We will calculate the DCG of the textual method and the word embedding method of the evaluation (from above) and we will compare the result.

Calculation:

- Cumulative Gain (CG) is the sum of the graded relevance values of all results in a search result list:

  $$CG_p = \sum_{i=1}^{p} rel_i$$

  Where rel_i is the graded relevance of the result at position i

- Discounted Cumulative Gain (DCG) :

  The premise of DCG is that highly relevant documents appearing lower in a search result list should be penalized as the graded relevance value is reduced logarithmically proportional to the position of the result

  $$DCG_p = \sum_{i=1}^{p} \frac{rel_i}{log_2(i+1)}$$

- In our evaluation, $p = 4$ and rel_i is one of the values 0, 1 or 2

- Auxiliary calculation :
    - Position $i = 1$, $log_2(1 + 1) = 1$
    - Position $i = 2$, $log_2(2 + 1) = 1.58$
    - Position $i = 3$, $log_2(3 + 1) = 2$
    - Position $i = 4$, $log_2(4 + 1) = 2.32$

For example:
For the query “Which Dependency Injection Tool Should I Use?”

- The relevant score of the **embedding iteration**
    - returned 4 documents, which were labeled with relevant score of: $D_1 = 1,\ D_2 = 2, D_3 = 0, D_4 = 2$
    - CG_p = 1+2+0+2 = 5
    - DCG_p = 1+1.3+0+0.8613531161 = 3.1 (sum DCG)
- The relevant score of the **textual iteration**
    - returned 4 documents, which were labeled with relevant score of: as: $D_1 = 0,\ D_2 = 0, D_3 = 0, D_4 = 2$
    - CG_p = 0+0+0+2 = 2
    - DCG_p = 0+0+0+0.8613531161 = ~0.9

Ignoring recall issue:

Since not every query result returns the same number of documents, we do not want to “punish” queries that return less than x documents (in our case 4)

We will calculate the “max relevant” score for each query result set in which we treat each document as if it had received the maximum relevant score (that is 2)

Calculation:
Max Score for each document with relevant score = 2, per position

- Max score, position $i = 1, \frac{2}{log_2(1+1)} = 2$
- Max score, position $i = 2, \frac{2}{log_2(2+1)} = 1.26$
- Max score, position $i = 3, \frac{2}{log_2(3+1)} = 1$
- Max score, position $i = 4, \frac{2}{log_2(4+1)} = 0.86$

Max Score for each query result (where documents are tagged with relevant score = 2)

MAX DCG score:

- Search query return of 4 documents:
  - DCG_p = 2 + 1.26 + 1+0.86 = 5.12
- Search query return of 3 documents
  - DCG_p = 2 + 1.26 + 1 = 4.26
- Search query return of 2 documents
  - DCG_p = 2 + 1.26 = 3.26
- Search query return of 1 documents
  - DCG_p = 2

Now, we will calculate the "relative query score" by normalizing the DCG.
For each query result, we will divide the DCG Sum value by the MAX DCG score (depending on the number of results returned to the query)

Back to previous example:

- Embedding mechanism received 4 documents for the query "Which Dependency Injection Tool Should I Use?"
  - The DCG sum is 3.1 (based on the relevant score of each document)
  - The " "relative query score" is: 3.1/ 5.12 = 0.605
- Textual mechanism received 4 documents for the query "Which Dependency Injection Tool Should I Use?"
  - The DCG sum is 0.9 (based on the relevant score of each document)
  - The " "relative query score" is: 0.9/ 5.12 = 0.1681275363

Total Relevant score

$$\frac{\sum_{i=1}^{p} relative\ query\ score}{p}$$

Where $p$is the number of queries that participated in the evaluation

The iteration total relevant score that i tagged is:

- For the Word Embedding mechanism : 24.84%
- For the Textual mechanism : 15.5%

These numbers are very small in relation to the relevance score we expect from a search engine in general, but we can see very clearly a proof of feasibility, as no research has been done on the types of users and how the queries they use, the sample of questions (building the iteration) was taken from the database itself and not from a real data of users queries. By using a basic income of a word embedding mechanism we improved the overall search engine relevance score by 60% from the textual relevance score.

Response time metric:

Average response time metric:

- Word embedding technique - 27.762 ms
- Textual matching technique - ~16 ms

Upper 90 percentile response time metric:

- Word embedding technique - 32.853 ms
- Textual matching technique - 19.575 ms

As we can see the response time increases when we search with the word embedding technique, the reason why is:

1. For each user query , first we are converting it from a text query to a vector query representation (for calculating the cosine similarity between the user query and the DB documents)

2. Using the “match_all” query in elastic before calculating the documents scores based on the cosine similarity script function, due to the limitation of Elastic search (mentioned in the previous section). There are many ways to deal with the above, which are not part of this work .A light touch on the subject is for example, creating clusters from the corpus of word embedding document representation (the clustering of the documents are based on the k nearest neighbors), and performing the similarity function between the user query to the corpus only in front of a specific cluster (a selected subset) of course understanding in front of which cluster subset to calculate the similarity, is a task in itself

   For example, splitting the corpus into 3 subsets, each subsets has a representative vector (whose calculation is a challenge in itself)

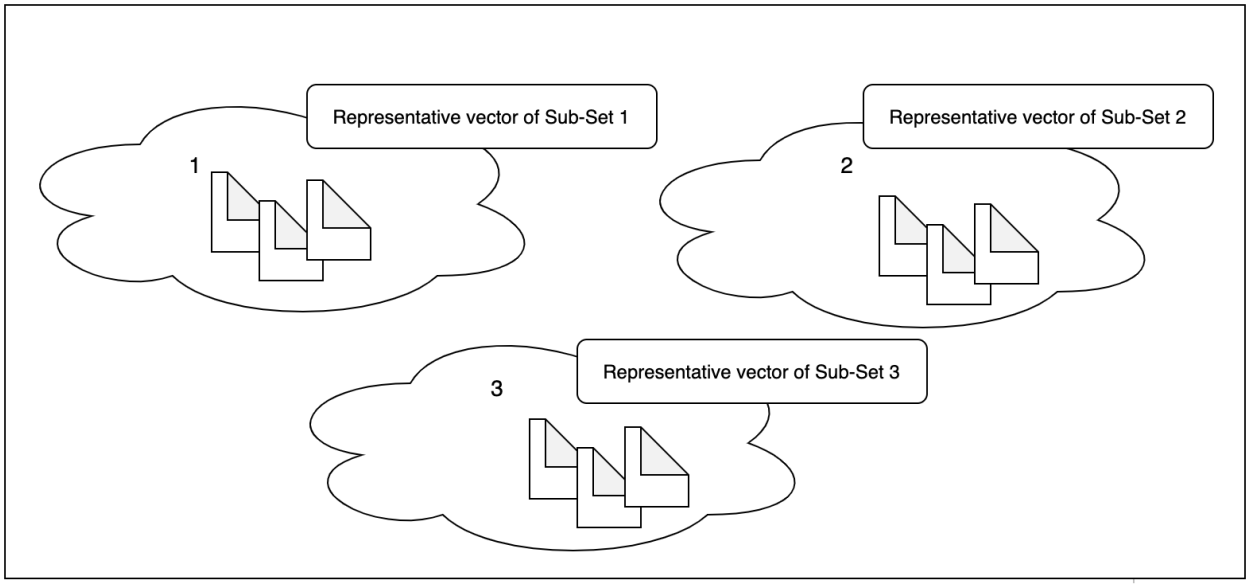


Figure 13: high-level architecture of clustering the corpus

Searching for results based on a user text query:

1. first the query will be converted into a vector representation
2. Then, a match will be made in front of the vectors that representing the subsets clusters
3. The similarity calculating between the query vector and the corpus, will be performed in front of the selected subset documents (from step 2)

This approach can reduce significantly the irrelevant documents that the word embedding method using ES returns (as we saw in the example of the “zipping up files”)

Development environment and tools

The system was developed in Python language, in a work environment of MacOS Catalina (version 10.15.7), Memory 16 GB, 2.8 GHz Quad-Core Intel Core i7.
I installed and ran locally elasticsearch version 7.13.2, and used elastic SQL to perform the information retrieval of the two main methods of the project. The representation of the title documents embedding vectors were done using the Universal Sentence Encoder that is public and available in Tensorflow-hub.

The project was developed on the purity of the open source code.

Code architecture

GitHub repository link: https://github.com/Avigailbr/text-embeddings

Main function (**main.py**) that responsible for:

1. Downloading the pre-trained embeddings from tensorflow hub
2. Creating a model object that enables to use universal-sentence-encoder model [5]
3. Creating a tensorflow session
4. Creating elasticsearch client object
5. For loops that enables
    a. Option 1 - querying the collection through word-embedding similarity match
    b. Option 2 - querying the collection through textual match
    c. Option 3- creating the evaluation data (queries was selected via the random function score mentioned above, and stored inside a “questions.csv” file inside the repository)

Elastic module (**elastic.py**) responsible for:

1. Preparing the data and Index it into the elastic database
2. Perform the search tougher a SQL elastic via the database
    a. Creating the query
    b. Manipulating the response data (metrics and documents) to the user

Embed module (**embed.py**) responsible for receiving a text title and returning an embedding that represents the text title via the tensorflow model

Iteration module (**iteration.py**) responsible for Creating the iteration end-to-end using elastic and embed modules

Sources of information

[1] elasticsearch installation -
https://www.elastic.co/downloads/elasticsearch

[2] Kaggle - Stackoverflow Database -
https://www.kaggle.com/muhammedabdulazeem/500k-stackoverflow-questions#

[3] universal-sentence-encoder-models -
https://www.kaggle.com/dimitreoliveira/universalsentenceencodermodels

[4] cosine Similarity function -
https://www.elastic.co/guide/en/elasticsearch/reference/7.6/query-dsl-script-score-query.html#vector-functions

[5] universal-sentence-encoder model
https://tfhub.dev/google/universal-sentence-encoder/2

References

[1] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzwei, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Cornell University, Universal Sentence Encoder

[2] Pavel Siroktin, CoRR abs/ 1302.2318 (February 2013), On Search Engine Evaluation Metrics

[3] Doug Turnbull, John Berryman, Manning Publications, ISMN: 8781617292774, 2016, Relevant Search: With applications for Solr and Elasticsearch

[4] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, ICLR (Workshop Poster) 2013, Efficient Estimation of Word Representations in Vector Space, ar Xiv:1301.3781