Topical Paper Classifier

Semantic Data Processing Project Report

Paweł Golik

Przemysław Olender

January 23, 2023

1 Topical paper classification

An ontology is a formal representation of concepts and their relationships to each other. It defines a common vocabulary and provides a structure for representing and organizing knowledge in a particular domain (e.g., Computer Science). Ontologies deliver a common understanding of gathered knowledge, simplify the integration and sharing of information, and help computers understand our data.

Ontology can be formulated using Resource Description Framework (RDF, and its extensions, such as RDF Schema and the Web Ontology Language - OWL). We can understand RDF-represented knowledge as a directed graph, where each RDF statement: a triple "subject-predicate-object" (e.g., [Chair] [has] [leg]), is a connection between two vertices in the graph, i.e., the subject is a starting node, the predicate is an edge, and the object is an end node. RDF also provides several serialization format types. Turtle (Terse RDF Triple Language) is the serialization format in which the OpenCS ontology [1] stores its data.

In our project, we aim to provide a tool that, for scientific articles, returns a ranking of the most relevant concepts from the OpenCS ontology based on their title and abstract. The OpenCS ontology provides knowledge about scientific publications and related entities, such as their fields of study. Our task is to return a ranking, i.e., a list of concepts present in the ontology with their scores ordered from most relevant to least relevant.

Even though the name "classification," our task is not supervised - we do not know precisely what the domains of articles used in our project are. We can only rely on our assessment of the articles based on reading them. The ontology provides us with concepts representing the possible fields of study. A hypothetical tool for solving our problem based on the provided text (title and abstract of the article) tries to find relevant concepts in the ontology.

2 Our solution

2.1 Elasticsearch

Elasticsearch [2] is a search engine based on the Lucene library. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is released as open source under the terms of the Apache License. It is often used for log analytics, full-text search, and for business intelligence use cases. It can also be used as a NoSQL data store.

Elasticsearch can be used to search text within an ontology by indexing the ontology data in Elasticsearch and then using the search capabilities of Elasticsearch to query the indexed data. This can be done by creating an index in Elasticsearch for the ontology data and then adding the data to the index using the Elasticsearch API. Once the data is indexed, it can be searched using Elasticsearch's query language, called the Query DSL, which allows for complex queries to be constructed using a JSON-based syntax.

3 Implementation

3.1 Environment

Our solution relies on a local Elasticsearch instance deployed using Docker images. The whole solution utilizes Python, and both Elasticsearch and Docker are accessed through Python APIs corresponding to these frameworks, which means no advanced knowledge about any of them is required to run the Topical Paper Classifier. Our solution was created in the Conda environment using Python 3.10.8. We recommend installing it to ensure running the Topical Paper Classifier correctly using instructions specified in the readme file. If any Elasticsearch instance is deployed, our Docker-ready solution can be used.

After downloading the project from the repository, you must visit the config.py file to adjust the configurations options; variables BASE_DIR and ONTOLOGY_DIR should be set to absolute paths to the directory with the downloaded project repository and the directory with the downloaded ontology repository. It is also possible to adjust other options, like a version of Elasticsearch images, Docker container names, ports on which Elasticsearch and Kibana will run, name of the Elasticsearch index. More information about the setup procedure is available in the readme file.

```
### paths ###
ENV_PATH = path.join(r"C:\Users\golik\.conda\envs\opencs_paperclassification")
BASE_DIR = path.join(r"C:\Users\golik\Desktop\mgr\semantic\project\
                                              opencs_paperclassification")
ONTOLOGY_DIR = path.join(r"D:\OpenCS\OpenCS")

DATA_DIR = path.join(BASE_DIR, "data")
RESULT_DIR = path.join(BASE_DIR, "results")
ONTOLOGY_CORE_DIR = path.join(ONTOLOGY_DIR, r"ontology\core" )

##############

### docker ###

# docker images
ES_IMAGE = "docker.elastic.co/elasticsearch/elasticsearch:7.12.1"
KB_IMAGE = "docker.elastic.co/kibana/kibana:7.12.1"

# containers
ES_CONTAINER_NAME = "opencs-pc-es"
KB_CONTAINER_NAME = "opencs-pc-kb"

ES_PORTS = {9200:9200, 9300:9300}
KB_PORTS = {5601:5601}

PORT = list(ES_PORTS.keys())[0]

# docker network
NETWORK = ’elastic_net’

##############

### Elastic Search index ###
IDX_NAME = ’ontology_index’
```

Figure 1: sample content of *setup.py*

First step after adjusting *config.py* file is to download Docker images with Elasticsearch containers, code to do that is present in *env_setup.ipynb* notebook. This step requires Docker running on the PC. To download the containers you have to run function *start_docker_containers()*:

```
containers = start_docker_containers()
```

Figure 2: Running function to setup Docker environment

After that images should be downloaded and running, it is now possible to access Elasticsearch GUI in the browser on http://localhost:5601/ (or other port, if it was changed in *config.py*)

In case you want to remove docker containers you can do it manually or run *remove_docker_containers()* function.

```
remove_docker_containers(containers)
```

Figure 3: Running function to remove Docker containers

For more detail about setting up the environment please visit *env_setup.ipynb* notebooka and *readme.md*.

3.2 Elasticsearch Index

An index is like a 'database' in a relational database. It has a mapping that defines multiple types [3]. In order to allow classifying article topics, it is necessary to build an index containing all the data from the ontology. We have prepared two ways of doing so. It is possible to define an index manually or use our function, which generates a basic index based on the ontology's proprieties. Thanks to that, with future changes of the OpenCS ontology will keep the solution updated. We have tested many approaches to create index templates, mostly experimenting with properties types and options connected with them, testes approaches are in 3.2.1. After having an index template, it has to be filled with data. To do this, we prepared functionality to read all .ttl files present in OpenCs ontology and collect everything inside them. After that, we created a dictionary translating concepts to its prefLabel. Using that dictionary, we have translated all names like' C11' present in broader and related predicates to human-readable labels. With data in this form, we could start fulfilling indexes with data. Our first approach uploaded one row (constructed from one turtle file) at once. It was very inefficient, and uploading all 45917 took over 40 minutes. We used a bulk function present in the Elasticsearch library for Python to fix that. It loads all rows at once, shortening the time to less than one minute.

3.2.1 Testing index templates

In the first attempt we used dimple index with fields of type 'text', template looked like this:

```
    {
    "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "prefLabel": {
                "type": "text"
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "text"
            },
            "broader": {
                "type": "text"
            }
        }
    }
}
```

Figure 4: First index template

In the next approach used type keyword which is used for structured content such as IDs, email addresses, status codes or tags.

```
        {
    "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "label": {
                "type": "keyword"
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "keyword"
            },
            "broader": {
                "type": "keyword"
            }
        }
    }
}
```

Figure 5: Template for index with keyword types

Next approach, like the first one, used *text* property type, but this time we also included *analyzer* for English language for fields which will be used in queries. Analyzer removes most punctuation, lowercases terms, and supports removing stop words.

```
{
    "settings": {
    "number_of_shards": 2,
    "number_of_replicas": 1
    },
    "mappings": {
        "dynamic": "true",
        "_source": {
        "enabled": "true"
        },
        "properties": {
            "name": {
                "type": "text"
            },
            "label": {
                "type": "text",
                "analyzer":"english"
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "text",
                "analyzer":"english"
            },
            "broader": {
                "type": "text",
                "analyzer":"english"
            }
        }
    }
}
```

Figure 6: Index with text analyzers

In the last two approaches we have used the similarity setting, it defines how matching documents are scored. We tried it in two ways, standard and using tf-idf.

```
{
    "settings": {
        "index": {
        "similarity": {
            "my_similarity": {
                "type": "DFR",
                "basic_model": "g",
                "after_effect": "l",
                "normalization": "h2",
                "normalization.h2.c": "3.0"
                }
            }
        }
    },
    "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "label": {
                "type": "keyword",
                "similarity" : "my_similarity"
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "keyword",
                "similarity" : "my_similarity"
            },
            "broader": {
                "type": "keyword",
                "similarity" : "my_similarity"
            }
        }
    }
}
```

Figure 7: Index with similarities

```
{
  "settings": {
    "number_of_shards": 1,
    "similarity": {
      "scripted_tfidf": {
        "type": "scripted",
        "script": {
          "source": "double tf = Math.sqrt(doc.freq); double idf = Math.log((field.docCount+
                                                        1.0)/(term.docFreq+1.0)) + 1.0;
                                                        double norm = 1/Math.sqrt(doc.length
                                                        ); return query.boost * tf * idf *
                                                        norm;"
        }
      }
    }
  },
  "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "label": {
                "type": "keyword",
                "similarity" : "scripted_tfidf"
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "keyword",
                "similarity" : "scripted_tfidf"
            },
            "broader": {
                "type": "keyword",
                "similarity" : "scripted_tfidf"
            }
        }
    }
}
```

Figure 8: Index with similaritt tf-idf

To decide which index is the best we have run the same two queries on each one of them - using only title and using both title and abstract. Unfortunately results weren't very impressive, all indexes return similar results. It is probably caused by data present in the ontology, prefLabels are single words or very short phrases, in most cases withot any stopwords or punctuation, with no need for lemmatization.

Specific test results are present in *test_other_indexes.ipynb* notebook.

3.3 Querying the index

After choosing the best index template we have tested a lot of approaches to building queries, some of them are presented below. All results are accessible in *querying_es.ipynb* notebook.

3.3.1 Querying only with title

Title of used article: 'Brain Tumor Detection from MRI using Adaptive Thresholding and Histogram based Techniques'.

```
query = {'query': {
    "match": {"prefLabel" : a0['article_title']}
    }
}
```

Figure 9: The simplest possible query, using only *prefLabel* property.

```
query = {
  "query": {
    "multi_match" : {
      "query":    a0['article_title'],
      "fields" : ["prefLabel^3", "related", "broader"]
    }
  }
}
```

Figure 10: This query uses *multi_match* which allows multi-field queries, so properties *prefLabel*, *realted* and *broader* are used, ${}^{3}meansthat$*prefLabel*$isvaluesthreetimesmorethandefaulty$

```
query = {
  "query": {
    "multi_match" : {
      "query":      a0['article_title'],
      "type":       "bool_prefix",
      "fields":     ["prefLabel^4", "related", "broader^2"]
    }
  }
}
```

Figure 11: Similar to the previus one, experimentng with scoring fileds, also using *bool_prefix* which Creates a match_bool_prefix query on each field and combines the _score from each field

3.3.2 Querying with both title and abstract

Abstract of article: 'This paper depicts a computerized framework that can distinguish brain tumor and investigate the diverse highlights of the tumor. Brain tumor segmentation means to isolated the unique tumor tissues, for example, active cells, edema and necrotic center from ordinary mind tissues of WM, GM, and CSF. However, manual segmentation in magnetic resonance data is a timeconsuming task. We present a method of automatic tumor segmentation in magnetic resonance images which consists of several steps. The recommended framework is helped by image processing based technique that gives improved precision rate of the cerebrum tumor location along with the computation of tumor measure. In this paper, the location of brain tumor from MRI is recognized utilizing adaptive thresholding with a level set and a morphological procedure with histogram. Automatic brain tumor stage is performed by using ensemble classification. Such phase classifies brain images into tumor and non-tumors using Feed Forwarded Artificial neural network based classifier. For test investigation, continuous MRI images gathered from 200 people are utilized. The rate of fruitful discovery through the proposed procedure is 97.32 percentage accurate.'

Title is the same as previously.

```
query = {
  "query": {
    "dis_max": {
      "queries": [
        {
          "multi_match" : {
          "query":    a0[’article_title’],
          "fields" : ["prefLabel^3", "related", "broader"]
    }
        },
        {
          "multi_match" : {
          "query":    a0[’article_abstract’],
          "fields" : ["prefLabel^2", "related", "broader^2"]
          }
        }
      ]
    }
  }
}
```

Figure 12: Thanks to *dis_max* it is possible to use two queries combined

```
query = {
  "query": {
    "dis_max": {
      "queries": [
        {
          "multi_match" : {
          "query":      a0[’article_title’],
          "type":       "best_fields",
          "fields":     ["prefLabel^3", "related", "broader"],
          "tie_breaker": 0.3
         }
        },
        {
          "multi_match" : {
          "query":      a0[’article_abstract’],
          "type":       "best_fields",
          "fields":     ["prefLabel^3", "related", "broader"],
          "tie_breaker": 0.3
         }
        }
      ]
    }
  }
}
```

Figure 13: *best_fields* finds documents which match any field, but uses the _score from the best field, most useful when you are searching for multiple words best found in the same field

```
query = {
  "query": {
    "dis_max": {
      "queries": [
        {
          "multi_match" : {
          "query":      a0['article_title'],
          "type":       "most_fields",
          "fields":     ["prefLabel^3", "related", "broader"],
          "tie_breaker": 0.5
         }
        },
        {
          "multi_match" : {
          "query":      a0['article_abstract'],
          "type":       "most_fields",
          "fields":     ["prefLabel^3", "related", "broader"],
          "tie_breaker": 0.5
         }
        }
      ]
    }
  }
}
```

Figure 14: *most_fields* finds documents which match any field and combines the _score from each field, most useful when querying multiple fields that contain the same text analyzed in different ways

4 Results

4.1 First attempts results

After conducting all the experiments we have decided to use queries below and the simplest index template with all properties of type *text*.

```
query = {
    "query": {
        "multi_match" : {
            "query": title,
            "fields" : ["prefLabel^3", "related", "broader"]
            }
        }
    }
```

Figure 15: Query using only title

```
query = {
        "query": {
            "dis_max": {
                "queries": [
                    {
                        "multi_match" : {
                            "query": title,
                            "type": "best_fields",
                            "fields": ["prefLabel^4", "related", "broader"],
                            "tie_breaker": 0.5
                            }
                        },
                        {
                            "multi_match" : {
                            "query": abstract,
                            "analyzer" : "standard",
                            "type": "best_fields",
                            "fields": ["prefLabel^4", "related", "broader"],
                            "tie_breaker": 0.5
                            }
                        }
                    ]
                }
            }
        }
```

Figure 16: Query using both title and abstract

Results were rather positive, for article with title 'Brain Tumor Detection from MRI using Adaptive Thresholding and Histogram based Techniques'

and abstract 'This paper depicts a computerized framework that can distinguish brain tumor and investigate the diverse highlights of the tumor. Brain tumor segmentation means to isolated the unique tumor tissues, for example, active cells, edema and necrotic center from ordinary mind tissues of WM, GM, and CSF. However, manual segmentation in magnetic resonance data is a time consuming task. We present a method of automatic tumor segmentation in magnetic resonance images which consists of several steps. The recommended framework is helped by image processing based technique that gives improved precision rate of the cerebrum tumor location along with the computation of tumor measure. In this paper, the location of brain tumor from MRI is recognized utilizing adaptive thresholding with a level set and a morphological procedure with histogram. Automatic brain tumor stage is performed by using ensemble classification. Such phase classifies brain images into tumor and non-tumors using Feed Forwarded Artificial neural network based classifier. For test investigation, continuous MRI images gathered from 200 people are utilized. The rate of fruitful discovery through the proposed procedure is 97.32 percentage accurate.'

```
[{’prefLabel’: [’Balanced histogram thresholding’], ’score’: 43.706383},
 {’prefLabel’: [’Tumor detection’], ’score’: 43.226555},
 {’prefLabel’: [’Brain tumor segmentation’], ’score’: 42.221302},
 {’prefLabel’: [’Adaptive histogram equalization’], ’score’: 36.658276},
 {’prefLabel’: [’Thresholding’], ’score’: 33.2383},
 {’prefLabel’: [’Mri segmentation’], ’score’: 32.949432},
 {’prefLabel’: [’Histogram’], ’score’: 30.420723},
 {’prefLabel’: [’Blocking techniques’], ’score’: 27.061342},
 {’prefLabel’: [’Recovery techniques’], ’score’: 27.061342},
 {’prefLabel’: [’Visualisation techniques’], ’score’: 27.061342}]
```

Figure 17: Query using only title

```
[{’prefLabel’: [’Brain tumor segmentation’], ’score’: 501.49963},
 {’prefLabel’: [’Tumor segmentation’], ’score’: 427.25647},
 {’prefLabel’: [’Liver tumor segmentation’], ’score’: 408.45203},
 {’prefLabel’: [’Recurrent Gastrointestinal Stromal Tumor’], ’score’: 363.9802},
 {’prefLabel’: [’Tumor detection’], ’score’: 339.48605},
 {’prefLabel’: [’Tumor region’], ’score’: 339.48605},
 {’prefLabel’: [’Nerve tumor’], ’score’: 337.33698},
 {’prefLabel’: [’Stromal tumor’], ’score’: 337.33698},
 {’prefLabel’: [’Duodenal submucosal tumor’], ’score’: 333.46344},
 {’prefLabel’: [’Gastrointestinal stroma tumor’], ’score’: 333.46344}]
```

Figure 18: Query using both title and abstract

4.2 Mediocre results

Unfortunately, articles we choose for testing (the were chosen randomly) gave all good results, but there are some which give results far from desired, especially if we use abstract in the query. It was caused by the length and complexity of abstract text, containing many words which were not vert valuable in terms of searching for article title but still used equally as all others. It led to finding more and more general topics. Query using only title reurned better results because title is short and often contain the most important words. For example, if we use title and abstract below, results are not satisfying.

title: 'Map-reduce based distance weighted k-nearest neighbor machine learning algorithm for big data applications'

abstract: 'With the evolution of Internet standards and advancements in various Internet and mobile technologies, especially since web 4.0, more and more web and mobile applications emerge such as e-commerce, social networks, online gaming applications and Internet of Things based applications. Due to the deployment and concurrent access of these applications on the Internet and mobile devices, the amount of data and the kind of data generated increases exponentially and the new era of Big Data has come into existence. Presently available data structures and data analyzing algorithms are not capable to handle such Big Data. Hence, there is a need for scalable, flexible, parallel and intelligent data analyzing algorithms to handle and analyze the complex massive data. In this article, we have proposed a novel distributed supervised machine learning algorithm based on the MapReduce programming model and Distance Weighted k-Nearest Neighbor algorithm called MR-DWkNN to process and analyze the Big Data in the Hadoop cluster environment. The proposed distributed algorithm is based on supervised learning performs both regression tasks as well as classification tasks on large-volume of Big Data applications. Three performance metrics, such as Root Mean Squared Error (RMSE), Determination coefficient (R2) for regression task, and Accuracy for classification tasks are utilized for the performance measure of the proposed MR-DWkNN algorithm. The extensive experimental results shows that there is an average increase of 3% to 4.5% prediction and classification performances as compared to standard distributed k-NN algorithm and a considerable decrease of Root Mean Squared Error (RMSE) with good parallelism characteristics of scalability and speedup thus, proves its effectiveness in Big Data predictive and classification applications.'

```
[{’prefLabel’: [’Fuzzy k nearest neighbor’], ’score’: 51.334732},
 {’prefLabel’: [’Map reduce’], ’score’: 50.358086},
 {’prefLabel’: [’Nearest-neighbor chain algorithm’], ’score’: 45.369736},
 {’prefLabel’: [’k-nearest neighbors algorithm’], ’score’: 43.930107},
 {’prefLabel’: [’Weighted distance’], ’score’: 41.160805},
 {’prefLabel’: [’Nearest neighbor search’], ’score’: 40.77295},
 {’prefLabel’: [’Transitive nearest neighbor’], ’score’: 40.77295},
 {’prefLabel’: [’Nearest neighbor clustering’], ’score’: 40.77295},
 {’prefLabel’: [’Nearest neighbor classifier’], ’score’: 40.77295},
 {’prefLabel’: [’Nearest neighbor imputation’], ’score’: 40.77295}]
```

Figure 19: Query using only title

```
[{’prefLabel’: [’Suicide and the Internet’], ’score’: 690.5829},
 {’prefLabel’: [’Sociology of the Internet’], ’score’: 539.9591},
 {’prefLabel’: [’Programming in the large and programming in the small’],
  ’score’: 527.02966},
 {’prefLabel’: [’The E and B Experiment’], ’score’: 494.9974},
 {’prefLabel’: [’Outline of the Internet’], ’score’: 492.57883},
 {’prefLabel’: [’The Internet’], ’score’: 470.5413},
 {’prefLabel’: [’Agreement on the Application of Sanitary and Phytosanitary Measures’],
  ’score’: 453.43387},
 {’prefLabel’: [’Determining the number of clusters in a data set’],
  ’score’: 451.27924},
 {’prefLabel’: [’Confusion of the inverse’], ’score’: 450.44818},
 {’prefLabel’: [’Data, context and interaction’], ’score’: 446.75644}]
```

Figure 20: Query using both title and abstract

To fix this issue we tried to preporcess input text to clear abstract of unnscessary words, so the topics will became less general. At first we tried to do that using another Elasticsearch template, but the results still were not good enough.

title: 'Experiences with mesh-like computations using prediction binary trees'

abstract: 'In this paper we aim at exploiting the temporal coherence among successive phases of a computation, in order to implement a load-balancing technique in mesh-like computations to be mapped on a cluster of processors. A key concept, on which the load balancing schema is built on, is the use of a Predictor component that is in charge of providing an estimation of the unbalancing between successive phases. By using this information, our method partitions the computation in balanced tasks through the Prediction Binary Tree (PBT). At each new phase,current PBT is updated by using previous phase computing time for each task as next-phase's cost estimate. The PBT is designed so that it balances the load across the tasks as well as reduces dependency among processors for higher performances. Reducing dependency is obtained by using rectangular tiles of the mesh,of almost-square shape (i. e. one dimension is at most twice the other). By reducing dependency, one can reduce inter-processors communication or exploit local dependencies among tasks (such as data locality). Furthermore, we also provide two heuristics which take advantage of data-locality. Our strategy has been assessed on a significant problem, Parallel Ray Tracing. Our implementation shows a good scalability, and improves performance in both cheaper commodity cluster and high performance clusters with low latency networks. We report different measurements showing that tasks granularity is a key point for the performances of our decomposition/mapping strategy.'

```
template = {
  "settings": {
    "analysis": {
      "filter": {
        "english_stop": {
          "type":       "stop",
          "stopwords":  stopwords.words('english')
        }
      },
      "analyzer": {
        "rebuilt_cjk": {
          "tokenizer":  "standard",
          "filter": [
            "cjk_width",
            "lowercase",
            "cjk_bigram",
            "english_stop"
          ]
        }
      }
    }
  },
  "mappings": {
        "properties": {
            "name": {
                "type": "text"
            },
            "prefLabel": {
                "type": "text",
            },
            "type": {
                "type": "text"
            },
            "closeMatch": {
                "type": "text"
            },
            "related": {
                "type": "text",
            },
            "broader": {
                "type": "text",
            }
        }
    }
}
```

Figure 21: Another index template, with removing stopwords and and advanced analyzer

```
[{'prefLabel': ['Is-a'], 'score': 524.8042},
 {'prefLabel': ['The purpose of a system is what it does'],  'score': 420.18066},
 {'prefLabel': ['IS-IS'], 'score': 396.01587},
 {'prefLabel': ['Determining the number of clusters in a data set'],  'score': 372.15686},
 {'prefLabel': ['Everything is a file'], 'score': 367.01727},
 {'prefLabel': ['Computation in the limit'], 'score': 352.42493},
 {'prefLabel': ['Confusion of the inverse'], 'score': 344.44702},
 {'prefLabel': ['Model in the loop'], 'score': 337.43103},
 {'prefLabel': ['Software in the loop'], 'score': 337.43103},
 {'prefLabel': ['An Essay towards solving a Problem in the Doctrine of Chances'],  'score':
                                               322.15088}]
```

Figure 22: Results for querying using title and abstract and new template

4.3 Final results

Out last idea was to provide prepossessing of abstracts and titles in Python instead of Elasticsearch. Prepossessing consists of lowering the letters, removing punctuation, removing stopwords and removing duplicated words.

This approach made results better, but still not perfect.

```
[{’prefLabel’: [’Large margin nearest neighbor’], ’score’: 88.40973},
 {’prefLabel’: [’Machine-generated data’], ’score’: 81.23446},
 {’prefLabel’: [’Scalable parallel algorithms’], ’score’: 79.411194},
 {’prefLabel’: [’Task parallelism’], ’score’: 76.62481},
 {’prefLabel’: [’Map reduce’], ’score’: 73.412056},
 {’prefLabel’: [’Web based learning environment’], ’score’: 73.21821},
 {’prefLabel’: [’Supervised learning’], ’score’: 72.44982},
 {’prefLabel’: [’Web based social networks’], ’score’: 72.13786},
 {’prefLabel’: [’Complex data structures’], ’score’: 71.019615},
 {’prefLabel’: [’Distributed data structures’], ’score’: 70.52696}]
```

Figure 23: Results for querying using title and abstract from 4.1

```
[{’prefLabel’: [’High performance computing clusters’], ’score’: 81.80024},
 {’prefLabel’: [’High performance parallel computing’], ’score’: 76.64901},
 {’prefLabel’: [’Load balancing (computing)’], ’score’: 73.98726},
 {’prefLabel’: [’High performance cluster’], ’score’: 73.03627},
 {’prefLabel’: [’Network Load Balancing’], ’score’: 72.57456},
 {’prefLabel’: [’Ray coherence’], ’score’: 71.407646},
 {’prefLabel’: [’Distributed ray tracing’], ’score’: 70.37132},
 {’prefLabel’: [’High performance computation’], ’score’: 69.77369},
 {’prefLabel’: [’Commodity computing’], ’score’: 68.97159},
 {’prefLabel’: [’Self-balancing binary search tree’], ’score’: 67.03011}]
```

Figure 24: Results for querying using title and abstract from 4.2

5 End-to-end pipeline

To enable possibility of using whole solution without deep insight into the code we have prepared script *pipeline.py*. It runs whole flow of the solution:

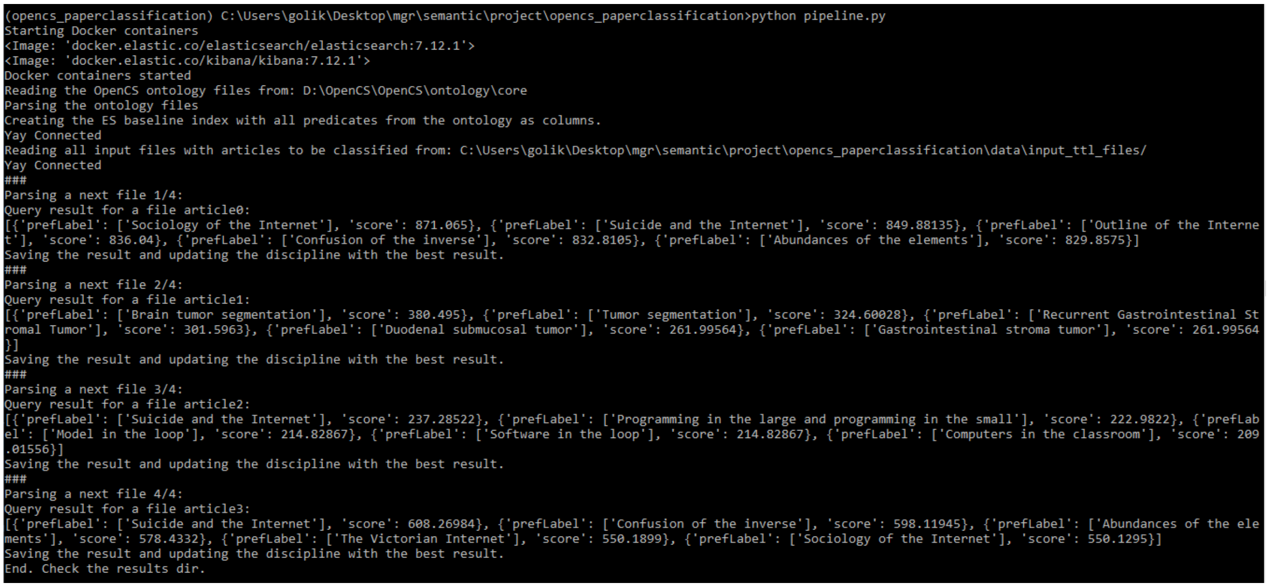
- starts Docker containers,
- reads/parses the ontology,
- build a baseline index (all predicates from the ontology as columns),
- perform classification for the files from the input directory (by default: ./data/input_ttl_files),
- saves the results (moves updated files) in the output directory (by default: ./results/results),
- removes the deployed Docker containers.

Input data for the script are *.ttl* files from *data/input_ttl_files* directory - examples of these in our solution come from the scraping team. With topic classified for the article new *turtle* file is saved in directory *results/results*, it is input file appended with *fabio:hasDiscipline* predicate and concept being a classified topic.

There are three possible parameters you can pass to the script:

- *–move_original_files* (True/False) determines whether the input files are deleted from the input directory after they are updated and moved to the result directory (by default True)
- *–ask_to_override* (True/False) specifies whether to request a confirmation of results ([y/n]) before saving them from the user (note that if set to True you need to monitor the script and make a decision for each of the files after results are printed. By default False)
- *–n* (int) indicates how many best results will be displayed for each article (by default 5)

To run the script values in file *config.py* must be adjusted and conda environment has to enabled.



```
(opencs_paperclassification) C:\Users\golik\Desktop\mgr\semantic\project\opencs_paperclassification>python pipeline.py
Starting Docker containers
<Image: 'docker.elastic.co/elasticsearch/elasticsearch:7.12.1'>
<Image: 'docker.elastic.co/kibana/kibana:7.12.1'>
Docker containers started
Reading the OpenCS ontology files from: D:\OpenCS\OpenCS\ontology\core
Parsing the ontology files
Creating the ES baseline index with all predicates from the ontology as columns.
Yay Connected
Reading all input files with articles to be classified from: C:\Users\golik\Desktop\mgr\semantic\project\opencs_paperclassification\data\input_ttl_files/
Yay Connected
###
Parsing a next file 1/4:
Query result for a file article0:
[{'prefLabel': ['Sociology of the Internet'], 'score': 871.065}, {'prefLabel': ['Suicide and the Internet'], 'score': 849.88135}, {'prefLabel': ['Outline of the Interne
t'], 'score': 836.04}, {'prefLabel': ['Confusion of the inverse'], 'score': 832.8105}, {'prefLabel': ['Abundances of the elements'], 'score': 829.8575}]
Saving the result and updating the discipline with the best result.
###
Parsing a next file 2/4:
Query result for a file article1:
[{'prefLabel': ['Brain tumor segmentation'], 'score': 380.495}, {'prefLabel': ['Tumor segmentation'], 'score': 324.60028}, {'prefLabel': ['Recurrent Gastrointestinal St
romal Tumor'], 'score': 301.5963}, {'prefLabel': ['Duodenal submucosal tumor'], 'score': 261.99564}, {'prefLabel': ['Gastrointestinal stroma tumor'], 'score': 261.99564
}]
Saving the result and updating the discipline with the best result.
###
Parsing a next file 3/4:
Query result for a file article2:
[{'prefLabel': ['Suicide and the Internet'], 'score': 237.28522}, {'prefLabel': ['Programming in the large and programming in the small'], 'score': 222.9822}, {'prefLab
el': ['Model in the loop'], 'score': 214.82867}, {'prefLabel': ['Software in the loop'], 'score': 214.82867}, {'prefLabel': ['Computers in the classroom'], 'score': 209
.01556}]
Saving the result and updating the discipline with the best result.
###
Parsing a next file 4/4:
Query result for a file article3:
[{'prefLabel': ['Suicide and the Internet'], 'score': 608.26984}, {'prefLabel': ['Confusion of the inverse'], 'score': 598.11945}, {'prefLabel': ['Abundances of the ele
ments'], 'score': 578.4332}, {'prefLabel': ['The Victorian Internet'], 'score': 550.1899}, {'prefLabel': ['Sociology of the Internet'], 'score': 550.1295}]
Saving the result and updating the discipline with the best result.
End. Check the results dir.
```

Figure 25: Example of running *pipline.py* script

6 Summary

In summary, in our project, we have successfully delivered a tool for assigning scientific articles to the domains they belong to from the OpenCS ontology. Our solution based on the search engine Elasticsearch was thoroughly tested regarding the performance of the various functionalities offered by Elasticsearch and text preprocessing. The classification results were not always accurate, but we identified possible directions for further improvements to the solution. Finally, the project results are fully reproducible, and the use of the tool is easy, thanks to the documentation created.

7 Team members contribution

Paweł Golik

1. Embedding approach research and tests
2. Providing Docker solution
3. Implementation of connection to ES
4. Code refactoring
5. Researching Elasticsearch solutions
6. Github readme creation
7. Contributing to writing the report

Przemysław Olender

1. Elasticsearch research and tests
2. Preparing code reading ontology with Python and loading it to Elasticsearch
3. Creating and testing Elasticsearch templates
4. Creating and testing Elasticsearch queries
5. Contributing to writing the report

References

[1] Opencs-ontology/opencs: Main opencs ontology repository. https://github.com/OpenCS-ontology/OpenCS. (Accessed on 01/18/2023).

[2] Welcome to elastic, creators of elasticsearch & kibana — elastic. https://www.elastic.co/. (Accessed on 01/18/2023).

[3] What is an elasticsearch index? https://www.elastic.co/blog/what-is-an-elasticsearch-index. (Accessed on 01/20/2023).