FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Documents classification using machine learning methods
Student:	Artem Ustynov
Supervisor:	Ing. Zdeněk Buk, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of winter semester 2021/22

Instructions

The problem of searching in uncategorized documents is that users are often presented with results that contain searched keywords, but are not relevant to the user.
The goal of this work is to extend the documents with tags based on their content. Each tag contains the category as well as the confidence of the classification. The document can have multiple tags.

Tasks:
- Create a testing dataset.
- Compare basic "out of the box" Elasticsearch text classification tool with "Semaphore" and state of the art techniques: LSTM, BERT (consider the ease of integration with current infrastructure, performance, and accuracy).
- Make an argument for the best solution and implement it.

Minimum performance requirements:
- The accuracy of predictions should be at least 70 %.
- Evaluation time 5s per document 1MB of text.
- Time to train a model: under 48 hours.

The application should be written in Java Spring Boot as a callable service or in Python as a Jenkins job.

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 18, 2020

Bachelor's thesis

Documents classification using machine learning methods

Artem Ustynov

Department of Informatics
Supervisor: Ing. Zdeněk Buk, Ph.D.

June 3, 2020

Declaration

In Prague on June 3, 2020 . . . .

Citation of this thesis

Ustynov, Artem. *Documents classification using machine learning methods.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Problém hledání v nekategorizovaných dokumentech spočívá v tom, že uživatelům jsou často prezentovány výsledky, které obsahují hledaná klíčová slova, ale nejsou pro uživatele relevantní. Cílem této práce je rozšířit dokumenty o štítky na základě obsahu dokumentů. K dosažení cíle bylo zvažováno několik přístupů: Elasticsearch, Semaphore, LSTM, BERT. Cílem práce je zjistit, která technika má největší potenciál a poskytuje nejlepší výsledky. Všechny uvedené přístupy byly testovány a vyhodnoceny. Bylo zjištěno, že modely BERT fungovaly nejlépe a splnily všechny vstupní požadavky. Zlepšení kvality klasifikace pomocí BERT bylo dosaženo použitím počátečního modelu a manuální klasifikací malé sady dokumentů s nízkým skóre spolehlivosti.

Klíčová slova Elasticsearch, Semaphore, LSTM, BERT, klasifikace textu

Abstract

The problem of searching in uncategorized documents is that users are often presented with results that contain searched keywords, but are not relevant to the user. The goal of this work is to extend the documents with tags based on their content. To accomplish this several approaches were considered: Elasticsearch, Semaphore, LSTM, BERT. The objective of this thesis is to determine which technology has the most potential and provides the best results. All listed approaches were tested and evaluated. It was found that BERT models performed the best and satisfied all of the initial business requirements. Some improvements in the quality of classification with BERT were achieved by utilizing the initial model and manually classifying a small set of documents with a low confidence score.

Keywords Elasticsearch, Semaphore, LSTM, BERT, text classification

Contents

List of Figures

Introduction

Motivation and objectives

A medical company has a data source that can be used by people from different departments to store any sort of text documents. Elasticsearch is utilized to provide a search on terms from user's queries, however, results are often not relevant to the end-user, since this search engine returns documents based on TF-IDF [1] score without taking into account the context of the document itself.

To solve this problem idea of tagging documents based on their context was proposed since data source is already mature and has more than five million documents indexed already manual tagging is not an option.

Several approaches were considered:

- ElasticSearch – classification based on manually tagging some volume of documents and then running "more like this" query on them utilizing engines capabilities of stemming and synonyms [2]
- Semaphore – a commercial tool that allows classifying documents based on predefined ontology structure [3]
- LSTM – machine learning method that was shown to get a high score in classification problem in sample data sets [4]
- BERT – attention based machine learning method that also showed good results in classification problems [5]

Given the scope of the project and the fact that more tags can be requested from users in the future, it is crucial to choose the approach that allows to bootstrap data set: train model on relatively small data sets, and leverage initial results to find more documents that would contain given tag and repeat until the desired accuracy is achieved. To aid the creation of the initial data set users are expected to provide some highly specific queries that will help a person to identify and find more documents that will serve as initial data to train the model.

Requirements

Requirements that were agreed upon were the following:

- The accuracy of predictions should be at least 70
- Evaluation time 5s per document with 1MB of text.
- Time to train a model: under 48 hours.

Problem statement

The main sight of the thesis is to determine which approach will combine the best performance of the end model with high accuracy of the results. To provide users with the context of the documents. How to achieve this without creating a massive original data set. And how to bootstrap from a relatively small volume of documents.

Chapter 1

State-of-the-art

One way to determine the current state of the art for natural language processing is to check what models are currently holding top positions in well-established benchmarks. Currently, leading positions are occupied by Bert and it's modifications such as "Albebert", "Roberta", "T5" [6].

Bert is an attention-based model trained on a large corpus of general text and is later fine-tuned to perform various tasks such as text prediction, summarization, and classification.

Elasticsearch is an open-source search and analytic solution that allows to performs classification of documents with minimal effort. It requires very little preparation time and relies on its ability to find similar documents, based on indexed data. From preliminary research, it was found that it can perform well enough to satisfy the requirements of the project [2].

LSTM is a special kind of recurrent neural network. LSTM models were shown to perform well in classification tasks and are close in it's performance to BERT [7]. LSTM and BERT rely on different approaches when it comes to natural language processing, LSTM is a recurrent model, while BERT is an attention [8] based model. LSTM in our use case is justified since our task requires a model, that can perform well from relatively small initial data set and drawbacks such as having a problem with retaining information for long periods of time or dealing with words that have a different meaning in different context [9] might not constitute themselves on small scientific / technical data set.

Semaphore is a commercial tool that takes a different approach to the classification problem. It positions itself as "Assisted intelligence" [3] that relies on a semantic model, which describes the concepts, topics, subjects, themes, and organizational structures that are unique to each use case. The model is published and rule bases - a series of templates, which can be modified to achieve specific classification outcomes - are generated [10]. This tool is already available at our company and there exists an ontology model that describes one of the data sources. Our use case would be determining if the existing model will overlap with our data source, it should be evaluated if creating a new ontology model is justified. Given that model is created by experts in a field this approach can theoretically outperform other methods of classification.

Even though Bert and it's variants are currently leading in natural language processing[6], other methods should not be disregarded, without a test. They can be more efficient in speed or even provide better results in our particular use case with a small original data set.

Chapter **2**

Analysis and design

Currently, several data sources are utilized to provide users with search capabilities. One such data source contains over 5.5 million documents (over 100GB of data) and has no order to it. It contains documents from different departments that are related to various topics.

To provide users with improved results several enhancers are utilized. Optical image recognition enhancer – utilized to extract text from images attached to documents. Author normalization enhancer – used to normalize authors' names and enable users to filter documents by author. The additional enhancer was proposed to classify data and tag documents based on their context. See figure 2.1

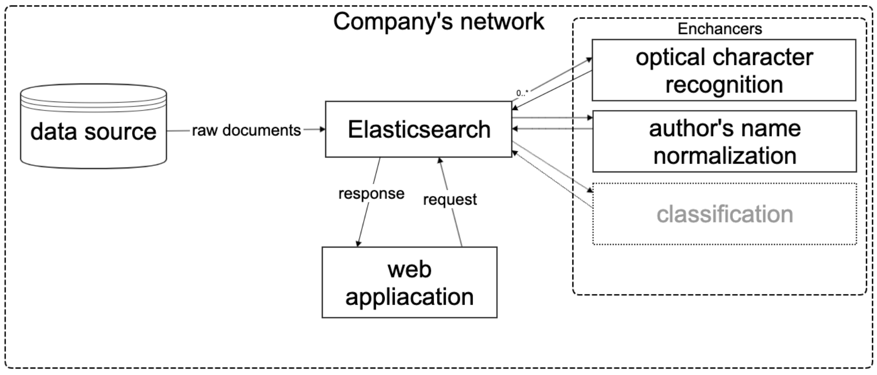

Figure 2.1: Overview of current document flow infrastructure.

To have a direct comparison between different classification methods python function "train_test_split from sklearn.model_selection was applied to split data set into train and test. Then true positive, false positive, true negative and false-negative results were computed along with the Matthews correlation coefficient (MCC).

MCC was chosen since it is robust to changes and is easy to interpret [11]. MCC takes a value between -1 and 1, where 1 is a perfect model. Higher Matthews correlation coefficient implies a better result. I chose MCC because it takes into account all of the cells of the confusion matrix:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.1}$$

Where: TP – true positive, FP – false positive, TN – true negative, FN – false negative.

These metrics (TP, FP, TN, FN) will be used in all graphs related to the performance of different models. The exact value of each metric will be provided on top of the respected bar in every bar chart.

In all implementations, the same random seed was used, and the test size was 20% of the total size of the data set. Split was performed with python function "train_test_split from sklearn.model_selection".

Emphasis was made on finding a solution with the smallest amount of false positives. Since tags are not distributed equally among all documents and overall relatively high score might not be representative of a good model. The high false-positive count would mean that the assigned tag is likely does not belong there. In that case, it will present little value to the end-user.

Measuring the time of training and evaluation was done from the code without using external programs.

Python was chosen as a language for all implementation, except Semaphore, since its API is only available via Java.

To interface with Elasticsearch python package "elasticsearch" was used. Since it allows to index documents and perform "more like this" query in a form that is close to interacting with Elasticsearch directly. See figures 2.2 and 2.3

```
response = es.search(index=index, body={'query': {
    "more_like_this": {
        "fields": ["text","tags"],
        "like": row["TEXT"],
        "analyzer":"english",
        "min_word_length":4,
    }
}
}, _source = "tags")
```

Figure 2.2: Example of python interface with Elasticsearch

```
GET /_search
{
    _source = "tags",
    "query": {
        "more_like_this" : {
            "fields": ["text","tags"],
            "like" : "text to find",
            "analyzer":"english",
            "min_word_length":4,
        }
    }
}
```

Figure 2.3: Example of native Elasticsearch interface

To work with LSTM models ”keras” package was used to set up the model and train it, since it allows to create a new model with few simple lines of code, and created models can be stored as binary files, and distributed between different machines, without need to re-train the model. See figure 2.4

```
model = Sequential()
model.add(Embedding(n_most_common_words,
emb_dim, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(256, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(16, activation='softmax'))
model.compile(optimizer='adam',

loss='categorical_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=epochs,

                    batch_size=batch_size,
                    validation_split=0.2,
                    callbacks=[EarlyStopping

                    (monitor='val_loss',patience=4,
                    min_delta=0.001)])
model.save("path")
```

Figure 2.4: Example of using ”keras” to train and save an LSTM model

To work with BERT originally ”pytorch” models were considered, but they require a lot of pre-processing of the data and boilerplate code, so package ”simpletransformers” was used to interface with models. It enabled to write very short code that is easy to read and maintain.

```
model = ClassificationModel(model_type, model_name,

use_cuda=False, args={
      'output_dir': 'path',
      'max_seq_length': 256,
      'train_batch_size': 36,
      'eval_batch_size': 28,
      'num_train_epochs': 10,
      "save_eval_checkpoints": False,
      "save_model_every_epoch": False,
      'save_steps': 0
  })

  # Train the model
  model.train_model(data_train)

  # Evaluate the model
  # this step also saves TP,TN,FP,FN values and
  # MCC score in eval_results.txt file
  result, model_outputs, wrong_predictions =

  model.eval_model(data_test)
```

Figure 2.5: Example of using ”simpletransformers” to train and save BERT model

To interface Semaphore java API was used since it is the only supported way.

```
ClassificationClient classificationClient
= new ClassificationClient();
ClassificationConfiguration classificationConfiguration
= new ClassificationConfiguration();
classificationConfiguration.setUrl(CloudConfiguration.csURL);
classificationConfiguration.setApiToken

(token.getAccess_token());

classificationClient.setClassificationConfiguration
(classificationConfiguration);
Result result = classificationClient
.getClassifiedDocument(("text").getBytes(), "title");
```

Figure 2.6: Example of using Semaphore Java API.

All tests were run on the same machine CentOS Linux release 7.8 CPU: 16 x Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz with 128 GB of RAM.

Chapter 3

Realisation

3.1 Creating an initial data set

To create initial data set first of all target tags were established. To do so, the corporate structure was utilized.

Then using google analytic I found documents that are most frequently accessed by users and manually tagged 500 of them. Two major groups of tags were established: technical and scientific. One document can contain multiple tags from one of the groups.

Tags were not distributed evenly among the documents, so I will only concentrate on those tags that have a bigger presence see figure 3.1.

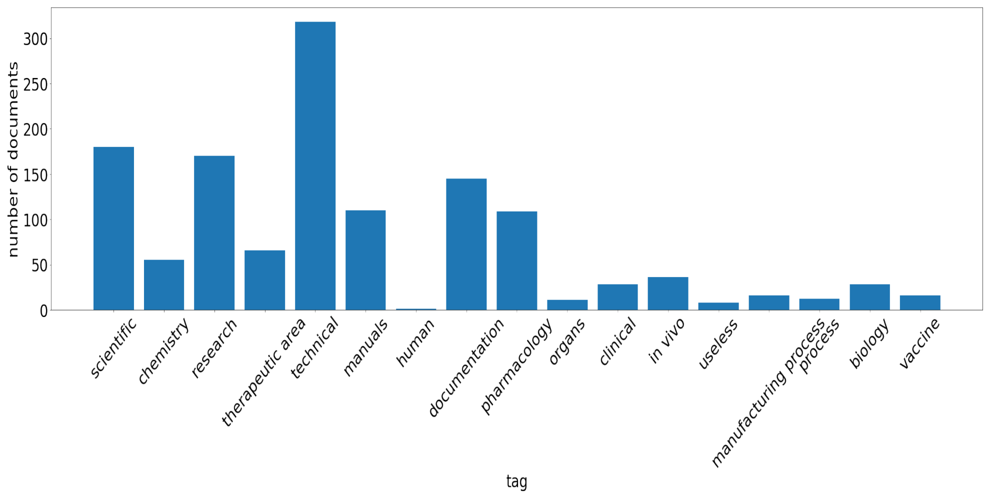

Figure 3.1: Distribution of tags between documents

Distribution of characters in documents grouped by tags was not even as well see figure 3.2.

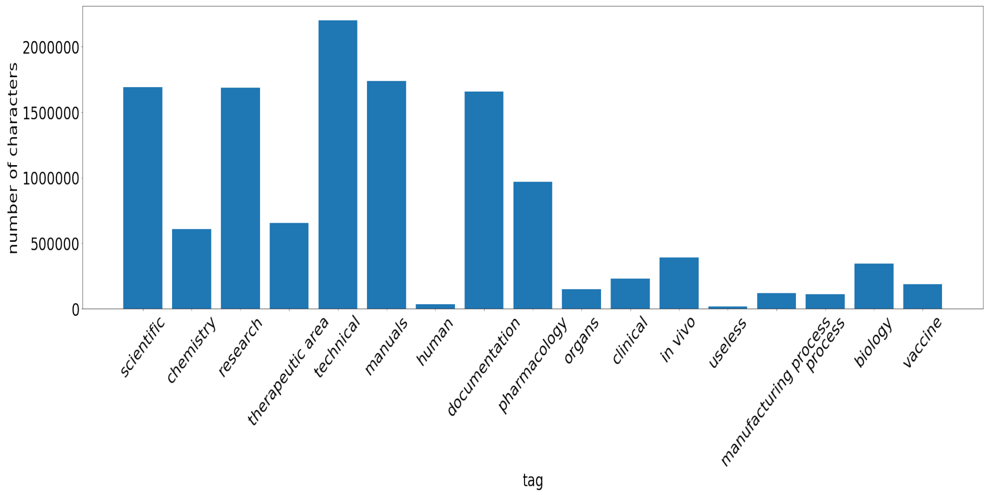

Figure 3.2: Distribution of text between tags.

Creating a bigger better-balanced data set on documents without any structure in original data would be a too time-consuming task. And since new tags can be requested by users later selected classification method should be able to "bootstrap" itself from a relatively small initial set.

3.2 Elasticsearch

Elasticsearch is a distributed open-source search engine that has build-in functionality – "more_like_this" query. Elasticsearch enables users to apply stemming and analysis [12] of the text. The default analyzer and stemmer were used.

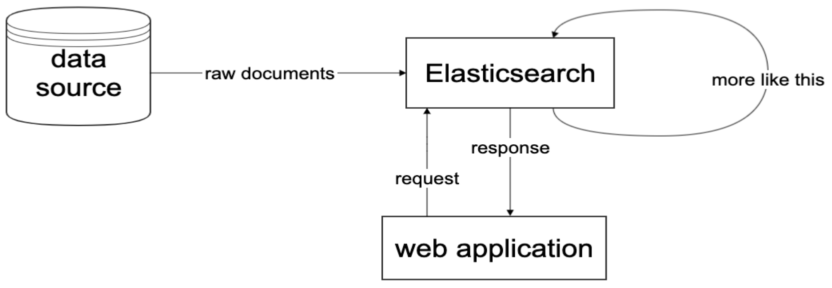

Figure 3.3: Elasticsearch classification infrastructure

3.2.1 How it works

If one wants to find all documents similar to a given input document he can run ”more like this” query. Obviously, the input document itself should be its best match for that type of query. And the reason would be mostly, according to Lucene scoring formula[1], due to the terms with the highest TF-IDF. Therefore, the terms of the input document that have the highest TF-IDF are good representatives of that document and could be used within a disjunctive query (or OR) to retrieve similar documents. The MLT (more like this see figure 3.4 query simply extracts the text from the input document, analyzes it, usually using the same analyzer at the field, then selects the top K terms with highest TF-IDF to form a disjunctive query of these terms.[13]

```
# es - elasticsearch object
def mlt(input_text):
    #transforms text to set of stemmed words
    analyzed_text = es.analyze(input_text)
    # map {key=word, value = TF-IDF}
    map_of_terms= {}
    for word in analyzed_text:
        #returns TF-IDF score
        score = es.search(term=word, text=input_text)
        map_of_terms[word]=score
    #sort terms
    sorted_terms=map_of_terms.sort_by("values")
     # get top k terms default value of K is 25
    top_k_terms = sorted_terms[0:K]
    #searches for all terms in es index
    #using disjunctive query (OR)
    #then returns all documents that match
    #and their TF-IDF score.
    result = es.search_all(top_k_terms)
    return result
```

Figure 3.4: Algorithmic description of ”more like this” query. (pseudocode)

3.2.2 Implementation

To perform a classification I split documents from my data set with python function ”train_test_split” using random seed – 41. After that, I index train split, such that each document contained ”text” and ”tags” fields and run ”more_like_this” query that returns score of the document and tags that it contained. For each tag, I summed up the score of the document, so as a result I got a map where the key is tag and value represents combined score of all similar documents. After initial analysis of the results I found out that there are multiple documents, where values are very different between themselves and there are too many tags in a returned query, to determine what tags should be kept I applied ”knee” [1]function and only kept tags before pivoting point see figure 3.5

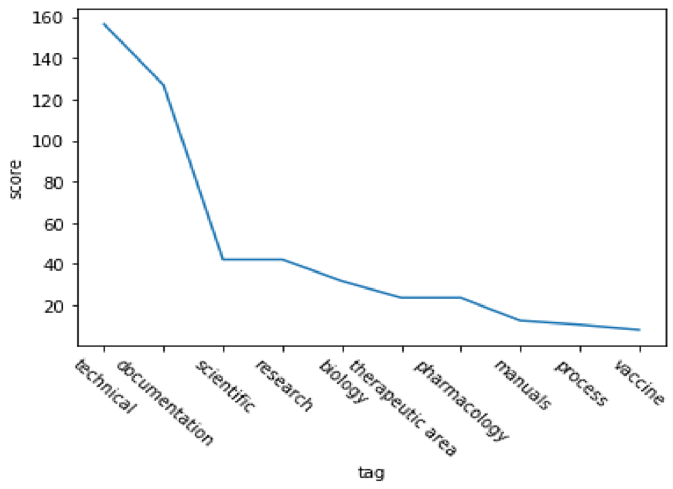

Figure 3.5: Example of a knee function applied to a search result (3rd item – pivoting point)

[1]Knee of a curve can be defined as a vertex of the graph where the curvature has a maximum. Knee function returns index of the vertex at the point of maximum curvature.

3.2.3 Results and conclusions

To measure the performance of the method Matthews correlation coefficient (MCC) was used. Results that were acquired are high in false-positive count that sometimes exceed true positive results see figure3.6.

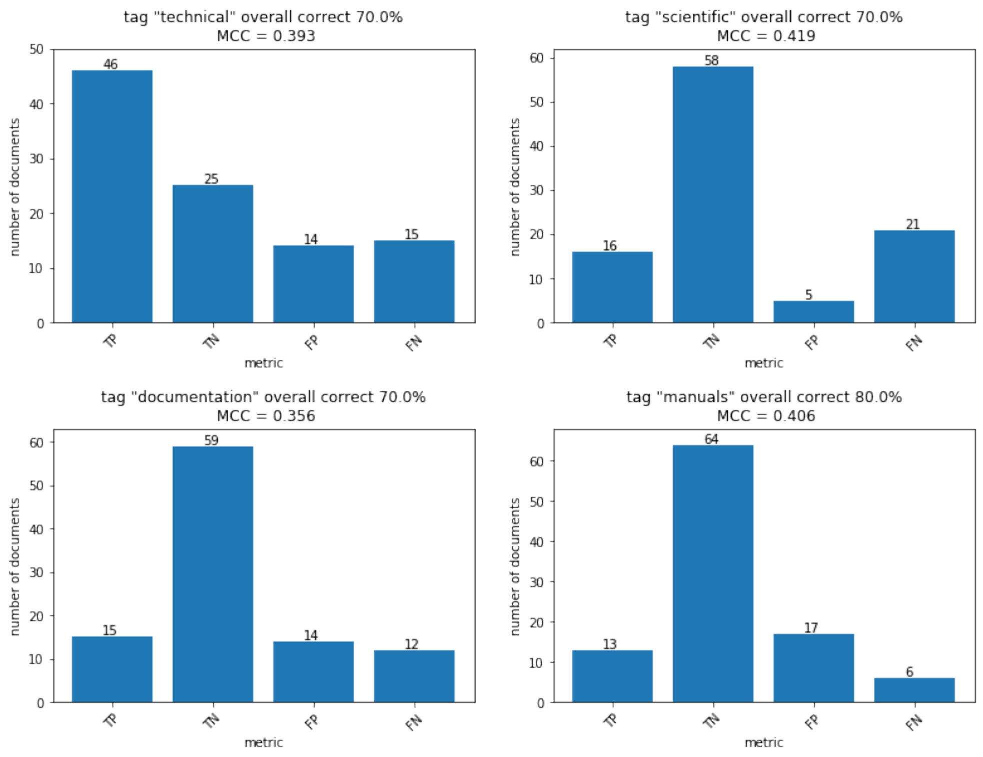

Figure 3.6: Elasticsearch classification results

The time required to index 400 (4 MB of text) manually tagged to query documents to be classified – 10 seconds. The time required to classify 100 (1 MB of text) documents – 7 seconds.

Since tags are not distributed evenly in the documents applying threshold for did not yield in the improvement of the results and only cause an increase in a false negative count.

Benefits of using Elasticsearch for classification:

- Fast document indexing
- Fast evaluation time
- Utilizes software that is already used in the project
- Does not require any additional libraries or software

Drawbacks of using Elasticsearch for classification:

- Even when using a large well-established data set only achieves 72% of correct classification [2], and that is barely above the minimum requirement set by the business.
- Relies heavily on having very good ”pre-classified data” meaning an equal amount of text distributed among tags
- Elasticsearch scores are difficult to interpret and they don’t scale. More text in ”pre-classified data” often results in a higher score, even though when manually reviewed relevancy is not better.

Overall, Elasticsearch as a classification engine does not provide good results when working from small data set with unevenly distributed text among tags. For it to start producing results large data set is required and it does not fit in our use case when new tags can be added by request without users providing large data set to index beforehand.

Elasticsearch is better suited to tag new documents given a large corpus of existing classified data with few tags to be identified. And that does not fit into the described use case.

3.3 Semaphore

Semaphore is a commercial ”assisted intelligence”[3] product that can classify data according to predefined rules.

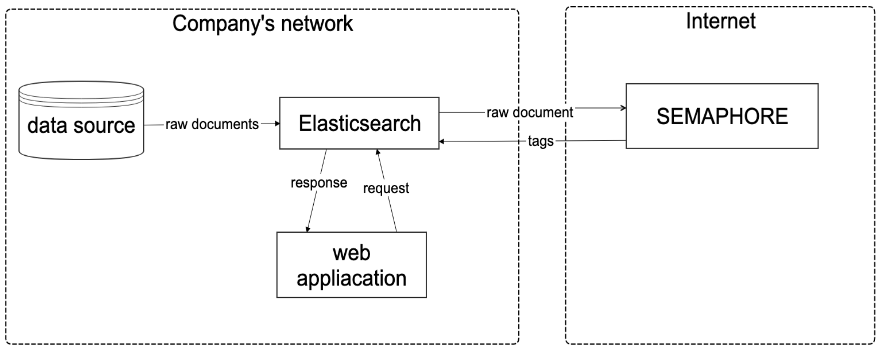

Figure 3.7: Semaphore classification infrastructure

3.3.1 How it works

Semaphore starts with a semantic model, that describes the concepts, topics, subjects, themes and organizational structures that are unique to your organization and use case. The model is published as rules - a series of templates, which can be modified to achieve specific classification outcomes - are generated. These rule-based are combined with sophisticated classification strategies to examine enterprise information. Precise and consistent metadata is applied. Organizations can leverage it to improve search and retrieval, power information-intensive processes. Comply with governmental mandates, secure sensitive information and aid e-discovery.[10]

3.3.2 Rulebasses

Semaphore expects rules to be generated beforehand by an expert in a field, and later, uses them to perform classification of the documents. Rulebasses are provided in the RDF format.

RDF is a standard model for data interchange on the Web. RDF (see figure 3.8) has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). This simple model, allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.[14]

```
        ...
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
  rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:country>USA</cd:country>
  <cd:company>Columbia</cd:company>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

</rdf:RDF>
        ...
```

Figure 3.8: Example of RDF document [15].

3.3.3 Implementation

Since creating new rulebase requires expert knowledge in a field, it was decided to use the ontology model that already exists in a company and is applied to a different data source.

To match tags that are defined for our use case it was decided to create a mapping from final categories of the existing model to tags from our classification model.

To run classification task Semaphore provides Java API that allows users to send the text from a document and returns a list of pairs tag - confidence score. To determine what tags should be kept ”knee” function was applied, similar to usage with Elasticsearch.

3.3.4 Results and conclusions

To measure the performance of the method Matthews correlation coefficient was used [11]. Results that were acquired are high in false positive and false count that indicates that the model that is currently being applied to a different organized data source does not translate well enough to different data set. The high amount of false-negative indicates that there are documents that do not fit into the already defined category. The high amount of false positives indicates that keywords that are being used in the different data sources as a differentiating factor occurs in current documents, not following the logic that was established earlier.See figure 3.9.

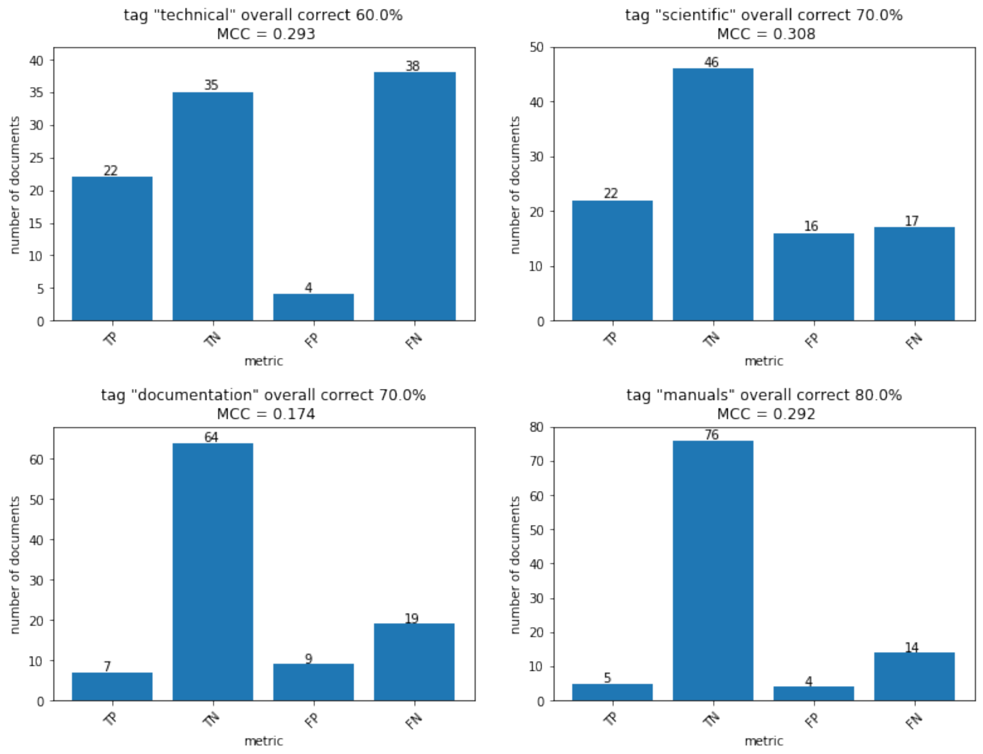

Figure 3.9: Semaphore classification results

The time required to upload the ontology model to a server - 5 seconds
The time required to classify 100 (1 MB of text) documents – 110 seconds.

Benefits of using Semaphore for classification:

- There is no need to have training data, so evaluation can be performed on the whole data set. However, to have a direct comparison with different models, I classified the same data as with other approaches.
- Easy setup
- Computational load is redirected from local machine to servers that are maintained by service provider
- Score system that is easy to interpret
- Existing ontology model that somewhat overlaps with our current task

Drawbacks of using Semaphore for classification:

- Ontology has to be created manually by respected fields professionals
- Only one instance of the product is available, which means it has to be shared with other projects, and rules that are applied within one rulebase can be conflicting for different teams.
- Evaluation bottlenecked by internet speed.
- Has a poor understanding of the document's context if keyword drug name is repeated multiple times, f.e. in some table that keyword will trigger system to classify the document as being scientific, however, it can contain technical information in the rest the of the columns.
- It doesn't learn with time, meaning that ontologies have to be manually edited to improve results. Implementation of a user feedback loop will still require manual work from the person that developed ontology.
- Hard to add new tags: if some group of users would want to add a new tag they would have to edit current ontology. Not a lot of people have experience with that and experimenting is rather hard, because there is only one instance of semaphore available.
- Distance between related terms has to be set manually as a number, and the only way to test if it results in improvement is to run classification manually and see if results are better. Given that there can be thousands of parameters doing this work manually to achieve optimal performance can take unjustifiably large amounts of time.
- Black-box approach to classification. Users can only provide ontology rest of the evaluation is hidden from control.

- Documents have to be sent via the internet outside of the company's network and even though the Semaphore product page claims to be secure [10] that is an additional attack vector for a potential hacker and unnecessary risk for our company to trust to a third party service.

Overall, Semaphore does not fit into our use case. Since the goal of our service is to provide high level-tags, creating an ontology for that purpose would be a job for an expert in a field and that is not what we have at our disposal.

Given a large list of drawbacks, I can't justify recommending purchasing another instance of Semaphore. Or spending time to create ontology specific to our use case, since it will not scale well. When new tags will be requested, an expert who can create an ontology for that specific tag without breaking any existing rules will be required. Finding such an expert is a difficult task.

Semaphore would better fit to assign low-level tags to documents that have a similar structure, yet there exists differentiating logic known to some expert and this logic can be expressed in RDF format.

3.4 LSTM

LSTM is a machine learning model that allows to work with word tokens to predict the category of the document.

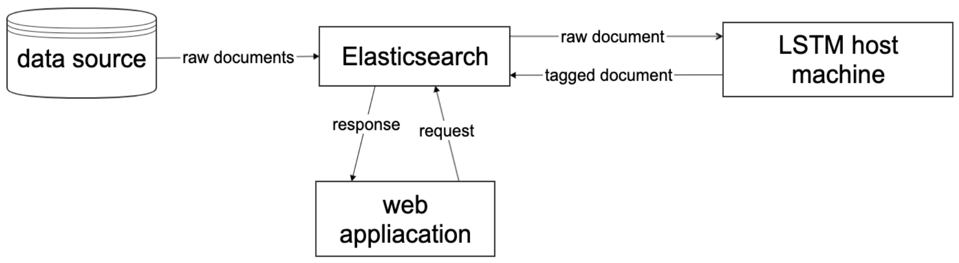

Figure 3.10: LSTM classification infrastructure

3.4.1 How it works

Long Short-term memory is a recurrent neural network that can learn to follow time intervals in excess of 1000 steps, without loss of short time capabilities. This is achieved by an efficient, gradient-based algorithm for an architecture enforcing constant (thus neither exploding nor vanishing) error flow through internal states of special units (provided the gradient computation is truncated at certain architecture-specific points this does not affect long-term error flow though).[16] See overview of a single LSTM unit 3.11

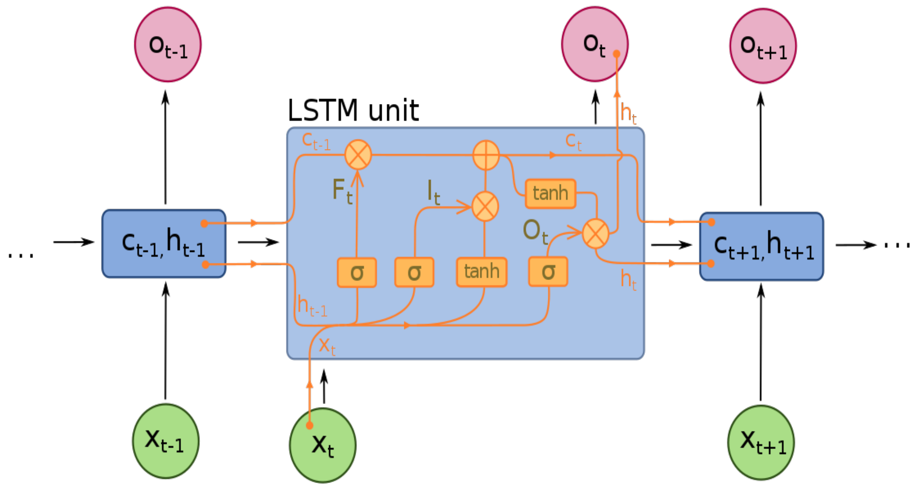

Figure 3.11: Single LSTM unit

While processing text each word is converted to vector with a given amount of dimensions. Then words are fed into LSTM that produces LSTM representation. And after that final layer makes decision about the document's class. See figure 3.12

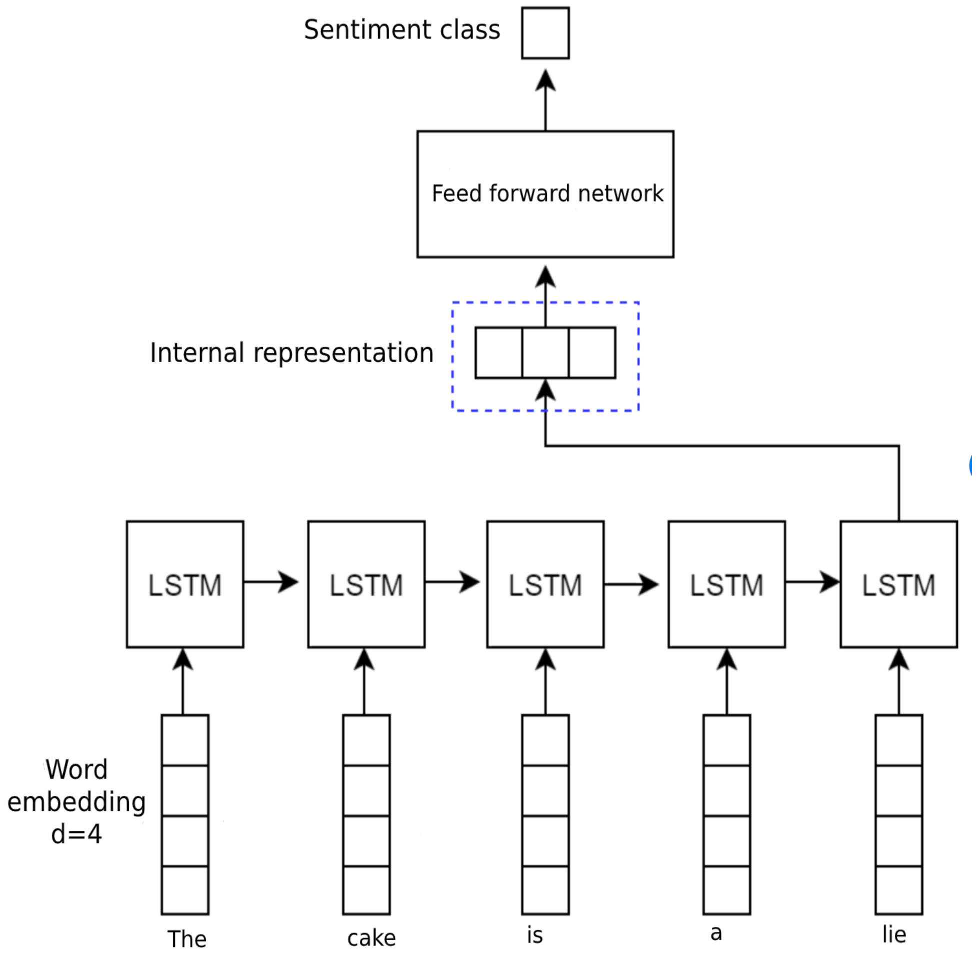

Figure 3.12: High level lstm overview [17]

3.4.2 Implementation

Original approach of recognizing multiple tags was tested, however final results were extremely poor with MCC score close to zero, so classification using multiple classifiers was implemented.

Multiple binary classifiers will allow adding new tags to documents, without need to retrain the big model. Also, that allows to share models and to train specific model independently.

Text tokenization was performed with python library ”keras”. No restriction on the number of recognized tokens was imposed.

Further parameters:

- Number of epochs – 20
- Batch size – 40
- Embedded dimensions – 512
- Dropout – 0.1
- Recurrent dropout – 0.1
- Optimizer – Adam
- LSTM layers – 512
- Loss function – categorical cross entropy
- Early stopping
  - patience – 4
  - min delta – 0.001

These parameters were tweaked multiple times during the implementation and testing process. Presented configuration resulted in the best performance on inspected tags. Since the training and evaluating process is very time-consuming, I started from values that were 4 times smaller and increased them by a factor of two, except dropout and min delta that I was decreasing at the same rate. Respectively larger and smaller values were also tested. However, the results were worse than presented and will not be included here, since they don't present much value. Worsening of the results can be explained by model overfitting on train data. Early stopping was applied after 10th iteration as opposed to 18th before that.

3.4.2.1 Testing without text processing

Originally no modification to text was made. It was tokenized as is without any processing. Since original documents contain a lot of obscure symbols and numbers due to them often being optically recognized or converted from some data format to text automatically.

Results that were received were barely satisfactory. See figure 3.13.

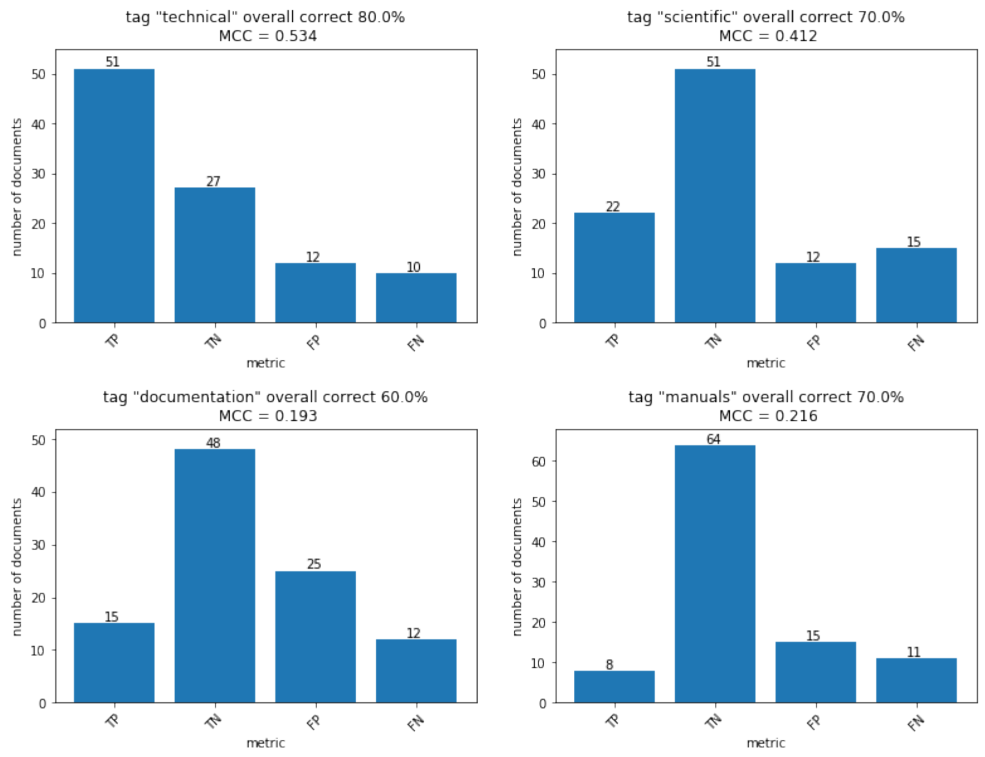

Figure 3.13: LSTM classification results. No text processing

3.4.2.2 Testing with text stemming

During the process of tokenization, all unique words are encoded as numbers – token. For example ”hello the world world#$##” can be tokenized as ”1 2 3 4”. To improve upon previous results, Elasticsearch capabilities of stemming words and removing stop-words [18] (list of all default Elasticsearch English stop-words [19]) were applied, that allowed to avoid all obscure characters as well as to recognize different variations of the same word as a single token. For example ”It is commonly recognized that the metabolic reaction” was transferred to ”commonli recogn metabol reaction” and words like ”do, did, done” would be transformed into ”do”. After applying Elasticsearch word-stemming string from the previous example will be transformed from ”hello the world world#$##” into ”hello world world” and subsequently tokenized as ”1 2 2”. From this example, we can observe that after applying word-stemming we reduce our vocabulary size as well as remove artifacts of text recognition and preserve better relationship between words by recognizing different variations of the same word as one token.

Since Elasticsearch can be run locally, there was no bottleneck from using the internet.

After stemming text with Elasticsearch new results showed some improvement in the MCC score. Except for manual, however the amount of false positives decreased there as well so, that is not a total downgrade. See figure3.14.

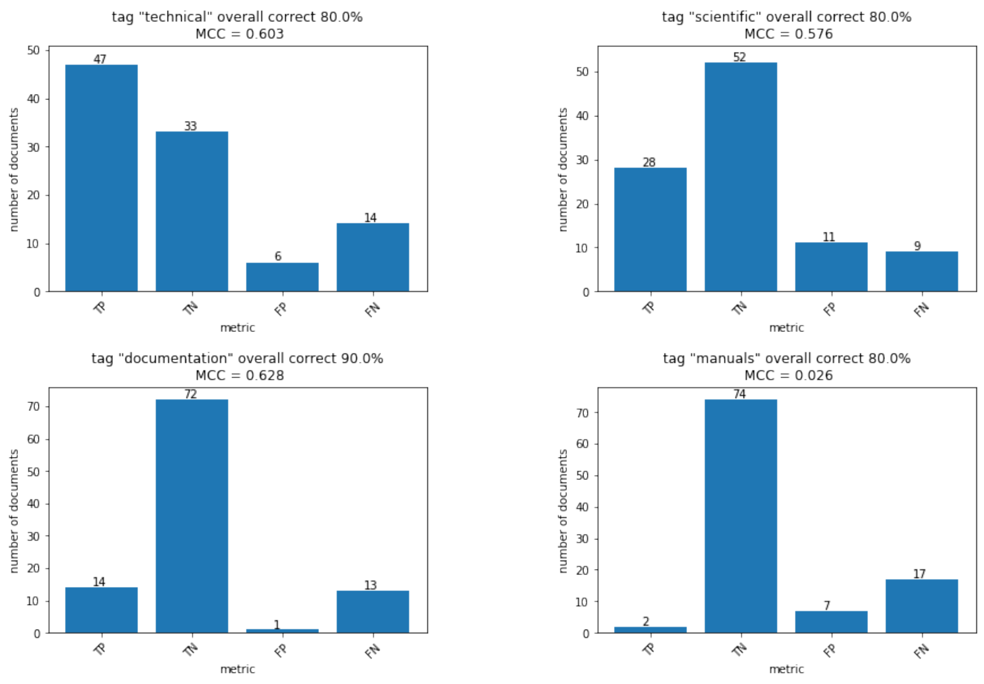

Figure 3.14: LSTM classification results with text processing

3.4.2.3 Testing with text stemming and truncating of the data

Trying to combat false positive and false negative results I tried to even out the number of documents representing each tag as well as the amount of text between those tags. I made it so that there is not more than 1,5 times more of a tag present than tag missing. I did it with a combination of text stemming with Elasticsearch.

However, this did not improve my results.See figure 3.15.

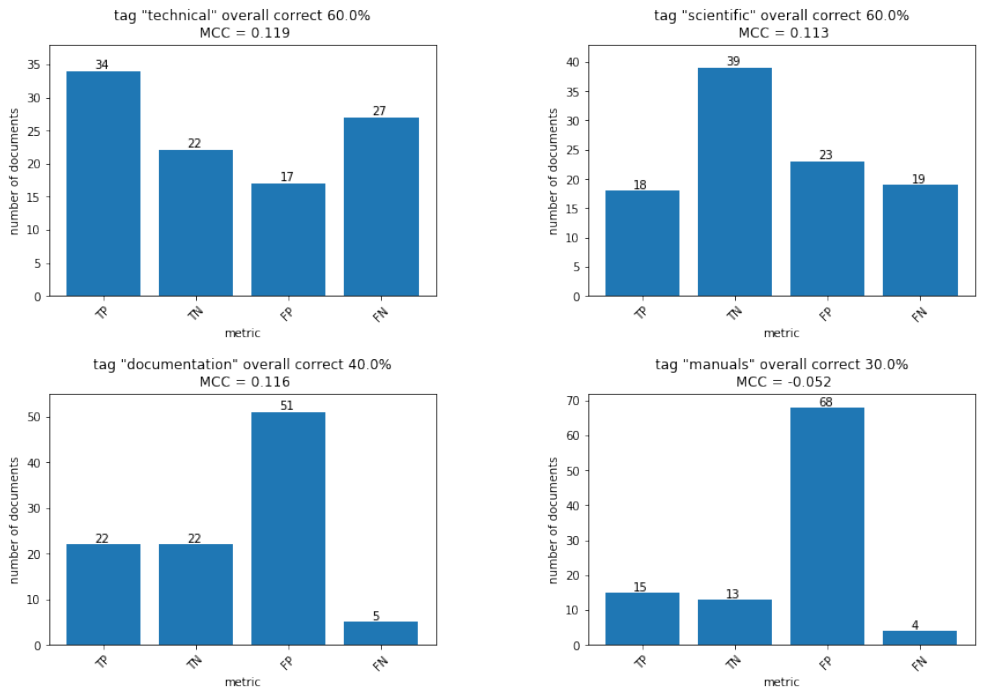

Figure 3.15: LSTM classification results with text processing

3.4.3 Results and conclusions

Results with stemming from Elasticsearch showed an improvement almost entirely across the board. See figure 3.14

Time to train model – 1,5 hrs per model. However, this should be done only once.

Time to evaluate 100 documents (1MB of text) – 110 seconds. To test if the speed of evaluation depends on the number of documents or the amount of text in a single document large file with 1.2MB of text was created and evaluated. The process took up to 1.4 seconds, indicating that the amount of text in a single document does not affect the speed of evaluation as much as the number of documents themselves. The speed of evaluation is within requirements defined for this task.

Benefits of using LSTM for classification:

- Mobility each classifier can be saved copied and distributed across multiple machines.
- Modularity since each classifier makes an independent decision it can be improved without affecting others. Hypothetical scenario when this can be very handy - Group of people G wants to add a new tag T and they can provide a set of documents that contain information relative to that tag and set that doesn't train a new model, in that case, would not be a difficult task, and that tag can be added on as many data sources as they wish, as long as the quality of original data is guaranteed the result can be very good [20]
- Multiple parameters that can be fine-tuned for each separate model to produce the best result, without affecting other models. This would be very hard to do with approaches such as Elasticsearch or Semaphore.
- If users, that are experts in researched fields would be given an option to tag documents, from within a search app, then with time as train data set grows accuracy of each classifier will be increased, something that is missing with Semaphore.
- Easy to interpret confidence score, unlike Elasticsearch score provided by LSTM is a number between 0 and 1 that is easy to understand and process.

Drawbacks of using LSTM for classification:

- Time to train the model (about an 60 min) and tags assigning times are inferior to Elasticsearch as well as ease of implementation.
- Multiple parameters that can be almost endlessly fine-tuned to optimize the result. So desired performance can be achieved without too

much trouble, however, achieving optimal performance is a very time and computationally consuming process.

Overall, LSTM produced a good result. Slower evaluation process and training time as well as more complex implementation should not be a big concern since all those processes will not be run continuously. While benefits that users will get from a better search result will outweigh the initial ”sunk” cost of LSTM.

From my unsuccessful experiment with truncating data, it is clear that the increase in the number of documents benefits the accuracy of predictions by a lot. Thus if the search app will give a way for users to tag documents, then the accuracy of predictions will increase rapidly.

3.5 BERT

Bert is a language representation model. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.[21]

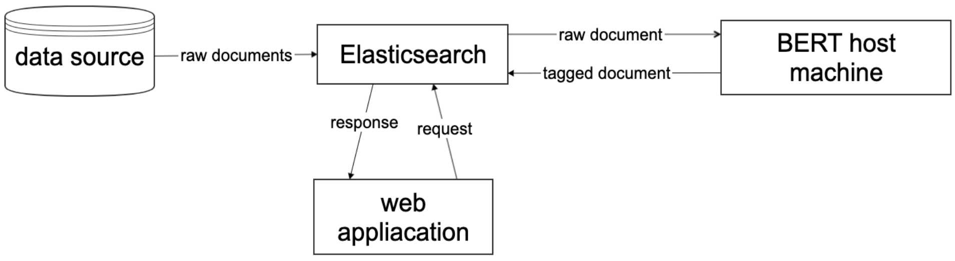

Figure 3.16: BERT classification infrastructure

3.5.1 How does it work

Bert is pre-trained on a large corpus of data such as Wikipedia-dump or IMDB movie review database. There are several models available online. Such as BERT-base BERT-large – original models from Google, Roberta base and large – models from Facebook, Albert, distil-BERT, and many other models from different teams. Each model has a different amount of hidden layers and total parameters, however, they are all pre-trained on some large corpus of data and can perform different natural language processing tasks based on the final layer and fine-tuning applied to them.

3.5.2 Implementation

To interface with BERT models I used python library ”simpletransformers” that allows to fine-tune BERT and perform an evaluation of the results measured in MCC.

Since BERT by default is using WordPiece embedding [21], the application of Elasticsearch for word stemming was not necessary. By default words will be transformed from for example ”doing” to ”do+##ing”

Parameters used for BERT model:

- max sequence length – 256

- train batch size – 36
- evaluation batch size – 28
- number of train epochs – 10

Increasing max sequence length and batch size caused a python kernel failure on my machine. I do not include experiments with smaller parameters since they yielded results that were worse across the board and thus have little value.

3.5.3 Results

In this section, I will present the results that I got from using different initial BERT models.

3.5.3.1 BERT-base

Using the standard version from Google yielded good results that were better than previously tested methods. See figure3.17

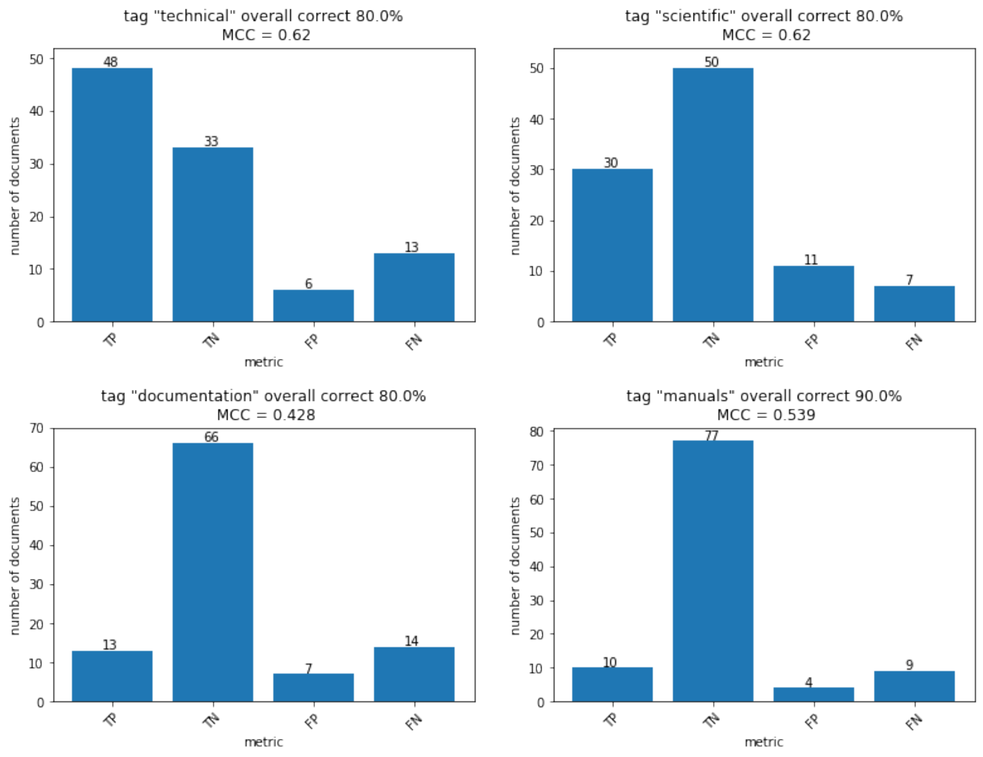

Figure 3.17: BERT-base model results

Time to fine-tune one model – about 3 hours.

Time to evaluate 100 documents (1MB of text) – 50 seconds. To test if the evaluation time depends on the amount of text in a single document single entry with 1.2MB of text was created. Evaluation time for that document was 0.94 seconds which is within specified requirements.

3.5.3.2 BERT-large

To test if a bigger model will result in better results I tried running BERT-large, however, results did not improve, in the contrary amount of false-positive increased across the board. See figure 3.18 Time required to fine-tune

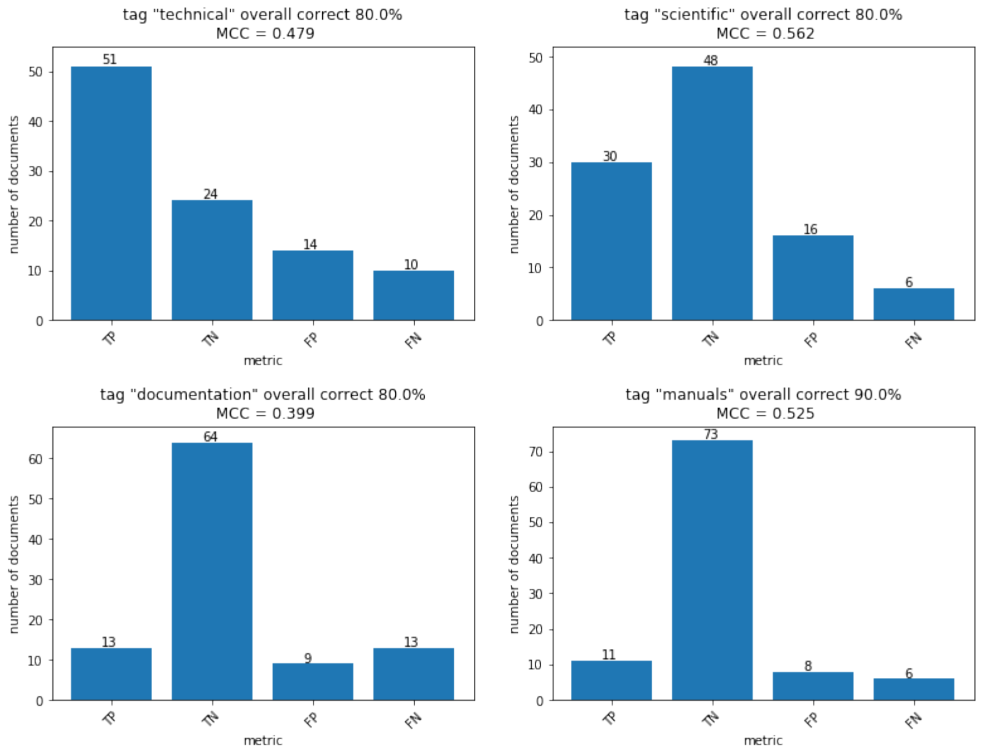

Figure 3.18: BERT-large model results

a single model is up to 7 hours.

Time to evaluate 100 documents (1MB of text) – 70 seconds. To test if the evaluation time depends on the amount of text in a single document single entry with 1.2MB of text was created. Evaluation time for that document was 1.12 seconds which is within specified requirements.

3.5.3.3 RoBERTa-base

RoBERTa is a model from Facebook that uses 120 million total parameters as opposed to 110 million in BERT-base and has a higher place in GLUE benchmark [6] Since fine-tuning of the model is a very time-consuming process I will only work with ”technical” tag, since it has the best representation in the original data set. See figure 3.19

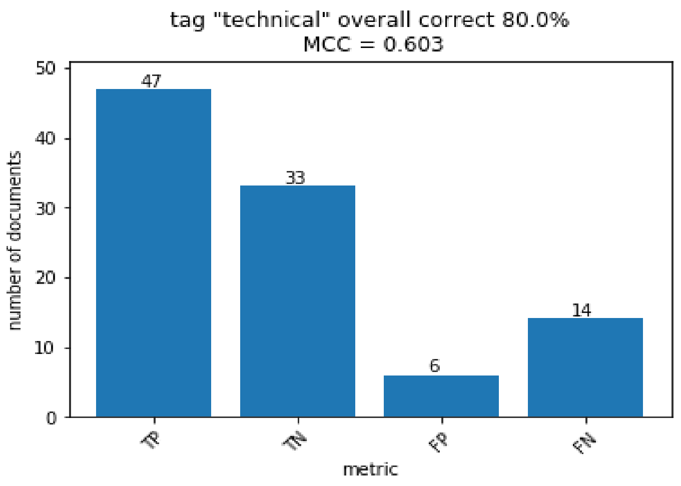

Figure 3.19: RoBERTa-base evaluation result

Time to train the model – 2.5 hours.
Time to evaluate – 53 seconds.

3.5.3.4 RoBERTa-large

RoBERTa-large is a model from Facebook that uses 755 million total parameters as opposed to 110 million in BERT-base and has a higher place in GLUE benchmark [6]

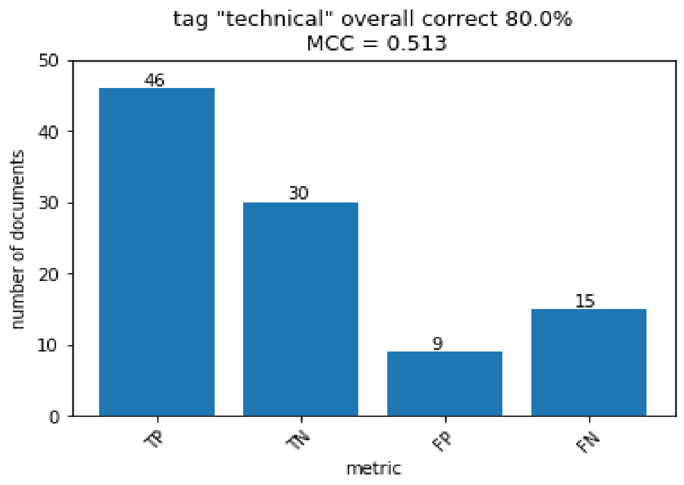

Figure 3.20: RoBERTa-large evaluation result

3.5.3.5 AlBERT-xxlarge-v2

Albert model is leading at glue benchmark [6] and did recognize the most of documents with tags, however also produced a lot of false positives.

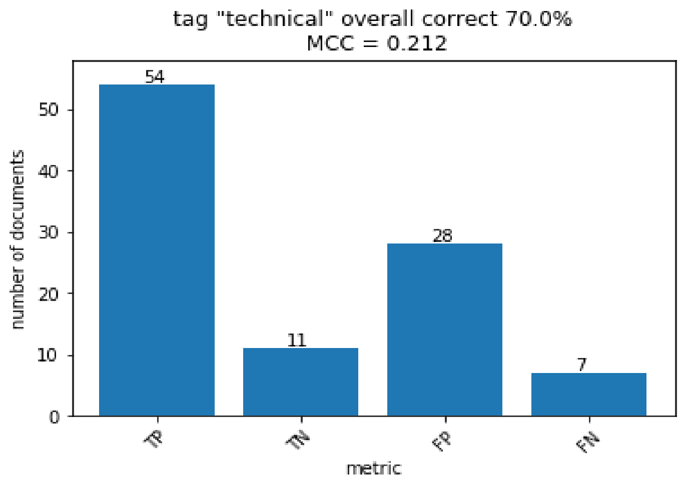

Figure 3.21: AlBERT-large evaluation result

Time to train model – 6 hours. Time to evaluate – 61 seconds.

3.5.3.6 distilBERT

Since smaller models seem to perform better I decided to test smaller model yet – distilBERT, with 60 million parameters as opposed to 110 million in BERT-base.

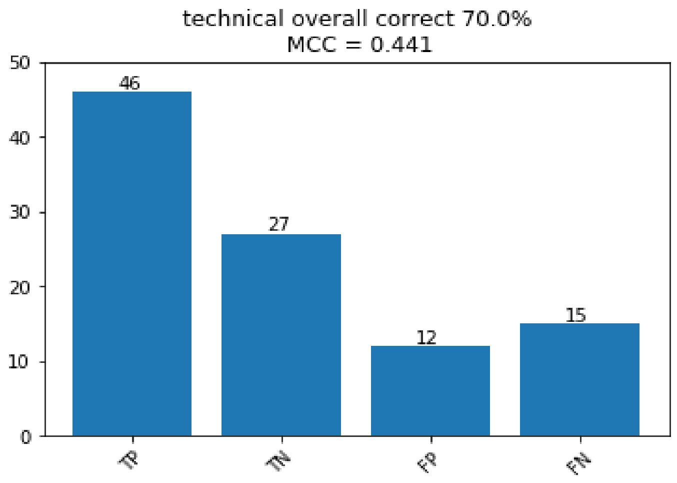

Figure 3.22: DistilBERT evaluation result

Time to train model – 2 hours. Time to evaluate – 61 seconds.

3.5.4 Conclusions

BERT-base produces the best results among other BERT models and overall. The most interesting results are for ”manuals” and ”documentation” tags. Even though their representation in the original data set is not very high 110 and 145 respectively out of 500. Even though LSTM with Elasticsearch did outperform BERT in ”documentation” tag figure 3.14 BERT was better in other categories and is more consistent. Also, there is a significant improvement with more documents even a slight increase to 180 scientific tags results in a better performance figure3.17 . From my testing of different BERT models, I observed that larger models tend to have more false-positive results figure 3.21 figure 3.18 and smaller model also didn't perform very well figure 3.22. Both RoBERTa and BERT-base had around 120 million parameters and both were rather close in the final result figure 3.17 figure 3.19, so models of that size seem to perform the best on a small data set in our particular use case.

Benefits of using BERT for classification:

- Results that are better than in other approaches.
- Modularity – allows to assign tags independently from each other.
- Easy to interpret confidence score.

- Does not require the use of external services, meaning that data does not leave the corporate network.
- Relatively fast evaluation time. Performed better than LSTM, while having a better score.

Drawbacks of using BERT for classification:

- Time to fine-tune a single model is very large. Depending on the model can take 6 or more hours.
- Computationally heavy – consumes 50-60 GB of ram when using ”large models” and 20-30 GB when using smaller models.
- Resulting models are 400-900 MB each.

Overall all drawbacks can be looked at as ”sunk” costs, so a longer approach is utilized, less important initial time investment is. So I think that BERT suits our use case the best.

Chapter 4

Application

In this chapter, I will answer questions of how good predictions are after training on the initial data set.

BERT model evaluates the text and returns a logit values for example [-3,0.5], I applied inverse function – expit and treated those as a confidence score.

I set a minimal threshold of reporting a tag to 0.9 and tested ”technical” tag. Not all documents that were tested contained text. See figure 4.1

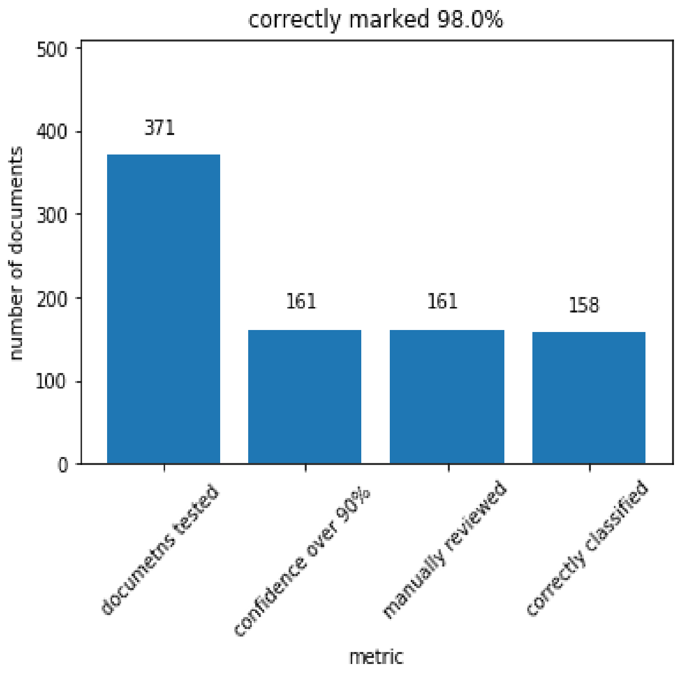

Figure 4.1: Result of manual review of documents with over 90% confidence score

Among 210 documents that did not pass the threshold of 90%, 150 were marked ”technical”, and 110 were marked correctly. That is still 73% accuracy which is above specified requirement. However, I think that a threshold of 90% confidence score should be applied, since it will provide a better user experience since it is better to have no tag than a misleading one.

Chapter 5

Improving data set

In this chapter, I will talk about approaches that I took to improve the original data set and about the results that I got out of it.

5.1 Using documents with high confidence score

I tried to improve the results by utilizing documents that had a high confidence score to improve evaluation results. For that, I kept the test data set without changes and enhanced train data set with documents that were marked by the model as those that had more than 90% confidence.

At first, I tried to increase the number of documents that did contain "technical" tag as well as those that didn't. There was no improvement in results. See figure 5.1

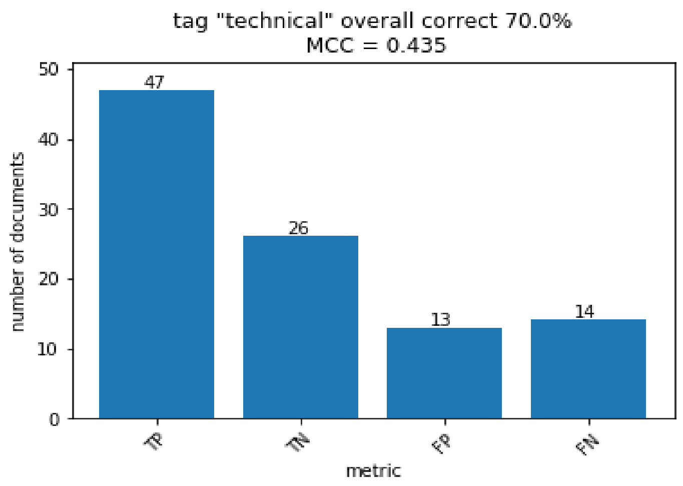

Figure 5.1: Result with train data set that was extended by technical and non technical documents automatically.

Then I extended the original data set by only documents that did not contain ”technical” tag thus making data set evenly contain ”technical” and non-technical documents. There was a significant increase in true positive documents and false-positive documents. See figure 5.2

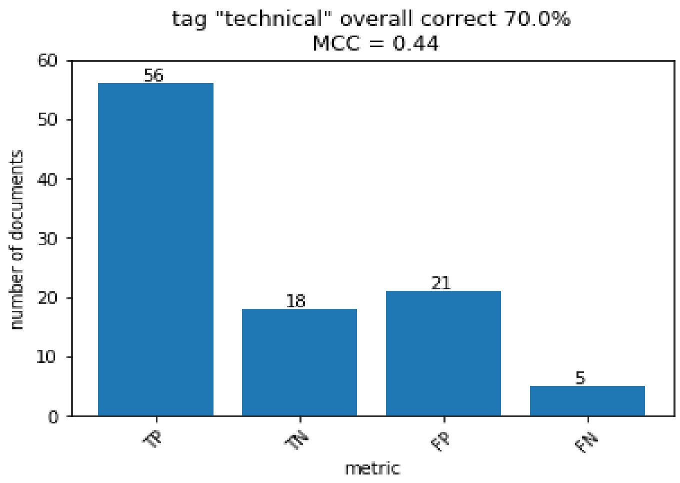

Figure 5.2: Results with train data set that was extended by non technical documents automatically.

I also tested the performance of the model on a new test set, for that, I added documents to the original data set and performed a new train-test split. See figure 5.3

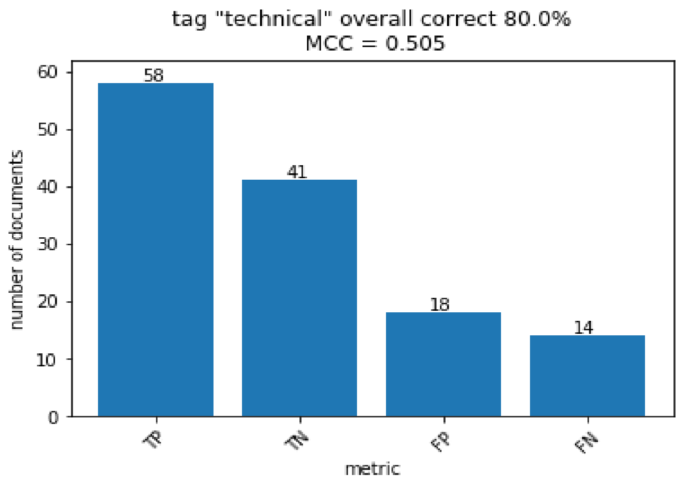

Figure 5.3: Result with data set that was extended by technical and non technical documents automatically. New train test split.

5.2 Using documents with low confidence score

To improve the original data set I run classification on random documents from the data source. I saved documents with a low confidence score (lower than 0.7) after that I manually tagged them.See figure 5.4

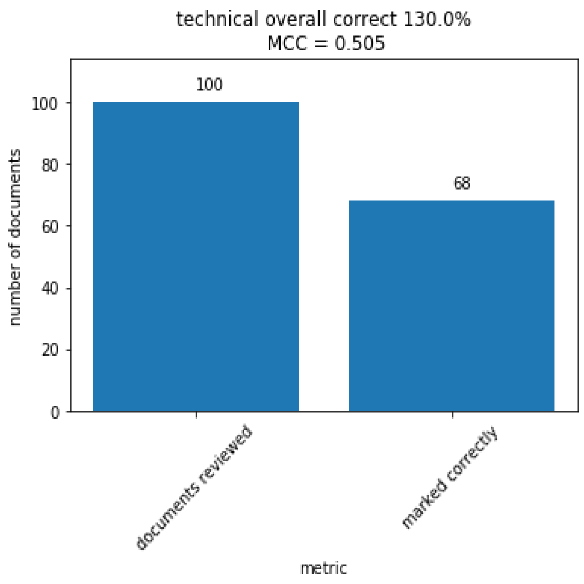

Figure 5.4: Manual review of documents with confidence score below 0.7.

I added manually tagged documents to train data set and evaluated results. See figure 5.5

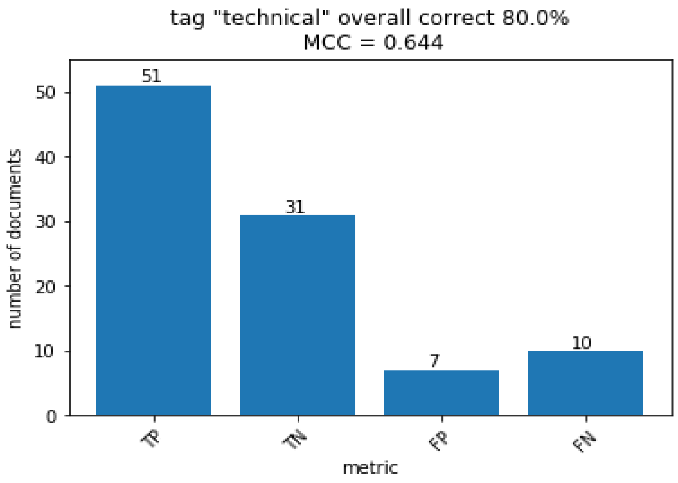

Figure 5.5: Results after adding manually reviewed documents with low confidence score.

Then I reduced amount of documents that contained tag ”technical” and added only documents that didn’t. See figure 5.6

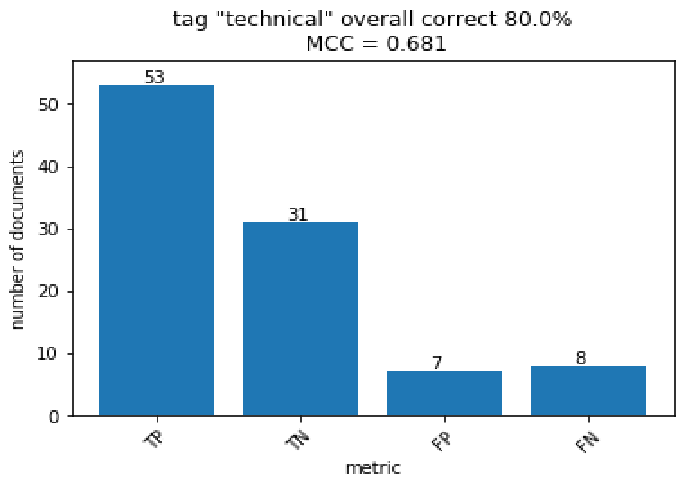

Figure 5.6: Results after adding manually reviewed documents with low confidence score that did not contain a tag, making data set balanced.

I tested how this model will perform evaluating the same random documents from the data source. See figure 5.7

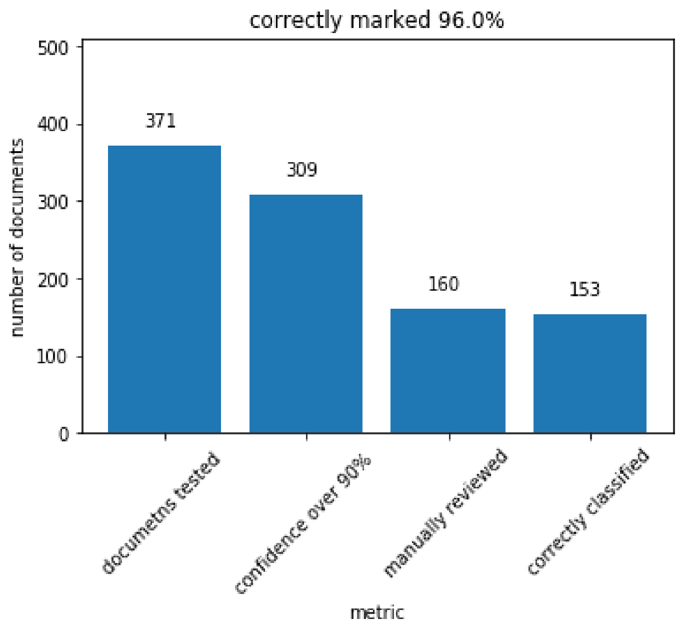

Figure 5.7: Result of manual review of documents with over 90% confidence score.

A lot more documents were marked with high confidence score 83% compared to previous 43% (figure 4.1). The resulting accuracy is slightly lower. Five out of seven documents that were marked incorrectly contained some binary data in their text field, since technical documents are often represented with tables odd characters probably caused the model to misinterpret the text. To avoid these documents that do not contain words from the English dictionary should not be sent for evaluation, removing all non-alpha characters from the text should help to resolve this issue.

5.3 Results

Manually reviewing documents with a low confidence score and fixing errors produced by automatic evaluation resulted in a better MCC score as well as an increasing amount of documents with a high confidence score. This approach requires more manual labor than utilizing documents with a high confidence score but produces better results.

Reviewing documents with a low score is more productive than reviewing random documents since it allows to avoid documents that the model is capable of correctly identifying from the initial set and concentrate on less typical documents providing a bigger variety and as the result better model.

Conclusion

Overview

Automatic text classification is the subject of this thesis. Text classification is an important field of research since it allows to provide users with insight into document context, filter unwanted results and speed up the search for an answer, by reducing the volume of documents that have to be considered.

In our particular use case (classification of large data source that contains unstructured documents of different types) usage of several different approaches was plausible to compare results of different classification techniques Matthews correlation coefficient (MCC) was used.

Finding an approach that can both be fast enough and be able to work from a small initial data set was crucial.

Work done

As an initial step data set of manually classified documents was created.

Elasticsearch, Semaphore, LSTM, BERT classifications were implemented and tested. For all approaches, different combinations of parameters were tested and those that yielded the best results are presented in this work.

After establishing an approach that allows to perform the best classification several different methods were tested to improve the initial data set. Such as - utilizing the initial model to find documents with a high confidence score of a tag being present in an evaluated text. Finding documents with a high confidence score of tag not being present. Finding documents with confidence scores close to 0.5 and manually classifying them.

Results

Some methods were found to be completely impractical in their implementation since they would require a team of experts to produce complicated rule bases that are hard to maintain and extend upon chapter 3.3. The other was easy and fast to implement and provided very fast evaluation with the minimal new infrastructure required chapter 3.2, but produced results inferior to other approaches. Both machine learning models produced decent results with over 80% accuracy and MCC score above 0.6 for well-represented tags figure 3.14 figure 3.17. BERT models were found to be slightly better and this result aligns with the glue benchmark [6]. BERT models take more time to train and produce better results. It was found that large BERT models did not perform better with small data set and were prone to produce false positive results.

Overall BERT approach produced the most accurate results as well as satisfied all original use case requirements. And will be implemented in production.

The manual classification of documents with a low confidence score was found to be the most productive one. Classification of a small additional set of documents yielded in the improvement of the model both in total accuracy and in assigning a higher confidence score to correctly classified results. For overview of all classification methods see table 5.1.

Future work

More research can be done in the area of bootstrapping of the initial data set finding new approaches and determining the minimal amount of manually classified documents required, before producing valuable results.

Another topic for further investigation would be creating own BERT pretrained model based on corporate documents and using it for classification.

Tag / Method	technical	scientific	documentation	manuals
Elasticsearch	0.393	0.419	0.356	0.406
Semaphore	0.293	0.308	0.174	0.292
LSTM without text stemming	0.534	0.412	0.193	0.216
LSTM with text stemming	0.603	0.576	0.628	0.026
LSTM truncated	0.119	0.113	0.116	-0.052
BERT-base	0.62	0.62	0.428	0.539
BERT-large	0.479	0.562	0.399	0.525
RoBERTa-base	0.603	–	–	–
RoBERTa-large	0.513	–	–	–
AlBERT-xxlarge-v2	0.212	–	–	–
distilBERT	0.441	–	–	–
BERT-base extended automatically	0.435	–	–	–
BERT-base extended automatically balanced	0.44	–	–	–
BERT-base extended automatically new split	0.44	–	–	–
BERT-base after extending data set manually	0.644	–	–	–
BERT-base after extending data set manually balanced	0.681	–	–	–

Table 5.1: MCC scores of different approaches

Appendix A

Acronyms

BERT	Bidirectional Encoder Representations from Transformers
LSTM	Long short term memory
RDF	Resource Description Framework
MCC	Matthews correlation coefficient
MLT	More like this
TP	true positive.
FP	false positive.
TN	true negative.
FN	false negative.

Bibliography

1. 2014. Available also from: `https://lucene.apache.org/core/4_9_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`.
2. VOLA, Saskia. *Text Classification made easy with Elasticsearch.* 2020. Available also from: `https://www.elastic.co/blog/text-classification-made-easy-with-elasticsearch`.
3. *AI.* Available also from: `https://www.smartlogic.com/trends/AI`.
4. JANNESKLAAS. *19 - LSTM for Email classification.* Kaggle, 2018. Available also from: `https://www.kaggle.com/jannesklaas/19-lstm-for-email-classification`.
5. TRIVEDI, Kaushal. *Multi-label Text Classification using BERT – The Mighty Transformer.* HuggingFace, 2019. Available also from: `https://medium.com/huggingface/multi-label-text-classification-using-bert-the-mighty-transformer-69714fa3fb3d`.
6. *GLUE Benchmark.* Available also from: `https://gluebenchmark.com/leaderboard`.
7. KEITAKURITA, Author. *Paper Dissected: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" Explained.* 2019. Available also from: `https://mlexplained.com/2019/01/07/paper-dissected-bert-pre-training-of-deep-bidirectional-transformers-for-language-understanding-explained/`.
8. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. *CoRR.* 2017, vol. abs/1706.03762. Available from arXiv: `1706.03762`.

9. KEITAKURITA, Author. *Paper Dissected: "Attention is All You Need" Explained*. 2019. Available also from: `http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/`.
10. *Semaphore Classification Provides Superior Results*. Available also from: `https://www.smartlogic.com/news-and-blog/entry/content-classification-for-superior-results`.
11. *The Best Metric to Measure Accuracy of Classification Models*. Available also from: `https://www.kdnuggets.com/2016/12/best-metric-measure-accuracy-classification-models.html/2`.
12. *Built-in analyzer reference: Elasticsearch Reference [7.7]*. Available also from: `https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html`.
13. *More like this query: Elasticsearch Reference [7.6]*. Available also from: `https://www.elastic.co/guide/en/elasticsearch/reference/7.6/query-dsl-mlt-query.html`.
14. *RDF*. Available also from: `https://www.w3.org/RDF/`.
15. Available also from: `https://www.w3schools.com/xml/xml_rdf.asp`.
16. HOCHREITER, Sepp; SCHMIDHUBER, Jurgen. *bioinf*. 1997. Available also from: `https://www.bioinf.jku.at/publications/older/2604.pdf`.
17. VO, Quan-Hoang; NGUYEN, Huy-Tien; LE, Bac; NGUYEN, Minh-Le. Multi-channel LSTM-CNN model for Vietnamese sentiment analysis. In: 2017, pp. 24–29. Available from DOI: `10.1109/KSE.2017.8119429`.
18. Available also from: `https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html`.
19. APACHE. *apache/lucene-solr*. Available also from: `https://github.com/apache/lucene-solr/blob/branch_7x/lucene/core/src/java/org/apache/lucene/analysis/standard/StandardAnalyzer.java#L47-L53`.
20. NGYPTR. *Multi class classification with LSTM*. Kaggle, 2018. Available also from: `https://www.kaggle.com/ngyptr/multi-class-classification-with-lstm/data`.
21. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, vol. abs/1810.04805. Available from arXiv: `1810.04805`.