Technische Universität Graz

Rok Hudobivnik

Keyword extraction and named entity recognition on Reddit submissions

master's thesis

the 2nd cycle master's study programme
computer and information science
Track: Computer and Information Science

Supervisor: Assoc. prof. dipl. ing. dr. techn. Denis Helic
Co-supervisor: prof. dr. Zoran Bosnić

Ljubljana, 2020

Technische Universität Graz

Rok Hudobivnik

Schlüsselwortextraktion und Erkennung benannter Entitäten in Reddit-Submissionen

masterarbeit

das studienprogramm des 2. zyklusmeisters
computer und informationswissenschaft
Studienweg: Computer und Informationswissenschaft

Mentor: Assoc. prof. dipl. ing. dr. techn. Denis Helic
Co-mentor: prof. dr. Zoran Bosnić

Ljubljana, 2020

Acknowledgment

To my family and friends who supported me all this time. Thank you for everything.

Rok Hudobivnik, 2020

"Intelligence is the ability to adapt to change."

— Stephen Hawking

Contents

List of used acronyms

acronym	meaning
AI	Artificial intelligence
KE	Keyword extraction
NER	Named entity recognition
POS	Part-of-speech
NLP	Natural language processing
MLP	Multilayer perceptron
CNN	Convolutional neural network
RNN	Recurrent neural network
LSTM	Long short-term memory
BERT	Bidirectional Encoder Representations from Transformers
MLM	Masked language model
QA	Question answering
NLI	Natural language inference
CRF	Conditional random field

Abstract

Title: Keyword extraction and named entity recognition on Reddit submissions

The goal of this thesis was to create a pipeline for extraction of valuable information from short natural language texts, more specifically Reddit submissions. The two main areas of research that we covered were keyword extraction and named entity recognition for the extraction of keywords and the recognition of actors and movie titles in the texts. In our thesis we implemented and evaluated four different approaches for keyword extraction (RAKE, TextRank, LSTM and biLSTM networks) and three different approaches for named entity recognition (Spacy library models, Stanford NER and Fine-tuned BERT models). The analysis of the algorithms showed that the best results were achieved when using a three layered biLSTM network for keyword extraction, an uncased BERT model fine-tuned on the MIT movie corpus dataset for the recognition of actors, and the BERT model fine-tuned on the Ontonotes 5 dataset for the recognition of movie titles.

Keywords

Deep learning, named entity recognition, keyword extraction, analysis

Kurzfassung

Titel: Schlüsselwortextraktion und Erkennung benannter Entitäten in Reddit-Submissionen

Ziel dieser Arbeit war es, eine Pipeline zur Extraktion wertvoller Informationen aus kurzen Texten in natürlicher Sprache, insbesondere Reddit-Beiträgen, zu erstellen. Die beiden Hauptforschungsbereiche, die wir behandelt haben, waren die Keyword-Extraktion und die Erkennung benannter Entitäten für die Extraktion von Keywords und die Erkennung von Schauspielern und Filmtiteln in den Texten. In unserer Arbeit haben wir vier verschiedene Ansätze für die Schlüsselwortextraktion (RAKE, TextRank, LSTM und biLSTM-Netzwerke) und drei verschiedene Ansätze für die Erkennung benannter Entitäten (Spacy, Stanford NER und fein abgestimmte BERT-Modelle) implementiert und bewertet. Die Analyse der Algorithmen ergab, dass die besten Ergebnisse erzielt wurden, wenn ein dreischichtiges biLSTM-Netzwerk für die Keyword-Extraktion verwendet wurde, ein BERT-Modell für Kleinbuchstaben, das auf den MIT-Filmkorpus-Datensatz zur Erkennung von Akteuren fein abgestimmt war, und das BERT-Modell, das auf den Ontonotes 5-Datensatz zur Erkennung von Filmtiteln fein abgestimmt war.

Schlüsselwörter

Tiefes Lernen, Erkennung benannter Entitäten, Keyword-Extraktion, Analyse

Chapter 1

Introduction

1.1 Motivation

With the explosive rise in internet usage and content creation worldwide [1] in the past decade, there has never been a time in history when such enormous amounts of data are being consumed every second of the day. Finding content that we actively seek for or accidentally finding that we did not know we might like in such a world becomes increasingly more difficult, opening up new space for creative solutions. This shift in the amount of content we could interact with on our daily basis lead to the creation of systems that could find relevant information, products or services for the user. With this in mind, data classification, content curation and content recommendation took the center stage.

Nowadays, systems dealing with recommendations can be found practically everywhere, from their use in stores to internet advertising. Although that is the case, the field of recommendation that is based on specific user requests and descriptions, possibly written in natural language, stays somewhat neglected. A possible reason for that is the lack of appropriate datasets, containing item descriptions and their respective recommendations in natural, human understandable language, that were written by other users, based on the original users request.

With that being said, this thesis will primarily focus on processing specific user requests in short text forms. We believe that by using text written in natural language, we can obtain more relevant information about the users preferences and hence construct a clearer picture of the users request. In other words, the goal of this endeavour is to create an accurate and more importantly an automated process of creating a set of processed data that contains the most important bits of the original text. This could provide future research and business opportunities by making the process of including natural language requests in models easier and more streamlined.

1.2 Methodology

1.2.1 Keyword extraction

Keyword extraction (KE) is an information extraction task that seeks to locate and extract words or phrases of higher importance from unstructured text. For KE, keywords are chosen from words that are explicitly mentioned in the original text. By definition, keywords are words that describe the main topics expressed in text, hence they provide a lot of information about the contents of the text and can thus be leveraged for a variety of different tasks. Such tasks include text summarizing, document classification etc.

In most cases, keyword extraction starts with a selection of potential keywords called candidates for which a certain metric is calculated (centrality, frequency, co-occurrence etc). Based on the calculated metrics a number of candidates are selected as keywords.

1.2.2 Named entity recognition

Named entity recognition (NER for short) (also known as entity identification, entity extraction and entity chunking) is a process of information extraction aimed at locating and the subsequent classification of named entities in unstructured text into pre-defined user created categories (Image

1.1). Such categories often include organizations, locations, person names etc. NER can be found in many different fields related to artificial intelligence.

The process is usually divided into segmentation, paired with chunking of the segmented data, and later classification. During segmentation, the input text is in most approaches divided into individual words which are then equipped with their respective part-of-speech tags (POS tags) and clumped together using chunking methods. Classification of the tokens depends heavily on the approach used.

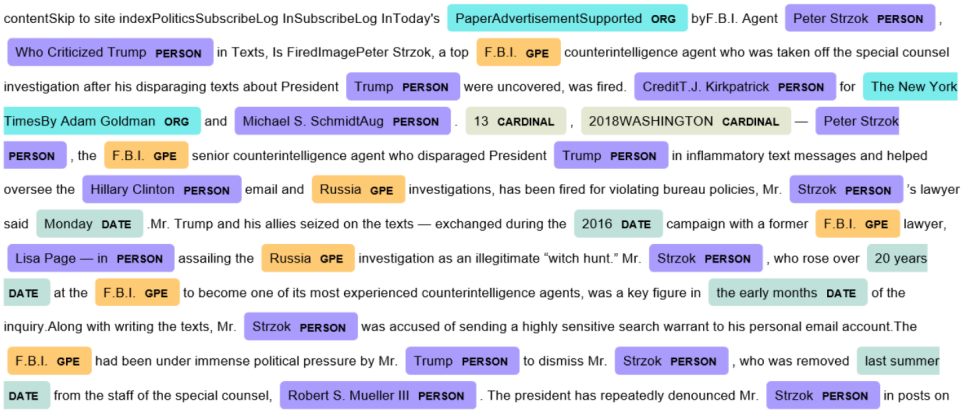

Figure 1.1: An illustration of a NER algorithm results.

1.3 Problem definition

The problem for this thesis is centered around extracting important words or phrases from short texts written in natural language. For the purposes of this thesis, we will primarily focus on processing specific request and recommendation from the website Reddit[1]. More specifically, from submission and their respective comments posted on a part of this website (a subreddit) named /r/MovieSuggestions[2]. On this subreddit the users post specific requests or questions about movies and other users try to recommend a movie

[1] `https://www.reddit.com/`

[2] `https://www.reddit.com/r/MovieSuggestions/`

to them, that they believe suits the expressed request. The processing of this kind of data includes the extraction of keywords and the recognition of named entities in the text, such as movie titles and names of actors, directors, etc. An example of such processing would be the following:

> `Spider-verse`[3]`:`
> I really like the artwork of the new Spiderman movie, Spider-Verse. Could anyone suggest me more movies like that?
>
> `From this submission we can extract the following details:`
> {artwork (keyword), Spiderman (named entity, name), Spider-Verse (named entity, movie title)}.

The problem can be divided into two main fields, keyword extraction and named entity recognition. For each of the two main fields, multiple different approaches will be used in order to determine how successful and suitable each one of them is for the problem in question. A part of this thesis will hence deal with the analytical evaluation of those algorithms. In the end, the ultimate goal is to present and analyse the results of the evaluation and based on those, construct a complete algorithm for automatic keyword extraction and named entity recognition of short texts.

1.4 Structure

This master's thesis is alongside the introduction divided into 5 chapters, detailing the research, experiments and the results of our work on this problem.

In Chapter 2 we will outline the work of other authors that was instrumental for our work. For each one of the algorithms we used, we try to explain the underlying architecture and principles behind it.

[3] `https://www.reddit.com/r/MovieSuggestions/comments/a8bpk8/spiderverse/`

In Chapter 3 we provide a detailed explanation on how we approached the problem, how the algorithms were implemented and what combination of test cases and parameter values were used.

Chapter 4 contains the results we obtained from the experiments explained in Chapter 3 and a thorough analysis of them. We compare the algorithms for both keyword extraction and named entity recognition against each other, respectively.

In Chapter 5 we take a critical look at our research and outline possible potential problems with our input dataset and approaches to the problem that could skew the obtained results. We propose a few potential solutions for the outlined problems.

Chapter 6 summarizes the results and gives the discussion of possible future work that could meaningfully expand our research.

Chapter 2

Related work

In the following chapter we will outline the related work that served as a basis for this thesis. We will try to explain the techniques and methods behind the used algorithms for the purpose of this work. The goal of this chapter is to present the existing algorithms and explain why they were selected for this thesis.

2.1 TextRank

In 2004, Mihalcea and Tarau [2] proposed a novel solution for the problem of KE, by representing the input text as a graph. The idea behind this is that keywords as a part of a text can be described as having connections to other words in that text. These connections can then be used as a sort of a voting, or rather a sort of recommendation system, where the connections can be seen as votes and the weight of these connections is determined by the importance of the vertex they originate from.

Formally, let the undirected graph be described as *G=(V,E)*, with a set of V vertices and a set of E edges (connections between vertices), where E is a subset of $V \times V$. Given this notation we declare, for a certain vertex V_i, a set of all vertices that point to V_i, to be $In(V_i)$ and the set of all vertices that V_i points to, to be $Out(V_i)$. With this we can define the importance of

a single vertex by:

$$S(V_i) = (1 - d) + d * \sum_{j \in In(V_i)} \left(\frac{1}{|Out(V_j)|} S(V_j) \right) \tag{2.1}$$

where d is the damping factor that can be set between 0 and 1 (set to 0.85 in this implementation of the algorithm). Due to this equation, the algorithm is iterative and by starting from a random starting assignment of node values, requires multiple passes through the graph to converge. The application of the graph based ranking algorithm to the natural language text follows the next selection of steps:

1) Identify text units that best define the task at hand, and add them as vertices in the graph. In the case of this task, singular words are used as vertices.

2) Identify relations that connect such text units, such as neighbouring words, parts of phrases etc. and use these relations to draw edges between vertices in the graph. Edges can be directed or undirected, weighted or unweighted, but we will use unweighted undirected edges (Figure 2.1).

3) Iteratively update the vertex scores until convergence.

4) Sort vertices based on their final score. Use the values attached to each vertex for ranking/selection decisions.

TextRank as an algorithm works due to the fact that keywords are highly connected to the rest of the text, meaning that a high number of words should vote for them. This algorithm does not require any deep linguistic or domain knowledge to run and is fairly simple implementation-wise, which is why we decided to use this algorithm in our work.

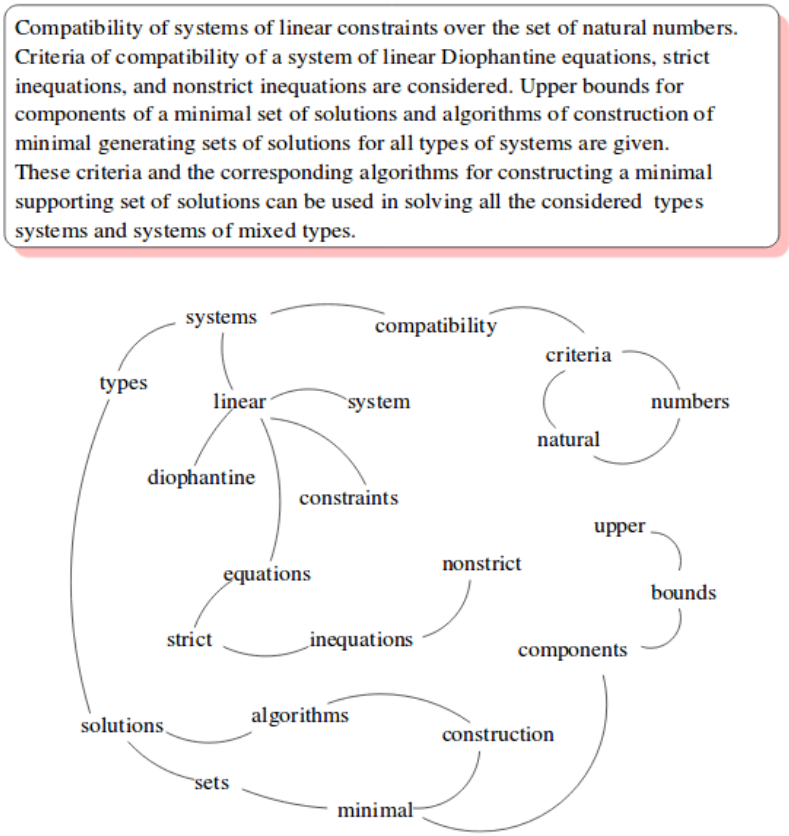

Figure 2.1: An illustration of an unweighted undirected graph created by TextRank.

2.2 RAKE

Rapid automatic keyword extraction (RAKE) was proposed in 2012 by Rose et al. [3], as a simple and efficient algorithm that could rival its concurrent algorithms for KE. The basic premise of this algorithm is the assumption that most keywords do not contain any punctuation marks or so-called stopwords, words like and, of, the etc. that provide minimal lexical meaning. The reasoning behind that is that due to the frequency of such words, they do not provide any meaningful contribution to analyses and search tasks.

The algorithm starts with an input set of stopwords, which are used alongside phrase delimiters to split the original text into sequences of words.

Words in a single sequence are assigned the same position in the text and are together considered to be a single candidate keyword. After all candidate keywords in the original text have been identified, a graph of word co-occurrence is computed and a score is calculated for each candidate keyword. The score of candidate keyword is defined as the sum of all of co-occurrence scores of the words that comprise the candidate keyword. Due to the minority of keywords that contain an interior interim stopword, the algorithm corrects its behaviour by joining together certain keyword candidates (and the interior interim stopword that connects them) that adjoin each other at least two times in the same order as in the same document (this becomes more common in longer input texts). Based on their previously calculated scores, the candidates are sorted and the top T of them are selected as keywords, where T is a user defined number.

The efficiency, low complexity, as well as the different way of approaching KE, were all factors in deciding to use this algorithm in our work.

2.3 Word2Vec

Words written in a natural language are in a sense discrete states connected to each other through language. Each language has its own vocabulary of words and a set of rules on how to connect them. Based on that, it is possible to compute transitional probability between words, discrete states. This notion is found at the very base of NLP. Since neural networks are designed to work with numerical data, these discrete states are transformed into numerical vectors in a process called word embedding. Words carry meaning, some more than the others, and have underlying connections to other words based on that meaning (i.e. words 'boy' and 'man'), which has to be accounted for when performing word embedding. Due to this fact, a simple One-hot encoding performs only a part of the task by vectorizing words, since the newly embedded representations do not retain any dependencies between each other.

In our thesis we will be using a word embedding method called Word2Vec, that was first proposed by Mikolov et al. [4] in 2013. The method makes use of a shallow, two layered neural network to process text and output numerical vector representations of words. In its architecture the network resembles an autoencoder, but rather than trying to reconstruct the same word as an output, the network attempts to predict neighbouring words (context) found in the training corpus. This is done in one of two ways, either using the input word to predict the target context (Skip-gram) or the other way around, using context to predict the target word (continuous bag of words, or in short CBOW) (Figure 2.2). Both approaches represent viable ways of computing the word embeddings, although the skip-gram approach is preferred due to a higher accuracy when compared to CBOW, which is a faster, yet less accurate approach. During training, similar words are pushed together due having similar contexts, preserving the dependencies between words. The output of this method is a vocabulary of all input words and their corresponding numerical vector representations.

2.4 LSTM and BiLSTM neural networks

Long short-term memory neural networks (LSTM for short) were first proposed by Hochreiter and Schmidhuber in 1997 [5]. Since then, the LSTM architecture has been improved and built upon, the most notable change was the addition of a forget gate by Schmidt and Schmidhuber [6] in 1999. In the recent years, the architecture has been adapted to natural language processing (NLP) due to its ability to keep track of dependencies between elements of input sequences, without the vanishing gradient problem that occurs in the simpler recurrent neural networks (RNN) for longer sequences. Unlike the classic feed-forward neural networks, LSTM takes advantage of a feedback loop (Figure 2.3), which is more suitable for processing not only single data points, but also entire sequences. A LSTM network is hence a good match for NLP. LSTM network is a special kind of a RNN, capable

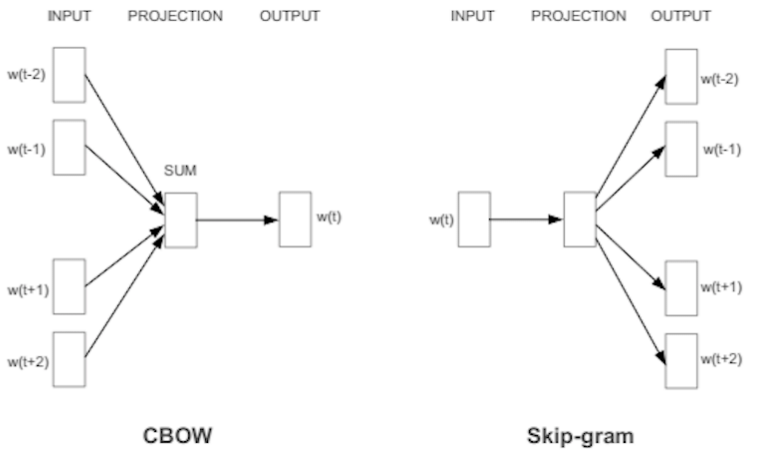

Figure 2.2: A visual comparison of the skip-gram and CBOW approaches to word embedding. In the CBOW model, the or surrounding words (word embeddings, represented by the rectangles) are combined to predict the word in the middle. On the other hand, in the Skip-gram model, the embedded input word is used to predict the surrounding words.

of learning long-term dependencies. Like any RNN, LSTM network uses a chain structure of repeating modules (Figure 2.4).

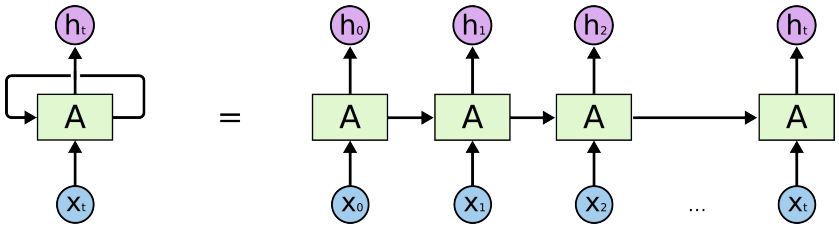

Figure 2.3: An illustration of a LSTM network. In this figure A represents a single LSTM cell, X_t input to this cell and h_t the output of the cell. Source: colah's blog[1].

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

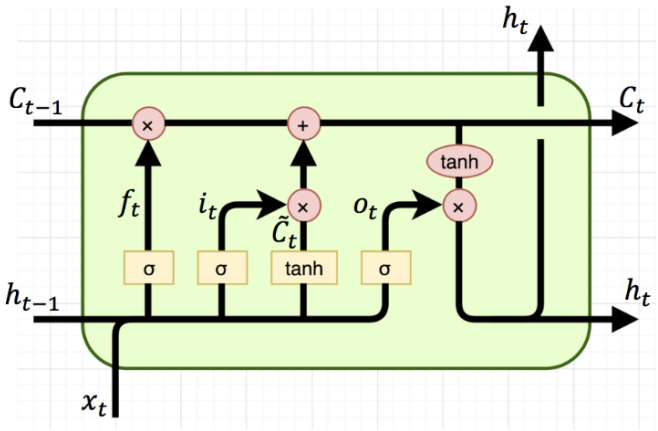

Figure 2.4: An illustration of a single LSTM cell. In this figure, C_{t-1} and C_t represent the previous and the new cell states, similarly h_{t-1} and h_t represent the output of the previous cell and the output of the current cell. Additionally, X_t is the input, f_t is the forget gate, i_t in combination with $\tilde{C}_t$ represent the input gate and o_t represent the output gate. Source: colah's blog[2].

The main intuition behind this architecture is for the LSTM cell to keep track of dependencies between the elements in the input sequence. This is done via the cell state (C), a memory of the LSTM cell that goes through a forgetting phase and a an updating phase each time new input is presented to the cell.

Inside a single LSTM cell (Figure 2.4) we have an input (X_t), the cell state (C_{t-1}), passed on from the previous cell in the sequence, a hidden state (h_{t-1}), also passed from the previous cell and the weights ($W_{f,i,c,o}$). To update the cell state and form an output of the cell, these input variables pass through three gates in the following order: a forget gate, an input gate and an output gate. These gates control the flow of information and the extent to which the cell state will be modified by the new information. Firstly, the forget gate combines the previous hidden state and the new input, which are

[2] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

then passed through a sigmoid function (σ). The output of this function is between 0 and 1 for each number in the cell state, where 0 means forgetting and 1 means keeping it. This result is then then multiplied with the cell state in a point-wise fashion to simulate the forgetting of previous knowledge given a new input:

$$C_f = C_{t-1} \odot \sigma(W_f \odot |h_{t-1}, X_t|) \tag{2.2}$$

where C_f is the cell state after passing through the forget gate.
Secondly, the input gate controls how much new information is added to the cell state. Similarly to the previous gate, the previous hidden state and the new input are combined and passed into the sigmoid function. They are be used in a point-wise multiplication with the combined hidden state and the new input. Both of those were passed through the *tanh* function, to regulate the size of the input. The result of this equation is then added to the cell state using a point-wise addition, to produce the new cell state (C_t):

$$C_t = C_f + (\sigma(W_i \odot |h_{t-1}, X_t|) \odot tanh(W_c \odot |h_{t-1}, X_t|)) \tag{2.3}$$

Lastly, we need to produce an output. This output relies on the current cell state (regulated using the *tanh* function) and the result of the sigmoid function over the combination of the previous hidden state and the new input. The two parts are then combined using a point-wise multiplication into the output of the cell (h_t). This will also be our new hidden state, that will be output to the next cell:

$$h_t = \sigma(W_o \odot |h_{t-1}, X_t|) \odot tanh(C_t) \tag{2.4}$$

The connections from and into the gates are weighted (W) and can be recurrent (multiple possible variant of the LSTM cell that we do not be cover in this work). Said weights need to be trained during the training phase.

An extension to the LSTM architecture that we will be making in this thesis is the addition of a second layer of LSTM cells (Figure 2.5). The

difference being in the input, this second layer of LSTM cells will process the same text sequence as the first layer, just inverted. The results of the two layers are combined in the end. This architecture of the LSTM network is called the bidirectional LSTM network, or BiLSTM for short, and is often used in NLP. With this addition, the network is able to utilize both the past and the future context of a specific word, potentially resulting in better performance of the algorithm. For the purposes of this work we will be using a similar network to the one in Basaldella and Antolli et al. [7].

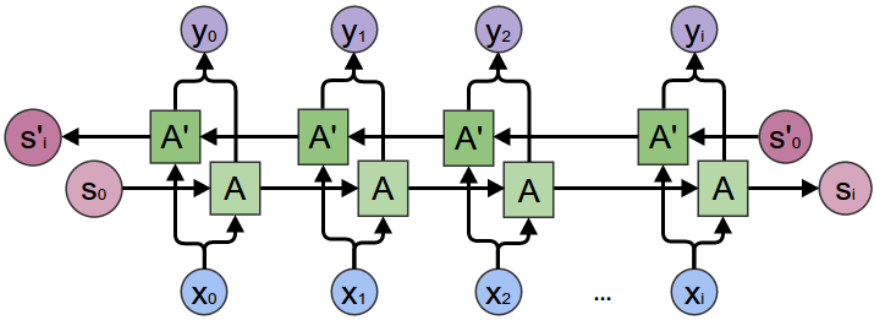

Figure 2.5: An illustration of a biLSTM network. Here, S_i and $S_i^{'}$ represent the direction in which cell states and hidden states are passed on, X_i is the input, Y_i is the output and A and $A^{'}$ are the LSTM cells. Source: colah's blog [3].

2.5 Spacy

Spacy is an industrial-strength natural language processing free open-source library in Python. The library contains multiple tools for text processing, tokenization, serializing etc. of natural language text. Among all these tools it also provides pre-trained NER models, as well as the possibility of training your own model or updating an existing one, although with some restrictions.

As far as we can say, at the time of writing this thesis, no official paper has been released about the Spacy library [8] or parts of it. Due to this we

[3] `http://colah.github.io/posts/2015-09-NN-Types-FP/`

will be relying on a video presentation[4] of the Spacy NER algorithm, that was made by Matthew Honnibal, one of the authors of the Spacy library.

The NER algorithm is a form of a transition based NER 2.6. In other words, it is a method for recognizing entities by modelling the transitional probabilities between states, or in our case, words. On the surface, the algorithm behind this method seems simple. Firstly, it starts with an empty stack and an input sequence of words. Continuing from there, the actions that change the state are defined. Finally, a prediction for the most probable transition (action), for each word in the input sequence, is made. The result of the algorithm is the sequence of actions taken.

Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Watney, visited, Mars]	
SHIFT	[]	[Mark]	[Watney, visited, Mars]	
SHIFT	[]	[Mark, Watney]	[visited, Mars]	
REDUCE(PER)	[(Mark Watney)-PER]	[]	[visited, Mars]	(Mark Watney)-PER
OUT	[(Mark Watney)-PER, visited]	[]	[Mars]	
SHIFT	[(Mark Watney)-PER, visited]	[Mars]	[]	
REDUCE(LOC)	[(Mark Watney)-PER, visited, (Mars)-LOC]	[]	[]	(Mars)-LOC

Figure 2.6: An example of a transition-based NER.

Transitional probabilities and in turn the sequence of actions is predicted using a so-called Embed, Encode, Attend, Predict statistical framework. As its name suggests, the framework works in 4 distinct steps, that result in the prediction of the next transition:

Embed: The first part of this model deals with constructing a numerical vector representation of the input words. This is done by extracting four features of each word, namely norm, suffix, prefix and shape. Each feature is then passed through a hashing function that transforms the string data to a set of numbers. The hashes are then concatenated and passed on to a shallow multilayer perceptron (MLP), which produces the final representation for the input word.

Encode: In this part, the word representations are enriched by adding some contextual information to them. It starts with taking a window of size

[4] `https://spacy.io/models`

1 around the input word (meaning, the words left and right of the input word). The words in this window are then concatenated and passed to a convolutional neural network (CNN) that compacts the concatenated representations back to the size of a single representation. To avoid changing the initial representation too much, the result of this step has a residual connection, which means that the output of the CNN is added to the input representation. This series of computations is then repeated a number of times, taking the resulting representation of each iteration as the new center of the window. Doing all this, the context of the neighbouring words (the number equals the number of iterations) is infused into the input word.

Attend: The final step before the prediction is used for the summary of inputs with respect to query. Here, a few word representations and entity representation, from entities that were previously recognized by this algorithm, are concatenated and passed to a MLP. The result of this step is a feature vector of the length of the input word representation.

Predict: In this last part, the features, extracted during the attend part, are fed into a MLP, which then produces a set probabilities for every defined action. Out of those, the most probable is chosen and added to the result sequence.

The Spacy library tools have been tested against the state-of-the-art benchmarks and have shown to achieve high accuracy[5].

2.6 Stanford NER

Stanford NER is a Java implementation of a Named Entity Recognizer that uses supervised conditional random field (CRF) sequence models for recognizing possible entities in the natural text [9]. CRF models are a class of

[5] `https://spacy.io/usage/facts-figures`

discriminative models suited for prediction tasks, where contextual information is important. In essence CRFs are a special kind of Markov Random fields which satisfy certain graph conditions.

CRFs were pioneered by Lafferty et al. [10] in 2001, as a framework for building probabilistic models to segment and label sequence data. The underlying principle behind them is their application of logistic regression to sequential inputs. Like any classifier, CRFs make heavy use of a set of feature functions (Equation (2.5)), to calculate probability scores. The feature functions use the current word (s), the current word label (l_i), label of the previous word (l_{i-1}) and the position of a word in a sentence (i), to output a real-valued number (most often either 0 or 1). Assigning each feature function a weight (λ_j) and calculating the sum over each feature function and each position in the sentence, the score of a label for the given sentence can be computed:

$$score(l|s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_{i-1}, l_i) \tag{2.5}$$

By using exponents and normalizing the equation, the probability of a label, given a word, can be derived. The form of this equation closely resembles logistic regression:

$$p(l|s) = \frac{\exp[score(l|s)]}{\sum_{l'} \exp[score(l'|s)]} \tag{2.6}$$

The last part of the algorithm are the weights, which need to be adjusted for the problem at hand. This is done iteratively until a certain stopping condition is met:

$$\lambda_i = \lambda_i + \alpha[\sum_{j=1}^{m} f_i(s, j, l_{j-1}, l_j) - \sum_{l'} p(l'|s) \sum_{j=1}^{m} f_i(s, j, l'_{j-1}, l'_j)] \tag{2.7}$$

, where λ_i is the weight of the $i-th$ word in the sentence, α is the adjustment rate, f_i is the feature function and $p(l'|s)$ is the probability of a

label given a word. Same as before, s represents a word and l_i represents a label.

The Stanford NER pre-trained models are largely considered to be a standard for NER. This fact and the ease of implementation being the reasons enough that we decided to include the algorithm in our work.

2.7 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers or more commonly referred to as BERT [11] is a natural language processing technique for pre-training, developed by Google. The technique is designed to pre-train deep bidirectional representations from unlabeled plain text data, by jointly conditioning on both left and right context in all layers. This results in the models ability to be fine tuned with just one additional output layer. Such a task leads to the creation of state-of-the-art models for natural language processing tasks, such as named entity recognition and keyword extraction.

BERT uses a masked language model (MLM) objective, that randomly masks some of the input tokens, making the prediction of these masked tokens the training goal. This MLM goal enables the representation to fuse the left and the right context, which allows the technique to pre-train a deep bidirectional Transformer. In addition to the MLM, a next sentence prediction training is also used, jointly pre-training text-pair representations.

The usage of BERT technique is divided into two parts (Figure 2.7), the pre-training and the fine-tuning. Pre-training of models is done using unlabeled data, over multiple pre-training tasks and is extremely computation expensive. On the other hand, the fine-tuning of these models is relatively inexpensive. Fine-tuning is done using labeled data of a specific task and although multiple tasks might use the same pre-trained parameter initialization, a separate fine-tuned model is produced for each one of them. BERT's architecture is that of a multi-layered bidirectional Transformer encoder, in-

stead of a LSTM one, based on the design described by Vaswani et al. [12] in 2017. The input and output representations that BERT uses are produced by an embedding algorithm WordPiece [13] with the addition of certain special tokens ([CLS] and [SEP]) that signify the start of every sentence and the separator of two joint sentences in a single sequence, respectively (Figure 2.8).

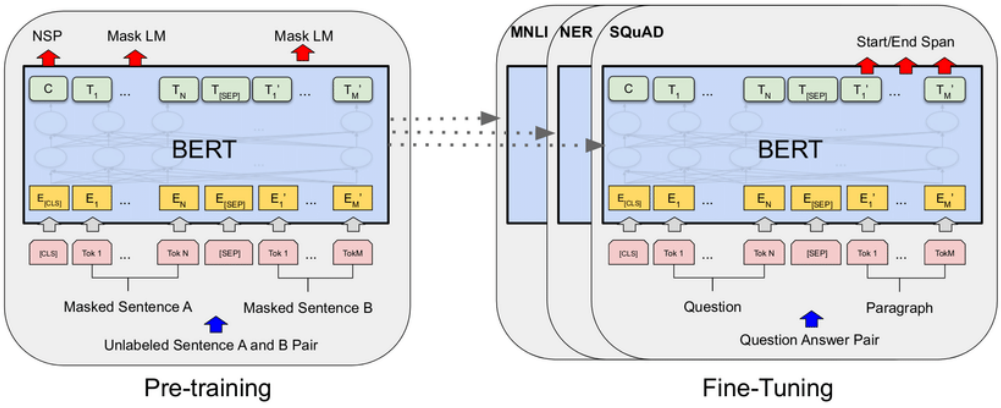

Figure 2.7: An illustration of the two parts of training a BERT model. On the left we can observe how the pre-training of the BERT model is carried out, using MLM (mask or masked language model) and next sentence prediction (NSP). On the right, it is shown how the pre-trained BERT model can be fine-tuned to handle multiple tasks, such as named entity recognition (NER), question answering (SQuAD stands for Stanford question answering dataset) or as a natural language interface (MNLI stands for multi natural language interface).

In the following we provide more details about the two parts of the BERT process. Firstly, as was mentioned in previous paragraph, the pre-training process includes two tasks, the identification of masked tokens in a sentence and the prediction of the next sentence. During the first of these two tasks, prediction of masked tokens, 15% of the input tokens are randomly chosen to be replaced as a part of the MLM objective. Out of these tokens, 80%

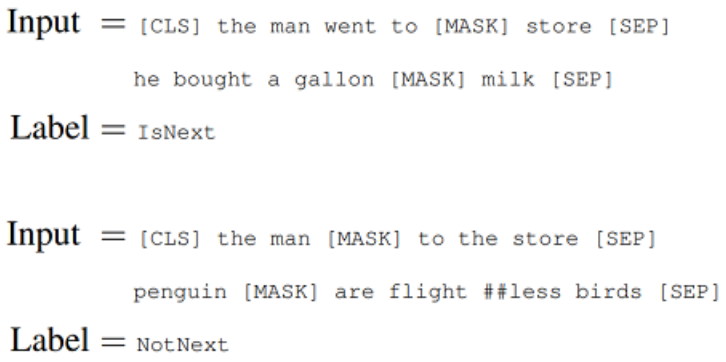

Figure 2.8: An example of the MLM objective. We can observe how tokens (words) are replaced with [MASK] and how the model learns by predicting the next sentence.

are replaced with a [MASK] token, 10% are replaced with random tokens and the remaining 10% stay unchanged. This is implemented to combat the miss-match between pre-training and fine-tuning, since fine-tuning does not make use of the [MASK] token. The second of these two tasks, next sentence prediction, tries to capture the relationship between two sentences, which is inherently important for certain problems, i.e. Question Answering (QA) and Natural Language Inference (NLI). This part makes use of the joint sentences in a single sequence, trying to predict on based on the other as an input. The pre-training procedure using these two tasks follows the existing literature on language model pre-training.

While the pre-training is used to create deep bidirectional representations from unlabeled text, it is only the start. From the general representations, it is possible to configure the model for a specific task, such question answering or in our case named entity recognition. The process differs for any given problem, but in its essence, it involves providing the input-output pairs for the task at hand. Given those, the model can adjust its representations, to deliver the desired output. In comparison with the pre-training, the inputs and outputs at this step are very task specific and do not include any masking whatsoever. For any task, it is possible to simply plug in the task

specific inputs and output and fine-tune all the parameters end-to-end. Due to this, fine-tuning is usually carried out with a dataset much smaller than the one used for pre-training, over a shorter period of time, making this step computationally quite inexpensive.

Chapter 3

Experiments

In this chapter we will describe the datasets used in this thesis and the implementation of algorithms and methods that were presented in the previous chapter. All of implementations and data processing were written in Python[1] version 3.7.1, with the help of the PyCharm Python IDE[2]. Any specific Python library that we have used for the implementation of the presented algorithms we mention when describing the implementation that used it.

3.1 Data preparation

In our thesis we will mostly deal with a dataset of submissions from the platform Reddit[3], more specifically from a Subreddit called Movie Suggestions[4]. The dataset is a product of crowd-sourced work and is comprised of submissions, comments, and a pool of movie titles and their respective IDs. The submissions contain their respective original texts and titles, alongside the ID, under which they were posted on Reddit, keywords, movie titles, actors and genres. The latter four were extracted from the original texts and split

[1] `https://www.python.org/`
[2] `https://www.jetbrains.com/pycharm/`
[3] `https://www.reddit.com/`
[4] `https://www.reddit.com/r/MovieSuggestions/`

according to their sentiment (positive and negative). In total we are dealing with 1480 instances of submissions and a movie pool of 11578 movie titles in their original languages. Additionally, we will use the movie pool for the NER of movie titles.

Analysis of the submissions data shows, that the texts are relatively short, with the median of 45 words and first (Q1), and third quartile (Q3) at 28 and 77 words respectively (Figure 3.1). The texts contain between 0 and 19 keywords, with the median of 2 keywords, Q1 at 1 keyword and Q3 at 4 keywords (Figure 3.2). The keywords in this dataset are either singular words or key phrases divided into singular words.

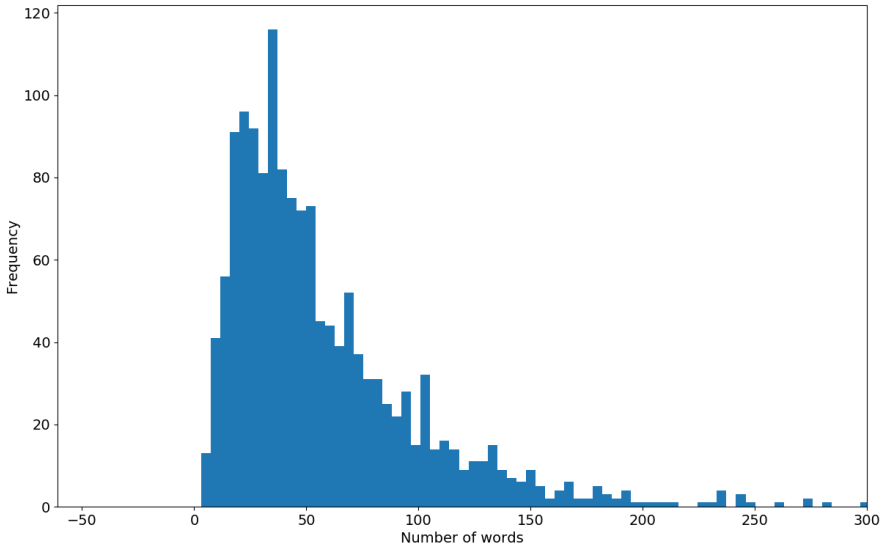

Figure 3.1: A plot of submissions word counts per instance.

On a different note, named entities in the dataset are divided into two parts, firstly into person names (actors, directors, movie characters etc.) that were taken straight out of the original submission text. The number of actors in the text is somewhat low, with there only being between 0 and 3 actors per instance, with the median, Q1 and Q3 all being at 0. Secondly, the movie titles found in the text are denoted by a tt-ID (based on IMDb movie IDs). It is important to mention that the original texts more often than not, do

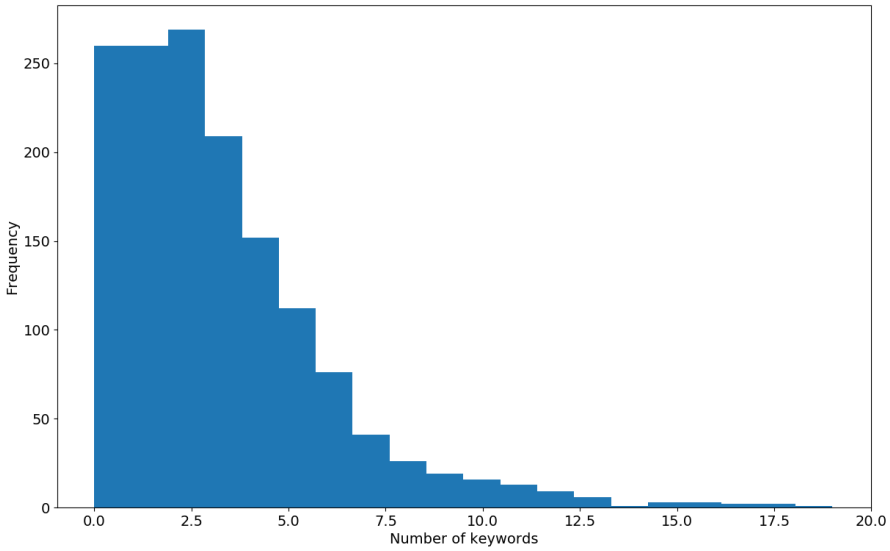

Figure 3.2: A plot of submissions keyword counts per instance.

not contain the full titles of the movies denoted by the tt-IDs (shorter titles, misspelling etc.). On that note, each text contained in the dataset has at least 1 movie ID linked to it (and at most 48 IDs), with the median of 3, Q1 at 1 IDs and Q3 at 5 IDs (Figure 3.3). The number of unique movie title IDs found was 5521, which is less than half of the size of the movie pool. For further references, this (the Reddit submissions dataset) will be the dataset used in the experiments under the name Reddit data and all future evaluations of the results will be based on it.

In addition to the described datasets, we will also be using some external datasets as a part of our experimentation. The main purpose for this being so-called transfer learning, a method of training a model on a dataset pertaining to a similar problem to the one we are trying to solve (usually because such dataset is bigger) and only fine-tuning the model on the problem related dataset. In our thesis, we will be using the following additional dataset:

INSPEC dataset: A well known dataset constructed by Hulth et al. [14] in 2003, for use in their research in keyword extraction. The dataset consists of 2000 abstracts in English from papers published between 1998 and 2002.

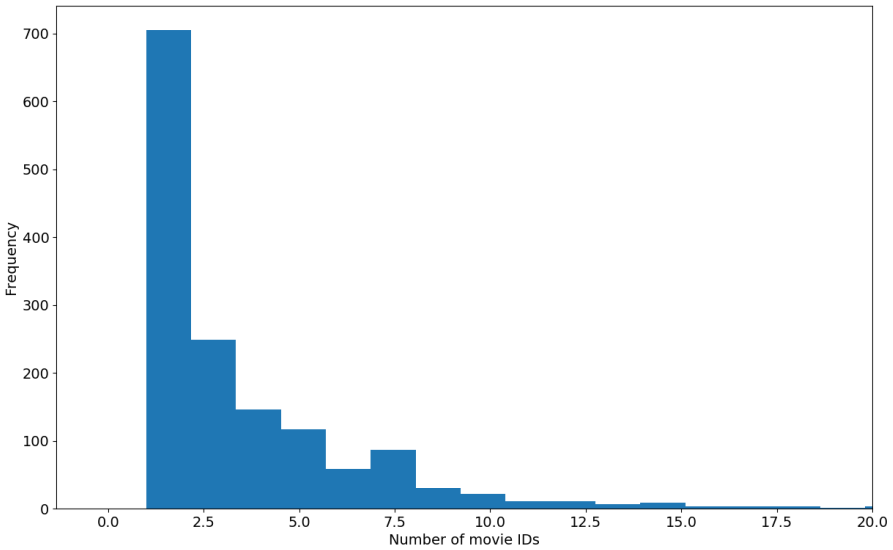

Figure 3.3: A plot of submissions movie ID counts per instance.

The dataset has two sets of keywords assigned to it, each annotated using a different approach. The keywords in both sets might or might not be present in the abstract itself. For the purpose of this thesis and due to the nature of the problems we are tackling, we will be using the controlled keywords with the added restriction of only using keywords that appear in the text.

Krapivin dataset: The dataset proposed by Krapivin et al. [15] in 2009 is described as a high quality dataset consisting of 2000 scientific papers from the field of computer science, that were published by ACM[5]. For each paper, the keywords were assigned manually by authors and verified by reviewers. This dataset was chosen due to its size, contents and most importantly the quality of its keywords.

CoNLL-2003: This dataset, first proposed in 2003 by Sang et al. [16], is a part of the CoNLL-2003 shared task, that concerns itself with language-independent NER. The data itself is provided in English and German

[5] `https://www.acm.org/`

and uses four entities to annotate its texts, namely persons, locations, organizations and names of miscellaneous entities. Over the years this dataset has become a sort of a baseline for NER research and algorithm development, which is also the reason why we decided to use it in this thesis. The tokens in the dataset are annotated with one of four entity types, namely PER (people, names, etc.), LOC (locations), ORG (names of organizations), MISC (other).

MIT movie corpus: A semantically tagged training and test corpus in BIO (Beginning-Inside-Outside) format, constructed by the Spoken language systems group at Stanford [17]. It is comprised of documents containing data obtained from simple queries, and ones containing more complex queries. The queries forming this dataset are centered around movie titles, actors, directors etc., making it a good choice for the problem of recognizing movie titles in the text. The data contains the following entity types: TRAILER (trailer names), YEAR, SONG (song titles), TITLE (movie title), ACTOR, RATING, PLOT (parts of the text pertaining to the plot of the movie), RATINGS_AVERAGE, GENRE, DIRECTOR, CHARACTER (names of movie characters) and REVIEW (parts of the text that could be considered a review).

Ontonotes 5: This dataset is the fifth release of a large multilingual richly-annotated corpus with hundreds of thousands of texts in multiple language, published by the Linguistic Data Consortium[6]. The dataset is widely used for training NER models, due to its size and annotation quality. The dataset divides entities into 18 different categories, namely PERSON (people, including fictional names), NORP (nationalities or religious or political groups), FAC (buildings), ORG (organizations), GPE (geopolitical entities), LOC (locations), PRODUCT (objects, products), EVENT, WORK_OF_ART, LAW (named documents made into law), LANGUAGE, DATE, TIME, PERCENT, MONEY (monetary

[6] `https://www.ldc.upenn.edu/`

values, including units), QUANTITY, ORDINAL (ordinal numbers), CARDINAL (cardinal numbers).

To prepare the data for use in the various algorithms that were described in the previous chapter, we implemented a pre-processing pipeline. We started off by removing any markdown that was present in the text, as well as removing accents and letters that might not be found in the English language (i.e. Greek letters). In the end, the processed texts were input to the NLTK library word tokenizer to produce tokens, which can then be used in the algorithms. Afterwards, depending on the context in which the datasets were used, the tokens were passed to the embedding function, transformed into the BIO format or left unchanged.

3.2 Keyword extraction

3.2.1 TextRank and RAKE

The implementation of RAKE and TextRank was fairly standard, using specific functions from a Python library for each. For RAKE we made use of the `rake_nltk` library[7] and for the TextRank algorithm we used the implementation found in the `gensim` library[8]. Due to the nature of the algorithms, we were only able to experiment with changing a few parameters when taking into account the whole input corpus and computing the possible keywords: the input (the pre-processed natural language text) and the parameter limiting the number output keywords. Aside from using the pre-processed text, described in the previous section, we made an additional experiment by running the two algorithms with input texts in which the stopwords (defined by the `nltk` library[9]) were removed. Coming back to the parameters limiting the number of output keywords, we experimented with the minimum and the maximum number of keywords that could possibly be found in text for

[7] `https://pypi.org/project/rake-nltk/`

[8] `https://radimrehurek.com/gensim/summarization/keywords.html`

[9] `http://www.nltk.org/`

the RAKE algorithm, and with the maximum number of keywords that were chosen (based on their ratings) for the TextRank algorithm.

3.2.2 LSTM and BiLSTM

All of the LSTM and biLSTM networks that we experimented on were implemented using the Keras Python library[10]. For this set of algorithms, in addition to testing the difference between the unidirectional and bidirectional LSTM networks, we tested how transfer learning impacted the end results, as well as the difference a certain architecture of the network could have on the results and overall performance of the network.

We decided to test seven different network architectures, trying to encompass as many different shapes of the network as possible: narrow and shallow (100), wide and shallow (300), narrow (100-100), wide (300-300), outward cone (200-300), inward cone (300-200), and a deeper three-layered architecture (400-300-200). The numbers here refer to the number of nodes in a single layer of the neural network. We have mostly stayed within a range of 100-400 nodes in a single layer, mostly due to the discoveries made by the preceding research on this topic [7] and additionally by our initial experiments. We also stayed within a range of 1-3 hidden LSTM or biLSTM layers, due to the decaying results after adding more than 3 layers. Following the LSTM or biLSTM layers, we used a time-distributed dense layer with 150 nodes and a softmax layer, to produce the results of this network. Aside from the number of layers and their sizes, the networks try to mimic the original design proposed by Basaldella and Antolli et al. [7], with the use of *tanh* activation function in the LSTM, or biLSTM, layers, RELU in the dense layer, hard Sigmoid for the recurrent connections and a softmax function for computing the outputs of the network. Throughout the network we used a dropout layer between all of the LSTM, or biLSTM, layers, with the dropout set to 0.25. We found this to improve the results the most during our initial research. For the purposes of this thesis we tested transfer learning

[10] `https://keras.io/`

using three types of networks, one of which was the control network which was trained normally on the Reddit data, without any additional transfer learning, while the other two used the INSPEC [14] and Krapivin datasets [15] as their bases for transfer learning.

3.3 Named entity recognition

Taking a slightly different route in comparison within the KE experiments, the experiments for this problem were divided into two tasks for each algorithm. Namely, the task dealing with recognizing persons in the text (actors, directors, etc.) and the task dealing with the recognition of movie titles. In the most cases, the difference between these two tasks is only in the evaluation of the results, but, as can be seen later, in some cases, one of these two tasks might not be possible to complete, usually due to the design of the model and the entities it produces. This split is only possible due the different entities that describe actors and movies.

3.3.1 Spacy

As the name and the description in the previous chapter would suggest, we have implemented the this approach to NER with the use of the Spacy library[11]. The library offers a wide range of natural language processing and prediction tools, among which we can also find some pre-trained NER models. Out of 8 available models for texts written in English, we will be using the `en_core_web_lg` model[12], since the model is compatible with our work. As described on the official Spacy library web page, this is an English multi-task CNN model trained on OntoNotes, with GloVe vectors trained on Common Crawl. It can be used for assigning word vectors, context-specific token vectors, POS tags, dependency parsing and NER. Since the model was trained

[11] `https://spacy.io/`

[12] `https://github.com/explosion/spacy-models/releases//tag/en_core_web_lg-2.2.5`

on the Ontonotes 5 dataset, it is able to recognize 18 different entity types (previously described in this chapter), out of which we are only interested in PERSON type for actor recognition and WORK_OF_ART for movie titles (possibly also LOC, location, and ORG, organization). In addition to using the model, Spacy library also offers the possibility to update preexisting models, or to train Spacy models from scratch using your own training data.

We will be using the four different types of models, namely a control model in the form of the Spacy pre-trained model, without any modifications and three versions of this model, updated with the English titles data (from the movie pool), MIT movie corpus and CoNLL-2003 data, respectively. Because we do not know how much the new data will impact the model, we will train each of the aforementioned models three times, using dropout values of 0.35, 0.5 and 0.7, presenting the best combinations in the end.

3.3.2 Stanford NER

Originally the Stanford NER algorithm was implemented in Java, but due to its usefulness, there have been many successful attempts of wrapper implementations that will allow us to use it in Python. The library we will be using for this is NLTK[13].

Included with the Stanford NER are three pre-trained models, that we will be using in our experiments:

3-class: A model with only three entity types, namely Person, Location and Organization, was trained on CoNLL-2003 [16], MUC-6[14] and MUC-7[15] datasets, in addition to some other not specified data.

4-class: A pre-trained model with four entity types: Person, Location, Organization and Misc (miscellaneous). The model has been trained on the English texts from the CoNLL-2003 [16] dataset.

[13] https://www.nltk.org/_modules/nltk/tag/stanford.html
[14] https://cs.nyu.edu/cs/faculty/grishman/muc6.html
[15] https://catalog.ldc.upenn.edu/LDC2001T02

7-class: A 7-class model, that was trained on the MUC-6 and MUC-7[16] training data. The two datasets were presented at the sixth and seventh Message understaning conferences. As the name suggests, this model includes seven entity types: Location, Person, Organization, Money, Percent, Date and Time.

To expand the number of models for this approach to NER, we will also train three new models ourselves, using the same training methods as the pre-trained models. The three datasets that will be used for this are the Reddit English titles from the movie pool dataset, MIT movie corpus, and Ontonotes 5.

3.3.3 BERT

As explained in the previous chapter, BERT models for NER are constructed in two steps, the pre-training step and the fine-tuning step. Because the pre-training step for such models exceeds our computational and time capabilities, we will be using pre-trained models provided by Google[17]. Given that, we will focus solely on the fine-tuning aspect of this process. For the implementation of BERT models in our thesis, we took advantage of DeepPavlov, an open source conversational AI framework, for its fine-tuned BERT models. The framework also provides the added functionality of fine-tuning the pre-trained models using your own data.

Going into more details about the pre-trained models, we will be using two models– both of them fine-tuned on the same pre-trained model, more specifically a twelve-layered cased pre-trained model, released by Google. Cased in this context means that the model has been trained on data that has not been lowercased before embedding, retaining some additional information. Normally, the uncased models perform better than their cased counterparts, but since we are dealing with a case-sensitive problem, the cased variant

[16] `https://catalog.ldc.upenn.edu/LDC2001T02`

[17] `https://github.com/google-research/bert`

of the model should work better. The fine-tuning of these two models is accomplished using CoNll-2003 and Ontonotes 5 datasets, respectively.

Additionally, we will also create two new models, one based on the cased pre-trained model and one on the uncased variant. Both will be then fine-tuned with the MIT movie corpus dataset. The premise of this experiment is to test how the cased and uncased pre-trained models affect the end results.

Chapter 4

Results

For evaluation of the experiments, we calculated precision, recall and F1 score of the extracted keywords and the recognized named entities. To keep the results comparable to each other, the evaluation was always performed using 20% of the original Reddit data as the testing data. The remaining 80% of the data was further split into the training set (70%) and the validation set (10%).

4.1 Keyword extraction

To establish a baseline, we pre-computed the baseline accuracy of 0.9522 for the KE part of this thesis. This is the accuracy we would have achieved if we chose to label every word in the input text as not a keyword.

4.1.1 RAKE

The amount of experiments we were able to perform using the RAKE algorithm was quite limited, due to the nature of the algorithm. Although this is the case, the algorithm, alongside TextRank was meant to serve the purpose of a baseline for other KE algorithms. This being said, we can observe quite an improvement in the algorithms performance (Figure 4.1) if we remove any stopwords from the text during the pre-processing. The improvement in

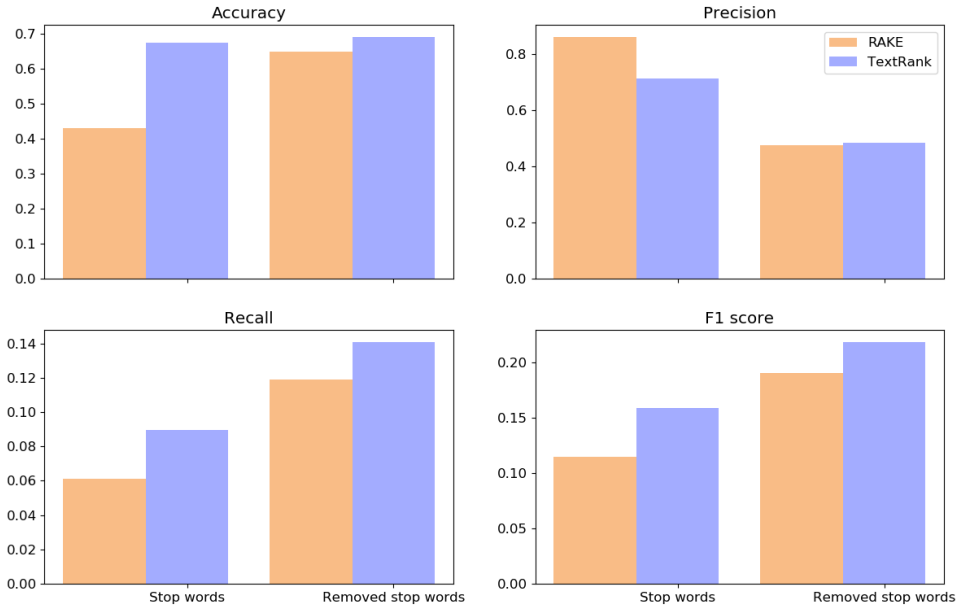

Figure 4.1: Evaluation results for the RAKE and TextRank keyword extraction algorithms. In this figure we are comparing the results of RAKE and TextRank based solely on the input text, which either did or did not include stopwords. The exact results for this experiment can be found in Appendix A (Tables A.1 and A.2).

this case comes as a minor surprise as the RAKE algorithm, based on the description of the algorithm [3], relies heavily on stopwords found in the text. Such a result could be attributed to the fact that we used a different, more general, stopword list, for the removal of stopwords, when compared to the default one used by the RAKE algorithm.

4.1.2 TextRank

Similarly to the RAKE algorithm evaluation, we can observe (Figure 4.1) an improvement in the results after the removal of stopwords. In case of this algorithm, the observed improvement is potentially easier to explain. The

removal of words that do not carry any meaningful information affects the structures and nodes of graphs, created by this algorithm. This in turn affects how the node weights converge and ultimately, which words are considered to be keywords. Since stopwords are at most only parts of keywords, the change in the input translates into less cluttered graphs and hence into better results.

4.1.3 LSTM and biLSTM

The evaluations performed for this subsection (Figures 4.2 - 4.6) are threefold: firstly, the comparative evaluation of the seven distinct neural network architectures for each of the input datasets, secondly, the comparative evaluation of different input datasets (including transfer learning) and lastly, the comparison between the LSTM and biLSTM networks.

Starting off with the models trained solely on the Reddit dataset. We can observe from the results (Figure 4.2) that out of all LSTM architecures (300-300) architecture achieved the highest precision and (200-300) the highest recall and F1 score. On the biLSTM side, (300) architecture achieved the highest precision and the highest recall was shared between (100-100) and (200-300) architectures. The highest F1 score was achieved by the (200-300) architecture. From these results we can clearly see, that for this specific training dataset, the (200-300) architecture is the best choice for both LSTM and biLSTM networks.

Continuing with the comparison of the different architectures for the models trained solely on the Krapivin dataset (Figure 4.3). Looking at the results of the LSTM networks, the highest precision goes to the (100) architecture and both the highest recall and the highest F1 score go to the (400-300-200) architecture. With the biLSTM networks, the highest precision goes to the (300) architecture and both the highest recall and the highest F1 score go to the (300-300) architecture. The two architectures, (400-300-200) for the LSTM networks and (300-300) for the biLSTM networks, clearly being the best choices in this case.

Thirdly, the models trained solely on the INSPEC datset (Figure A.5).

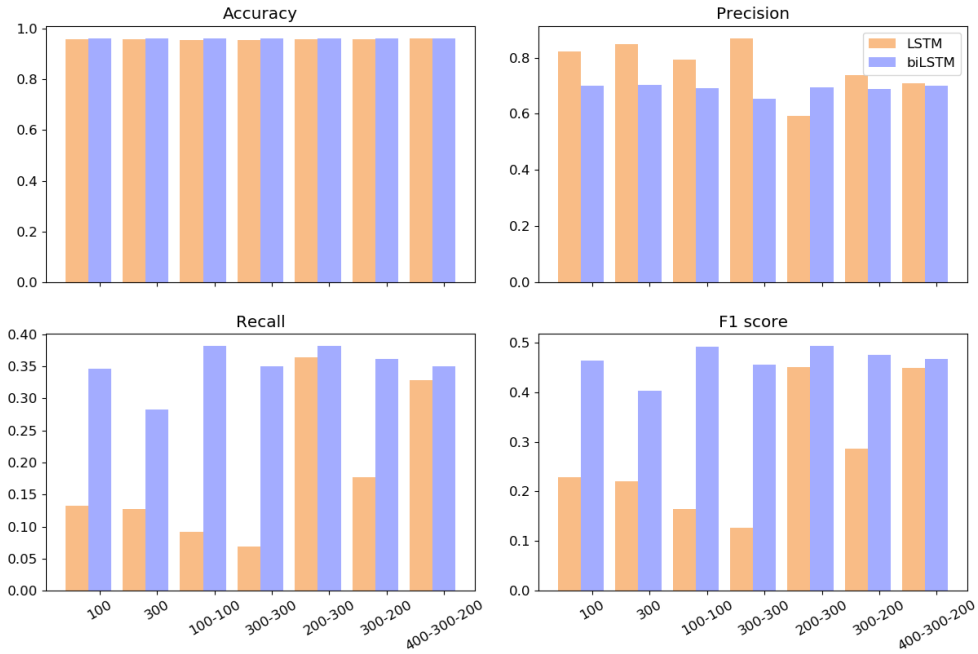

Figure 4.2: Evaluation results for the LSTM and biLSTM keyword extraction algorithms. In this figure we compare models with different architectures, denoted by the number of neurons in each of their hidden layers. All of these models were trained using the Reddit dataset. The exact results for this experiment can be found in Appendix A (Table A.3).

The results show us that, for the LSTM networks, (300-300) architecture achieved the highest precision and (100) architecture achieved the highest precision and F1 score. For biLSTM networks, (100) architecture achieved the highest precision, (300-300) the highest recall and (100-100) the highest F1 score. Given these results we can say that (100) architecture would be the best choice for the LSTM networks and (100-100) for the biLSTM ones.

Moving on to the first of the two sets of models that trained using transfer learning. Starting with the models trained on the Krapivin dataset and fine-tuned on the Reddit dataset (Figure A.6). For the LSTM networks, the highest precision was achieved by the (100-100) architecture and the highest recall, alongside the highest F1 score was achieved by the (400-300-

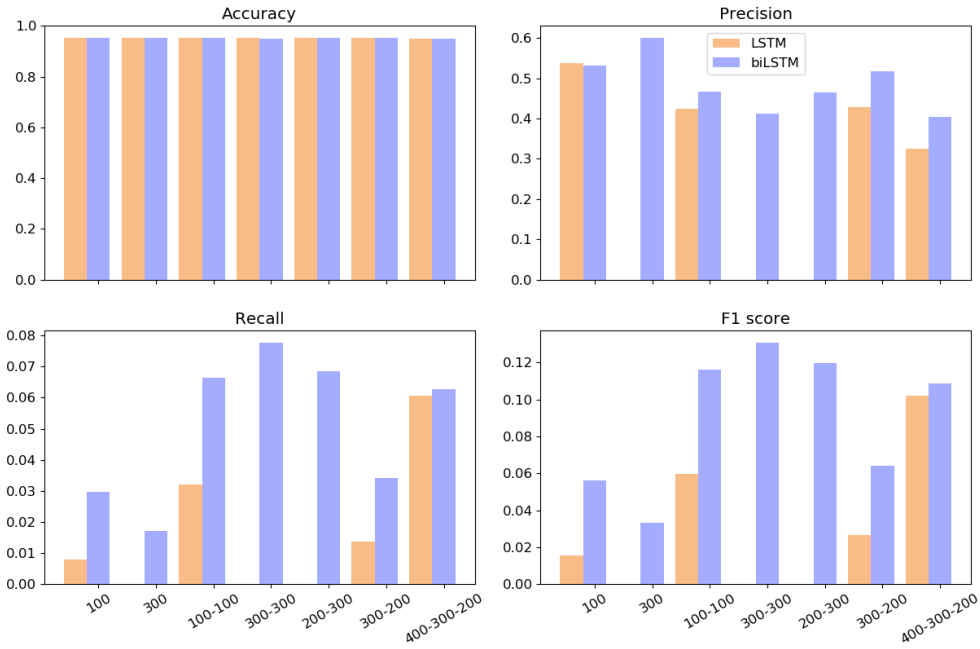

Figure 4.3: Evaluation results for the LSTM and biLSTM keyword extraction algorithms. In this figure we compare models with different architectures, denoted by the number of neurons in each of their hidden layers. All of these models were trained using the Krapivin dataset. The exact results for this experiment can be found in Appendix A (Table A.4).

200) architecture. Somewhat similarly, the highest recall and F1 score is attributed the (400-300-200) architecture and the highest precision to the (300-300) architecture, for the biLSTM networks. Summarizing that, the optimal choice out of these seven architectures would be the three-layered arcitecture (400-300-200) for both LSTM and biLSTM networks.

The second set of models trained using transfer learning was trained on the INSPEC dataset and then fine-tuned on the Reddit dataset (Figure A.7). Out of these models, the LSTM model with (300-300) architecture achieved the highest precision, (400-300-200) achieved the highest recall and (200-300) the highest F1 score. On the biLSTM side, the highest precision is attributed to the (300) architecture and both the highest recall and F1 score

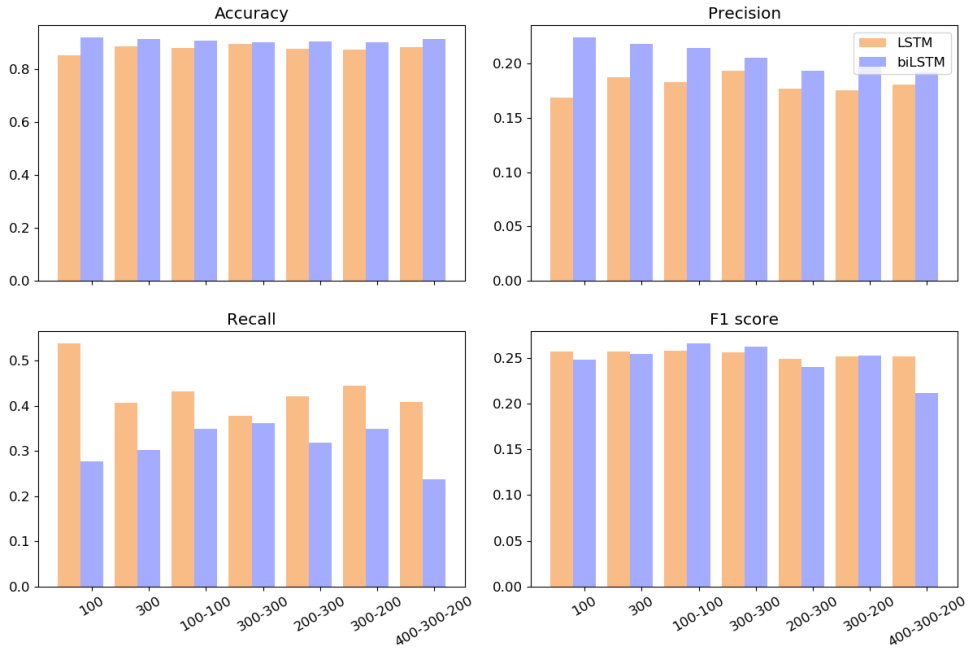

Figure 4.4: Evaluation results for the LSTM and biLSTM keyword extraction algorithms. In this figure we compare models with different architectures, denoted by the number of neurons in each of their hidden layers. All of these models were trained using the INSPEC dataset. The exact results for this experiment can be found in Appendix A (Table A.5).

to the (400-300-200) architecture. For biLSTM the best overall architecture for this training data is clearly the (400-300-200) architecture. On the other hand the best architecture for the LSTM networks, despite multiple models having very similar recalls and F1 score, the (200-300) architecture is still slightly better than the rest, for this training data.

Given these results, we can conclude, that for the most part, the three-layered architecture was one of the best for nearly all training methods. Furthermore, when comparing the results of the best models we observe, that the models that were trained on the data directly related to the data used for evaluation (Figures 4.2, 4.5 and 4.6) performed substantially better than the ones trained on datasets that were only somewhat related to the evaluation

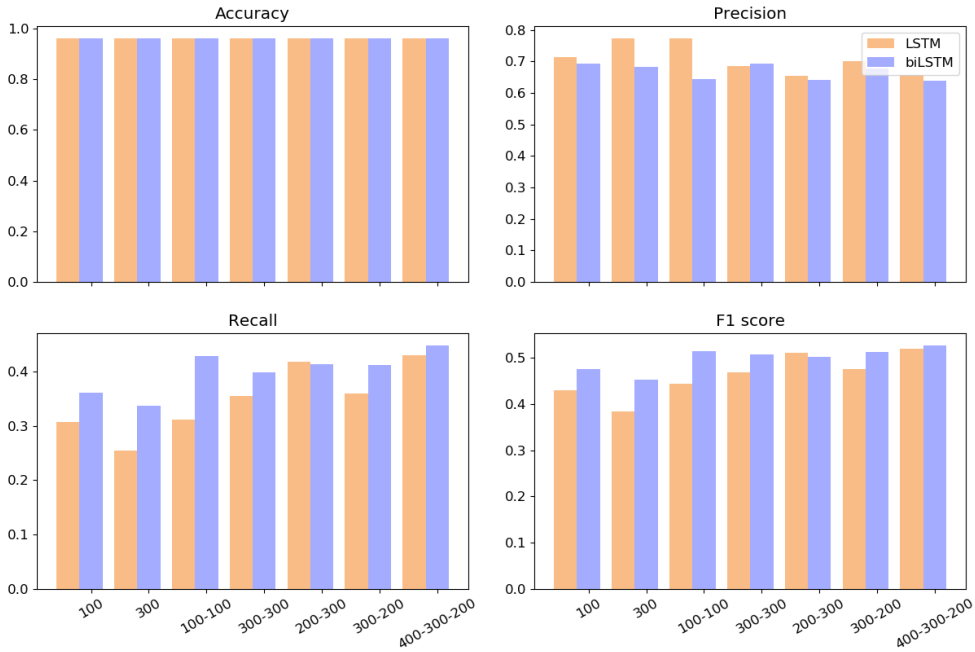

Figure 4.5: Evaluation results for the LSTM and biLSTM keyword extraction algorithms. In this figure we compare models with different architectures, denoted by the number of neurons in each of their hidden layers. All of these models were trained using the Krapivin dataset and fine-tuned using the Reddit dataset. The exact results for this experiment can be found in Appendix A (Table A.6).

data (Figures 4.3 and 4.4). When combining these two approaches, using transfer learning, we are able to obtain even better results than the ones obtained by only training the models on the Reddit dataset. Out of the two transfer learning approaches, the one utilizing the INSPEC dataset proved to be slightly better in regard to the F1 scores. One possible explanation for this occurrence is that the models retained more generality when trained using transfer learning.

Lastly, comparing the LSTM and biLSTM networks overall. As was somewhat expected, the biLSTM networks achieved better precision, recall and F1 scores in almost all comparisons, when comparing best models over all

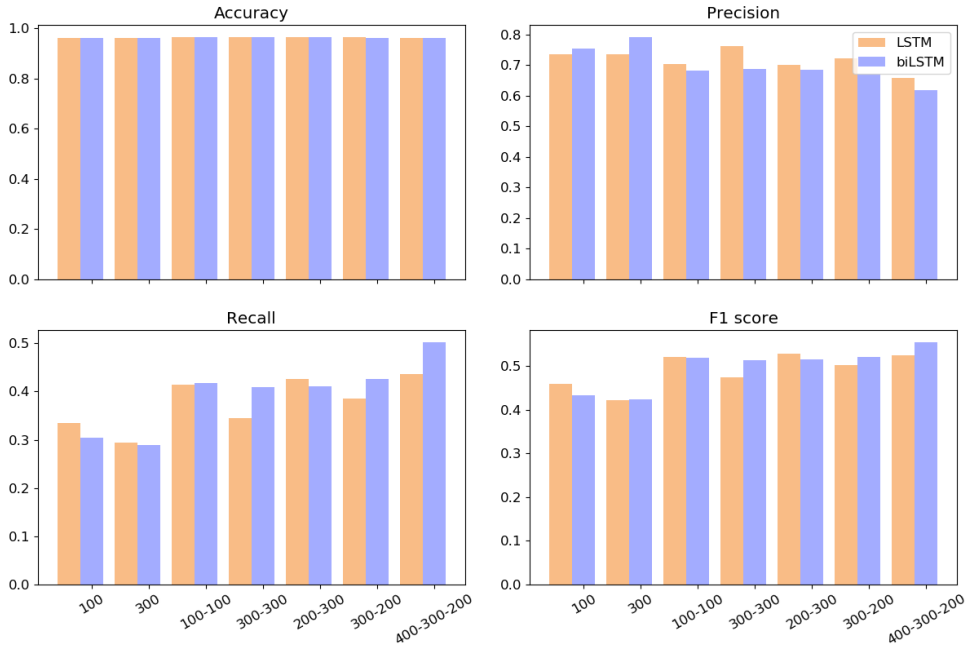

Figure 4.6: Evaluation results for the LSTM and biLSTM keyword extraction algorithms. In this figure we compare models with different architectures, denoted by the number of neurons in each of their hidden layers. All of these models were trained using the INSPEC dataset and fine-tuned using the Reddit dataset. The exact results for this experiment can be found in Appendix A (Table A.7).

different training methods.

4.1.4 Keyword extraction summary

Out of the three types of algorithms that we experimented with, we can conclude that the best ones are the biLSTM networks. When utilizing transfer learning, this algorithm outperforms all others in the majority of experiments. For the best performing model of this algorithm, we used a three-layered network architecture (400-300-200), trained the algorithm on the INSPEC dataset and fine-tuned it on the Reddit dataset. Looking back at the two

simpler algorithms, RAKE and TextRank, it was observed, that their performance is on par with the LSTM and biLSTM algorithms that did not use any specific domain data (Figures 4.3 and 4.4). This makes the two algorithms a computationally cheaper and simpler alternatives for problems for which we do not necessarily have any specific domain data.

4.2 Named entity recognition

The evaluation for NER is divided into evaluation of the recognition of names and the recognition of movie titles. The two evaluations are carried out on different subsets of entity types: entity types associated with persons and ones associated with movie titles.

For the first of the two evaluations (entity types associated with persons), we relied on the fact that all of the true names of persons (actors, characters, movie directors etc.) were directly extracted from the submissions in the Reddit dataset. Due to this, we could make a direct comparison between the predicted names and the true names, by simply computing the similarity between them. For this part of the evaluation we used the Ratcliff/Obershelp pattern recognition [18] to compute the similarity between the two names. The threshold for accepting a name as correctly predicted was set to 0.85. Due to some names appearing multiple times in a single submission, we also implemented a similarity check between the predicted names and removed any names that were deemed too similar to others. The threshold for this is signified by t in the plots (Figures 4.7-4.12) for this evaluation.

Secondly, the evaluation of the recognition of entity types associated with movie titles. As mentioned in the previous chapter, the true movie titles are not extracted word-by-word from the submission, but are in fact *ttids* of movies that the submission is refering to. To be able to compare the predicted results with the true results we convert the predicted results into *ttids*. For the purposes of this evaluation two different methods are employed: finding the most similar movie title from the movie pool dataset (using the Ratcliff/Obershelp pattern recognition, denoted by its threshold value in the plots and tables, e.g. $t = 0.55$) and using IMDbPY[1], a Python package for retrieving and managing the data of the IMDb movie database, to retrieve a *ttid* of a movie that best matches the predicted string, denoted by *API* in the plots (Figures 4.7-4.12). Although the movie search function for IMDbPY

[1] `https://imdbpy.github.io/`

could be considered a black box function, it provides us with the capability to search the whole IMDb movie database and find most relevant movie titles.

Additionally, during our initial experimentation we noticed that quite a few words or sequences of words that we would normally consider to be a part of a movie title, were incorrectly classified under other entity tags. To correct this and draw further conclusions from it, the results of evaluation also include slight variations of various models that account for this. This is meant in a way, where the same model is run twice, once normally and the second time by treating words tagged with the LOCATION and ORGANIZATION entity tags as tags that contain information about movie titles. Similarly, we apply the same logic to the evaluation of named entities relating to persons, where we, at a few points, also take in account CHARACTER entity tag. Such extension of viable tags for a model are signified by *ext.* in the evaluation plots (Figures 4.7-4.12).

4.2.1 Spacy

Updating the original Spacy model with additional data was done with three different dropout values, signified by d in the plots (Figures 4.7 and 4.8). Given the two sets of results: entities associated with persons and entities associated with movie titles, we first focus on the recognition of persons.

Taking the original model for NER, provided by the Spacy library, as the baseline of this algorithm, we can draw some conclusions about the custom updated models from our results (Figure 4.7). Aside from the Reddit English titles dataset, all the other models exhibit similar behaviour in terms of their precision and recall, with a high recall and very low precision. The results do vary between updating on different datasets, with both the CoNLL-2003 and MIT movie corpus datasets achieving better performance than the original model. The same cannot be said for the Reddit English titles dataset, a decrease in performance easily explained by the lack of any person related entity tags in this dataset, which is also the reason for receiving invalid results at lower dropout values. Interestingly enough, the two datasets (CoNLL-2003

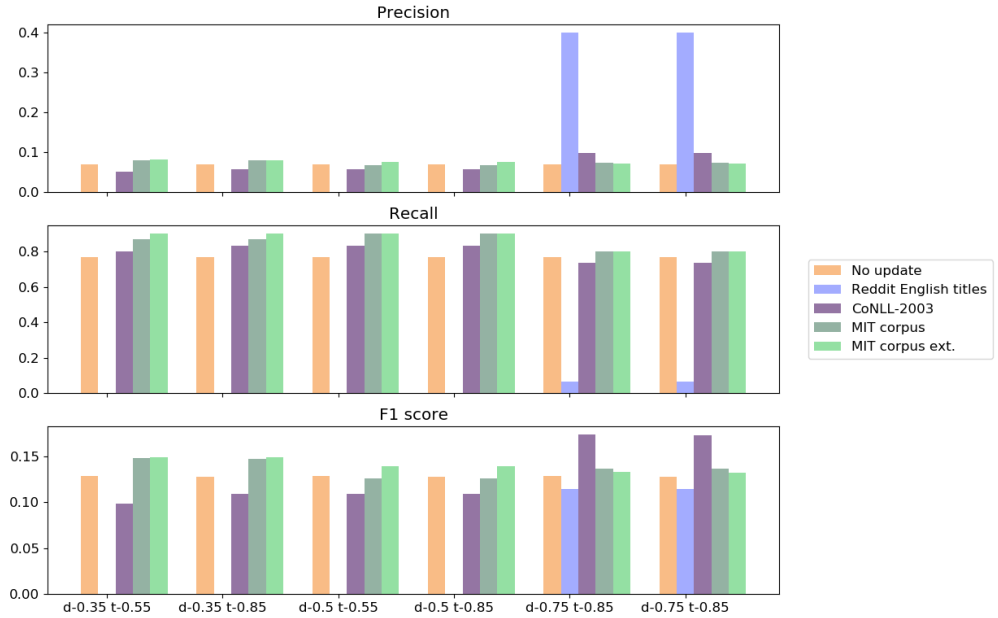

Figure 4.7: Evaluation results of Spacy models for entity types associated with persons. In this figure We compare models updated with different datasets and with different dropout values (d), evaluated using different similarity thresholds (t). The exact results can be found in Appendix B (Table B.1).

and MIT movie corpus) react differently, in terms of precision, to the increase in the dropout value, with the CoNLL-2003 dataset exhibiting a positive correlation and MIT movie corpus exhibiting a slight negative one. This leads us to believe that MIT movie corpus dataset is better suited for problem at hand. Additionally, it is worth mentioning that, by including CHARACTER entity tag in the tags used for recognising persons, the results slightly improve. A possible explanation of this occurrence could be the misclassification of real names under the movie character names during the prediction.

The evaluation of NER pertaining to the entity tags connected to movie titles (Figure 4.8), is very one-sided with the models updated using the MIT movie corpus dataset producing predominately better results when compared

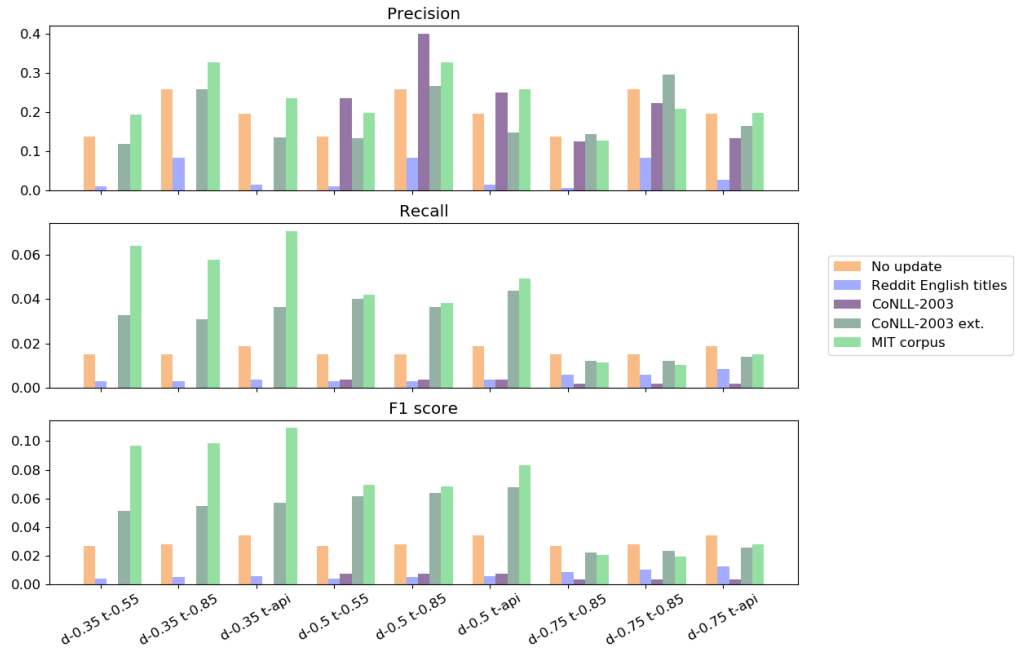

Figure 4.8: Evaluation results of Spacy models for entity types associated with movie titles. In this figure We compare models updated with different datasets and with different dropout values (d), evaluated using different similarity comparisons: threshold (t) or API. The exact results can be found in Appendix B (Table B.2).

with other models. Such results are expected, since this dataset is the only one out of the three that were used for these models, that contained movie titles in the context of bigger natural texts. It is true that the Reddit English titles dataset is also comprised of movie titles, but those are standalone titles with no additional text surrounding them. Comparing the other models with the original one, we can observe an improvement in the end results when using the models updated with MIT movie corpus and the Reddit English titles, while the deterioration of those when using the model updated on the CoNLL-2003 dataset. Unlike the evaluation of name recognition, it can be observed here, that over all models recall deteriorates with the increase in the dropout value, most likely due to the original model, not being intended for such a task, improving its recognition with any additional information

that we input. Lastly, the API evaluation method for the models that we used seems to be the best choice in the majority of cases, no matter the dropout. This comes as no surprise, as the similarity methods of comparison we used with the threshold evaluation are somewhat simple and limited by the number movies contained in the movie pool.

4.2.2 Stanford NER

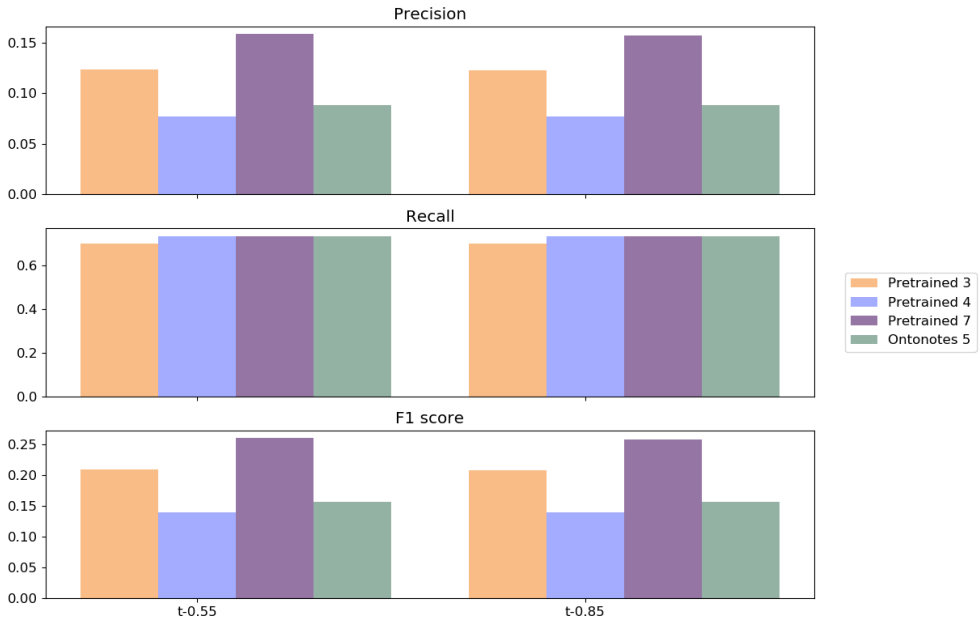

Figure 4.9: Evaluation results of Stanford NER models for entity types associated with persons. We compare different models, evaluated using different similarity comparisons: threshold (t) or API. The exact results can be found in Appendix B (Table B.3).

Looking at the resulting data (Figure 4.9) from the evaluation of Stanford models for the recognition of entity tags related to persons, we can draw some conclusions based on the threshold values and the models we used.

Firstly we explain the effect that different training datasets have on the results. This comparison is not an easy one to make, since it cannot be made simply on the number of entity types a certain model is capable of recognising. Due to this problem we perform comparison strictly on the datasets best results. The most successful model is the Pre-trained 7 (P7) model, which was trained on the MUC-6[2] and MUC-7[3] datasets. In contrast

[2] `https://cs.nyu.edu/cs/faculty/grishman/muc6.html`
[3] `https://catalog.ldc.upenn.edu/LDC2001T02`

to that, the least successful model was Pre-trained 4 (P4) that was trained on the CoNLL-2003 dataset. With the comparison of these two models to the Pre-trained 3 (P3) model, a model that was trained on the combination of MUC datsets, CoNLL-2003 dataset and some additional data, which can be ranked somewhere in the middle by its results, we can safely conclude that the combination of the Stanford NER with the CoNLL-2003 dataset is not the optimal choice. Furthermore, the addition of this dataset to other data can actively deteriorate the results of the model. The performance of the model that was trained using the Ontonotes 5 dataset is very similar to the one of the P4 model with a slightly higher precision, potentially signaling that this dataset could exhibit the same issues as the CoNLL-2003 one, in combination with the Stanford NER algorithm. Secondly, an analysis of the results can be made on the basis of the threshold values. We can observe that over all 4 models that we used, the increase in the threshold value translates to a slight decrease in the precision, or in the case of the Ontonotes 5 model, maintaining the same precision. This indicates, that the models are recognising some of the same word sequences or parts of the sequences, which are then removed later on with the threshold 0.55, but are not similar enough to be removed by the threshold of 0.85.

The comparisons that can be made based on the evaluation of the results for the recognized entities, pertaining to the movie titles (Figure 4.10), are twofold. Firstly, comparing the models results between each other. The performance of the P3, P4 and P7 models closely mimics their performance in recognising persons, with P7 achieving the best results of the three, P3 behind it and P4 performing the worst. With the addition of the P4 model with extended tags (P4 ext.), this hierarchy changes, with the P4 ext. achieving drastically better results than the P7. An explanation for this would the fact, that a lot of movie titles contain words or sequences of words related to locations or organisations (i.e. Shutter island[4] or The Firm[5]), which, if added

[4] `https://www.imdb.com/title/tt1130884/`

[5] `https://www.imdb.com/title/tt0106918/`

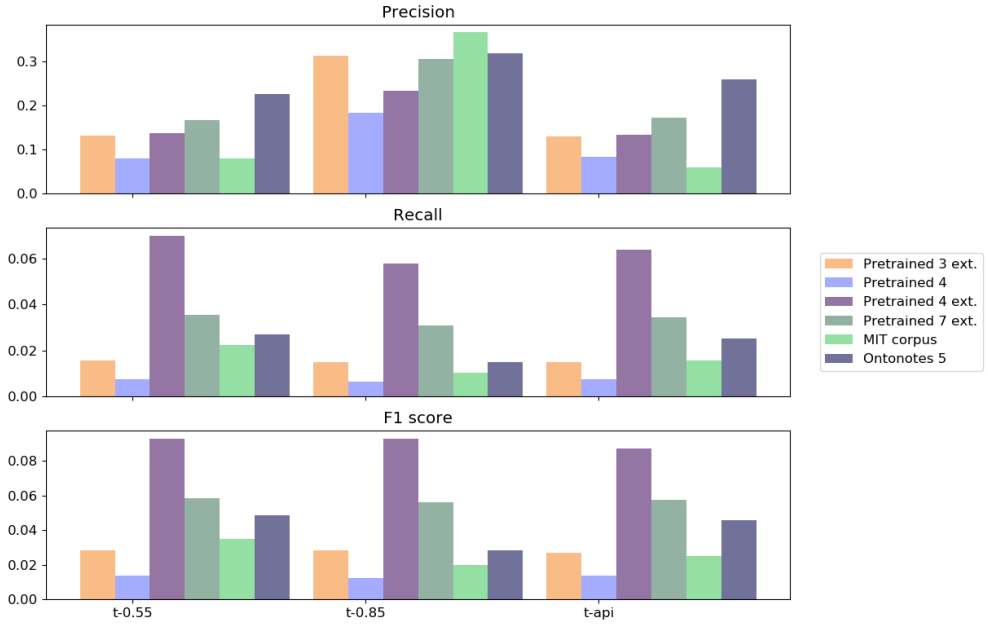

Figure 4.10: Evaluation results of Stanford NER models for entity types associated with movie titles. We compare different models, evaluated using different similarity comparisons: threshold (t) or API. The exact results can be found in Appendix B (Table B.4).

to the predictions of a model, could boost its performance. In contrast to the persons recognition evaluation results, the model trained on the Ontonotes 5 dataset performs much better for the recognition of movie titles, most likely due to the inclusion of the entity type WORK_OF_ART in the dataset, which also covers movie titles. Surprisingly enough, the MIT movie corpus model performs on par with the P4 model, even though this dataset was built exactly for these kind of problems. Secondly, we can observe that in the majority of evaluations the F1 scores computed with the threshold value 0.55 evaluation were the highest or very close to the highest score for the model in question. We suspect that this occurs because Stanford NER algorithm is not that well equipped for dealing with the recognition of movie titles. It produces outputs that are only somewhat similar to the true movie title, meaning that such results get cut off by the high threshold, while they

get passed by the lower threshold. This mostly true for the results obtained by the Ontonotes 5 and the MIT movie corpus models.

When we compare the results of this algorithm to the SPACY algorithm results, presented in the previous section, we can see that these algorithms perform very similar when it comes to the recognition of movie titles, while Stanford NER produces much better results for recognition of persons.

4.2.3 BERT

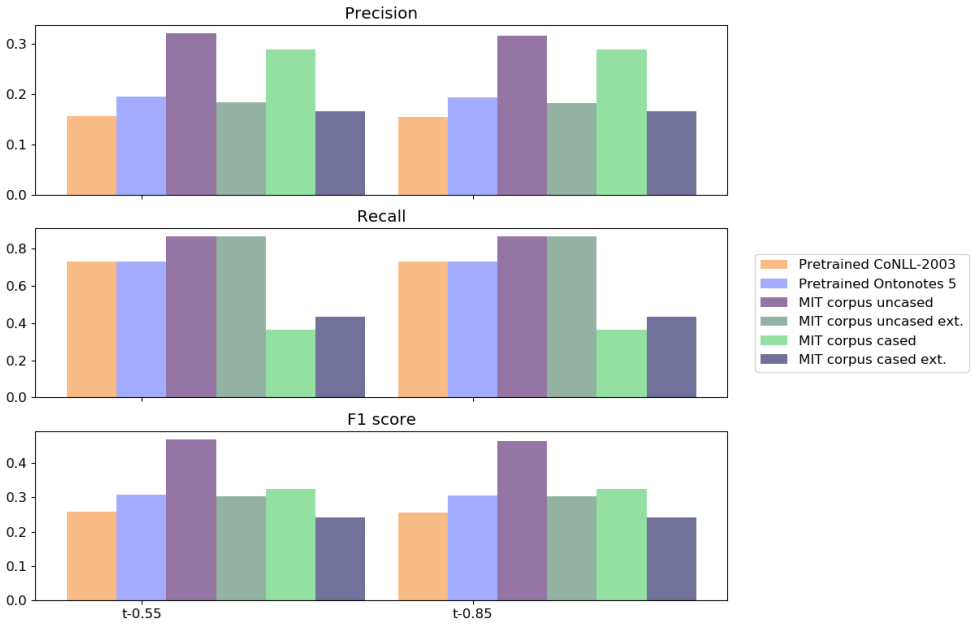

Figure 4.11: Comparison of the results of BERT models for entity types associated with persons. The comparison between models in this figure is based on the similarity threshold (t) when comparing the predicted result with the true result. The complete evaluation results can be found Appendix B (Table B.5).

From a quick glance at these results (Figure 4.11) we can immediately observe that even the worst results produced by the BERT models are still on par with the best ones obtained from the Spacy and Stanford NER algorithms. The difference in the results of the two pre-trained models comes mainly from the precision metric. Similarly to the previous section the precision for these two algorithms deteriorates ever so slightly with the increase of the threshold value, while the recall stays the same. Continuing on to the models fine-tuned on the MIT movie corpus data. Contrary to some minor expectations set up by Google-research[6], the uncased version of the pre-trained does

[6] `https://github.com/google-research/bert`

indeed perform better than the cased model for this problem. Given more thought about this, it makes sense, since the MIT movie corpus dataset is lower-cased, fitting neatly with the pre-trained BERT model. As with the CoNLL-2003 and Ontonotes 5 pre-trained models, the trend of deteriorating precision with the increased threshold value continues with these models as well. Another notable comparison is between the initial model evaluation and the evaluation with the extended entity type tags. The results of the extended tags evaluation for both the cased and uncased models are considerably worse than the initial evaluations. The true names contained in the Reddit submissions do not contain any fictional character names, meaning that if the algorithm is capable of accurately determining which names are fictional and which ones are not, the results will deteriorate if we would add such names to the end predictions. We believe that this is the reason for such a difference in the results.

Similarly to the evaluation of names recognition, the results of all the BERT models for the recognition of movie titles in text (Figure 4.12) are considerably better when compared with any results achieved by Spacy or Stanford NER models. It can be observed from the presented results, that the CoNLL-2003 and the Ontonotes 5 pre-trained models achieve better scores than their counterparts that were fine-tuned on the MIT movie corpus dataset. Similarly to the previous results for the movie titles recognition evaluation, recall seems to deteriorate with the increase of the similarity threshold value, while precision seems to increase with it. This hold true for both the pre-trained models as well as the ones fine-tuned on the MIT movie corpus dataset. On the other hand, when we compare the results of the threshold evaluation to the results returned by the API evaluation, there is a clear difference between the two sets of models. For the pre-trained models it holds true, that the API evaluation results in the highest scores, while for the MIT movie corpus models the highest scores are the ones computed by the similarity threshold evaluation with the threshold value of 0.55. Furthermore, when comparing the evaluations of the pre-trained models, it can be

observed, that extending the entity type tags has a positive influence on the results of the CoNLL-2003 pre-trained model, but has little to no effect on the results of the Ontonotes 5 pre-trained model.

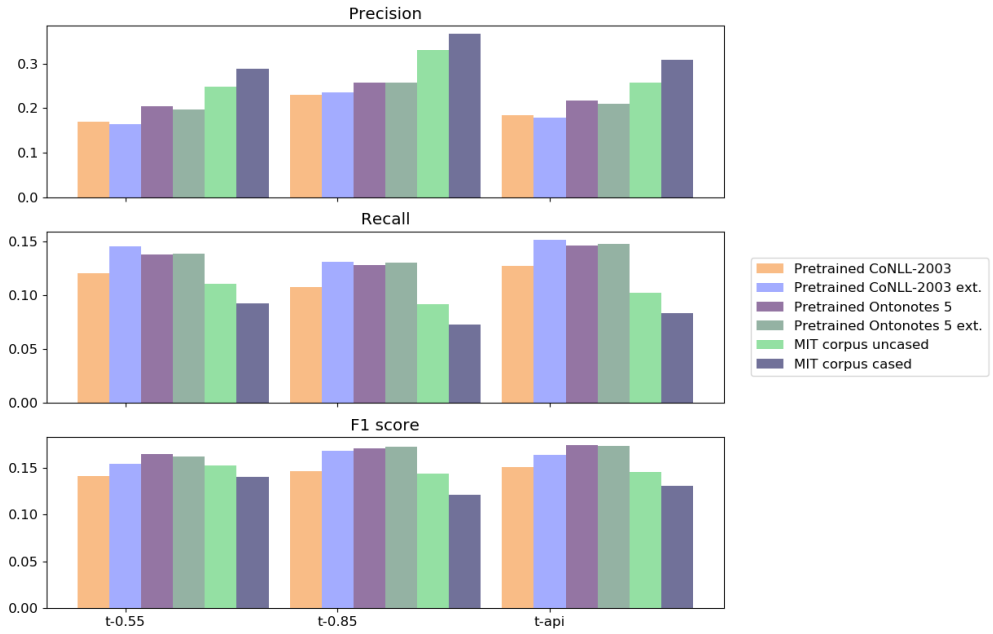

Figure 4.12: Comparison of the results of BERT models for entity types associated with movie titles. The comparison between models in this figure is based on the similarity threshold (t) when comparing the predicted result with the true result. The complete evaluation results can be found Appendix B (Table B.6).

4.2.4 Named entity recognition summary

We achieved the best results for this sub-problem using the state-of-the-art algorithm BERT. This was achieved by fine-tuning the uncased, pre-trained BERT network on the MIT movie corpus dataset. For both the recognition of entities associated with persons and those associated with movie titles, BERT models show better F1 scores than the models built using the other two algorithms.

4.3 Pipeline

The main contribution of this thesis is a pipeline for automated extraction of keywords and recognition of named entities. The pipeline is comprised of a preprocessing algorithm, a keyword extraction model and two named entity recognition models, one for the recognition of entities associated with persons and the other for ones associated with movie titles. The preprocessing part of this pipeline utilizes the same preprocessing algorithm that was described in the previous chapter. For the keyword extraction model we will use the biLSTM model with (400-300-200) architecture, that was trained using transfer learning utilizing the INSPEC dataset. We will recognize entities associated with persons using the uncased BERT model that was fine-tuned on the MIT movie corpus. For the recognition of named entities associated with movie titles we will use the BERT model pre-trained on the Ontonotes 6 dataset. With this setup we should be able to achieve the best overall results, given the evaluation we performed earlier in this chapter.

Chapter 5

Discussion

In our work we strove to acquire the best possible results we could from the data that was available to us. Yet, by diving deeper into the provided crowd-sourced data, we have noticed several problems that could not be completely ignored. As the data is annotated by people it includes a lot of subjectivity, which is inherent for the extraction of keywords and poses little to no problems on that part, but it does become a major problem when deciding on what the original user actually meant (i.e. movie Oldboy could refer to the original 2003 movie[1] or the 2013 remake of that movie[2]). Additionally, in some instances, certain ambiguous parts of the text were not classified as named entities when in reality they should be (such as actor nicknames, i.e. The Rock - Dwayne Johnson), creating an atmosphere of ambiguity in the dataset. Overall, this only becomes a relevant problem due to the smaller size of the of the original data and the fact that named entities are very scare in it. An idea to improve upon the results we achieved as a part of this thesis would be to correct the data and possibly even extend it for an easier training phase and potentially better end results.

Touching upon the evaluation of outputs, we need to discuss the evaluation of movie titles. As explained in the previous chapter, aside from the API

[1] `https://www.imdb.com/title/tt0364569/`
[2] `https://www.imdb.com/title/tt1321511/`

comparison, which is in all regards a black box to us, we only compare the extracted movie entities to a pool of around 12,000 hand-picked English movie titles. This includes only a small fraction of all movies ever made (if we take into account only non-adult movie titles that can be found on IMDb[3], we are looking at around a total of 500,000 movie titles). Due to this, the selection indubitably induces some bias to the results, which we need to be aware of. For the purposes of this thesis, such a bias was acceptable. Although this is the case, we also think that a potential improvement of this could prove to be a big improvement on the results. A possible solution for this problem could be an implementation of a custom-made document embedding algorithm, similar in its execution to an algorithm proposed by Le and Mikolov [19], called Doc2Vec, which would be trained on all possible movie titles. On that note, we also need to make an additional comment about the movie recognition process. The way the movies are recognized is by extracting a word, or a sequence of words, and find the one most similar movie title from the movie pool. This process works fairly well for the purposes of this thesis, but we noticed that some submissions requested a multitude of movie titles by only giving a broad term that could be used to describe a movie franchise or a series of movies (i.e. Bond movies refers to a series of spy movies with the same protagonist). Again, such a problem could be remedied by the use of Doc2Vec by extracting not only the most similar result, but also a small cluster around that result.

Additionally, we need to acknowledge a side of the named entity recognition that we deliberately ignored. We are talking about the existence of hypernyms among the extracted movie titles in the original Reddit datset. For example, a user expresses a request in their submission for spy movies that are similar to James Bond movies. When a model would extract the phrase „James Bond movies" from this submission, our algorithm would return a single movie with the most similar title to this phrase, when in reality it should have returned all or at least a subset of movies that feature the

[3] `https://www.imdb.com/?ref_=nv_home`

titular character James Bond. We have decided to ignore this side due to the complexity of the issue and the expansion to the scope of our thesis it would entail. Although that was the case, we believe that the inclusion of an algorithm that could handle such hypernyms could be beneficial to the evaluation of the models and as a valuable addition to the pipeline for extraction of keywords and named entity recognition.

We are also aware that when we approached the implementation of LSTM and biLSTM algorithms we relied on only one method of embedding the input text sequences, namely the pre-trained Google News Word2Vec model. This decision was made due to the scope of this thesis and some minor hardware limitations at the time of writing. We are aware that multiple promising new embedding techniques and algorithms have been proposed in the past few years, that could potentially be better suited for this problem. As we have not delved into the comparisons between different embedding techniques for this specific problem, we would like to propose such an experiment as potential extension of this research.

Lastly, as we have mentioned or at least implied in previous chapter, the difference between the best models of some of the simpler algorithms are on par with the average results of some of the more complex algorithms. With this we can theorize, that with some additional fine-tuning, the simpler models could still be potentially useful for this problem, as a lighter, less computationally expensive alternative.

Chapter 6

Conclusion

In our work we analysed four algorithms for KE and three algorithms for NER, under different circumstances, parameters and input datasets. We performed a multitude of tests to ascertain which algorithm for each sub problem fits our problem the best. The conclusion we have reached about that is that the best results are achieved by the combination of a three layered biLSTM network for keyword extraction, an uncased BERT model, fine-tuned on the MIT movie corpus dataset, for the recognition of actors and the BERT model, fine-tuned on the Ontonotes 5 dataset, for the recognition of movie titles.

The contribution of this thesis is twofold. Firstly, the analytical results of the evaluations of multiple different algorithm. Secondly, a functional pipeline for the extraction of keywords and named entities from the provided input texts.

We hope that our research opens the doors for others to continue where we have left off and improve upon our results. We firmly believe that our contributions could present themselves as crucial in systems (such as recommendation systems, etc.) that aim to improve their functionality with specific user generated requests. Our ideas for future work include the implementation of a better collection of movie titles instead of the current movie pool, perhaps even an implementation of more robust way of determining a

movie title from a string (in place of the current method, where we check for similarity with other movie titles). Additionally, a possible continuation of our research could focus on the adaptation of the BERT algorithm to work properly for the keyword extraction.

Appendix A

Keyword extraction tables

Input \Metric	Accuracy	Precision	Recall	F1
Normal	0.4303	**0.8616**	0.0613	0.1145
Removed stopwords	**0.6497**	0.476	**0.119**	**0.1904**

Table A.1: Complete evaluation results for the RAKE keyword extraction algorithm, comparing the performance of the algorithm based on the input text, with or without stopwords.

Input \Metric	Accuracy	Precision	Recall	F1
Normal	0.6746	**0.7138**	0.0894	0.159
No stopwords	**0.6909**	0.4851	**0.141**	**0.2185**

Table A.2: Complete evaluation results for the TextRank keyword extraction algorithm, comparing the performance of the algorithm based on the input text, with or without stopwords.

Model \Metric	**Accuracy**	**Precision**	**Recall**	**F1**
LSTM (100)	0.9572	0.8226	0.1325	0.2283
LSTM (300)	0.9572	0.8473	0.1268	0.2206
LSTM (100-100)	0.9554	0.7920	0.0914	0.1639
LSTM (300-300)	0.9550	**0.8695**	0.0685	0.1271
LSTM (200-300)	0.9576	0.5918	0.3645	0.4512
LSTM (300-200)	0.9577	0.7380	0.1771	0.2857
LSTM (400-300-200)	0.9614	0.7076	0.3291	0.4492
biLSTM (100)	0.9617	0.7013	0.3462	0.4636
biLSTM (300)	0.9600	0.7037	0.2822	0.4029
biLSTM (100-100)	0.9623	0.6915	**0.3817**	0.4918
biLSTM (300-300)	0.9600	0.6524	0.3497	0.4553
biLSTM (200-300)	**0.9624**	0.6943	**0.3817**	**0.4926**
biLSTM (300-200)	0.9617	0.6891	0.3622	0.4749
biLSTM (400-300-200)	0.9618	0.7002	0.3497	0.4664

Table A.3: Complete evaluation results for the LSTM and biLSTM keyword extraction algorithms using the Reddit dataset as training data. The names of the models in this table are taken from the used algorithm and the architecture of the model (number of neurons in each of its hidden layers).

Model \Metric	**Accuracy**	**Precision**	**Recall**	**F1**
LSTM (100)	0.9523	0.5384	0.008	0.0157
LSTM (300)	0.9522	0	0	0
LSTM (100-100)	0.9517	0.4242	0.032	0.0595
LSTM (300-300)	0.9522	0	0	0
LSTM (200-300)	0.9522	0	0	0
LSTM (300-200)	0.9520	0.4285	0.0137	0.0265
LSTM (400-300-200)	0.9491	0.3251	0.0605	0.1021
biLSTM (100)	0.9524	0.5306	0.0297	0.0562
biLSTM (300)	**0.9525**	**0.6**	0.0171	0.0333
biLSTM (100-100)	0.9518	0.4677	0.0662	0.1161
biLSTM (300-300)	0.9506	0.4121	**0.0777**	**0.1307**
biLSTM (200-300)	0.9517	0.4651	0.0685	0.1195
biLSTM (300-200)	0.9523	0.5172	0.0342	0.0643
biLSTM (400-300-200)	0.9508	0.4044	0.0628	0.1088

Table A.4: Complete evaluation results for the LSTM and biLSTM keyword extraction algorithms using the Krapivin dataset as training data. The names of the models in this table are taken from the used algorithm and the architecture of the model (number of neurons in each of its hidden layers).

Model \Metric	Accuracy	Precision	Recall	F1
LSTM (100)	0.8517	0.1688	**0.5371**	0.2569
LSTM (300)	0.8878	0.1878	0.4057	0.2567
LSTM (100-100)	0.8809	0.1832	0.432	0.2573
LSTM (300-300)	0.8950	0.1935	0.3782	0.2560
LSTM (200-300)	0.8790	0.1770	0.4205	0.2492
LSTM (300-200)	0.8741	0.1758	0.4434	0.2517
LSTM (400-300-200)	0.8834	0.1810	0.4091	0.2510
biLSTM (100)	**0.9195**	**0.2239**	0.2777	0.2479
biLSTM (300)	0.9149	0.2182	0.3028	0.2537
biLSTM (100-100)	0.9078	0.2141	0.3485	**0.2653**
biLSTM (300-300)	0.9029	0.2057	0.3611	0.2621
biLSTM (200-300)	0.9040	0.1931	0.3177	0.2402
biLSTM (300-200)	0.9012	0.1979	0.3497	0.2527
biLSTM (400-300-200)	0.9159	0.1918	0.2365	0.2118

Table A.5: Complete evaluation results for the LSTM and biLSTM keyword extraction algorithms using the INSPEC dataset as training data. The names of the models in this table are taken from the used algorithm and the architecture of the model (number of neurons in each of its hidden layers).

Model \Metric	**Accuracy**	**Precision**	**Recall**	**F1**
LSTM (100)	0.9610	0.7146	0.3062	0.4288
LSTM (300)	0.9608	0.7743	0.2548	0.3834
LSTM (100-100)	0.9627	**0.7727**	0.3108	0.4433
LSTM (300-300)	0.9614	0.6850	0.3554	0.4680
LSTM (200-300)	0.9616	0.6541	0.4171	0.5094
LSTM (300-200)	0.9621	0.7015	0.36	0.4758
LSTM (400-300-200)	0.9620	0.6573	0.4297	0.5196
biLSTM (100)	0.9618	0.6929	0.3611	0.4748
biLSTM (300)	0.9608	0.6828	0.3371	0.4514
biLSTM (100-100)	0.9613	0.6437	0.4274	0.5137
biLSTM (300-300)	**0.9628**	0.6924	0.3988	0.5061
biLSTM (200-300)	0.9609	0.6412	0.4125	0.5020
biLSTM (300-200)	0.9625	0.6766	0.4114	0.5117
biLSTM (400-300-200)	0.9615	0.6384	**0.448**	**0.5265**

Table A.6: Complete evaluation results for the LSTM and biLSTM keyword extraction algorithms using the Krapivin dataset as training data and Reddit dataset for transfer learning. The names of the models in this table are taken from the used algorithm and the architecture of the model (number of neurons in each of its hidden layers).

Model \Metric	Accuracy	Precision	Recall	F1
LSTM (100)	0.9624	0.7343	0.3348	0.4599
LSTM (300)	0.9612	0.7350	0.2948	0.4208
LSTM (100-100)	0.9637	0.7042	0.4137	0.5212
LSTM (300-300)	0.9635	0.7620	0.344	0.4740
LSTM (200-300)	**0.9638**	0.6992	0.4251	0.5287
LSTM (300-200)	0.9635	0.7216	0.3851	0.5022
LSTM (400-300-200)	0.9622	0.6580	0.4354	0.5240
biLSTM (100)	0.9620	0.7535	0.304	0.4332
biLSTM (300)	0.9624	**0.7906**	0.2891	0.4234
biLSTM (100-100)	0.9629	0.6828	0.4182	0.5187
biLSTM (300-300)	0.9629	0.6884	0.4091	0.5132
biLSTM (200-300)	0.9628	0.6857	0.4114	0.5142
biLSTM (300-200)	0.9626	0.6708	0.4262	0.5213
biLSTM (400-300-200)	0.9614	0.6183	**0.5017**	**0.5539**

Table A.7: Complete evaluation results for the LSTM and biLSTM keyword extraction algorithms using the INSPEC dataset as training data and Reddit dataset for transfer learning. The names of the models in this table are taken from the used algorithm and the architecture of the model (number of neurons in each of its hidden layers).

Appendix B

Named entity recognition tables

Model \Metric	**Precision**	**Recall**	**F1**
No update (t=0.55)	0.0699	0.7666	0.1281
No update (t=0.85)	0.0696	0.7666	0.1277
Reddit English titles (d=0.75, t=0.55)	0.4	0.0666	0.1142
Reddit English titles (d=0.75, t=0.85)	0.4	0.0666	0.1142
CoNLL-2003 (d=0.35, t=0.55)	0.0524	0.8	0.0983
CoNLL-2003 (d=0.35, t=0.85)	0.0522	0.8	0.0981
CoNLL-2003 (d=0.5, t=0.55)	0.0584	0.8333	0.1091
CoNLL-2003 (d=0.5, t=0.85)	0.0582	0.8333	0.1089
CoNLL-2003 (d=0.75, t=0.55)	**0.0982**	0.7333	0.1732
CoNLL-2003 (d=0.75, t=0.85)	0.0977	0.7333	0.1725
MIT corpus (d=0.35, t=0.55)	0.0807	0.8666	0.1477
MIT corpus (d=0.35, t=0.85)	0.0802	0.8666	0.1468
MIT corpus (d=0.5, t=0.55)	0.0678	**0.9**	0.1261
MIT corpus (d=0.5, t=0.85)	0.0676	**0.9**	0.1258
MIT corpus (d=0.75, t=0.55)	0.0747	0.8	0.1367
MIT corpus (d=0.75, t=0.85)	0.0745	0.8	0.1363
MIT corpus ext. (d=0.35, t=0.55)	0.081	**0.9**	**0.1487**
MIT corpus ext. (d=0.35, t=0.85)	0.0808	**0.9**	0.1483
MIT corpus ext. (d=0.5, t=0.55)	0.0754	**0.9**	0.1391
MIT corpus ext. (d=0.5, t=0.85)	0.0752	**0.9**	0.1388
MIT corpus ext. (d=0.75, t=0.55)	0.0722	0.8	0.1325
MIT corpus ext. (d=0.75, t=0.85)	0.072	0.8	0.1323

Table B.1: Complete evaluation results of Spacy models for entity types associated with persons. We compare models updated with different datasets and with different dropout values (d), evaluated using different similarity comparisons: threshold (t) or API.

Model \Metric	Precision	Recall	F1
No update (t=0.55)	0.1367	0.0149	0.0269
No update (t=0.85)	0.258	0.0149	0.0282
No update (API)	0.196	0.0186	0.0341
Reddit English titles (d=0.75, t=0.55)	0.005	0.0056	0.0085
Reddit English titles (d=0.75, t=0.85)	0.0833	0.0056	0.0105
Reddit English titles (d=0.75, API)	0.027	0.0084	0.0128
CoNLL-2003 (d=0.5, t=0.55)	0.2352	0.0037	0.0073
CoNLL-2003 (d=0.5, t=0.85)	**0.4**	0.0037	0.0074
CoNLL-2003 (d=0.5, API)	0.25	0.0037	0.0073
CoNLL-2003 (d=0.75, t=0.55)	0.125	0.0018	0.0036
CoNLL-2003 (d=0.75, t=0.85)	0.2222	0.0018	0.0037
CoNLL-2003 (d=0.75, API)	0.1333	0.0018	0.0036
CoNLL-2003 ext. (d=0.35, t=0.55)	0.1182	0.0327	0.0512
CoNLL-2003 ext. (d=0.35, t=0.85)	0.2578	0.0308	0.055
CoNLL-2003 ext. (d=0.35, API)	0.1344	0.0364	0.0573
CoNLL-2003 ext. (d=0.5, t=0.55)	0.1327	0.0401	0.0616
CoNLL-2003 ext. (d=0.5, t=0.85)	0.2671	0.0364	0.0641
CoNLL-2003 ext. (d=0.5, API)	0.1473	0.0439	0.0676
CoNLL-2003 ext. (d=0.75, t=0.55)	0.1444	0.0121	0.0224
CoNLL-2003 ext. (d=0.75, t=0.85)	0.2954	0.0121	0.0233
CoNLL-2003 ext. (d=0.75, API)	0.1648	0.014	0.0258
MIT corpus (d=0.35, t=0.55)	0.1927	**0.0644**	0.0966
MIT corpus (d=0.35, t=0.85)	0.328	0.0579	0.0984
MIT corpus (d=0.35, API)	0.2352	0.071	**0.1091**
MIT corpus (d=0.5, t=0.55)	0.1973	0.042	0.0693
MIT corpus (d=0.5, t=0.85)	0.328	0.0383	0.0686
MIT corpus (d=0.5, API)	0.2572	0.0495	0.083
MIT corpus (d=0.75, t=0.55)	0.1276	0.0112	0.0206
MIT corpus (d=0.75, t=0.85)	0.2075	0.0102	0.0195
MIT corpus (d=0.75, API)	0.1975	0.0149	0.0278

Table B.2: Complete evaluation results of Spacy models for entity types associated with movie titles. We compare models updated with different datasets and with different dropout values (d), evaluated using different similarity comparisons: threshold (t) or API.

Model \Metric	**Precision**	**Recall**	**F1**
Pre-trained 3 (t=0.55)	0.1235	0.7	0.21
Pre-trained 3 (t=0.85)	0.1228	0.7	0.2089
Pre-trained 4 (t=0.55)	0.0774	**0.7333**	0.1401
Pre-trained 4 (t=0.85)	0.0771	**0.7333**	0.1396
Pre-trained 7 (t = 0.55)	**0.1582**	**0.7333**	**0.2603**
Pre-trained 7 (t=0.85)	0.1571	**0.7333**	0.2588
Ontonotes 5 (t=0.55)	0.088	**0.7333**	0.1571
Ontonotes 5 (t=0.85)	0.088	**0.7333**	0.1571

Table B.3: Complete evaluation results of Stanford NER models for entity types associated with persons. We compare different models, evaluated using different similarity comparisons: threshold (t) or API.

Model \Metric	Precision	Recall	F1
Pre-trained 3 ext. (t=0.55)	0.1328	0.0158	0.0283
Pre-trained 3 ext. (t=0.85)	0.3137	0.0149	0.0285
Pre-trained 3 ext. (API)	0.13	0.0149	0.0268
Pre-trained 4 (t=0.55)	0.0808	0.0074	0.0136
Pre-trained 4 (t=0.85)	0.1842	0.0065	0.0126
Pre-trained 4 (API)	0.0842	0.0074	0.0137
Pre-trained 4 ext. (t=0.55)	0.1368	**0.07**	0.0927
Pre-trained 4 ext. (t=0.85)	0.2348	0.0579	**0.0929**
Pre-trained 4 ext. (API)	0.1339	0.064	0.087
Pre-trained 7 ext. (t=0.55)	0.1666	0.0355	0.0585
Pre-trained 7 ext. (t=0.85)	0.3055	0.0308	0.056
Pre-trained 7 ext. (API)	0.1728	0.0345	0.0576
MIT corpus (t=0.55)	0.0794	0.0224	0.0349
MIT corpus (t=0.85)	**0.3666**	0.0102	0.02
MIT corpus (API)	0.0596	0.0158	0.025
Ontonotes 5 (t=0.55)	0.2265	0.0271	0.0484
Ontonotes 5 (t=0.85)	0.32	0.0149	0.0285
Ontonotes 5 (API)	0.2596	0.0252	0.0459

Table B.4: Complete evaluation results of Stanford NER models for entity types associated with movie titles. We compare different models, evaluated using different similarity comparisons: threshold (t) or API.

Model \Metric	**Precision**	**Recall**	**F1**
Pre-trained CoNLL-2003 (t=0.55)	0.156	0.7333	0.2573
Pre-trained CoNLL-2003 (t=0.85)	0.1549	0.7333	0.2558
Pre-trained Ontonotes 5 (t=0.55)	0.1946	0.7333	0.3076
Pre-trained Ontonotes 5 (t=0.85)	0.1929	0.7333	0.3055
MIT corpus uncased (t=0.55)	**0.3209**	**0.8666**	**0.4684**
MIT corpus uncased (t=0.85)	0.317	**0.8666**	0.4642
MIT corpus uncased ext. (t=0.55)	0.1843	**0.8666**	0.304
MIT corpus uncased ext. (t=0.85)	0.183	**0.8666**	0.3023
MIT corpus cased (t=0.55)	0.2894	0.3666	0.3235
MIT corpus cased (t=0.85)	0.2894	0.3666	0.3235
MIT corpus cased ext. (t=0.55)	0.1666	0.4333	0.2407
MIT corpus cased ext. (t=0.85)	0.1666	0.4333	0.2407

Table B.5: Complete evaluation results of BERT models for entity types associated with persons. We compare different models, evaluated using different similarity thresholds (t).

Model \Metric	Precision	Recall	F1
Pre-trained CoNLL-2003 (t=0.55)	0.169	0.1205	0.1407
Pre-trained CoNLL-2003 (t=0.85)	0.229	0.1074	0.1463
Pre-trained CoNLL-2003 (API)	0.1845	0.1271	0.1505
Pre-trained CoNLL-2003 ext. (t=0.55)	0.1643	0.1448	0.1539
Pre-trained CoNLL-2003 ext. (t=0.85)	0.2348	0.1308	0.168
Pre-trained CoNLL-2003 ext. (API)	0.1792	0.1514	0.1641
Pre-trained Ontonotes 5 (t=0.55)	0.2044	0.1373	0.1643
Pre-trained Ontonotes 5 (t=0.85)	0.2565	0.128	0.1708
Pre-trained Ontonotes 5 (API)	0.2169	0.1457	**0.1743**
Pre-trained Ontonotes 5 ext. (t=0.55)	0.1965	0.1383	0.1623
Pre-trained Ontonotes 5 ext. (t=0.85)	0.2564	0.1299	0.1724
Pre-trained Ontonotes 5 ext. (API)	0.2092	**0.1476**	0.1731
MIT corpus uncased (t=0.55)	0.2489	0.1102	0.1528
MIT corpus uncased (t=0.85)	0.331	0.0915	0.1434
MIT corpus uncased (API)	0.257	0.1018	0.1459
MIT corpus cased (t=0.55)	0.2877	0.0925	0.14
MIT corpus cased (t=0.85)	**0.3661**	0.0728	0.1215
MIT corpus cased (API)	0.3079	0.0831	0.1309

Table B.6: Complete evaluation results of BERT models for entity types associated with movie titles. We compare different models, evaluated using different similarity comparisons: threshold (t) or API.

Bibliography

[1] I. Facts, Figures-the world in 2015, Geneva: The International Telecommunication Union (ITU) (2015).

[2] R. Mihalcea, P. Tarau, Textrank: Bringing order into text, in: Proceedings of the 2004 conference on empirical methods in natural language processing, 2004.

[3] S. Rose, D. Engel, N. Cramer, W. Cowley, Automatic keyword extraction from individual documents, Text Mining: Applications and Theory (2010) 1–20.

[4] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013).

[5] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[6] F. A. Gers, J. Schmidhuber, F. Cummins, Learning to forget: Continual prediction with LSTM (1999).

[7] M. Basaldella, E. Antolli, G. Serra, C. Tasso, Bidirectional lstm recurrent neural network for keyphrase extraction, in: Italian Research Conference on Digital Libraries, Springer, 2018, pp. 180–187.

[8] M. Honnibal, I. Montani, spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing, to appear (2017).

[9] J. R. Finkel, T. Grenager, C. Manning, Incorporating non-local information into information extraction systems by gibbs sampling, in: Proceedings of the 43rd annual meeting on association for computational linguistics, Association for Computational Linguistics, 2005, pp. 363–370.

[10] J. Lafferty, A. McCallum, F. C. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: ICML 2001 (2001).

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

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.

[13] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, arXiv preprint arXiv:1609.08144 (2016).

[14] A. Hulth, Improved automatic keyword extraction given more linguistic knowledge, in: Proceedings of the 2003 conference on Empirical methods in natural language processing, Association for Computational Linguistics, 2003, pp. 216–223.

[15] M. Krapivin, A. Autaeu, M. Marchese, Large dataset for keyphrases extraction, Tech. rep., University of Trento (2009).
URL `http://eprints.biblio.unitn.it/1671/1/disi09055-krapivin-autayeu-marchese.pdf`

[16] E. F. Sang, F. De Meulder, Introduction to the conll-2003 shared task: Language-independent named entity recognition, arXiv preprint cs/0306050 (2003).

[17] Spoken language systems group2013, laungage Systems Group. 2013. The MIT Movie Corpus. https://groups.csail.mit.edu/sls/downloads/, MIT Computer Science and Artificial Intelligence Laboratory.

[18] P. E. Black, Ratcliff/obershelp pattern recognition, Dictionary of algorithms and data structures 17 (2004).

[19] Q. Le, T. Mikolov, Distributed representations of sentences and documents, in: International conference on machine learning, 2014, pp. 1188–1196.