Keyphrase Extraction from Scholarly Documents

Author: Chrysovalantis Giorgos Kontoulis
Supervisor: Dr. Grigorios Tsoumakas

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in the Department of Computer Science

May 5, 2021
Abstract

The problem of keyphrase extraction is essential for many academic research tasks such as document indexing, recommendation systems, and, many more. Keyphrases could be used even as a special kind of text summarization for humans to assist in the comprehension of the discussed topics in some text without having to read it first. Despite its importance, it is a particularly challenging problem as even the state-of-the-art models cannot achieve top-tier performance.

In this thesis, we explore the problem of automatic keyphrase extraction on academic publications by utilizing various methods on the Bi-LSTM-CRF model (Alzaidy et al., 2019), which is a state-of-the-art recurrent neural network for this task. To our knowledge, there is no public repository with a contemporary implementation of the model tuned for keyphrase extraction. So, we provide one with a tensorflow implementation of the model.

We approach the task as a binary sequence labeling problem, in which the words are classified into keywords and non-keywords. We also provide insightful information and general guidelines stemming from data analysis in view of improving the performance on the task. Additionally, we experimented with methods aiming to introduce new keyphrases in comparison to the staple combination of the title and abstract in hopes of further improving performance.

Specifically, considering the data analysis, we isolate parts of the documents that potentially are more keyphrase dense, e.g. the first few paragraphs of a scholarly document, and, we experiment with the full-text of the documents by splitting it into paragraphs and sentences. Also, we experiment with summaries of the full-text documents, which are generated by state-of-the-art deep learning models.

To check the validity of our methods, we employed them to popular unsupervised models as well. Our methods can be used with any keyphrase extraction, and possibly abstraction, model to achieve greater results. In our setting, we managed to increase the exact match F1-score by as much as 5.85%. We further provide guidelines to select the proper text excerpt for each of the used machine learning models.

Last but not least, we introduce a new method of calculating the F1-score, recall and precision metrics. This method aims to soothe the strictness of the exact matching by addressing the case where a gold keyphrase is a substring of a predicted. For that reason, we call the new method semi-exact matching.
Acknowledgements

I would like to express my great gratitude to my advisor professor Grigorios Tsoumakas for being present from the very start to the very end of the composure of this thesis. He provided me with assistance and ideas at every step and helped me to draw this thesis to a close. His passion and scientific curiosity motivated me to work harder and consistently throughout the whole thesis development, while his attention to detail lead to a fine end result.

Also, I would like to gratify Eirini Papagiannopoulou for sharing her domain expertise, providing insightful material and assisting in the writing of the thesis. She really helped me to grasp a deeper understanding of the problem, resulting in substantial expansion of my field knowledge, and she gave structure to a scientifically appropriate end product.

Special thanks to the professors and instructors of this Master’s program, as I am really grateful for all the hard work they have put to provide us with valuable information and knowledge as well as their expertise sharing. Last but not least, I would like to thank my friends and family for being pillars of support throughout this endeavour.
Το πρόβλημα της εξαγωγής λέξεων κλειδιών είναι σημαντικό για πολλές ακαδημαϊκές ερευνητικές εργασίες όπως η εισαγωγή εγγράφων, τα συστήματα συστάσεων και πολλά άλλα. Οι λέξεις κλειδιά θα μπορούσαν να χρησιμοποιηθούν ακόμη και ως ένα ειδικό είδος περίληψης κειμένου, το οποίο μπορεί να βοηθήσει τους ανθρώπους στην κατανόηση των συζητηθέντων θεμάτων κάποιον χωρίς να χρειάζεται να το διαβάσουν πρώτα. Παρά τη σημαντικότητά του, είναι ένα ιδιαίτερα δύσκολο πρόβλημα, καθώς ακόμη και τα υπερσύγχρονα μοντέλα δεν μπορούν να επιτύχουν κορυφαία απόδοση.

Σε αυτή τη διατριβή, αναλύουμε το πρόβλημα της αυτόματης εξαγωγής φράσεων κλειδιών σε ακαδημαϊκές δημοσιεύσεις, χρησιμοποιώντας διάφορες μεθόδους στο μοντέλο Bi-LSTM-CRF (Alzaidy et al., 2019), το οποίο είναι ένα υπερσύγχρονο επαναλαμβανόμενο νευρικό δίκτυο (RNN). Από όσα γνωρίζουμε, δεν υπάρχει δημόσιο αποθετήριο (repository) με σύγχρονη και επικαιροποιημένη υλοποίηση του μοντέλου που έχει ρυθμιστεί για να αντιμετωπίζει το πρόβλημα εξαγωγής φράσεων κλειδιών. Έτσι, δημιουργήσαμε εμείς ένα, με μια υλοποίηση του μοντέλου που στηρίζεται στην βιβλιοθήκη tensorflow.

Σε αυτή την εργασία, προσεγγίζουμε το πρόβλημα ως δυαδικό, όπου οι λέξεις των ακολουθιών ταξινομούνται σε λέξεις-κλειδιά και μη λέξεις-κλειδιά. Επίσης, παρέχουμε σημαντικές πληροφορίες, παρατηρήσεις και γενικές οδηγίες που προκύπτουν από την ανάλυση των δεδομένων με σκοπό τη βελτίωση της απόδοσης των μοντέλων στο πρόβλημα. Επιπλέον, πειραματιστήκαμε με μεθόδους που στοχεύουν στην εισαγωγή νέων φράσεων κλειδιών σε σύγχρονη κείμενα με τον κλασσικό συνδυασμό του τίτλου και της περίληψης που χρησιμοποιείται ευρέως, με την ελπίδα βελτίωσης της απόδοσης.

Συγκεκριμένα, λαμβάνοντας υπόψη την ανάλυση δεδομένων, απομονώσαμε τμήματα των εγγράφων τους είναι πιθανός πυκνότερα σε φράσεις κλειδιά, π.χ. τις πρώτες παραγράφους καθώς ένας επιστημονικός εγγράφος και, πειραματιστήκαμε με το πλήρες κείμενο των εγγράφων χωρίζοντας το σε παραγράφους και προτάσεις. Επίσης, πειραματιστήκαμε με περίληψης των εγγράφων των παραδείγματα αυτοματοποιημένα από το πλήρες κείμενο με τη χρήση υπερσύγχρονων μεθόδων βαθειάς μάθησης.

Για να ελέγξουμε την αποτελεσματικότητα των μεθόδων μας, τις χρησιμοποιήσαμε και σε δημοφιλή μοντέλα χωρίς επίβλεψη. Οι μέθοδοι μας
μπορούν να χρησιμοποιηθούν με οποιαδήποτε μέθοδο εξαγωγής φράσεων κλειδιών για την επίτευξη καλύτερων αποτελεσμάτων. Στα πειράματα μας, καταφέραμε να αυξήσουμε τη μετρική exact match F1-score κατά 5.85%. Επιπλέον, προτείνουμε οδηγίες για την επιλογή του κατάλληλου αποσπάσματος κειμένου για κάθε ένα από τα μοντέλα μηχανικής μάθησης που χρησιμοποιήσαμε στα πειράματα.

Τέλος, εισάγουμε μια νέα μέθοδο υπολογισμού των μετρικών απόδοσης F1-score, ανάκλησης (recall) και ακρίβειας (precision). Αυτή η μέθοδος έχει ως στόχο να μειώσει την αυστηρότητα της ακριβούς αντιστοίχισης (exact match) αντιμετωπίζοντας την περίπτωση όπου μια φράση βασικής αλήθειας (ground truth) είναι υποσύνολο μιας προβλεπόμενης. Για αυτόν τον λόγο, νομοθέτουμε τη νέα μέθοδο ημι-ακριβή αντιστοίχιση (semi-exact match).
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>Περιλήψη στα Ελληνικά</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Contributions</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Thesis Overview</td>
<td>2</td>
</tr>
<tr>
<td>2 Keyphrase Extraction</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Unsupervised Learning</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Statistics-based Methods</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Graph-based Ranking Methods</td>
<td>6</td>
</tr>
<tr>
<td>2.1.3 Topic-based Methods</td>
<td>7</td>
</tr>
<tr>
<td>2.1.4 Information based on Similar Documents/Citation Networks</td>
<td>8</td>
</tr>
<tr>
<td>2.1.5 Graph-based Methods with Semantics</td>
<td>8</td>
</tr>
<tr>
<td>2.1.6 Keyphrase Extraction based on Embeddings</td>
<td>9</td>
</tr>
<tr>
<td>2.1.7 Language Model-based Methods</td>
<td>10</td>
</tr>
<tr>
<td>2.2 Supervised Learning</td>
<td>10</td>
</tr>
<tr>
<td>2.2.1 Traditional Supervised Methods</td>
<td>10</td>
</tr>
<tr>
<td>Keyphrase Extraction as Binary Classification Task</td>
<td>11</td>
</tr>
<tr>
<td>Keyphrase Extraction as Learning to Rank Task</td>
<td>12</td>
</tr>
<tr>
<td>Keyphrase Extraction Using Supervision</td>
<td>12</td>
</tr>
<tr>
<td>Sequential Labeling</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2 Types of Supervised Learning features</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Deep Learning</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Evaluation</td>
<td>19</td>
</tr>
<tr>
<td>2.3.1 Exact vs Partial Matching</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2 Subjectivity and Class Imbalance</td>
<td>23</td>
</tr>
<tr>
<td>2.3.3 Datasets</td>
<td>24</td>
</tr>
<tr>
<td>3 Neural Networks</td>
<td>27</td>
</tr>
<tr>
<td>3.1 Deep Neural Networks</td>
<td>27</td>
</tr>
<tr>
<td>3.1.1 Simple Feed-forward Networks</td>
<td>27</td>
</tr>
<tr>
<td>3.1.2 Forward Pass</td>
<td>28</td>
</tr>
<tr>
<td>3.1.3 Output layers</td>
<td>29</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Unsupervised keyphrase extraction methods. (Papagiannopoulou and Tsoumakas, 2020) ............... 5
2.2 Supervised keyphrase extraction methods. (Papagiannopoulou and Tsoumakas, 2020) ............... 10
2.3 Evaluation methods popularity based on the literature. (Papagiannopoulou and Tsoumakas, 2020) ........ 21

3.1 Three layer feed-forward neural network ........................... 28
3.2 Neuron of an artificial neural network ......................... 29
3.3 RNN unfolded through time (LeCun et al., 2015) .......... 33
3.4 LSTM block .................................................. 34
3.5 Sequence-to-sequence problem types and examples ........... 35
3.6 Simple sequence-to-sequence model architecture .......... 36
3.7 Sequence-to-sequence model with bidirectional encoder and attention mechanism ..................... 39
3.8 Bidirectional RNN with sigmoid output layer .............. 40
3.9 Bi-LSTM-CRF architecture .................................. 41
3.10 CRF architecture ........................................... 42

4.1 Data flow throughout each step of our experiments ......... 44
4.2 Percentage of keyphrase coverage per information source in the kp527k-training set (abstract and title) .......... 48
4.3 Histogram of document lengths of the kp527-train set ... 50
4.4 Histogram of document lengths of the kp527-train set after splitting it into sentences. ................. 51
4.5 Percentage of keyphrase coverage per information source in the kp20k-test set (abstract and title) ... 57
4.6 Percentage of keyphrase coverage for the variations of the title and abstract, the full-text, and, title and summarized text (distillated roberta) .......................... 58
4.7 Percentage of keyphrase coverage for the first 3 paragraphs for lengths 220 and 400 ................. 59
4.8 Semi-exact match: calculation example of the gold list .. 63
List of Tables

2.1 Well-known datasets grouped by the type of available text excerpts (Type). Papagiannopoulou and Tsoumakas, 2020 .......................................................... 24

4.1 Statistics for the datasets used to reproduce Alzaidy et al. (2019) (after pre-processing) ........................................ 49

4.2 Statistics for the main test datasets (after pre-processing), i.e., NUS, ACM, & SemEval-2010 with the full-text .... 49

4.3 Cropped documents per sequence size for continuous text and sentence model ............................................. 50

4.4 Bucket sizes .......................................................................................................................... 51

4.5 ACM, NUS and SemEval statistics for the paragraph split experiment, where a': full-text split in paragraphs, and, b': 3 first paragraphs (kp: keyphrase, kw: keyword, non-kw: non-keyword) ........................................... 56

4.6 Time needed to produce summaries for each full-text test dataset with distilled roberta ..................................... 60

4.7 F\textsubscript{1} based on sequence (S), exact (E) and partial (P) evaluation for the original Bi-LSTM-CRF approach and our implementation. Bi-LSTM-CRF\textsubscript{full} is trained on whole titles and abstracts, whereas Bi-LSTM-CRF\textsubscript{sent} operates at the sentence level ......................................................... 65

4.8 F\textsubscript{1} based on Exact (E) and Partial (P) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) .......................................................... 67

4.9 F\textsubscript{1}@10 based on Exact (E) and Partial (P) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ...................................................... 69
4.10 \( F_{1@20} \) based on Exact (E) and Partial (P) evaluation approach for TF-IDF on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . . . . . . . . 69

4.11 \( F_{1@10} \) based on Exact (E) and Partial (P) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . 70

4.12 \( F_{1@20} \) based on Exact (E) and Partial (P) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . 71

A.1 \( F_{1} \) based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . . . . . . . . 77

A.2 \( F_{1} \) based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . . . . . . . . 77

A.3 \( F_{1@10} \) based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for TF-IDF on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) . . . . . . . . . . . 78
A.4 F1@10 based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 78

A.5 F1@20 based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 79

A.6 F1@20 based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 79

A.7 F1@10 based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 80

A.8 F1@10 based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 80

A.9 F1@20 based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ................................................................. 81
A.10 $F_1@20$ based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400) ........................................ 81

B.1 $F_1$ based on extraction Exact (E) and Partial (P) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input. .................................................. 83

B.2 $F_1$ based on Exact (eE), Partial (eP) and Sequence (S) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input. .................................................. 83

B.3 $F_1$ based on Semi-Exact (SE) and Semi-Exact (eSE) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input. .................................................. 83
Chapter 1

Introduction

Keyphrase extraction is defined as the process of identifying representative phrases from a text excerpt, that summarize its content, in an automatic manner. Keyphrases play an important role in many language tasks. They can be utilized in the task of document indexing to incorporate semantics and thus allowing for greater performance. Keyphrases can also serve as a special type of summary that indicates the document content that can assist humans in the decision of whether to read a document. Additionally, they can be used in document summarization and as features in document clustering and classification. Keyphrases play an exceptionally important role in the publishing industry as they can be utilized in recommendation systems for books and articles, identification of candidate paper reviewers, suggestion of possible citations to authors, as well as detection of topic trends.

The problem can be tackled with different approaches that fall into two main categories, the supervised and unsupervised approaches. Each algorithm of the aforementioned categories is characterized by several category advantages and drawbacks. According to literature, supervised methods tend to model the problem better resulting in better performance but not without their limitations that mainly originate through the available training data. Unsupervised methods are domain independent and do not need any labeled data for training, which is a huge advantage as labeling data is expensive in time and money and it is heavily affected by the annotator/human subjectivity. It is worth mentioning that many famous companies, such as Microsoft, have their own commercial APIs that provide keyphrase extraction models.

1.1 Contributions

In this thesis, we explore the task of automatic keyphrase extraction on academic articles and the challenges it poses. First, we perform data analysis in multiple popular keyphrase extraction datasets with various information sources. We compare the results with each other in view of finding correlations of the gold keyphrase coverage between the various information sources. This is important as it can pinpoint
the maximum performance we can achieve with an extraction model. Also, this analysis gives direction to select the proper text preparation steps.

We examined a state-of-the-art keyphrase extraction model proposed by Alzaidy et al. (2019). We successfully re-produced their work after performing extensive experiments to find out their model set-up as important steps were omitted in their publication. In our knowledge, there is no public repository with a contemporary implementation of the model tuned for keyphrase extraction. So, we provide one with a tensorflow implementation of the model. In the replication process, we discovered some inconsistencies in their evaluation comparison with other state-of-the-art models and drew some conclusions that question its effectiveness.

We employed several evaluation methods to create a fair evaluation environment in order to compare its performance with other models. Also, we introduce a new method of calculating the $F_1$-score, recall and precision metrics. This method aims to soothe the strictness of the exact matching by addressing the case where a gold keyphrase is a substring of a predicted. For that reason, we call the new method semi-exact matching.

Lastly, we experimented with multiple methods that aim to increase the overall coverage of keyphrases in the scholarly documents in an effort to improve model’s performance. First, we attempted to address the intrinsic imbalanced nature of the problem. Also, we use specific parts of the documents that our data analysis indicated as highly keyphrase dense. We experiment with summaries of the full-text documents, which are generated by state-of-the-art machine learning models. To check the validity of our methods, we employed them to two popular unsupervised models and used three test datasets. By applying our methods to the models, we managed to achieve a maximum increase of 5.85% in the exact match $F_1$-score. Finally, we provide guidelines to choose the most appropriate text excerpt for each of the three machine learning algorithms and discuss potential future work.

1.2 Thesis Overview

In chapter 2, we perform a literature review of the keyphrase extraction task, in which we explore different approaches of handling the problem. The main categories are split into unsupervised and supervised learning. In this work, we focus on the supervised learning category. We deploy the Bi-LSTM-CRF model (Alzaidy et al., 2019), which is a state-of-the-art deep learning approach. Then, we examine and compare the evaluation methods used in the literature and present a list with the most popular datasets.
1.2. Thesis Overview

In chapter 3, we describe the basic architecture of neural networks, known as Simple Feed-forward Networks (FNNs), to set the foundations of understanding the architecture of our main model. Then, we define Recurrent Neural Networks (RNNs) and present how FNNs can be extended to RNNs. We expand in a specific type of RNN, the Long-Short Term Memory (LSTM), and we explain the sequence-to-sequence architecture, which is also known as encoder-decoder. The chapter finishes with the analysis of the Bi-LSTM-CRF model architecture, which is the model used in our experiments.

In chapter 4, we further expand on the selected datasets providing additional information and statistics. We describe the data processing steps, the setup of our experiments and the parameter tuning process. Then, we continue with the definition of our experiments and the evaluation methods we chose. Lastly, we provide the results of our experiments and make a discussion upon them. In the final chapter 5, we recapitulate the key findings of our experiments and we further give direction for future work.
Chapter 2

Keyphrase Extraction

2.1 Unsupervised Learning

Unsupervised methods can be split into several categories, as noted on the review Papagiannopoulou and Tsoumakas, 2020, which are based on (i) statistics (section 2.1.1), (ii) graphs that are built based on statistics (section 2.1.2), (iii) clustering that detects topics (section 2.1.3), e.g. LDA or knowledge graphs, (iv) citation networks or neighbors’ information (section 2.1.4), (v) semantics (section 2.1.5), (vi) embeddings (section 2.1.6), and, (vii) language modeling (section 2.1.7). We can see an outline in Fig. 2.1. According to this review, the most popular sub-category among the unsupervised category is the graph-based. Also, statistic methods are still a strong candidate in large documents, and semantic-based are rising in popularity.

![Unsupervised keyphrase extraction methods](image)

**Figure 2.1:** Unsupervised keyphrase extraction methods. (Papagiannopoulou and Tsoumakas, 2020)

In the papers Hasan and Ng (2010), Hasan and Ng (2014) authors formulated the unsupervised keyphrase extraction procedure as a set of three steps. In the first step, the documents should be cleared from redundant words based on some heuristics. These words could be stop words or a specific selection of part-of-speech (POS). In the second step,
a ranking of the remaining words is created and in the third and final step, keyphrases are formulated by selecting the words or phrases that have a high or top rank.

### 2.1.1 Statistics-based Methods

There are several statistic-based methods that are popular and provide competitive performance. Some of these are Tf-Idf and its alternatives, e.g. integrating the phrase frequency in the formula (Florescu and Caragea, 2017a), KP-Miner that utilizes statistical scores and a special candidate selection schema to create a ranking (El-Beltagy and Rafea, 2009), KeyCluster that calculates the semantic relatedness of the terms in order to group them into clusters and select the most exemplar of each cluster as keyphrases (Liu et al., 2009) and YAKE that combines both statistic and context information (Campos et al., 2018). In the recent paper (Won et al., 2019), researchers show that combining multiple statistical features could result in competitive performance, close enough to the state-of-the-art methods.

### 2.1.2 Graph-based Ranking Methods

In this category of methods, a graph is created where each node corresponds to a word of the document and the edges link words/nodes if they appear next to each other. Then, keyphrases are discovered by basically solving a graph optimization problem. Such algorithms are Google’s PageRank (Brin and Page, 1998), Positional Function (Herrings et al., 2005) and HITS (Kleinberg, 1999). TextRank (Mihalcea and Tarau, 2004) utilizes only nouns and adjectives, selected through Part-of-Speech (POS) tagging, to create an undirected and unweighted graph where nodes are the filtered words and edges appear between words that co-occur in between a specific window. Then, PageRank is applied and the keyphrases are generated when it converges.

SingleRank (Wan and Xiao, 2008) is an extension of TextRank that makes use of co-occurrence statistics as edge weights. RAKE (Rapid Automatic Keyword Extraction) (Rose et al., 2010) is another graph-based method that uses word frequency and word degree to create a ranking for the candidate phrases. In the process of detecting keyphrases, the latest methods SGRank (Danesh et al., 2015) and PositionRank (PR) (Florescu and Caragea, 2017b), feature positional, word co-occurrence, and statistical information that results in a noticeable step up in performance. The rest methods of this category fall into three groups which are methods based on knowledge from similar documents or citation networks, topic-based methods, and graph-based methods that utilize semantics.
2.1.3 Topic-based Methods

Topic-based methods identify topics from documents by commonly performing clustering techniques or Latent Dirichlet Allocation (LDA) (Blei et al., 2003). Those phrases, that are representatives of the majority of the topics covered by a document, are keyphrases.

An algorithm of the subcategory of clustering-based methods is TopicRank (TR) (Bougouin et al., 2013). As a first step, the algorithm clears the document, picks the candidate keyphrases and groups them into topics utilizing hierarchical agglomerative clustering. Then, from those topics, a graph is created where the edges have weights based on a measure calculated from the offset position of the phrases. Finally, a ranking of the topics is created with the aid of TextRank and from each of the top N topics a keyphrase is extracted. An upgraded version of TopicRank is MultipartiteRank (MR) (Boudin, 2018) in which they alter the edge weights to capture position intel in order to give importance to the words that appear at the start of the document.

An algorithm that constitutes the base of many of LDA-based methods is Topical PageRank (TPR) (Liu et al., 2010). First, after selecting only the adjectives and the nouns, the topic distribution is extracted for each word using LDA and then for each document a graph is composed by considering word co-occurrences. Lastly, the key point of the algorithm is that a biased PageRank gets through each individual topic. A variation of TPR, that reduces the cost of running the PageRank for each topic, is the Single Topical PageRank (Single TPR) (Sterckx et al., 2015a) in which PageRank runs once for each document.

An interesting approach proposed by Sterckx et al., 2015b considers a combination of multiple methods of keyphrase extraction where each individual method is trained in a different set of documents and then a compound topical word importance is calculated utilizing each of the methods. Lastly, the word importance constitutes as weight in a Topical PageRank, resulting in enhanced performance. The rationale behind this method is that by training each keyphrase model on a different corpus we manage to capture different word contexts, resulting in different word importance from method to method. That way the method covers a wider range of topics, which is an improvement as generally words are not one-dimensional semantically wise as they are used differently in different types of topics. Some other examples of LDA-based methods are the Salience Rank (Teneva and Cheng, 2017) and the method proposed by El-Kishky et al., 2014 that instead of modeling text corpora with unigrams, it utilizes phrases.
2.1.4 Information based on Similar Documents/Citation Networks

The need to include information from related documents becomes apparent as documents that have correlated topics can assist in the extraction of keyphrases. Previously mentioned graph-based techniques consider that documents are uncorrelated to each other and thus, they utilize only information from the current examined document. A method that utilizes intel from neighboring documents is ExpandRank (Wan and Xiao, 2008), which is based on SingleRank. Another insightful information source stems from citation networks. Specifically, when a document is cited from another, the citation is accompanied with sort text excerpts that describe content from within the original document. An example of such a method is CiteTextRank (Gollapalli and Caragea, 2014a) which draws knowledge from such text excerpts.

2.1.5 Graph-based Methods with Semantics

The aforementioned unsupervised methods suffer from many defects. Topic-based methods suffer from abstractive and vague topics. The window size limits co-occurrence-based methods, and thus the information that can be derived is negatively affected. This is because semantically similar words might not have any edges connecting them in the graph-of-words, if they do not appear inside the same window. On the other hand, statistic-based methods face with the problem of information overflow where the semantical meaning of a word might be distorted from alien documents which is negative when considering a specific document. The solution to these obstacles is to integrate semantics in the keyphrase extraction process

A method to integrate semantics into graph-based solutions is to utilize knowledge graphs. Shi et al., 2017 suggested a method that incorporates knowledge graphs. The method performs clustering to group semantically equivalent words and named entities into clusters. Another related work that endeavors to allocate semantic meaning to words, is the WikiRank suggested by Yu and Ng, 2018.

Albeit the methods that use knowledge graphs to add semantical meaning to words provided good improvement, there is a limit to the information drawn from semantic relations. Another method to incorporate semantics is by utilizing pretrained word embeddings. Wang et al., 2014 suggested a graph-based ranking model that includes semantic information by building it based on distributed word representations. Another work of Wang et al., 2015 suggested a better version of their previous work that incorporates a personalized weighted PageRank with pre-trained word embeddings and improved edge weights. An interesting observation of the latter work is the utilization of domain-specific
word embeddings to possibly achieve better performance. Based on this observation, (Mahata et al., 2018) proposed Key2Vec, that utilizes a topic-weighted PageRank algorithm (Langville and Meyer, 2004) to rank the candidate keyphrases of a document that are represented by domain-specific embeddings. Another work on embeddings is the Fasttext (Bojanowski et al., 2017) which is developed for training embeddings on phrases that are constituted by multiple words and it is based on a collection of scientific abstracts.

2.1.6 Keyphrase Extraction based on Embeddings

In literature, there are many ways to represent words with numerical values. An option is to use methods that are based on co-occurrence matrix, such as Latent Dirichlet Allocation (LDA) (Blei et al., 2003) and Latent Semantic Analysis (LSA) (Deerwester et al., 1990). Another option is word embeddings which have risen a lot in popularity as they usually tend to outperform other representations in the task of keyphrase extraction, possibly because they can involve word semantics. The first technique of this kind proposed is word2vec by Mikolov et al., 2013, who introduced the Continuous Bag-of-Words (CBOW) and the Continuous Skip-gram model. Following up, another similar method is GloVe (Global Vectors) (Pennington et al., 2014) and also there are phrase embeddings, such as Doc2Vec (Lau and Baldwin, 2016) and Sent2vec (Pagliardini et al., 2017). A method that utilizes embeddings to extract keyphrases is EmbedRank (Korhonen and Titov, 2018). First, it extracts candidate keyphrases by consulting a POS schema and then it represents the document as well as the candidate keyphrases in the same high-dimensional vector space by utilizing phrase embeddings, Doc2vec or Sent2vec. Keyphrases are extracted based on a ranking that occurs by considering the cosine similarity between the embeddings of the candidate phrase and the document.

Another method using embeddings is the one proposed by Papagiannopoulou and Tsoumakas, 2018 named as Reference Vector Algorithm (RVA). The novelty of this approach lies in the way the GloVe embedding is trained and specifically vectors are produced by training only at the examined document. It is worth noting that the GloVe embedding can be deemed as an alternate method of the graph-based approaches as both methods are based on textual statistics of word-to-word co-occurrence. Then, the document vector is calculated as the mean of the word vectors in the title and the abstract of the document, while candidate keyphrases are extracted by the same sources as well. To identify the keyphrases, a ranking is produced based on the cosine similarity between the document vector and each candidate keyphrase.
vector. The more similar the candidate keyphrase vector is to the document vector, the more representative the candidate keyphrase is for the document.

2.1.7 Language Model-based Methods

A language model sets the probability that any sequence of words has to occur. Language modeling is a valuable asset in natural language processing tasks (Chen and Goodman, 1999). In the work of Tomokiyo and Hurst, 2003, they proposed a method that utilizes a language model to detect keyphrases. The method creates two language models, a unigram and an n-gram on a foreground corpus (target document) and a background corpus (document set). Then, using the Kullback-Leibler divergence, that can calculate the loss between two language models, they calculate two scores called phraseness and informativeness. Both of those scores are summed and based on this final score, a ranking of the candidate keyphrases is produced.

2.2 Supervised Learning

Supervised methods can be divided into two main categories, which are traditional learning and deep learning. A taxonomy of supervised methods is shown in Fig. 2.2. A concise review of the sub-categories presented in Fig. 2.2 follows along with examples of popular algorithms in the following sections. Additionally, the types of existing features will be examined.

![Figure 2.2: Supervised keyphrase extraction methods.](Papagiannopoulou and Tsoumakas, 2020)

2.2.1 Traditional Supervised Methods

Traditional supervised methods require a training set that includes both the text of the documents as well as the actual keyphrases annotated by humans. This is a binary classification problem where the positive
class corresponds to a word/phrase being a keyphrase and the negative class being a non-keyphrase.

### Keyphrase Extraction as Binary Classification Task

One of the earliest approaches is KEA (Witten et al., 2005), in which a Naive Bayes model is composed by using as features Tf-Idf to represent the candidate keyphrases, as well as the position that the candidate keyphrase appears for the first time. Another system suggested by Hulth (2003) makes use of linguistic knowledge by including four features into a rule induction system with bagging. Those features are the part-of-speech (POS) of the keyphrase, the position of the first occurrence, the frequency of the candidate keyphrase within the document at hand, and, the frequency on the whole collection. Maui introduced by Medelyan et al. (2009), is another extension of KEA, suggesting a set of features that are utilized by a bagged decision tree model to produce predictions. The set of features includes Tf-Idf, the first occurrence, keyphraseness which is the frequency of a candidate keyphrase that appears as a tag in the training set, Wikipedia-based keyphraseness which is the probability that a candidate keyphrase is a link in Wikipedia, the spread of a phrase which is the distance between first and last occurrence of candidate keyphrase, as well as additional statistical features based on an external language source, Wikipedia.

In their paper, Nguyen and Luong (2010) suggest WINGNUS, a method that picks out candidate keyphrases using regular expressions and then utilizes various features with Naive Bayes. They concluded that the best features are Tf-Idf, the first occurrence of the phrase, the length and the term frequency of the phrase substrings. Most importantly, in this research, they observed that most of the keyphrases are located in the full-text or in a text excerpt that includes headers, title, abstract, introduction, related work, conclusion and the first sentence of each paragraph.

CeKE, a method introduced by Caragea et al. (2014), uses Naive Bayes to solve the keyphrase extraction problem as a binary classification problem. It introduces a new set of features that are based on citation information, which are the Tf-Idf score of each phrase calculated from citations and boolean features that their value depends on whether the phrase exists in any citation. Also, they use features that were proposed in former papers, such as phrase POS tags, Tf-Idf, first position and relative position. Finally, they expand on existing features, in particular, a boolean feature that its value stems from the comparison between Tf-Idf score and a certain threshold and a boolean feature that its value depends on the distance between the first time a phrase occurs and the start of a target paper and how it compares to a threshold.
Wang and Li (2017) introduced a method named PCU-ICL, which ranks candidate keyphrases with the aid of linear models, random forest and an ensemble of unsupervised models. The method includes most of the previously mentioned features and, in addition, a boolean feature that signifies whether the IEEE taxonomy list includes the phrase, Wikipedia-based Idf, GloVe embedding of the phrase, distance between the phrase and the citation and features generated from unsupervised methods. SurfKE, introduced by Huang et al. (2019), is a method that makes predictions utilizing the Gaussian Naive Bayes model. The method extracts feature representations for a single document from the graph of words of the document itself.

Keyphrase Extraction as Learning to Rank Task

While classification models make hard decisions, this kind of methods follow a different approach as they learn how to rank (ranking function) the candidate keyphrases based on their attributed score of being a keyphrase. One of the most popular ranking models is Ranking SVM (Jiang et al., 2009) which transforms the training data, from pairs of feature vectors and their respective ranks to a set of ordered phrase pairs, to train an SVM classifier. Subsequently, the problem is transformed into an optimization problem where the weights are the optimal solution. Another, more advanced method named MIKE is suggested by Zhang et al. (2017). First, the document at hand is cleaned from redundant words and then a graph of words is composed utilizing co-occurrence relations of the words. Additional features are combined into a parametric random-walk model and then by composing a loss function, which is gradient descent optimized, the model computes the parameters, and ranks each candidate word based on the produced score. Lastly, the score of phrases or consecutive words is computed as the sum of the scores of the words that a phrase is consisted of.

Keyphrase Extraction Using Supervision

TopicCoRank, proposed by Bougouin et al. (2016), is an extension of the unsupervised method TopicRank. The method allows the allocation of domain-specific keyphrases, even though they might not appear in the document. This is achieved by integrating another graph whose domain is the same as the primary topic graph. Yang et al. (2018) introduced a method that extracts event-related keyphrases. To achieve that, they obtain a set of candidate keyphrases connected to specific events. They utilize a Task-oriented Latent Dirichlet Allocation model (ToLDA), where events correspond to topics from their topic model. Lastly, they acquire the synonyms of the candidate keyphrases by deploying the PMI-IR algorithm (Turney, 2001).
Sequential Labeling

Another interesting approach is proposed by Gollapalli et al. (2017). In this work, they consider the problem of keyphrase extraction as a sequence tagging task utilizing Conditional Random Fields (CRFs). The selected features are based on structural, linguistic, and orthographic information. Moreover, they attempt to introduce domain information through posterior regularization and feature-labeling.

2.2.2 Types of Supervised Learning features

Supervised keyphrase extraction methods utilize various types of features to learn the word importance of a document. These features fall into several categories based on the information they capture, which are statistical, positional, linguistic, context and external knowledge. Additionally, stacking is a popular method that allows for a combination of different models and usually results in performance improvements. A more detailed report of the feature categories and how different works utilize them can be found in the review published by Papagiannopoulou and Tsoumakas (2020).

**Statistical Features** This category is very popular in the community of keyphrase extraction. It includes features such as the Tf-Idf score and its components, the term frequency (Tf) and the inverse document frequency (Idf), as well as statistical scores, e.g., phrase entropy.

**Positional Features** This type of features refer to the actual position of words in the document. The locality of a word is related to its importance, especially words that appear in the title or at the beginning of a document tend to be more valuable. The structural information of a document, e.g., defining the sentence boundaries, can be exceptionally advantageous for various methods (e.g., sequential labeling approaches).

**Linguistic Features** Some well known linguistic features are the POS tag sequence as well as morphological features, such as the acronym status and the suffix sequence. Additional popular features are the stem of the words, the length of the phrases and boolean features that capture word information, e.g., if the word has capitalized letters, contains punctuation and is a stopword.

**Context Features** This kind of features tend to be very beneficial for the task of keyphrase extraction as the context information can help capture the meaning of the words. Generally, the meaning of a word can be guessed depending on its context. There are several methods to capture context information, e.g., provide next or previous words of a word/phrase. Additional ways to attribute context to words is by using positional information, such as the relative position of a phrase in a document, and by deploying embeddings generated from neural networks and graphs.
External Knowledge There are multiple features that utilize information from external sources. The most notable features are statistical, the keyphraseness, and boolean features that capture knowledge about the existence of document words in an external source, e.g., Wikipedia.

Stacking This method ensembles multiple models, either supervised, unsupervised or combination, into one stacked model. The main objective is to integrate different models into one by using the individual model predictions as features into another final model. In the literature, stacking methods tend to outperform the individual models used to compose the stacking model.

2.2.3 Deep Learning

According to literature, deep learning techniques tend to outperform all other methods, with the top and most consistent performer being CorrRNN proposed by Riloff et al. (2018). There are several ways to approach the problem of keyphrase extraction with deep learning. There are techniques that convert the problem into sequence labeling, while others generate the keyphrases. The main shortcoming of the sequence labeling approach is that it cannot extract keyphrases that are not present in the given text. Generative approaches solve that problem, but even themselves have limitations, e.g., they lack a variety of generated keyphrases and suffer from duplications. One of the earliest deep learning attempts (Zhang et al., 2016) utilizes a two hidden layer recurrent neural network (RNN) to extract keyphrases from tweets. The first layer captures information about the keywords, while the second utilizes that information with a sequence labeling approach to make predictions. Chan et al. (2019) proposed a keyphrase generation approach that utilizes reinforcement learning to address the problem of the existing generative methods, which is the generation of few keyphrases.

Another popular method is seq2seq, introduced by Meng et al. (2017). This method generates keyphrases through a framework that incorporates both an encoder and a decoder. A deep learning model (CopyRNN) is utilized to capture text semantics. In particular, an RNN Encoder-Decoder framework takes as input the words of the document and their keyphrase status and learns the mapping from the input sequence to the output sequence. The encoder converts the document into a hidden layer representation and then the decoder utilizes this information to generate keyphrases. The main flaws of this method are that it generates duplicate keyphrases (duplication issue), as well as, it might not produce major topics of the document (coverage issue), which is an effect of the correlation ignorance between the target keyphrases. CorrRNN was suggested by Riloff et al. (2018) as an improved version of the sequence-to-sequence architecture. This approach
manages to capture the correlation between the target keyphrases and thus to solve both of the aforementioned problems. In order to cover all the topics of the document, a coverage mechanism (Tu et al., 2016) is integrated that keeps track of which document segments have been covered by the already generated keyphrases. Lastly, to ensure the uniqueness between the generated keyphrases, a review mechanism is employed to calculate the correlation between the keyphrases that have already been generated and the keyphrases that are going to be generated.

The latter two methods are dependent on large amounts of labeled data for training, which is very difficult and expensive to acquire. To cease these problems, Ye and Wang (2018) proposed a semi-supervised approach that uses deep learning techniques (LSTMs) trained on both labeled and unlabeled data to generate keyphrases. Specifically, they suggest two methods to exploit unlabelled data. The first method annotates unlabelled data by utilizing 2 unsupervised keyphrase extraction methods, TfIdf and TextRank (Mihalcea and Tarau, 2004). The final assigned keyphrases stem from the union of the two keyphrase sets, produced by running each unsupervised method separately. Lastly, they combine the synthetic dataset with an already labeled one, and, they feed the resulting data to a sequence-to-sequence model. The second method utilizes a framework that integrates the basic keyphrase generation task with a supplementary task, which is the title generation that was studied by Rush et al. (2015) as a summarization problem. The two proposed methods have the same encoder but different decoder. Specifically, on the first epoch, the initial parameter values are calculated on the synthetic dataset created by the supplementary task, which is annotated with titles as keyphrases. For the next 3 epochs, the model is trained on the final dataset produced by the first method. This way of training is continued until the model of the first task converges.

Wang et al. (2018) advance the performance in the field of unlabeled data by introducing a Topic-based Adversarial Neural Network (TANN), which takes advantage of both labeled and unlabeled data instances by utilizing knowledge from a resource with related domain information (cross-domain perspective). The TANN is composed by four stages, a topic-based encoder, a domain discriminator, a target bidirectional decoder, and a keyphrase tagger. The topic-based encoder consists of a Bidirectional Long Short-Term Memory (Bi-LSTM) Network that encodes sequential knowledge. Then, weights are attributed to the words to highlight the most important ones as those that are the most relevant to the documents’ topics. This is achieved by a topic correlation mechanism that considers the correlation between the vectors of the document words and the vector of the topic. The domain discriminator uses adversarial learning to ensure that the encoder extracted features are domain-independent. This is important
Chapter 2. Keyphrase Extraction

as there are phrases, that, independently of the domain, tend to be followed by keyphrases, such as “we introduce/propose”. The target bidirectional decoder utilizes bidirectional reconstruction loss in the frame of a bidirectional decoder that is applied in the target domain, in order to counterpoise the effects of the adversarial procedure that aim to eradicate the domain-specific information in the target domain. Lastly, the keyphrase tagger takes as input the output of the topic-based encoder and predicts whether a word of the examined document is a keyphrase.

Sahrawat et al. (2019) is another work that utilized the Bi-LSTM-CRF architecture on the problem of keyphrase extraction on scholarly articles. They conclude that, in the frame of keyphrase extraction, the use of contextualized embeddings (e.g., BERT) as well as genre-specific contextualized embeddings (e.g., SciBERT) is preferred, instead of fixed word embeddings (e.g., Glove) as they observed improved results. SciBERT (Beltagy et al., 2019), utilizes the plain BERT model to build domain-specific language models.

A major problem of supervised and unsupervised methods is that most of them cannot produce phrases that do not appear in the text. This problem is tackled by Çano and Bojar (2019) that consider keyphrases as an abstractive summary of the document title and abstract. They deployed popular text summarization neural architectures such as MERGE and INJECT proposed by Tanti et al. (2017). Also, they included ABS suggested by Rush et al. (2015) and Pointer-Generator network (POINTCOV) introduced in See et al. (2017). The models are expected to learn and paraphrase the text, while also introducing new words that do not appear in the text. Experiments concluded that the selected models could not achieve better performance than even the simplest supervised and unsupervised models. Lastly, they suggest that utilizing more advanced summarization techniques and optimized hyperparameter tuning might change the final result.

Chen et al. (2019) stress the importance of the document title in the keyphrase extraction process by exploring a method that handles the title individually. They propose the Title-Guided Network (TG-Net) that follows the encoder-decoder architecture. They incorporate two title-based new features, which are the use of the title as a query-like input and the use of an encoder to capture information between the title and the document words. The title-guided encoder comprises three layers, a sequence encoding, a matching, and a merging layer. The sequence encoding layer learns contextual representations from the input of the context and the title, individually. Then, for each representative word of the essential excerpts of the context, the matching layer finds the relevant title information. The merging layer produces the final title-guided context representation by integrating the aggregated title information, created by the matching layer, into each context word. The
2.2. Supervised Learning

evaluation process showed that the proposed models achieve better performance than state-of-the-art approaches, despite the title size. Generally, cluster-related documents seem to share a lot in common, as word context, usability and meaning. Based on this observation, Zahedi et al. (2019) propose a method that tries to introduce the keyphrases missing from the available text. They name those keyphrases as unseen keyphrases. First, the model tries to introduce the unseen keyphrases by capturing text concepts utilizing the text itself and then adds them in the hidden layers of the neural network during training. Then, the obtained information is used to make a sequence-to-sequence encoder-decoder RNN model to predict them. To introduce the unseen keyphrases, they utilize a deep clustering. The method seems to capture the documents’ basic concepts and efficiently introduce unseen keyphrases, as the evaluation process showed a slight improvement of the DeepUnseen over the next best performing model, which is CopyRNN.

Mu et al. (2020) introduce a novel feature representation called span-based. Specifically, span-based features are the relations between two words in a sentence for any set of words. First, they utilize BERT to isolate token features in order to feed them into a Bi-LSTM to create the span-based representations. Lastly, a Bi-LSTM is utilized to capture the interaction of phrases and create the phrases’ representation. As a final step to extract keyphrases, they approach the problem with two methods, as a binary classification (i.e., KP or Non-KP) and as a ranking problem (i.e., numbers/rankings that denote how probable a word is to be a keyphrase). For both methods, they deploy the same fully connected feed-forward network, but with different output layer to properly fit the needs of each approach.

Zhang et al. (2020) suggest a target-center based LSTM (TC-LSTM) that creates word representations by encoding information about the context located before and after the selected word. They also introduce a self-attention mechanism that allows the TC-LSTM model to capture the importance of words that aids in distinguishing the keyphrases more effectively. The attention mechanism is similar to Bi-LSTM, as it combines the outputs of a forward and a backward LSTM with a softmax function to capture information before and after each word, respectively. In addition, a method that annotates large amounts of unlabeled data is suggested, in order to utilize more data in the training process. The method produces a labeled dataset with the help of a ranking process based on the way they created the original/labeled training dataset. Finally, the training process consists of two steps. First, the model is pre-trained on the artificial dataset created by the suggested method. Then, the model is re-trained with the original/labeled dataset.

In their work, Basaldella et al. (2018) utilized a Bi-LSTM RNN in
order to capture the context of the words that appear before and after the examined word. They follow a pretty standard approach, where the whole text is split into sentences and thereafter into words that are converted into word embeddings and are fed into the Bi-LSTM RNN model. One of the latest and most promising works is the Bi-LSTM-CRF model on scholarly documents, suggested by Alzaidy et al. (2019). This was the paper that motivated the present work. The Bi-LSTM layer is utilized for encoding the sequence of the text data while incorporating the text semantics. Then, the Bi-LSTM output is fed into a Conditional Random Field (CRF) layer that encodes the dependencies of the whole word sequence in the output (labels). Finally, by utilizing those label dependencies, the CRF layer produces the probability distribution of the tag sequence. The labels can be one of the following values, either keyphrase (KP, positive class) or non-keyphrase (Non-KP, negative class).

Patel and Caragea (2019) investigate the keyphrase extraction problem as sequence labeling with Conditional Random Fields (CRF). A CRF model utilizes a transition parameter matrix that stores the transition probabilities from one label to the neighboring label to learn the label dependencies. The main focus of the study is to use as features the combination of word embeddings with document-specific features from the literature. Results from the conducted testing analysis show that the combination of word embeddings and document specific features provide top tier performance when compared to other state-of-the-art approaches. Additionally, they conclude that a simple and easy way to configure a strong model is to feed word embeddings directly to a CRF model as they result in increased performance. Also, they studied the combination of a Bi-LSTM layer with a CRF and found that the effect of the non-linearity of the Bi-LSTM results in a representation that it might not be useful for the CRF layer, resulting in lesser performance. In other words, Bi-LSTM encodes information of a sentence both forward and backward, while the CRF layer expects the sequence to be by default forward. Although, adding document features alongside word embeddings might confuse the Bi-LSTM model. Bi-LSTM model seems to perform better when the keyphrases and the text of the document tend to be longer as it manages to successfully capture long-term information. The size of the dataset also plays a role as neural networks tend to benefit from large size of data.

Another model that combines deep learning and a CRF layer is the Multi-Level Memory network withs CRFs (MLM-CRF) that was proposed by Zhou et al. (2020) and it is basically a GRU-CRF model, which uses a GRU layer to capture long term information.

Zhu et al. (2020) propose a Bi-LSTM-CRF model trained on data produced by a self-training method (SL-BLSTM-CRF-H) that is introduced to deal with the problem of expensive data annotation. The
self-training method chooses unlabeled data instances based on the model’s confidence and adds them into the training set. A heuristic function is deployed to the output of the neural network based on the observation that words with capital first letter have high chances to be a keyphrase. Despite the fact that they do not validate the model’s performance on many well-known datasets, the performance seems promising.

2.3 Evaluation

Evaluating methods are a necessity as they allow us to compare the performance of each model, select the proper one and have an indication of how well the model is able to solve the problem. However, evaluating the performance of a method is not always an easy task, especially in the case of keyphrase extraction. This is due to the complexity that semantics introduce to the problem and thus making the metric calculation unreliable. The most popular evaluation methods adopted by the scientific keyphrase extraction community are the following:

1. Precision, Recall and $F_1$-score:

   \[
   \text{Precision} = \frac{\text{number of correctly matched}}{\text{total number of extracted}} = \frac{TP}{TP + FP} \tag{2.1}
   \]

   \[
   \text{Recall} = \frac{\text{number of correctly matched}}{\text{total number of assigned}} = \frac{TP}{TP + FN} \tag{2.2}
   \]

   \[
   F_1 - \text{score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.3}
   \]

   where $TP$ is the number of true positives, $FP$ the number of false positives and $FN$ the number of false negatives.

2. Ranking quality measures are utilized to evaluate models that produce rankings of extracted phrases. This category of measures considers the order that the keyphrases are extracted by the models. The most popular metrics in this category are:

   - Mean Reciprocal Rank (MRR) (Voorhees et al., 1999):

     \[
     \text{MRR} = \frac{1}{|D|} \sum_{d \in D} \frac{1}{\text{rank}_d} \tag{2.4}
     \]

     where $d$ is a single document of the whole document collection $D$ and $\text{rank}_d$ is the rank of the first correct keyphrase of the document $d$. 
• Mean Average Precision (MAP) (Liu and Özsu, 2009) considers the order of the returned keyphrases. First, the average precision (AP) is computed:

\[ AP = \frac{\sum_{r=1}^{\mid L \mid} P(r) \cdot rel(r)}{\mid L_R \mid} \]  

(2.5)

where \( \mid L \mid \) is the cardinality of the returned keyphrases, \( P(r) \) is the precision that occurs from the first \( r \) returned keyphrases, \( rel(r) \) is 1 if the \( r^{th} \) returned keyphrase belongs in the golden set and 0 otherwise, and, \( \mid L_R \mid \) is the cardinality of relevant keyphrases.

Finally, the Mean Average Precision (MAP) is the average AP of \( n \) documents.

\[ MAP = \frac{1}{n} \cdot \sum_{i=1}^{n} AP_i \]  

(2.6)

where \( AP_i \) is the average precision of the extracted keyphrases from a specific document.

3. Binary preference measure (Bpref) (Buckley and Voorhees, 2004) is basically a summary of the cardinality of all the candidate phrases that are actually keyphrases and appear before the candidate phrases that are non-keyphrases.

\[ Bpref = \frac{1}{R} \sum_{r \in R} 1 - \frac{\mid n \text{ ranked higher than } r \mid}{M} \]  

(2.7)

where, for a specific document, \( R \) is the cardinality of candidate phrases that are actually keyphrases, \( M \) is the cardinality of all extracted keyphrases, \( r \) is a phrase from the set of correctly predicted keyphrases and \( n \) is a phrase of the set of the predicted keyphrases that are actually not a keyphrase.

4. Average of Correctly Extracted Keyphrases - (ACEK) is the average number of the correctly extracted keyphrases. This was the first evaluation metric that was utilized in the task of keyphrase extraction, but Precision, Recall and \( F_1 - \text{score} \) are preferred at present, as they provide a more complete outlook of the method.

The evaluation process for the problem of keyphrase extraction should be dealt with additional caution as it has more depth than simply calculating the standard evaluating metrics used in the relevant literature. Papagiannopoulou and Tsoumakas (2020) created a ranking of the evaluation methods based on the number of appearances in the reviewed papers, which can be seen in the Fig. 2.3.
2.3. Evaluation

There are several metrics used in the literature, but the most adopted ones are the use of $F_1$-score, Accuracy, Precision, and Recall, as noted in the aforementioned review. To calculate these metrics, someone can employ one of the three following techniques or a combination of them as used in the literature.

1. **Exact match evaluation** utilizes exact string matching to compare the predicted set with the golden keyphrase set. Usually, the words are stemmed as a pre-processing step to treat word comparisons more fairly. This method is strict and it generally results in lower evaluation scores than its alternatives.

2. **Partial match evaluation** was introduced by Rousseau and Vazirgiannis (2015) as an alternative method to the exact match in order to compute Precision, Recall and $F_1$-score. This method approaches the evaluation process in a much looser way, as it compares the set of words resulted by joining each keyphrase of the predicted set with the set of words of the golden set, produced in the same way. The words are stemmed before the comparison. The downsides of this method are that it cannot assess the syntactic soundness of the phrases and it cannot cope with overlapping and over-generation problems that may characterize the predicted keyphrase set.

3. **Manual evaluation** involves a team of experts that evaluate which predicted keyphrases are relative and which not. The drawbacks of manual evaluation are that the final score is heavily affected by human subjectivity (Zesch and Gurevych, 2009) and at the same time it is expensive and time-consuming.
Chapter 2. Keyphrase Extraction

As we can see from Fig. 2.3, the most popular evaluation method is exact string matching. This method is too strict as it does not account for semantic similarities between words. For example, it equally punishes methods for predictions that have no relation to an actual keyphrase and predictions that are a subset or superset of an actual keyphrase. Suppose we have a keyphrase golden set that consists of the phrases “approximate search” and “similarity search” and the extracted keyphrase is “approximate similarity search”. Despite the fact that this prediction is correct, the exact match approach will assess it as wrong.

This is a major obstacle in the keyphrase extraction problem as it might restrict the actual performance of some methods and boost the performance of others, incorrectly. Hence, the need for a fair evaluation method that handles those inefficiencies is apparent. To deal with those inconsistencies, each research should specifically clarify the exact evaluation approach the writers chose in order to promote reproducibility and fair comparison, something that is yet to be adopted from all researchers in the field, but it slowly becomes more popular. Furthermore, the metrics Precision, Recall and $F_1$-score at the top N phrases, where N = 5, 10, 15, 20, are usually better than the ranking quality measures, as generally the success rate on the top phrases is more important than the order of the returned keyphrases.

Papagiannopoulou and Tsoumakas (2020) performed a comparative empirical analysis of the problem and ended up with several proposals. They sliced the evaluation problem into three methods, exact match, partial match, and the average of the aforementioned methods, and compared them with the manual evaluation. They found that the average of exact and partial match scores achieves the closest score to the manual evaluation than any other popular method.

2.3.1 Exact vs Partial Matching

As noted in the review of Papagiannopoulou and Tsoumakas (2020), the most popular evaluation method in the literature is the exact matching of the predicted keyphrase set on the golden set. Despite its popularity, exact matching misses the predicted keyphrases that are different but semantically equivalent to those on the golden set (Rousseau and Vazirgiannis, 2015; Wang et al., 2015). In contrast, partial matching favors models that their predicted keyphrases contain words that appear in the gold set, even though the predicted keyphrases themselves might not be suitable. Additionally, Papagiannopoulou and Tsoumakas (2020) compared the difference (mean squared error - MSE) between partial/exact matching and manual evaluation. They underline that the evaluation approach closest to the manual is the average of partial and exact matching. Partial matching finished in second place, while
exact matching ends up in the last place, with respect to the their corresponding MSE values. Lastly, they emphasize that the best evaluation method should be automatic and consider word semantics.

An attempt to this direction is made by Papagiannopoulou and Tsoumakas (2018) in which they propose an interesting evaluation method that aims to handle the subjectivity issues. They suggest the use of embeddings in the evaluation process to capture the semantics of the words in view of deeming extracted candidates as actual keyphrases, if they are expressed with different but synonym words. The similarity score of the phrases is given by the cosine similarity of the average of the word vectors of candidate keyphrases with the average vectors of the gold keyphrase set. That way, the semantics of the words are preserved and it can actually result in a more objective evaluation. As mentioned before, currently, the most common evaluation method is the exact string matching. Subsequently, the need for more reliable evaluation methods that integrate semantics is apparent.

2.3.2 Subjectivity and Class Imbalance

A major problem that impacts the supervised keyphrase extraction approaches is the problem of class imbalance. This is due to the fact that phrases that are not stated as keyphrases are deemed as negative examples in the training process. The root behind this issue stems from the way that authors pick the keyphrases. They might choose a phrase depending on the popularity of terms in their field or depending on the way they grasp the importance of the terms based on how well they represent their work, etc. Hence, the real problem lies in the human subjectivity, which can be partly solved by incorporating keyphrases from different annotators.

Sterckx et al. (2016) suggest that phrases that are not annotated as keyphrases are not necessarily negative instances. To solve this problem, they transformed the conventional supervised keyphrase extraction process to Positive Unlabeled Learning proposed by Elkan and Noto (2008). Basically, they attempt to replicate the annotation process by multiple annotators and insert uncertainty in the negative instances. This is achieved by attributing weights to training instances. For the negative class instances, a classifier is trained on data annotated by a single person and the predictions on the unlabeled phrases serve as weights on another classifier. The latter is trained on the same data but with the addition of the weights to predict the final labels or the ranking of the phrases. For the positive class instances, the assigned weight equals 1, while the instances of the negative class are replicated so that one copy is considered negative and the other positive. Lastly, Sterckx et al. (2018) suggest that new datasets should be formed for
evaluation of keyphrase extraction methods that incorporate multiple sources and are annotated by many different annotators.

In the review of Hasan and Ng (2014), authors dive into an analysis of the errors made by keyphrase models. Such errors include (i) evaluation errors (the extracted keyphrase is evaluated as non-keyphrase while it is semantically equivalent to a gold one), (ii) redundancy errors (multiple keyphrases that are semantically equivalent), (iii) infrequency errors (keyphrases that are rare in the document and thus not detected), and (iv) overgeneration errors (detecting frequent phrases that are actual keyphrases while also detecting additional phrases that contain frequent words that are not keyphrases). A previous review of the beforementioned authors (Hasan and Ng, 2010) draws some conclusions, which are: (i) evaluation should be done on several datasets, (ii) performance is heavily affected by post-processing, e.g., the format of phrases, and (iii) Tf-Idf is a strong baseline.

### 2.3.3 Datasets

<table>
<thead>
<tr>
<th>Type</th>
<th>Dataset</th>
<th>Created By</th>
<th>Docs</th>
<th>Language</th>
<th>Annotation Type</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-text</td>
<td>NUS</td>
<td>Nguyen and Kan (2007)</td>
<td>211</td>
<td>English</td>
<td>Authors/Readers</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Krupivin (ACM)</td>
<td>Krupivin et al. (2009)</td>
<td>2,304</td>
<td>English</td>
<td>Authors</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Semeval2010</td>
<td>Kim et al. (2010)</td>
<td>244</td>
<td>English</td>
<td>Authors/Readers</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Citeulike-180</td>
<td>Mediyan et al. (2009)</td>
<td>180</td>
<td>English</td>
<td>Readers</td>
<td>1</td>
</tr>
<tr>
<td>Paper</td>
<td>Inspec</td>
<td>Hallih (2003)</td>
<td>2,000</td>
<td>English</td>
<td>Indexers</td>
<td>21</td>
</tr>
<tr>
<td>Abstracts</td>
<td>KDDB</td>
<td>Gollapalli and Caragea (2014a)</td>
<td>755</td>
<td>English</td>
<td>Authors</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>KP20K</td>
<td>Meng et al. (2017)</td>
<td>590,089</td>
<td>English</td>
<td>Authors</td>
<td>3</td>
</tr>
<tr>
<td>News</td>
<td>WWW</td>
<td>Gollapalli and Caragea (2014a)</td>
<td>1,330</td>
<td>English</td>
<td>Authors</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>50N-KPCrowd</td>
<td>Marujo et al. (2013b)</td>
<td>500</td>
<td>English</td>
<td>Readers</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>110-PT-BN-KP</td>
<td>Marujo et al. (2013a)</td>
<td>110</td>
<td>Portuguese</td>
<td>Readers</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Wikinews</td>
<td>Bougouin et al. (2013)</td>
<td>100</td>
<td>French</td>
<td>Readers</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 2.1:** Well-known datasets grouped by the type of available text excerpts (Type). Papagiannopoulou and Tsoumakas, 2020

Various datasets are utilized in the problem of keyphrase extraction and can be categorized into several categories based on their type of available text excerpts. In the Table 2.1, well-known datasets are grouped by the most popular types of available text excerpts (Type), which are full-text scientific publications, paper abstracts and news documents. The table provides information for each dataset about the (i) total number of the documents (Docs), (ii) text language (Language), (iii) annotators (Annotation Type), which refers to the way that the keyphrases are annotated and can be either author-assigned, reader-assigned or professional-indexers-assigned, and iv) the number of times that the dataset is used (Freq.) based on the total reviewed papers in Papagiannopoulou and Tsoumakas (2020). In addition, there are many other datasets proposed that are either private or used only for evaluation or they are not widely used. Evaluation datasets can
be found on the Github repositories: keyword-extraction-datasets\textsuperscript{1}, AutomaticKeyphraseExtraction\textsuperscript{2} and on the website of Sujatha Das Gollapalli\textsuperscript{3}.

Lastly, there are multiple methods to create such datasets, but some suggestions are given by Medelyan et al. (2009) and Sterckx et al. (2018). A collection of keyphrase extraction datasets was composed by Sterckx et al. (2018) and is available upon request for research purposes. Various people annotate these datasets that are based on on-line news/sports, lifestyle magazines and newspaper articles.

In the present chapter, we reviewed the history of the keyphrase extraction task from the earlier years through contemporary publications, with a focus on deep learning work. We provided a brief presentation on the majority of the most impactful works throughout the keyphrase extraction problem lifetime, covering different domains and approaches of the fields of the unsupervised and supervised learning. We also addressed the major restrictions in the domain, presented a collection of popular datasets, and analyzed the most adopted evaluation methods. In the next chapter, we will dive more into neural network architectures, starting from the base models to grasp a basic understanding and finishing with state-of-the-art techniques. We will further expand on the selected model (Bi-LSTM-CRF) for our experiments and the theory behind its success.

\textsuperscript{1}\url{https://github.com/zelandiya/keyword-extraction-datasets}
\textsuperscript{2}\url{https://github.com/snkim/AutomaticKeyphraseExtraction}
\textsuperscript{3}\url{https://sites.google.com/site/sujathadas/home/datasets}
Chapter 3

Neural Networks

On the present thesis, inspired by Alzaidy et al. (2019), we study the effectiveness of the Bi-LSTM-CRF on the problem of keyphrase extraction on scholarly documents. We further extend that work with new methods that aim to further increase both the overall performance and the range of the predicted keyphrases. Before proceeding further, we will present the basic notion behind artificial networks and then we will analyze the Bi-LSTM-CRF model and its encoder-decoder function. On the following presentation of the neural networks, we will emphasize on Natural Language Processing (NLP) tasks.

3.1 Deep Neural Networks

The name of Artificial Neural networks (ANNs) stems from their purpose of trying to simulate the function of a human brain. Specifically, ANNs are mathematical models that are consisted from many neurons that are basically small processing units. Neurons are connected with each other with weighted connections. Those weights define the decision output an ANN will produce given a certain input. Multiple neurons are put together to form a layer. By stacking multiple layers with neurons we form a Deep Neural Network (DNN). An example of a neural network can be seen in the Fig. 3.1 and an example of a neuron can be seen in Fig. 3.2 (Ojha et al., 2017).

3.1.1 Simple Feed-forward Networks

Over the years, many variants of ANNs have been formed, with the simplest one being the feed-forward neural network (FNN). The neuron connections between layer in FNNs are acyclic. That means that each layer is connected only with the previous and its next layer or, in other words, the neurons of a specific layer are connected only to neurons from the previous and the next layer. In Fig. 3.1, we can see an example of a feed-forward network.

The output of a FNN is basically the product of a non-linear function formed by the connections between the hidden layers. The input of
the FNN is disseminated through the neurons in the hidden layers to produce an output which differs based on the connection weights between neurons. This procedure is known as forward pass. Neurons have activation functions that produce an output by basically mutating their input with the weights of the connected neurons from the previous layer. In parallel, the weights of the connections between neurons are basically the coefficients of their activation functions. FNNs can model non-linear problems effectively given an optimal set of weights.

### 3.1.2 Forward Pass

Forward pass is the propagation of the FNN input through all of its hidden layers. Given the input $a_h$:

$$a_h = \sum_{i=1}^{I} w_{ih} x_i + b_h$$  \hspace{1cm} (3.1)

where $I$ is the number of input units, $x$ is the input vector, $h$ the hidden units (Fig. 3.2), $w_{ij}$ is the weight of the link between units $i$ and $j$, and, $b_h$ is the bias of the neuron.

The output $c_h$ would be:

$$c_h = \theta_h(a_h)$$  \hspace{1cm} (3.2)

where $\theta_h$ is the activation function. After computing the output of the units of the first hidden layer, calculating the outputs of the units in the rest layers is simply a matter of using the same equations as above with
3.1. Deep Neural Networks

Generally, non-linear activation functions tend to be used more often than linear, as they can model higher complexity problems with greater success, especially when stacked in multiple layers. Although, non-linear activation functions can severely impact a network’s performance, they must be paired with a great optimization algorithm to define as better weights as possible. A popular optimization algorithm is the Gradient Descent that focuses on finding local minima. Such algorithm, though, requires that the activation functions must be characterized as differentiable. Some of the most popular activation functions are the sigmoid:

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]  

(3.3)

and the hyperbolic tangent:

\[ \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \]  

(3.4)

3.1.3 Output layers

An output layer is required in order to generate predictions from the output of the previous neural network layers. The input of the output layer \( k \) follows the same computation process as in any other hidden layer. For a network with \( m \) hidden layers, the input is:

\[ a_k = \sum_{h=1}^{H_m} w_{hk}c_h + b_k \]  

(3.5)

The output layer has an activation function with a number of activation units, as any other network layer, but both of those differ depending on the type of problem we are solving. For instance, let us consider a binary
classification task. The output layer will have the sigmoid activation function with one unit. This is because the sigmoid function produces outputs in the range of (0,1). Thus, the sigmoid output can be deemed as the probability of the input vector being classified in the first class $C_1$. The output of the sigmoid unit $\hat{y}_k$ is:

$$\hat{y}_k = \sigma(a_k) = p(C_1|x) \quad (3.6)$$

For $K$ classes, with $K > 2$, the output layer should have $K$ units and the output itself should be normalized with the softmax function (Bridle, 1990) to acquire the probability distribution of the $K$ classes.

$$\hat{y}_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} = p(C_k|x) \quad (3.7)$$

In binary classification, neural networks’ loss functions require the target class $z$ to be represented as a binary vector with columns equal to the number of categories in the data. All vector values are zero except from the column that belongs to the class that corresponds to the input, which equals one. For example, for 3 classes ($K = 3$) with the class $C_2$ being the correct, $z$ is represented by the vector $(0, 1, 0)$. Using this notation, the target probabilities can be computed as:

$$p(z|x) = \prod_{k=1}^{K} \hat{y}_k^{z_k} \quad (3.8)$$

Considering the definitions described till this point, one could easily employ a neural network for classification tasks. Given an input vector $x$, a forward pass is performed and the probability of the vector being classified in each class is computed. Then, from all the unit outputs, the class label with the highest probability is selected.

### 3.1.4 Training

Conclusively from the previous sections, the output of a neural network depends on the input. The output is directly correlated with the weights connecting the hidden units and the hidden unit biases, which are the network’s parameters. In order to make predictions, we need to train the neural network to compute the values of those parameters. The training includes several mechanisms to define those parameters, such as objective function optimization and backpropagation.

**Optimization Objective**

Given a set of data, we train a neural network on them by optimizing the objective function. Basically, this implies that we need to find those
parameter values that optimize the objective function. This can be achieved by using Stochastic Gradient Descent (SGD).

In a classification supervised learning task, the training data consist of pairs of input vectors accompanied with the desired output for each vector. For every vector, a forward pass is performed in order to generate an output. Then, the predicted output is compared with the desired one and the weights are updated in view of minimizing the selected loss function.

The selection of the loss function depends on the task. Specifically, for a network that produces a probability distribution, the most suitable function would be the negative log probability of the right answer. This function is implemented by the cross-entropy cost function (Bishop et al., 1995). As stated above, we try to minimize the cost function by reaching to proper network parameters. We follow this process because the minimization of the cross-entropy function tries to maximize the probability of producing the correct answer.

Then, to minimize the cost function, the Stochastic Gradient Descent (SGD) searches for the proper values of the network’s parameters. SGD iteratively alters the network’s parameters in accordance to the cost function’s derivative for each hidden unit. This weight updating process is known as the delta rule. The delta value is multiplied with a lambda factor, which is known as the learning rate, which is an important tuning parameter.

Basically, the learning rate determines the amount of influence the new information has on the old information. This simulates the speed which the network "learns" the data. For every iteration, learning rate defines the step size at which we move towards a local minimum of the loss function. The selection of the learning rate is very important as a very large value will not be able to locate minima, while a very small value will considerably increase the learning time or the learning process will converge on an undesirable local minimum.

**Backpropagation**

We described the general process of minimizing the loss function by updating the network’s weights according to the loss function’s derivative (gradient). The computation of the loss function’s gradient for each hidden unit utilizes the chain rule. Using the chain rule, the gradient is computed for each layer recursively, starting from the last layer to avoid redundant chain rule computations. This procedure is known as Backpropagation (Williams and Zipser, 1995; Werbos, 1988; Rumelhart et al., 1985).

Backpropagation solves the problem of computing the derivatives of the loss function with respect to the hidden weights for the units of each hidden layer. The problem halts the learning process of the
layers of hidden units, because the desired output for the hidden units is not known. Backpropagation addresses that by computing first the derivative of the units in the output layer with respect to the output. Then, we can easily compute the derivative of the loss function for the weights connecting to the units in the previous hidden layer. This process continues until we reach the first hidden layer.

3.2 Recurrent Neural Networks

As described in the previous section, simple neural networks use only feed-forward connections. The innovation of Recurrent Neural Networks (RNNs) is that except from the units being connected with both the previous and the next layer, they are also connected with units within the same layer. In a certain time step, RNN units produce an output based on the input from the units in the previous layer, as well as from the hidden state of the previous time step. Basically, RNNs remember information stemming from the previous time steps.

The strength of RNNs stem from two key qualities. The first one is that several hidden states can be active simultaneously as they are distributed. This allows RNNs to remember different information at the same time. Secondly, RNNs are non-linear, which allows them to update their hidden states in a non-linear convoluted manner. This property empowers them to model very complex functions which renders them as strong function approximators.

The category of RNNs is consisted of several different variations, such as echo state networks (Jaeger, 2001), Long Short Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Cho et al., 2014). At this current point, we briefly highlight the main differences between FNNs and RNNs, and, then briefly describe how the FNNs are extended to RNNs. In continuation, we will get in more detail in a specific variation of an RNN, which is the LSTM. The LSTM is a top performer in natural language tasks, which is the reason of including it in our experiments.

3.2.1 Simple RNN

Let us consider a simple RNN with just one hidden layer. When unfolded through time, it is basically the same as a feed-forward neural network with one hidden layer for every time step. An example is shown in Fig. 3.3. Thus, the conversion of the FNN architecture into the RNN is pretty straightforward.
3.2. Recurrent Neural Networks

Backpropagation Through Time

In RNNs, the computation of the loss function’s derivative with respect to the network outputs is the same as for FNNs. Starting from the last layer, the network’s output is compared to the desired output. Then, the derivatives are computed with respect to the weights and the delta rule is applied.

The weight derivative calculations are performed by the Backpropagation Through Time algorithm (BPTT) (Williams and Zipser, 1995; Werbos, 1990), which is an extension of the backpropagation. As in the standard backpropagation, BPTT performs a repeated application of the chain rule. The main differentiation is that in RNNs, the loss function’s derivative of a hidden unit, with respect to its input at time $t$, is computed as a function of two inputs. The first input comes from the loss function derivative of the units in the output layer at time $t$, which is the same as in FNNs. The second input comes from the loss function derivative of the units in the same hidden layer at the previous time step $t-1$, which differs from FNNs.

3.2.2 Long-Short Term Memory (LSTM)

RNNs have the ability to learn contextual information over information present in an earlier point of the sequence. Though, the standard version suffers from limited distance for which information can be remembered making them effective only on shorter sequences. This occurs because, over multiple time steps, the gradient tends to get too small that prevents the weights from changing or too big that hinders the ability to find local minima as the steps are very large. These phenomena are known as the vanishing and exploding gradient problems, respectively. There have been multiple attempts to solve this issue with the most successful being the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU). As the work of this thesis was based on a LSTM architecture, in this section, we will concentrate on the analysis of this specific architecture.
The main idea of the LSTM architecture is based on the notion of the memory cell. A memory cell has the property of maintaining its state over time. The state includes a cell state vector that simulates the memory function, and, gating units that disseminate the information through and out of the memory. LSTMs solve the problem of vanishing gradient with the introduction of those gates, and thus they remember long-term information. The gates are multiplicative units and they are a total of three, which are the input, the output, and, the forget gate.

The input gate introduces new information to the memory and controls the changes introduced to the old memory. The output gate decides the output of the memory and specifically the quantity of the new memory to output. The forget gate is responsible of forgetting of old information. No memory will be saved if it is completely closed, while all old memory will be kept if it is completely open. The determination of how open the gate is, depends on the element-wise multiplication operation. Specifically, if the old information is multiplied with a vector close to 0, then most of the memory will be forgotten. The opposite holds for a vector close to 1, which signifies that most old information will be kept.

![LSTM block](image)

Multiple memory cells form a hidden layer just like memory units in simple RNNs. Memory cells are recurrently connected with each other from layer to layer. Fig. 3.4 depicts an example of such memory cell. In an LSTM, the final output layer could be any differentiable one, which is the same as with simple RNNs. Learning of the weights can be achieved with gradient descent and derivatives could be calculated with backpropagation through time.

We can calculate the updates for the sub-unit-inputs $i_t$, memory cell $c_t$, forget gates $f_t$, and, output $o_t$ of an LSTM unit at time step $t$ as follows:
where $x_t$ is the input vector (usually word embedding) at time step $t$, $\odot$ is the Hadamard product, $\sigma$ is the element-wise logistic sigmoid function, and, $h_t$ is the hidden state at time step $t$. $W$, $b$, and $U$ are model parameters that are calculated at the training process.

### 3.3 Sequence to Sequence Models

Sequence to sequence models were introduced by Google associates at 2014 (Sutskever et al., 2014) and then have rose in popularity. Especially over the last few years, and for a good reason, they tend to outperform any other method in many difficult natural language tasks. Such collection of tasks includes, but is not limited to, language (machine) translation, chatbots, speech recognition, and video/image automatic captioning.
The appeal of this method, that makes it stand out, is the ability to have different sequence sizes for the input and the output, which simple LSTMs cannot achieve. Generally, this type of model can be applied on sequence modeling problems with any input-to-output correlation such as one-to-one, one-to-many, many-to-one and many-to-many, which are shown in Fig. 3.5. For example, in language translation, there are many instances were a fixed set of input words is mapped to a different number of output words in another language, i.e., translating English into Greek, the 2 word input “good morning” is mapped to one output word “καλημέρα”.

The input of such models is required to be numeric, so the words are converted to a vector representation known as word embeddings. These embeddings can be learned by input/embedding layer of the model, among other ways. A sequence model is consisted of 3 main parts, which are an encoder, an intermediate (encoder) vector and a decoder, as it is demonstrated in Fig. 3.6. Essentially, the encoder and the decoder are two different recurrent neural networks merged into one. Basically, the encoder aims to understand the input sequence and create a summary of it in a smaller dimensional space, while the decoder attempts to use this information to create its own output sequence.

Here, we will present the basic structure of such models, but we should note that the simplicity of using just one single intermediate (encoder) vector to encode all the sequence information will not suffice for complex problems. There are multiple works in the literature that introduce new methods to cease the shortcomings of the base version in order to achieve great performance in problems with long input and output sequences. Such approaches would be to use LSTM or GRU units instead of base RNN, add information by considering the sequence in reverse (e.g. bidirectional-LSTM), introduce an attention mechanism, incorporate beam search, and, many others.
3.3. Sequence to Sequence Models

3.3.1 Encoder

The encoder component is constituted by multiple recurrent units, where each unit processes a single token of the input and then disseminates the collected information to the following units. Those units usually are LSTM (Sutskever et al., 2014) or GRU (Nallapati et al., 2016) units as they usually offer better performance. The hidden states of the encoder can be computed as any other typical recurrent neural network. Given the input sequence \( x = (x_1, ..., x_{T_X}) \), where \( T_X \) is the total time steps, the equation for a hidden state \( h_t \) of the encoder at time step \( t \) is:

\[
h_t = f(x_t, h_{t-1})
\]

where \( x_t \) is the input sequence of word embedding vectors at time step \( t \). The product of the encoder is a context vector \( c \), which tries to capture information of each input token to assist the decoder in making better decisions. The context vector serves as an initial hidden state of the decoder. The equation of the context vector is:

\[
c = q(h_1, ..., h_T)
\]

where \( f \) and \( q \) are non-linear functions, which are dependent on the selection of the RNN encoder, which is either LSTM or GRU.

3.3.2 Decoder

The decoder component is a model that utilizes the information of the context vector \( c \), generated by the encoder, to produce a sequence of words. In order to predict the next word \( y_t \), in addition to the context vector, the decoder takes into consideration all the previous generated words \( y_1, ..., y_{t-1} \). Basically, to generate a sequence of words, at each decoder time step, the decoder calculates the joint probability of the sequence of words \( y = (y_1, ..., y_{T_y}) \) for all the possible output words that exist in our vocabulary and selects the most probable one. The joint probability of the sequence of words is computed as the product of the conditional probabilities for all the time steps \( T_y \) of the decoder and its equation is as follows:

\[
p(y) = \prod_{t=1}^{T_y} p(y_t | y_1, ..., y_{t-1}, c)
\]

The decoder is a recurrent unit that uses the softmax function to create the probability vector that will allow as to generate the better prediction \( y_t \) for each decoder time step. Each recurrent unit receives the output and the hidden state of the previous unit and it generates an output, as well as its own hidden state.
Chapter 3. Neural Networks

\[ p(y_t) = \text{softmax}(g(y_{t-1}, s_t, c)) \]  
\[ (3.18) \]

where \( s_t \) is the hidden state and \( g \) is the non-linear function of the RNN decoder.

3.3.3 Beam Search and Attention

**Beam Search** As we saw in the previous section, the decoder calculates the probability that any word in the vocabulary has to be the next word in the sequence. Then, at each time step, the decoder has to decide what will be the best word for the current sequence. A simple approach would be to take a greedy decision, where we select the most probable word as the next word at each time step. Although, this method selects the most probable word in a specific time step, it does not guarantee a maximal value of the joint probability of the sequence of words.

For a certain sentence as input, there could be many probable sentences as output. We could consider the space of possible outputs as a tree structure, where each word, represented by a tree node, is connected only to nodes that might be possible to be next in the sequence. The greedy approach would follow the sequence of nodes that have the highest probability at each time step. A better approach would be to utilize beam search. Beam search considers the summation of the probabilities of all of the next \( k \) words in the sequence. Then, it selects the sequence with the maximum cumulative probability.

**Attention** Another common problem in the natural language tasks is that when considering text semantics, there are only a few words in a sentence or even a document that shape its meaning. Thus, some words are more important than others. The attention mechanism (Bahdanau et al., 2014) tries to identify and focus on the semantically important words while decoding the input sequence. The mechanism correlates each word in the input sequence with a weight. It adds importance to the vital words by assigning them a larger weight so that the decoder would produce optimal predictions. This weight is used in combination with the output of the encoder at each time step.

To implement this function, for each target word \( y_t \), a distinct context vector \( c_{t*} \) is calculated. The difference with the plain encoder-decoder model is that now of the conditional probability of a target word \( y_t \) is the context vector \( c_{t*} \). Another name for the probability distribution of all possible words \( y_t \) at decoding step \( t \) is vocabulary distribution:

\[ p(y_t | y_1, ..., y_{t-1}, c_{t*}) = g(y_{t-1}, s_t, c_{t*}) \]  
\[ (3.19) \]

The context vector \( c_{t*} \) is a weighted summation of the hidden states \( h_t \) of the encoder and is calculated as:
3.3. Sequence to Sequence Models

\[ c_{t}\ast = \sum_{i=1}^{T_x} a_{ti} h_t \]  

(3.20)

Fig. 3.7 showcases the process of computing the context vector \( c_{t}\ast \). The weights \( a_{ti} \) of the hidden states \( h_t \) are known as attention distribution. They could be considered as a probability distribution over all input words.

\[ a_{ti} = \text{softmax}(e_{ti}) \]  

(3.21)

\[ e_{ti} = v^T \text{tanh}(W_h h_t + W_s s_t + b_{attn}) \]  

(3.22)

where \( s_t \) is the hidden state of the decoder and \( v, W_s, W_h, \) and \( b_{attn} \) are parameters that can be learned.

**Figure 3.7:** Sequence-to-sequence model with bidirectional encoder and attention mechanism

3.3.4 Bidirectional Encoder

The basic version of RNNs takes as input a sequence of words in a forward manner, where the network processes each word in the exact order they are present in the sequence. In a specific time step of a sequence, often times, it would be beneficial to have knowledge about the following words in addition to the information from the previous words. This drove the authors Schuster and Paliwal (1997) to introduce the bidirectional RNN (BRNN) architecture. Such a model achieved great success in many tasks, such as speech recognition Graves et al. (2013).

The BRNN is composed of two RNNs, one that reads the input in a forward manner and one that reads it backwards. The forward RNN reads the input from the first sequence token \( x_1 \) to the last \( x_{T_x} \) and produces a forward hidden state \( h^\rightarrow \). The backward RNN read the input from the last token \( x_{T_x} \) to the first \( x_1 \) and produces a backward hidden
Bidirectional RNNs achieved state-of-the-art results in many problems when used as a decoder. The most popular RNNs for this architecture are LSTM and GRU. Note that bidirectional RNNs can be used only as an encoder and not as decoders. This is due to the fact that the decoder has access only to the already generated sequence till the current time step and not for future time steps. Fig. 3.8 showcases the notion of bidirectional RNNs as a separate entity.

3.3.5 Bi-LSTM-CRF Encoder-Decoder

The Bi-LSTM-CRF model has been successful in multiple natural language tasks, achieving state-of-the-art performance. Such tasks include noun phrase chunking (NPC), named entity recognition (NER), and, part-of-speech tagging (POS) (Liu et al., 2017; Ma and Hovy, 2016; Huang et al., 2015). In the spectrum of keyphrase extraction from scholarly documents, Alzaidy et al. (2019) deployed this model, and as they stated, their work is the first that considered text semantics and long-distance sequence information in combination with output sequence dependencies. The output sequence dependencies include information from the input sequence, from Bi-LSTM, as well as the label dependencies of neighboring words of the output.

Their approach considers the keyphrase extraction task as a sequence labeling problem. The Bi-LSTM component of the network
3.3. Sequence to Sequence Models

Bi-LSTM-CRF architecture offers text semantics and long-distance information from the input sequence. Then, the Bi-LSTM output is passed to the CRF layer, where the probability distribution of the tag sequence is computed. The calculation of the probability distribution takes into account the label dependencies of the entire sequence. In order to decide which output sequence is the most suitable for the input sequence, the Viterbi algorithm is utilized. The architecture of this network is shown in Fig. 3.9.

As described in Section 3.2.2, LSTMs have the ability to utilize long-distance information due to their memory cell architecture, which is a valuable asset. Traditional LSTMs store information only from the sequence that they have already seen. Though, having access to the upcoming parts of the sequence has been proven highly beneficial. So, in order to capture both forward and backward sequence information, we utilize a bi-directional LSTM network. As mentioned in Section 3.3.4, a Bi-LSTM consists of two hidden layers, one backward and one forward. Note that the nodes in the hidden layer are connected in order to preserve the long-distance information.

Conditional Random Field (CRF), proposed by Lafferty et al. (2001), is a class of discriminative models deployed in various sequence labeling tasks with great success. CRF is a special case of the Markov Random field and they are able to model the sequential dependencies among the predicted labels jointly. The conditional probability distribution of the output sequence $y$ given the input $x$ is defined as follows:

$$P(y|x; W, b) \propto \exp\left(\sum_{i=1}^{n} W_{y_{i-1},y_i}^T x_i + b_{y_{i-1},y_i}\right)$$  (3.24)

where $W_{y_{i-1},y_i}$ is the weight vector and $b_{y_{i-1},y_i}$ is the bias for neighboring labels $y_{i-1}$ and $y_i$.

In order to train a CRF model on a dataset $D = \{(x^{(j)}, y^{(j)})\}_{j=1}^{N}$, the weights $W$ and bias $b$ should be calculated. $W$ and $b$ are computed
during training by maximizing the following log-likelihood:

\[ L(W, b) = \sum_{j=1}^{N} \log p(y^{(j)} | x^{(j)}; W, b) \]  

(3.25)

Then, to find the most suitable output sequence at the decoding step, the Viterbi algorithm is utilized. Basically, the Viterbi algorithm tries to find the sequence \( y \) for which the log-likelihood is maximized, and, is computed as:

\[ y^* = \arg\max_{y \in y(x)} p(y|x; W, b) \]  

(3.26)

An example of a CRF model is seen in Fig. 3.10. As we can see from the figure, the nodes in the output layer are connected, which provides the ability to capture label dependencies.

![Figure 3.10: CRF architecture](image)

In the present chapter, we analyzed basic as well as cutting-edge architectures of neural networks. The encoder-decoder architecture is a powerful tool for natural language tasks. Especially for the problem of keyphrase extraction, the Bi-LSTM-CRF model that follows the encoder-decoder architecture seems to be effective compared to other state-of-the-art approaches as shown in Alzaidy et al. (2019). In the present work, we explored the effectiveness and weaknesses of this model and we further attempted to optimize its results in the task of keyphrase extraction by employing various techniques. In the following chapter, we will discuss our experimental setup as well as the results and conclusions we draw.
Chapter 4

Experiments

This chapter focuses on the details of the conducted experiments that were part of the work of the present thesis. This work aims to increase the performance of deep learning keyphrase extraction methods by increasing their exposure to words from the whole text. We merely focus on the current state-of-the-art keyphrase extraction method of Alzaidy et al. (2019) to demonstrate the effectiveness of our technique.

As of now, mainstream methods usually utilize just the title and the abstract of publications to extract keyphrases. This limits the overall performance of the methods as the vast majority of the text is omitted and subsequently a large portion of keyphrases are never seen. This does not reflect a real case scenario. Further, it is worth noting that the whole text of scholarly publications tends to be very lengthy, extending to multiple pages. This poses a problem as training such models is very costly in time and money due to hardware limitations.

In an effort to solve the problems of training directly on the full-text but retain the benefits of more words, we deploy text summarization techniques to introduce new words by creating a summary of the whole publication document and then use the produced text on the Bi-LSTM-CRF model to extract keyphrases.

Two types of models were trained, one on the unified abstract and title and one on the abstract and title split into sentences. We further experimented with evaluation datasets containing the full-text of scholarly documents to monitor the effectiveness of our methodology on a real case scenario where we have the full-text of a document. Those datasets are the ACM, the NUS and the SemEval-2010. To reproduce the Bi-LSTM-CRF model of Alzaidy et al. (2019), we experimented with the kp20k datasets and the two trained models. To set the comparison environment and to explore the effectiveness of our method through different methods, we used the Bi-LSTM-CRF model, as well as two popular unsupervised techniques as baseline, which are the TfIdf and the MultipartiteRank (Boudin, 2018) from the pke\(^1\) library.

In addition to the sequence evaluation method of calculating F\(_1\)-score, precision and recall, used by Alzaidy et al. (2019), we also include

\(^1\text{https://boudinfl.github.io/pke/build/html/index.html} \)
the most popular evaluation techniques of keyphrase extraction models, which are the calculation of those metrics with the exact and partial matching approaches. This is done in view of a fair comparison between previous approaches.

In the next sections, we provide detailed descriptions of all the steps performed from the start to the end of our experiments. The steps include data analysis, data processing (4.3), data annotation, experiments set-up and hyper-parameter tuning (4.4), and, evaluation (4.4.2). The data flow during our experiments is presented in Fig. 4.1. The background of our experiments is briefly mentioned in (4.4.1). Finally, at the end of this chapter, we present and discuss the experiments’ results and deductions (4.4.3). Results demonstrate the effectiveness of the proposed methods and highlight their advantages.

**Figure 4.1:** Data flow throughout each step of our experiments

### 4.1 Problem Formulation

We formulate the task of keyphrase extraction as a sequence labeling problem, where given an input sequence \( x = (x_1, ..., x_n) \), we predict a sequence of labels \( y = (y_1, ..., y_n) \). The input is consisted of a sequence of vectors, where each input vector of the \( i^{th} \) word is represented as \( x_i \). The output contains a sequence of labels, where each label is matched with its respective word in the input sequence. The label \( y_i \) is binary and its values are KP for a keyphrase word, and, Non-KP for a non keyphrase word. By using the Bi-LSTM-CRF model, we manage to incorporate the dependencies of the neighboring labels and further to jointly decode the best output sequence of labels for the input sequence,
instead of decoding each label individually. We train two models on
the documents’ title and abstract. The first one, considers them as a
whole text excerpt, while for the second, we split them in sentences.

4.2 Datasets’ description

For our experiments we utilized a training, a validation and four differ-
et test sets that are made up by scientific documents. The training and
validation sets come from Meng et al. (2017) and are known as kp527k
and kp20k-validation (or kp20k-v), respectively. Additionally, Meng
et al. (2017) provided a test set as well, that is referred to as kp20k-test
(or simply kp20k). This test set is mainly used for the reproduction of
the Bi-LSTM-CRF model of Alzaidy et al. (2019). The main test sets in
our evaluation set up are the NUS (Nguyen and Kan, 2007), the ACM
(Krapivin et al., 2009), and the SemEval-2010 (Kim et al., 2010).

The datasets from Meng et al. (2017) are composed by publications
found in ACM, Wiley, and Web-of-Science. All three of them are sep-
arated in distinct files and include the title and the abstract of the
scholarly publications, as well as the keywords/keyphrases assigned
by the authors of each publication. The total number of papers con-
tained across all three datasets is 570,809. The kp527k dataset contains
530,809 papers, while the kp20k-validation and the kp20k-test set con-
tain 20,000 papers each.

The title is added into the beginning of the corresponding abstract
and the resulting document is then lightly processed before it is con-
verted into readable format for our Recurrent Neural Network (RNN).
The processing we followed is described in the Section 4.3.

The final version of each document consists of two sequences, one
containing the word tokens and one for their corresponding keyphrase
labels. Words that are part of keyphrases take the label 1 (or KP), while
the rest of the words take the label 0 (or Non-KP). The label sequence
is generated with string matching between the document text and the
keyphrases, after both of them are lowered cased and stemmed.

The ACM dataset consists of 2,304 scholarly scientific documents
from the domain of computer science, which were published by ACM.
The papers were published in the time period from 2003 to 2005 and
they are all in English. The publications were extracted from the Cite-
SeerX autonomous digital library and were stored in UTF-8 text format.
The papers of the dataset are composed by the full-text of each pub-
lication and there are identifiers in the text that provide the ability to
split it into 4 text excerpts, the title, the abstract, the main body and
the references. The keyphrase set is author-assigned and the process of
creating the label sequence for each document is the same as mentioned
above.
Chapter 4. Experiments

The NUS dataset has a total of 211 papers. The documents were collected by the Google SOAP API using the query “keywords general terms filetype:pdf”. The collection is constituted only by scientific conference publications and their length varies between 4 and 12 pages. It is a rather small dataset, but this is because the keyphrase set not only includes keyphrases assigned by authors, but in addition it incorporates reader-assigned keyphrases. Again, the generation of the label sequence is the same, as previously mentioned. Each document is split into title and the full-text of the paper, including abstract and references.

The SemEval-2010 dataset contains a collection of full-text papers. Keyphrases are annotated from both the authors and readers. It is comprised of a train set with 144 papers and a test set with 100 papers. Words that were hyphenated (-) on line break were restored to their original form. Also, readers were instructed to annotate papers with keyphrases from any part of a document as soon as they exist in its text. On the test set, the final result contains an average of 15% keyphrases that do not exist in the document. To put that into perspective, author assigned keyphrases are comprised of 19% that do not appear in the text. Thus, the maximum recall that extraction models can achieve is upper bounded to 81% for the author-assigned and 85% for the reader keyphrases. Note that as the SemEval-2010 dataset is pretty small, we combined both the train and the data set, resulting in a set with a total of 244 documents.

Note that, in our experiments, we did not incorporate all of the datasets used in Alzaidy et al. (2019), because our primary purpose is to replicate their work sufficiently and then expand on techniques that attempt to add more keyphrases to increase performance, and not to conduct a full in-depth performance comparison of our methods. As one can see, our selection of test datasets features three sets with full-texts of scientific publications in order to experiment with various text excerpts extracted/generated from the documents’ full-text.

Last but not least, we stress the importance of a good dataset. Any model is as good as the quality of the training dataset is. A high-quality test set is also important, as, in the opposite position, it may obscure the actual efficiency of the approach. Generally, a quality dataset should be reviewed by multiple people in an effort to increase keyphrase coverage as well as reduce the effects of subjectivity. This kind of data annotation is extremely time complex, which means that a large amount of financial funds are required. At the same time, large amounts of data are needed for neural networks to shine, which in combination with those restrictions, this might never see the light of day, or at least in substantial capacity.
4.3 Data Processing

Machine learning models require clean and high-quality data to perform as best as possible. This is true for our case as well with deep learning models. As deep learning models do not require heavy processing as other machine algorithms may need, we performed some basic cleaning steps to assure better quality and thus greater performance in the task of keyphrase extraction.

For all three selected datasets, kp527k, NUS and ACM, the same processing steps were followed. Particularly, the data are first preprocessed and then annotated on their capacity of being keyphrases. There was no removal of stop-words as in neural networks; stop-words play an important role in learning text context. The datasets are structured, so they allow for an easy segmentation of the title, abstract, main body and references, where they apply. Note that we checked for duplicate papers among the training set and each of the other datasets in this work, including the kp20k validation and test sets as well as the ACM, NUS and SemEval-2010. The duplicate papers’ total is 425 and they were deleted from the kp527k-training set as the impact would be almost unnoticeable due to the large volume of documents.

The first pre-processing step is to expand contractions to their whole word counterparts, e.g. "it’s" becomes "it is". Then, utilizing several regex formulas, we convert the references located in the main body to a special token ("REFPUBL"), we remove mathematical expressions, and, all punctuation is replaced with a white-space and non-ascii characters are removed. In addition, we remove all the single letters that are not the letter ‘a’ as they do not serve a purpose and they are probably residues from mathematical expressions.

We remove mathematical expressions from the documents’ text in a series of steps as follows. We assume that everything inside brackets ‘[]’ is mostly references to publications or it serves as explanatory information, which both are not considered as high value source of keyphrases. Looking into the publications’ text of the datasets confirms this allegation on the vast majority of cases.

All digits located in the text of the papers and the keyphrase sets are replaced with a special token defined as "DIGIT_REPL". In continuation, empty documents and empty keyphrase sets are filtered, which is meaningful for the sentence trained model only. When splitting the documents into sentences, some sentences might end up with a single word or punctuation and thus after applying all the data cleaning steps they will end up empty. Finally, redundant white-spaces, new-lines and tabs are removed.

After these steps, the conversion of words into a numeric representation with the Keras tokenizer follows, which uses the token "<UNK>" for unknown words. The text is tokenized and lower-cased in order to
create an embedding matrix that matches the integer representation of
the words with GloVe word embeddings. Thereafter, the annotation
process of each of the document words takes place. The final step is the
addition of padding and the conversion of the label set to categorical.

As we can see from Fig. 4.2, after pre-processing, the combination of
the title and abstract covers 63.35% (1,774,628 keyphrases) of the total
gold keyphrases, which are 2,801,398. Also, the title introduces 121,971
new keyphrases that are omitted by the abstract, which equates to 4.45%
of the total gold keyphrases. Note that these statistics are produced
by checking the existence of the gold keyphrases in the text, without
considering the occurrence frequency. In addition to the statistics of
the Table 4.2, the sentence version of ACM with the title and abstract
has 17,486 sentences/training instances, the sentence version of NUS
dataset with the same information has 1,673 training instances, and,
the SemEval dataset has 1,979 sentences.

After our pre-processing steps, we are left with 530 thousand docu-
ments and a total of around 84 million tokens for the kp527k-training,
as we can see in Table 4.1. In comparison with Alzaidy et al. (2019), they
are left with 527 thousand documents with about 78 million tokens.
These differences might be caused for 2 different reasons. First, the
document difference could be because more documents are removed
from the duplicate document removal step at their part, which could be
a result of the kp30k-training having more duplicates with the WWW and KDD (Gollapalli and Caragea, 2014b) test datasets. Second, the difference in total tokens might be a combination of the first reason and because they might not include the title in their experiments.

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Training kp527k</th>
<th>Validation kp20k-val</th>
<th>Testing kp20k</th>
</tr>
</thead>
<tbody>
<tr>
<td># of documents</td>
<td>530,390</td>
<td>20,000</td>
<td>20,000</td>
</tr>
<tr>
<td># of sentences</td>
<td>4,136,306</td>
<td>156,519</td>
<td>155,801</td>
</tr>
<tr>
<td># total tokens</td>
<td>84,831,469</td>
<td>3,197,025</td>
<td>3,201,308</td>
</tr>
<tr>
<td># keyphrase tokens</td>
<td>6,584,027</td>
<td>248,314</td>
<td>248,357</td>
</tr>
<tr>
<td># of keyphrases</td>
<td>3,882,211</td>
<td>145,993</td>
<td>146,447</td>
</tr>
</tbody>
</table>

**Table 4.1:** Statistics for the datasets used to reproduce Alzaidy et al. (2019) (after pre-processing).

<table>
<thead>
<tr>
<th>Statistics</th>
<th>NUS</th>
<th>ACM</th>
<th>SemEval-2010</th>
</tr>
</thead>
<tbody>
<tr>
<td># of documents</td>
<td>211</td>
<td>2,304</td>
<td>244</td>
</tr>
<tr>
<td># of sentences</td>
<td>74,219</td>
<td>770,263</td>
<td>75,726</td>
</tr>
<tr>
<td># total tokens</td>
<td>1,427,743</td>
<td>16,285,971</td>
<td>1,673,430</td>
</tr>
<tr>
<td># keyphrase tokens</td>
<td>55,360</td>
<td>267,005</td>
<td>64,139</td>
</tr>
<tr>
<td># of keyphrases</td>
<td>38,382</td>
<td>172,087</td>
<td>43,289</td>
</tr>
</tbody>
</table>

**Table 4.2:** Statistics for the main test datasets (after pre-processing), i.e., NUS, ACM, & SemEval-2010 with the full-text.

In an attempt to identify their methodology, we tried removing the title in our set-up and we were left with about 79 million tokens. Note that, even though they have 3 thousand less documents, with a more moderate cleaning process the total token numbers could be matched. Assuming this was their set-up, we attempted to trained some models, but without any success in reproducing their results.

Also, the sentence count is also different. This is because sentence splitting is based exclusively on the full-stops, thus there is plenty of opportunity to not get a fully accurate split. Before splitting the text into sentences, we first remove the full-stops (.) from acronyms using regular expressions and from three abbreviations (e.g., i.e., etc.) to avoid noise for sentence boundary detection. After the pre-processing steps, we further remove the empty sentences, which ultimately positively affect the total count of sentences.

Experiments ran for the sequence length as well. The histograms in Figs. 4.3 and 4.4 show the lengths of the documents of the kp527-train set. The first figure is about the title and the abstract combined, while
Chapter 4. Experiments

Figure 4.3: Histogram of document lengths of the kp527-train set.

For the second experiment, the data is split into sentences instead. We experimented with a total of three different padding values, which are 300, 400, and 500. As we can see from the first case of combined title with abstract, a good padding value would be 400 as only 2,204 documents are cropped (Table 4.3) and it produces the best performance.

<table>
<thead>
<tr>
<th>length</th>
<th>cropped documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>14,914</td>
</tr>
<tr>
<td>400</td>
<td>2,204</td>
</tr>
<tr>
<td>500</td>
<td>653</td>
</tr>
</tbody>
</table>

Table 4.3: Cropped documents per sequence size for continuous text and sentence model

For the case of sentences, a good padding value would be 40, again for the same reasoning. From the 4,136,306 sentences, only 188,384 sentences are cropped with a padding value of 40 (Fig. 4.3). In this instance, we experimented with the values 40, 50, and 70. Additionally, in the experiments, we included the approach followed in the Alzaidy et al. (2019) work. The pytorch implementation that the authors utilized follows a bucket approach for training.
For the bucket method, the documents are split into 4 different segments regarding their length with comparison to 4 numbers. The first number is the max length of the documents, the second is the average length, the third is the average length of the documents that have length lesser than the average. For simplicity, we will refer it as lower average. The fourth and final number is the average length of those documents that have length greater than the average, referred as upper average. Each bucket contains the documents that have lesser length than that of its respective number. After assigning all the documents in buckets, each one is padded to match the length of the assigned bucket number. For the kp527k-train dataset, those values can be seen in Table 4.4. We observed that the bucket splitting values and the document distribution in each bucket are all very close for all three kp20k sets.

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Size</th>
<th># of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower average</td>
<td>110</td>
<td>124,574</td>
</tr>
<tr>
<td>average</td>
<td>160</td>
<td>156,678</td>
</tr>
<tr>
<td>upper average</td>
<td>216</td>
<td>150,878</td>
</tr>
<tr>
<td>max length</td>
<td>2,507</td>
<td>98,260</td>
</tr>
</tbody>
</table>

Table 4.4: Bucket sizes
Chapter 4. Experiments

For our experiments, we used a batch size of 64 for the model that was trained on the kp527-train with the title and abstract combined, and a batch size of 256 for the sentences model, in an effort to match the total words contained in a single batch to that of the first model. We also trained and compared the results of the 256 batch size with a batch size of 64 in the same sentences model. In order to finalize the batch size, we run several trials for 32, 64 and 128 batch sizes for the title and abstract model. For the sentences model, we tried the values of 224, 256, and 352.

4.4 Experiments Setup and Parameter tuning

The base model architecture for all experiments was the Bi-LSTM-CRF model proposed by Alzaidy et al. (2019) on a keras with tensorflow implementation. The same hyperparameter configuration was used for both models trained on the combination of title and abstract, and, on sentences. Experiments with the implementation of Liu et al. (2018) were taken place as well to confirm the results across different platforms, as the work of Alzaidy et al. (2019) was based on this pytorch implementation of the Bi-LSTM-CRF model. It is important to clear that this model is far from ready for the task of keyphrase extraction and that it does not include any kind of text pre-processing. At the same time, there is no existing public repository of their work to replicate their research. In combination with them omitting to expand on any text pre-processing detail, results in the increase of the replication difficulty.

Also, note that our testing was limited in the pytorch implementation as the pytorch and cuda versions needed to run the model were outdated for our available hardware. Additionally, training with just the CPU was terribly slow, halting our efforts to reproduce the model in a timely manner, especially due to the thesis time restrictions, as the implementation details were inadequate. We mainly used the pytorch implementation in combination with the writers’ details as a guide to build an equivalent tensorflow model.

The sequence-to-sequence model follows the encoder to decoder approach. The encoder is a bidirectional LSTM with 100 units to capture long distance information from both forward and backward word sequence directions. The decoder is a CRF layer that aims to define the most probable sequence of labels by considering the label dependencies through the entire sequence.

The word embedding layer is consisted of embeddings with 100 dimensions. The embedding layer was initialized with GloVe pre-trained embeddings and then continued updating them based on the dataset at hand during training as suggested by Alzaidy et al. (2019). The word embedding layer initialization is done with a GloVe embedding
matrix. Essentially, this matrix is a dictionary that matches the numeric representation of words, produced by keras Tokenizer, to GloVe vectors. The numeric representation is simply a unique number assignment to each unique word. Values start from ‘1’ through ‘dictionary size + 2’. The value ‘0’ is reserved as a padding value and the token ‘<UKN>’ is used for unknown words and is part of the dictionary.

After applying all the aforementioned pre-processing steps, it is worth noting that a GloVe embedding vector is matched to only 30% of the train set vocabulary, which comes to a total of 321,351 unique words. Despite this fact, we manage to reach a 97.69% coverage of all text. Basically, that means that there are many infrequent words. This generates the need to learn word embeddings for them as well, but because their appearances are limited, the embedding layer will not have much information to learn them optimally.

The top 5 most frequent unknown words accompanied by their frequency are: (‘digit_repl’, 831736), (‘notmal’, 14093), (‘refpubl’, 9748), (‘notm’, 5787), (‘multiobjective’, 3168). As we can observe, except from the special digit token ‘digit_repl’, most of the words are mistypes, abbreviated words (e.g. svms, USA), and multi-compound words without any dash in splitting them. The words that are not matched to a GloVe embedding are initialized with the average vector of the whole embedding space, as suggested by one of the creators of GloVe, Jeffrey Pennington, and then we attempt to learn them in the embedding layer during training.

For the training of all models, we used Stochastic Gradient Descent with a batch size of 64 and a learning rate schedule with 0.9 momentum, initial learning rate 0.01 and decay rate 0.5 applied on epoch end. Also, we used gradient value clipping in order to constrain the gradients to have a maximum norm of 5.

In order to mitigate the effects of over-training, in the embedding and in the Bi-LSTM layer, l1 activity regularization was used of values $1e^{-8}$ and $1e^{-10}$, respectively. Also, in the Bi-LSTM layer a recurrent constraint was set restricting the max norm to 2. No dropout was used in the model. The model is trained with respect to CRF loss and accuracy. The experiments were taken place on a Ryzen 5 3600 CPU with 16GB of RAM and were run for a total of 5 epochs. Training the model on title and abstract takes approximately 24 hours, while training on title and abstract split in sentences takes about 5 hours to complete.

We save all the models produced in each epoch. We select the model with the higher $F_1$-score on the validation set as the final model to generate predictions on the test set. The selected parameters were a mix of suggestions of the Alzaidy et al. (2019) work, that were verified through our own testing with different ranges of values, and also, some measures to prevent over-fitting as the model would over-fit after the first couple of epochs. For the model tuning we utilized the keras tuner.
4.4.1 Experiments’ specifics

In the experimental setup, we tried different data processing steps and also we experimented with a cost-sensitive learning method to handle the task’s intrinsic nature of class imbalance. We also conducted an analysis on the type of the dataset and the included information that can be derived from contained text excerpts (full-text or title and abstract). The purpose of this experiment is to identify some guidelines as to what would be great fuel for a model in order to reach an optimal level of performance.

In our comparison setting, we include both full-text information as well as just the title and abstract. We perform a total of six experiments. For the test sets that we keep just the title and abstract, we create two different versions. We use them as continuous text excerpts and as split into sentences. For the full-text test sets, we split them into sentences and into paragraphs. In another experiment, we keep just the first three paragraphs of the full-text. Lastly, we replace the abstract and the full-text by summaries generated with the use of a state-of-the-art deep learning model.

Data Processing Experiments

In view of reproducing the work of Alzaidy et al. (2019), we also explored the difference between exact and partial match text annotation to KP and non-KP. Basically, our experiment showed that annotating with partial string matching increases the number of keywords in the text, which in turn increases the probability of a random word being a keyphrase. In other words, this method increases the density of keywords in the text, coherently unfairly increasing the evaluation metrics. This is unfair because individual words, that have no other keywords as neighbors to form a gold keyphrase, are annotated as keywords simply because they exist in a keyphrase, which might not even exist in the document text.

Considering that the task is approached as keyphrase extraction, these results are more generous results than it should be. To put that into perspective, we calculated the number of keyphrases and non-keyphrases with both partial and exact string matching. The partial match method detects 12,498,484 keywords and 72,408,484 non-keywords, while the exact detects 6,584,027 keywords and 78,247,442 non-keywords. As we can see, the partial match annotation detects almost double the keywords of the exact. Also, the non-keyword to keyword ratio is around 11.88 (78,247,442/6,584,027) for exact and 5.8 (72,408,484/12,498,484) for partial. This means that in the partial match, for each existing keyword, 5.8 non-keywords exist, which is more than half of the exact (lower value means higher keyword density in the text). This translates into the evaluation metrics as the scores of
the partial match are double than those of the exact match annotation. Note that those are keyword counts for both exact and partial matching and not keyphrase counts. For the partial match, we cannot count the number of keyphrases as the notion does not exist. For the exact match, the count of keyphrases is 3,882,211, which results in an average of almost 2 words per keyphrase.

In Alzaidy et al. (2019), after pre-processing, they detect 5,458,743 keywords and 2,806,381 keyphrases, while the total word tokens are 78,441,075. Comparatively with our pre-processing, we detect 1 million more keyphrases, which translates in an 1 more million in keywords. This might be a combination of not including the title in their experiments, the differences in the pre-processing step as well as the 2,560 difference in size of their training dataset. Considering that in neural networks, pre-processing is kept at a minimal when it comes to removing words, we believe that the different pre-processing steps have minimal effect on the difference of the total words counts. Also, assuming the average number of words per document is about 159 (84,831,469/530,390), the difference of 2,560 documents adds up to about 407,040 words. Thus, we speculate that the most substantial difference comes from not considering the title, as it reduces the total tokens by about 5 million (as measured in our experiment set-up).

Even though we tried learning a model on the data without considering the title, we were not able to match their performance. Assuming that the title is not included in their experiments, we expect that the sequence metric performance will not be much affected, but the exact match metrics will. In our experiments, the title is consisted of about a million keywords and around 5 million word tokens. Thus, it is easy to conclude that the total coverage of gold keyphrases is reduced as there are keyphrases that exist in the title and not in the abstract. This could translate into a substantial reduction in evaluation metrics that consider the whole gold keyphrase set, because the total unique keyphrases in the text are reduced by as much as 4.35% (Figures 4.2 and 4.5). At the other hand, evaluation metrics that consider only the gold keyphrases existing are not expected to change.

Class Imbalance

Keyphrase extraction inherently is an imbalanced problem, as from the whole collection of words in a document we seek to find only a small portion of them that captures the document meaning. Thus, it is sound to integrate some mechanism that addresses class imbalance. In our experiments, we utilized a cost-sensitive learning approach, where we accompany the training instances with a weight. For the negative class, we assign a value of 1, while for the positive class we assign a weight based on the keyword to non-keyword ratio. This weight is a product
of the total count of keywords and non-keywords found in the text, which equates to 11.88 \((78,247,442/6,584,027 = 11.88)\). As an addition, we tried using a weight given from the count of keyphrases and the count of non-keywords, which turns to be a more aggressive approach as the number of keyphrases is smaller than that of the keywords. This weight equals to 20.15 \((78,247,442/3,882,211 = 20.15)\).

**Evaluation Dataset Experiments**

The evaluation dataset experiments aim to extract and combine the parts of the documents that contain keywords in order to introduce new keyphrases that cannot be found in the title nor the abstract. A rather simple method to add more words would be to utilize the full-text of documents. Though, this method comes with great limitations as the length of most documents usually extends into several pages that leads to great memory requirements very quickly. A workaround would be to split the full-text into smaller text excerpts, e.g. sentences or paragraphs, but this method is still limited by the great amount of irrelevant words introduced. Also, if the chosen length of the excerpts is too small, the Bi-LSTM will not be able to fully utilize its long learning mechanism and thus, some performance might be deprived.

**Full-text split in paragraphs** We split the full-text of the ACM, NUS and SemEval-2010 test datasets into paragraphs. Though, we could not split the documents into the actual paragraphs as there were no paragraph identifiers in the ACM and NUS datasets. Instead, we split them into sentences, and then, we combine the sentences till the length reaches 400 (same length as our training data). With this method, it is guaranteed that all of the paragraphs will have length below 400. We found that after splitting the ACM dataset into paragraphs, there are 53,083 paragraphs with keyword to non-keyword ratio 92.65. At the other hand, the NUS dataset has 4,744 paragraphs with keyword to non-keyword ratio 35.7. More statistics can be seen in Table 4.5. The SemEval-2010 datasets has 5,171 paragraphs and 37.17 keyword to non-keyword ratio.

<table>
<thead>
<tr>
<th></th>
<th>ACM</th>
<th></th>
<th>NUS</th>
<th></th>
<th>SemEval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>length: 220</td>
<td>length: 400</td>
<td>length: 220</td>
<td>length: 400</td>
<td>length: 220</td>
</tr>
<tr>
<td>a'</td>
<td>kp</td>
<td>-</td>
<td>168,345</td>
<td>-</td>
<td>36,392</td>
</tr>
<tr>
<td></td>
<td>kw</td>
<td>-</td>
<td>261,284</td>
<td>-</td>
<td>55,483</td>
</tr>
<tr>
<td></td>
<td>non-kw</td>
<td>-</td>
<td>15,598,434</td>
<td>-</td>
<td>1,369,180</td>
</tr>
<tr>
<td>b'</td>
<td>kp</td>
<td>26,537</td>
<td>42,692</td>
<td>5,298</td>
<td>8,682</td>
</tr>
<tr>
<td></td>
<td>kw</td>
<td>46,099</td>
<td>72,257</td>
<td>8,442</td>
<td>13,560</td>
</tr>
<tr>
<td></td>
<td>non-kw</td>
<td>1,148,941</td>
<td>2,152,343</td>
<td>105,160</td>
<td>197,356</td>
</tr>
</tbody>
</table>

**Table 4.5:** ACM, NUS and SemEval statistics for the paragraph split experiment, where a’: full-text split in paragraphs, and, b’: 3 first paragraphs (kp: keyphrase, kw: keyword, non-kw: non-keyword)
First 3 paragraphs Another approach we experimented with, considers only the first three paragraphs of the full-text documents of ACM, NUS and SemEval-2010. Again, the approach of splitting the documents into paragraphs is the same as described above. Except from the 220 length, we also experimented with a length of 400 to match the sequence size of the training set. This decision was based on an observation of the authors of Papagiannopoulou and Tsoumakas (2020) that generally the vast majority of keyphrases are located within the first introductory paragraphs. It is worth noting that there were some instances where documents contained a single sentence with length of over the 220 mark. Those were cropped to a length of 220.

To put that into perspective, for the ACM dataset with length 220, we found 6,910 paragraphs with keyword to non-keyword ratio 24.92, which is almost a third of the whole full-text in paragraphs, making it more keyword dense, but the total keywords are considerably less. For length size of 400, for the same number of paragraphs, the keyword to non-keyword ratio is 29.78, which is slightly higher than that of the 200 length, but the keywords are increased by about 40%. Those observations hold for the other datasets as well.

![Figure 4.5: Percentage of keyphrase coverage per information source in the kp20k-test set (abstract and title)](image)

Next, we compare the gold keyphrase coverage between all of our test sets. Note that the following statistics are produced by checking the existence of the gold keyphrases in the text, without considering the occurrence frequency. After pre-processing, the kp20k-test set with both the title and abstract covers 63.43% (66,893 keyphrases) of the total gold keyphrases, which are 105,459 (Figure 4.5). Again, the newly
introduced keyphrases by the title are the same percentage level as the train set (4.43% of the total gold keyphrases).

In contrast, from Figure 4.6, the combination of the title and abstract of the NUS and ACM test set covers about 10% less gold keyphrase coverage compared to the kp20k-test set, while the SemEval around 20% less. Introducing the full-text in addition to the title and the abstract, the gold keyphrase coverage increases substantially, especially for the case of NUS and SemEval. The high percentage on the NUS and SemEval full-text documents could be attributed to the double annotation from both the authors and readers. It seems that in both datasets, the readers were instructed to provide words from inside the text or they were not field experts thus providing in-text keywords. In contrast, authors seem to introduce new keywords that reduce the coverage, as we can also see from the ACM full-text coverage.

In the paragraph experiment, a paragraph is defined based on its tokens size. We take two approaches. In the first approach, a paragraph has 220 words, which is chosen after the upper-average length of the title and abstract training set. In the second approach, a paragraph has 400 words to match the sequence size of the training data. From figure 4.7, we see that the first 3 paragraphs, with paragraph size 220, increase the keyphrase coverage by 10% comparing to their title and abstract counterpart. Also, the first 3 paragraphs with length 400 seem to cover the vast majority of keyphrases as the keyphrase coverage is increased by 20% compared to the title and abstract versions.
4.4. Experiments Setup and Parameter tuning

Full-text summary Further, we explored a methodology that aims to introduce new words, as the words contained in the title and the abstract of document, more often than not, do provide a limited number of keyphrases that do not cover the full list of the provided gold keyphrases set. We achieve that by using state-of-the-art deep learning models that summarize text. We apply this model to the full-text datasets (NUS, ACM and SemEval). Note that we summarize the combination of the abstract and the full-text without including the title. The generated summary is then combined with the title and their combination is passed through our keyphrase extraction model. The difference of this approach is that it aims to add meaningful words that have great chance of being keyphrases. In theory, this way the generated text would be more condensed when it comes to important words.

We experimented with both an extractive and an abstractive summarization model for the creation of summaries. For the extractive approach, we deployed the pre-trained *distillated roberta* model "distilroberta-base-ext-sum" from the *TransformerSum*\(^2\) library. At the time of writing, *TransformerSum* suggests using this pre-trained model as it achieves state-of-the-art performance in a timely manner. *Distillated roberta* is a version of *RoBERTa* (Liu et al., 2019), which is based on *DistilBERT* Sanh et al., 2019. Basically, this version is a light, fast and small variation of the original RoBERTa, that achieves a time speed-up of 50%, while retaining 95% performance of the original model. As our abstractive option, we used the *LongFormer-Encoder-Decoder* (LED) (Beltagy et al.,

\(^2\)https://github.com/HHousen/TransformerSum
Chapter 4. Experiments

We experimented with both "led-base-16384" and "led-large-16384" pre-trained models for processing long sequences, but we did not manage to improve the keyphrase coverage. Though, this could change by fine tuning the pre-trained models in order to create a properly optimized set-up. For this reason, we conducted experiments only with the summaries produced by the extractive distilled roberta model.

As we can free from Figure 4.6, the extraction summarization model (distillated roberta) achieves better keyphrase density than the human written abstract. In our experimental set-up, it achieves an increase of 4% to 8% in gold keyphrase coverage across all three full-text test datasets. As the field of automated text summarization improves, especially the abstractive models, the keyphrase density is expected to improve as well. That said, the overall density of keyphrases in a summary will be always restricted by its length. General guidelines for selecting a proper length is yet to be defined. Note that for the model distillated roberta, we set the number of the summary sentences to 14 in order to achieve a total number of words that ranges from 300 to 400 to be inline with the 400 sequence padding of the train set. Time needed to produce summaries for each full-text dataset can be seen in Table 4.6.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SemEval</td>
<td>244</td>
<td>9</td>
</tr>
<tr>
<td>NUS</td>
<td>211</td>
<td>7.5</td>
</tr>
<tr>
<td>ACM</td>
<td>2,304</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 4.6: Time needed to produce summaries for each full-text test dataset with distillated roberta

Lastly, considering the analysis above, it is easy to understand that the evaluation of keyphrase extraction models should consider the keyphrase coverage of the documents as it may pose major limitations on extractive models’ performance, if the keyphrase coverage is poor. Considering this analysis, datasets with low keyphrase coverage throughout their full-text might not be suitable for keyphrase extraction, but that might not be the case for keyphrase generation.

4.4.2 Evaluation methods

The evaluation of keyphrase extraction models is very complicated and many could argue that till now there is no proper method for evaluation available. The most reliable method would be that of manual evaluation, but even this method has big flows. The process of evaluation is heavily affected by human subjectivity and even in this case we could not reach to a commonly agreed level of efficiency to take clear cut decisions of the better model. Studies, like Papagiannopoulou and
Tsoumakas, 2020, observed that taking the average of the exact and partial match scores comes close enough to that of the manual evaluation. In our experiments, we compute the $F_1$-score, recall and precision for the positive label (keyphrase-KP) with three different ways. The first one is exact matching, the second is partial matching, and, the third is sequence evaluation.

For the exact and partial matching, we first extract the predicted keyphrases from the pre-processed text by using the predicted sequence. A keyphrase is formulated by acquiring the sequence of words that are predicted as keywords. Precisely, words that are predicted as keywords and are located sequentially one after the other in a continuous manner in the sequence are considered as a single keyphrase. Finally, we delete duplicate predicted keyphrases and we compare them to the gold set.

For the comparison of those keyphrase sets, we first lower case and stem them, as suggested in the literature, and then we perform string matching. Note that the same pre-processing steps where applied to both sets to have a fair comparison.

The sequence evaluation is the method of evaluation adopted by the authors of Alzaidy et al. (2019) in their pytorch version. The notion of sequence evaluation exists only on models that map a sequence of inputs to a sequence of predictions of the same size. This method is especially popular on many other sequence prediction problems, such as named-entity recognition (NER) and part-of-speech (POS) tagging. To formulate the values of the metrics, this method compares the predicted sequence with the gold sequence. Essentially, each document is associated with a label sequence, where each label is matched to a word of the text and has information on whether the word is a keyword or not. Sequence evaluation can be deemed as a stricter kind of partial string matching. Also, by nature, it considers only keyphrases that appear in the text. Note that for this evaluation method, we remove the padding values before calculating the metrics.

Some clarifications on the evaluation methods follows. For the sequence evaluation, we do not take into account the padding values as they are removed so that the sequence would match its initial size. Also, before calculating the metrics with the exact and partial match evaluation on the test datasets that are split into sentences and paragraphs, we first re-assemble the original documents by combining each sentence/paragraph to its corresponding document. This is performed on both the test and the predicted keyphrase set.

Despite the sequence evaluation being a popular method, we cannot compare it with the results from the exact and partial matching, or, as a matter of fact, we cannot compare any of the used methods with another different other than itself. Thus, to fairly compare how the efficiency of the Bi-LSTM-CRF model stacks-up with approaches that were evaluated with the traditional and most popular exact string
matching, as well as those using partial matching, we utilize all three of aforementioned methods to cover a wide range of different approaches.

New Evaluation Metric

The metric aims to handle one of the problems the exact matching approach has. Specifically, let "keyphrase extraction" be a gold keyphrase and "keyphrase extraction task" a predicted keyphrase. Exact match will consider them as different, whereas our approach will consider them as the same.

The calculation of $F_1$-score, recall and precision with this methodology is focused on finding the values of two lists. Those values are binary (0 and 1) and indicate whether keyphrases from the gold set are matched with keyphrases of the predicted set and vice versa. The first list represents the gold keyphrase set. This list contains values that match to the number of gold keyphrases in the corresponding document. Basically, each keyphrase of a document is paired with its corresponding value. This value denotes whether the gold keyphrases matched with a predicted one. The second list represents the predicted keyphrase set. The notion is the same as for the gold list, but here the values indicate whether the predicted keyphrases have a gold keyphrase match.

From here, we compute true positives (TP) by summing all instances of the positive class in the gold keyphrase list (gold keyphrases that matched to a predicted). The false negatives (FN) are calculated by summing the instances of the negative class in the same list (gold keyphrases that did not match to a predicted). Lastly, for false positives (FP), we count the instances of the negative class in the predicted list (predicted keyphrases that did not match to a gold).

Now that we know the target, and how to calculate the metrics using it, we will describe the methodology that we use to fill the lists. The first dimension of those lists has length equal to the total document count. For each list respectively, the second dimension is equal to the number of predicted and gold keyphrases that each document has. Both of them are initialized with 0s and the positive class is assigned with the following method. We compare each gold keyphrase of a document with every predicted keyphrase of that document. From this comparison we calculate two numbers for each predicted keyphrase. The first number represents the keyword to non-keyword ratio of the predicted keyphrase. It results from the division of the number of words in the gold keyphrase that are covered by a predicted keyphrase with the total number of words in the predicted keyphrase.

$$\text{keyword to non-keyword ratio} = \frac{\text{# of gold matched keywords}}{\text{# of total words in predicted kp}} \quad (4.1)$$
The second number is the gold keyword coverage of the gold keyphrase. It results from the number of words in the gold keyphrase that are covered by a predicted keyphrase divided by the total number of words in the gold keyphrase.

\[
gold\text{ keyword coverage} = \frac{\# \text{ of gold matched keywords}}{\# \text{ of total words in gold kp}} \quad (4.2)
\]

Given those two numbers for a specific pair of a gold and a predicted keyphrase, we compute their average. Then, from all the pairs a gold keyphrase has with all the predicted keyphrases, we select the predicted keyphrase with the maximum average. Lastly, if this average is bigger than 0.5, the positive class is assigned to that gold keyphrase in the gold keyphrase list. The non-keyphrase to keyphrase ratio is used as a counter weight to select the appropriate match in the case of having both an exact match predicted keyphrase and a long predicted keyphrase that includes the whole gold keyphrase. An example of this process is described in Fig. 4.8. For the predicted keyphrase list, we assign the positive class to keyphrases that have matched to a gold keyphrase. This can be achieved by using the argmax function every time a gold keyphrase match is found with the described procedure. Basically, the argmax finds which is the predicted keyphrase that matched to the gold keyphrase. Note that for the string matching, we used the same processing steps we used for exact and partial string matching. We stemmed, lower cased and utilized the pre-processing steps described at Section 4.3.

As one can see, this metric allows for parameter tuning. Specifically, we can tune the strictness of the string matching by tuning two specific
values. First, we could adjust the condition which decides whether we have a string match. By setting a value larger than 0.5, we make the matching stricter, while for a value smaller than 0.5, we allow for a looser matching. Also, instead of simply filling the lists with binary values (0 and 1), we could substitute 1 with the gold keyword coverage of the gold keyphrase or even experiment with the keyphrase to non-keyphrase ratio of the predicted keyphrase or some other value. In this case, true positives, false negatives and false positives can be calculated by simple adding the non-zero values instead of the aces. Also, this metric allows us to retain the syntactic information of a sentence by considering the word ordering while in the condition that matches the gold and the predicted keywords.

4.4.3 Experiments’ Results

In this section, we will present and discuss the results of our experiments. First, we will start with the results of the Bi-LSTM-CRF replication, and then, we discuss the effects of cost-sensitive learning. Finally, we will conclude with the experiments on different sources of information for the evaluation datasets, which is our main focus. We evaluate our methods with 3 popular full-text datasets. In our experiments, we utilized 3 category representative algorithms, two of which are popular unsupervised methods and one is a state-of-the-art supervised learning model. We chose the Bi-LSTM-CRF model for the deep learning category, the TfIdf for the statistical category and the MultipartiteRank for the graph category.

Bi-LSTM-CRF Replication Results

The scores on the kp20k-test dataset of the best model trained on the title and abstract of the kp527k dataset can be seen in Table 4.7. We managed to achieve a score that is only 3.67% away from the score achieved in Alzaidy et al. (2019). Despite using the proposed hyper-parameters of the Alzaidy et al. (2019), we could not match its exact performance, neither on the keras nor the pytorch implementation. Even though we could not replicate their exact score, we did get very close. We also tried the bucket method they used and to further optimize hyper-parameters, as described in 4.4, but with no success. According to our experiments, this could be attributed to three things. First, considering the counts of keyphrases and words tokens of the train as well as the validation and test datasets, it seems that they did not include the title in their experiment (4.1 and 4.5). Thus, their performance could be matched if the hyper-parameters were tuned for the data without the title, but due to the thesis time limitation, the multiple and time consuming pre-processing experiments to reach to this observation and the high
demand for time needed to tune such model, we were not able to explore further. The second reason could be due to the difference in pre-processing, as the authors do not state anything about the actions performed in this part. Another factor could be due to the 3 thousand training instances difference between the training datasets.

The comparison scores of the sentence model are presented at the same table. More results of the sentence model are presented in the Appendix table B.1. In this case, we managed to achieve a score that is 6.87% away from the original model. Note that the bigger gap in the sentence model might be due to that the model hyper-parameters were optimized for the continuous title and abstract documents.

<table>
<thead>
<tr>
<th></th>
<th>S%</th>
<th>E%</th>
<th>P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Bi-LSTM-CRF_{full}</td>
<td>38.14</td>
<td>13.73</td>
<td>40.80</td>
</tr>
<tr>
<td>Original Bi-LSTM-CRF_{full}</td>
<td>41.81</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Our Bi-LSTM-CRF_{sent}</td>
<td>28.76</td>
<td>15.03</td>
<td>30.12</td>
</tr>
<tr>
<td>Original Bi-LSTM-CRF_{sent}</td>
<td>35.63</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.7: F1 based on sequence (S), exact (E) and partial (P) evaluation for the original Bi-LSTM-CRF approach and our implementation. Bi-LSTM-CRF_{full} is trained on whole titles and abstracts, whereas Bi-LSTM-CRF_{sent} operates at the sentence level.

It is worth noting that the comparison they made with other state-of-the-art models does not stand as the evaluation metrics were computed differently. Generally, most of the works in the literature, adopt the exact match approach that considers all the gold keyphrases. This holds for the models they used for comparison purposes as well. Contrariwise, their model scores are computed using the sequence evaluation. The metrics calculated with sequence evaluation have a major numeric gap in comparison with those computed with exact match as the sequence evaluation can be seen as a stricter variation of the partial match. The differences are further demonstrated in the appendix table A.1. Even though Bi-LSTM-CRF does not outperform the state-of-the-art copyRNN, it manages to outperform the graph-based approach MultipartiteRank and marginally outperform TfIdf, as presented in the following sections.

Last but not least, as the sequence F1-score is not widely used in the literature, we decided to select the final model to run our experiments based on the best exact match F1-score. Note that that model with the best sequence F1-score did not achieve the highest exact match F1-score as well. Thus, the final model we selected achieves 15.47% exact match F1-score and 32.36% sequence F1-score. Note that even though the sentence model was pretty close on the exact match F1-score of
the aforementioned model, it was not the best comparing the absolute numbers, thus it was not selected.

**Class Imbalance**

For the case of the sentence model, both cost-sensitive approaches produce similar performance, with the keyword to non-keyword ratio (11.88) resulting in a slightly better score, which could be due to the non-deterministic nature of neural networks. Comparing the best scores of the cost-sensitive model with the best scores of a simple model, we noticed a general increase of 1% in the $F_1$-score in behalf of the cost-sensitive model. For the case of the title and abstract model, the cost-sensitive approach appears to be malicious as the produced models have very high validation loss resulting in bad performance. Thus, for this type of model, it seems that handling imbalance with cost-sensitive learning is not beneficial. Next, we present and discuss the results of using different text excerpts from the documents of the evaluation sets.

**Evaluation Dataset Experiments**

We evaluated all of our experiments on three different full-text datasets with the $F_1$ calculated in multiple methods, which include exact and partial string match, the sequence $F_1$, and, the semi-exact string match. For the exact, partial and semi-exact string match, we computed two different versions. The first, considers all the keyphrases of the gold set. The second, considers only the gold keyphrases that exist in the text. In this section, we will focus on the first version, as in general, we think that the $F_1$-score that considers all the gold keyphrases is more important for the task as we can see the total overall performance that can be achieved with the different text sources of information. The results of the second version and the sequence $F_1$-score can be found in the appendix A. We compare our methods with the staple title and abstract, as well as full-text variations which is used in the vast majority of the publications in the literature.

For the latter two, we retrieve the 10 best predicted keyphrases and calculate metrics based on those. Also, for TfIdf, we compute term frequency on the full-texts for each method. Further, we use the default settings of the `pke`'s library documentation, which are word stemming, punctuation and stopword removal, and, selection of 1-3-grams. Default settings are also used for the MultipartiteRank, which are word stemming, punctuation and stopword removal, and, removal of all words except proper nouns, nouns and adjectives. Lastly, the model parameters are alpha=1.1, threshold=0.74 and method='average'.
4.4. Experiments Setup and Parameter tuning

Bi-LSTM-CRF

From the table 4.8, the best method for the Bi-LSTM-CRF model seems to be the title and abstract split in sentences (ABSE) and then the first three paragraphs with paragraph length 220 (3P220). The ABSE method seems to improve F1-score by about an average 4.4% over the popular title and abstract method. The 3P220 method seems to improve F1-score by about an average 3% over the title and abstract (TA) method. It is worth noting, that overall, the ABSE, 3P220, AS and FP outperform the plain TA, which is the most used piece of text.

<table>
<thead>
<tr>
<th>Bi-LSTM-CRF</th>
<th>SemEval E%</th>
<th>SemEval P%</th>
<th>NUS E%</th>
<th>NUS P%</th>
<th>ACM E%</th>
<th>ACM P%</th>
<th>KP20K-test E%</th>
<th>KP20K-test P%</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>10.27</td>
<td>19.59</td>
<td>12.85</td>
<td>27.01</td>
<td>14.75</td>
<td>32.54</td>
<td>15.47</td>
<td>33.14</td>
</tr>
<tr>
<td>ABSE</td>
<td><strong>16.12</strong></td>
<td><strong>32.49</strong></td>
<td><strong>18.18</strong></td>
<td><strong>35.95</strong></td>
<td><strong>17.89</strong></td>
<td><strong>38.68</strong></td>
<td><strong>18.76</strong></td>
<td><strong>40.35</strong></td>
</tr>
<tr>
<td>FP</td>
<td>15.68</td>
<td><strong>34.86</strong></td>
<td>14.36</td>
<td>31.85</td>
<td>8.20</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FS</td>
<td>13.23</td>
<td>31.63</td>
<td>10.24</td>
<td>22.61</td>
<td>6.75</td>
<td>17.45</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TS</td>
<td>9.72</td>
<td>19.19</td>
<td>12.75</td>
<td>26.49</td>
<td>13.89</td>
<td>31.67</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>11.82</td>
<td>22.64</td>
<td>14.47</td>
<td>29.95</td>
<td>15.13</td>
<td>34.54</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>14.29</td>
<td>26.40</td>
<td>16.77</td>
<td>33.71</td>
<td>15.65</td>
<td>35.24</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>8.75</td>
<td>18.65</td>
<td>10.20</td>
<td>23.89</td>
<td>13.78</td>
<td>33.63</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.8: F1 based on Exact (E) and Partial (P) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

For the ABSE experiment, we also included the kp20k-test dataset to further solidify the conclusion. In this case, we observe that the Bi-LSTM-CRF model achieves better exact match F1-score than the TA on all four datasets. The ABSE method outperforms the TA by a substantial amount. This could mean that the Bi-LSTM-CRF is unable to retain past information from longer sequences, which is a common issue for sequence-to-sequence models. This is further fortified by the results of the sentence model in the full-text as they are close or even higher than the TA. Results of the sentence model can be seen in Appendix table B.1. To solve this problem, it may be worth experimenting with the attention mechanism, that has risen in popularity in recent years. As we can observe, for the SemEval and the NUS datasets, the performance difference comparing to the TA is higher than that on the ACM and KP20K-test. Thus, it seems that the method achieves even better scores on datasets with gold keyphrases annotated by both writers and readers, which is to be expected as there are more potential keyphrases.

Also, the title and summaries (TS) method achieves close but inferior performance to the TA. The summaries manage to increase the overall keyphrase presence comparing to human written abstracts by
utilizing summaries that have about the same length as the abstracts. The increase ranges from 4% to 8%, but this is still not enough to achieve better performance than the plain title and abstract. Though, by considering the union of the predictions on the TA and TS methods (AS), we were able to improve the F$_1$-score by an average 1.18%, comparing to the plain TA scores. Despite the improvement of the AS method, 3P220 still achieves better performance by a considerable amount.

Considering the results in the NUS and Semeval datasets in comparison with ACM, the full-text in paragraphs (FP) method seems to be a good choice when keyphrases are assigned by both writers and authors. For the full-text split in sentences (FS), the performance is reduced in comparison with the FP, which goes to show that positional information of keyphrases captured through the memory mechanism is important. From our analysis and in the literature, it is observed that most of the keyphrases exist earlier in the text and that as we move to the end, they become scarcer. This is not taken into account by the model at all as its memory mechanism has only information for one sentence. Contrariwise, the FP method allows for some usage of the memory mechanism, thus the improvement on the results. Note that this effect is mitigated for ABSE, because of the smaller length and the increased keyphrase density abstracts have over full-texts.

**TfIdf**

From the table 4.9, we can observe that for the TfIdf algorithm, the title and summary (TS) method is the best among all methods, including the title and abstract (TA) and the full-text (F). It achieves about 2% average increase on the exact F$_1$-score. This could be a combination of the higher keyphrase density of the abstract produced by the summarizing model and the selection of only the top-10 predicted keyphrases. This is further validated from results of the table 4.10, as for the top 20 predicted keyphrases, the difference between the TS and full-text is closed down, while the performance increase of TS over TA remains. For both F$_1@10$ and F$_1@20$, the union of the abstract and the summaries (AS) is close to the TS, but achieves slightly lower scores. Though, it still outperforms the TA and for F$_1@10$, the full-text as well.

Interestingly enough, for the first 10 predicted keyphrases, the full-text (F) gives pretty close results to the title and abstract, which is rather unusual for statistical methods as they tend to benefit from larger text sizes. This may be caused for a couple of reasons. Firstly, we calculate the term frequency of all methods on the full-text of the documents as we consider it available for use. Secondly, the small text size of the abstract in addition to the increased keyphrase density might benefit metrics for lower numbers of retrieved predicted keyphrases. This is
4.4. Experiments Setup and Parameter tuning

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E%</td>
<td>P%</td>
<td>E%</td>
</tr>
<tr>
<td>TA</td>
<td>14.30</td>
<td>31.20</td>
<td>17.85</td>
</tr>
<tr>
<td>F</td>
<td>14.03</td>
<td>28.93</td>
<td>19.31</td>
</tr>
<tr>
<td>TS</td>
<td>16.15</td>
<td>32.45</td>
<td>20.12</td>
</tr>
<tr>
<td>AS</td>
<td>16.04</td>
<td>34.85</td>
<td>18.99</td>
</tr>
<tr>
<td>3P220</td>
<td>13.43</td>
<td>32.49</td>
<td>13.93</td>
</tr>
<tr>
<td>3P400</td>
<td>16.02</td>
<td>36.20</td>
<td>17.14</td>
</tr>
</tbody>
</table>

Table 4.9: $F_1@10$ based on Exact (E) and Partial (P) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

Further validated from the results of the top 20 predicted keyphrases (table 4.10), the full-text achieves better performance for a bigger number of retrieved predicted keyphrases.

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E%</td>
<td>P%</td>
<td>E%</td>
</tr>
<tr>
<td>TA</td>
<td>13.45</td>
<td>34.58</td>
<td>14.46</td>
</tr>
<tr>
<td>F</td>
<td>15.82</td>
<td>34.78</td>
<td>19.78</td>
</tr>
<tr>
<td>TS</td>
<td>15.68</td>
<td>36.09</td>
<td>17.35</td>
</tr>
<tr>
<td>AS</td>
<td>14.25</td>
<td>35.81</td>
<td>14.95</td>
</tr>
<tr>
<td>3P220</td>
<td>9.81</td>
<td>28.14</td>
<td>9.02</td>
</tr>
<tr>
<td>3P400</td>
<td>12.13</td>
<td>31.50</td>
<td>11.97</td>
</tr>
</tbody>
</table>

Table 4.10: $F_1@20$ based on Exact (E) and Partial (P) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

To sum up, for the first 10 predicted keyphrases, the TS method outperforms the TA method, while the AS method does not seem to provide any further performance improvement over the TS on the TfIdf. TS outperforms the full-text across all datasets while the AS seems to provide some decent improvement or at least similar scores. Overall, the best choice for TfIdf would be the TS method as it outperforms all others by a substantial amount. Also, for the first 3 paragraphs, length 400 (3P400) is better than the length 220, which is to be expected as statistical methods benefit from more text as it introduces more keyphrases. For the first 20 predicted keyphrases, the same observations hold, but for the case of TS, even though the strong performance is retained, the difference from the full-text is mitigated. Note that for all the experiments with TfIdf, we calculated the term frequency on the full-text of the documents.
Chapter 4. Experiments

MultipartiteRank

From the tables 4.11 and 4.12, we can observe that for the case of MultipartiteRank algorithm and both $F_{1@10}$ and $F_{1@20}$, the title and summary (TS) method is marginally better than the title and abstract (TA) across all datasets. For this algorithm, it seems that the increased density in gold keyphrases that the summaries offer over the title and abstract has minimal effect on performance. Even though the gold keyphrase density is increased by 4% to 8%, it is not enough to increase substantially the performance, but this could change if the density is further increased. Such thing could be possible in the future as automated summarization methods become more efficient.

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E%</td>
<td>P%</td>
<td>E%</td>
</tr>
<tr>
<td>TA</td>
<td>13.74</td>
<td>34.37</td>
<td>15.44</td>
</tr>
<tr>
<td>F</td>
<td>13.51</td>
<td>34.32</td>
<td>15.81</td>
</tr>
<tr>
<td>TS</td>
<td>14.46</td>
<td>35.79</td>
<td>15.67</td>
</tr>
<tr>
<td>AS</td>
<td>15.02</td>
<td>36.66</td>
<td>15.78</td>
</tr>
<tr>
<td>3P220</td>
<td>12.84</td>
<td>33.49</td>
<td>12.54</td>
</tr>
<tr>
<td>3P400</td>
<td>13.44</td>
<td>35.12</td>
<td>13.48</td>
</tr>
</tbody>
</table>

Table 4.11: $F_{1@10}$ based on Exact (E) and Partial (P) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

Considering the union of predictions of the TA and TS (AS), it seems that for the case of $F_{1@10}$, performance is improved in two out of the three datasets comparing to the individual performance of TA and TS, but this does not hold for $F_{1@20}$ as well. It seems that for the ACM dataset, AS is affected by the fact that data is annotated only by writers. The reduced number of gold keyphrases bounds the variety of the predicted keyphrases, thus there is a smaller chance of introducing new keyphrases to the TA with the summary.

Full-text (F) seems to perform similarly to the TA method for $F_{1@10}$, but the TA method seems slightly more superior as it achieves a decent performance increase on the ACM. For $F_{1@20}$, full-text manages to achieve better performance on two out of the three test datasets, again lagging behind on the ACM. This might be due to the writer annotation. More testing is needed to confirm that and to study the effect of different numbers of retrieved keyphrases on the $F_{1}$-score.

Overall, our methods do not seem to effect in a substantial way the performance on the MultipartiteRank, but more experiments should be done to come to a conclusion for the whole category of the graph based methods. For MultipartiteRank, it seems that TS has potential as
4.4. Experiments Setup and Parameter tuning

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E%</td>
<td>P%</td>
<td>E%</td>
</tr>
<tr>
<td>TA</td>
<td>14.64</td>
<td>35.91</td>
<td>14.26</td>
</tr>
<tr>
<td>F</td>
<td>15.31</td>
<td>38.12</td>
<td>16.12</td>
</tr>
<tr>
<td>TS</td>
<td>14.99</td>
<td>36.82</td>
<td>14.74</td>
</tr>
<tr>
<td>AS</td>
<td>14.71</td>
<td>35.90</td>
<td>13.54</td>
</tr>
<tr>
<td>3P220</td>
<td>11.08</td>
<td>28.48</td>
<td>9.38</td>
</tr>
<tr>
<td>3P400</td>
<td>11.78</td>
<td>30.51</td>
<td>10.44</td>
</tr>
</tbody>
</table>

Table 4.12: F1@20 based on Exact (E) and Partial (P) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

it performs slightly better than TA and full-text considering the F1@10. More summarization methods should be applied to such experiments to ultimately compare their effectiveness. Also, the effectiveness of our methods should be explored further for different number of retrieved predicted keyphrases to study performance fluctuations.

Evaluation Dataset Experiments Discussion

The experiments with the different sources of information were pretty insightful. The position of a sentence/paragraph in a text is important, because in general, most keyphrases are located early in the text. This can be further validated from our data analysis of the experiment where we select the first three paragraphs of the full-text. Especially for the case of documents annotated only by theirs author, the effects are exemplified as the vast majority of gold keyphrases are covered within the first three paragraphs including the abstract. The coverage percentage in the first three paragraphs drops a little bit when there are writers as well as reader annotators, but it is still substantially higher than that of the title and abstract.

For deep learning methods, it seems that the plain title and abstract combination is a very strong baseline, but if no attention mechanism is used, it might be worth splitting the title and abstract combination in sentences as it seems to boost performance considerably, at least for the Bi-LSTM-CRF model. Also, the first three paragraphs with 220 paragraph length is a strong runner up as it outperforms the title and abstract as a whole across all datasets. Another important observation is that even though the full-text covers a much wider range of the gold keyphrases than the plain title and abstract, deep learning models are not able to utilize it due to the extreme amounts of memory required.

Generally, when using full-text on unsupervised methods with a lower number of total retrieved predicted keyphrases, such as 10 in our
setting, it seems that some non-keyphrases take the place of keyphrases and that many keyphrases are left out. This happens because in the full-text there are many non-keyphrases that might appear as representative keyphrases, i.e., they may appear frequently. Thus, for a smaller amount of retrieved predicted keyphrases, full-text’s performance is reduced compared to a bigger number as noted by our results for $F_1@10$ and $F_1@20$. Subsequently, when using the full-text, it is suggested to use a higher number of retrieved predicted keyphrases. Evaluation results from the empirical study of Papagiannopoulou and Tsoumakas (2020) further indicate the validness of the observation.

For statistical methods, the deep learning generated summary seems to be superior and a clear winner on all test datasets and among all other methods. The small size of the text excerpt in combination with the increased gold keyphrase density seems to work better than the commonly used human written abstract and the whole full-text, for a low number of retrieved predicted keyphrases at least. For a bigger number of retrieved predicted keyphrases, full-text is a strong performer, while the summary is very close or even better in the case of ACM. Considering results from MultipartiteRank in our experiment setting, it seems that the text excerpts do not make a big difference for graph based methods and the best method would be the most easily accessible, which generally would be either the title and abstract or the full-text. Further experiments with additional representative algorithms of each category should be carried to confirm those indications.

**Semi-exact match - New Evaluation Metric**

In our experiments, we used a threshold value of 0.5 and for the gold keyphrase list, instead of marking existing gold keyphrases with the positive class, we used the gold keyword coverage ratio of each gold keyphrase. The gold keyword coverage ratio is the number of words of the gold keyphrase that exist in the matched predicted keyphrase divided by the total number of words in the gold keyphrase, as described in 4.4.2. The results for this metric can be found in the appendix A.

With the current parameter configuration of the semi-exact match, it seems that the scores tend to be closer to those of the partial match. Though, this case is not universal as there are instances were it is closer to the exact match scores. As the evaluation metrics were not the main contribution of this work, we did not include an in depth scientific exploration and evaluation of each aspect of the semi-exact match. Especially due to the extended parameter configuration and the multiple variations that the metric allows, it was left out as future work for a work that focuses on metrics of the keyphrase extraction task.
Chapter 5

Conclusion

In today’s world, there is a chaotic abundance of data, but concurrently, people become less patience and their time is limited to explore them all. Thus, it becomes apparent that techniques are required to speed up the process. For those reasons, the task of automatic keyphrase extraction is an essential problem, especially for the scientific community. A good set of keyphrases, that accurately represent a document’s topic, can give valuable insights to people with a quick glance. Even better, this process can be further expanded by utilizing search engines. Another domain, that keyphrase extraction can help, is document indexing from search engines. With a proper set of keyphrases, search engines can improve the quality of the retrieved results. At the other side of the coin, this could benefit the writers as keyphrases help with document indexing and thus it could help them propagate their work to the relevant audience. Concluding our work, we would like to compile the key notes and findings that we came across during the literature review as well as our experiments.

Papagiannopoulou and Tsoumakas, 2020 observed that statistical methods perform generally better than the graph-based methods when applied to full-text scientific publications. The observation applies on both exact and partial matching evaluation methods, even though partial match favors graph-based methods more. Statistical methods benefit more on the full-text, managing to form a better profile of the difference between keyphrases and non-keyphrases, while graph-based models are adversely affected by the excess information which results in the inability capture the correlations between words, despite the neediness for higher computation times. The opposite is true for datasets that provide only the title and the abstract, meaning that graph-based models outperform statistical ones. Generally, for short texts the graph-based category performs better with no considerable time overhead. Positional information plays a miniscule role in short documents that are constituted by a few lines and word frequency is unreliable as the text excerpts are inadequate.

In addition, for cross-domain collections of documents, both graph-based and statistical approaches work equally well considering the performances derived from partial and exact evaluation. The only
Chapter 5. Conclusion

parameter, that is worth accounted for the method selection, is the document length that affects directly the computation cost. Lastly, when word embeddings are deployed, given the available resources, they should be trained on the working corpus as they generally perform better than pre-trained models (e.g. GloVe) because of the personalized problem-specific information they capture.

A major hindering factor on the task of keyphrase extraction is that documents with textual content are overwhelmed from abundance of words while the keyphrase set is minute in comparison. Therefore, there are many words and phrases that are essential for the document’s topic but are not annotated as keyphrases.

Moreover, the annotators of the golden keyphrase sets play an important role when it comes to model evaluation. The main categories of annotators are the authors and the readers. Authors’ sets are usually consisted of lesser keyphrases than readers’ sets, but they are adequate to represent the document topics. Therefore, when using authors’ sets, lower performance should be expected compared to the readers’ sets, as the available range of keyphrases is eminently smaller.

Currently, on many natural language tasks, state-of-the-art performance is achieved by deep learning approaches and unsupervised language models. This holds for the case of keyphrase extraction as well. In the work of Papagiannopoulou and Tsoumakas (2020), they conclude that unsupervised approaches, like Tf-Idf2, serve as robust baselines and thus publications should consider including them on the evaluation comparison process. Also, representatives for the deep learning category are works like Alzaidy et al. (2019) and Meng et al. (2017), which achieve top performance across all other methods.

Next, we make a discussion upon our work’s experiments, results and observations.

5.1 Discussion

In this thesis, we managed to replicate the Bi-LSTM-CRF model of Alzaidy et al. (2019) for keyphrase extraction. We offer a public repository on github with the methods used in our work and with an implementation of the Bi-LSTM-CRF model in tensorflow with contemporary versions of components/libraries. Further, we showed that the model is not better than the copyRNN as suggested in Alzaidy et al. (2019), but it manages to outperform the graph-based approach MultipartiteRank and marginally outperform TfIdf. This is due to the difference of the way the evaluation metrics are computed in each work. For copyRNN, they use the exact match approach, while in the Alzaidy et al. (2019) work, they use the sequence evaluation method. We observed that the sequence evaluation method of calculating the metrics is pretty close to the partial match method.
We conclude that data analysis should be carried out for each dataset in order to identify how many keyphrases exist in the text, especially after applying the pre-processing steps. Especially for the case of keyphrase extraction, this plays a major role as the maximum performance is upper bounded by the total keyphrases existing in the text. Also, utilizing cost-sensitive learning did not manage to mitigate the effects of class imbalance in the Bi-LSTM-CRF model and it seems that it is not worth using. Semi-exact matching, our new evaluation method, seems to be promising in theory as a fair evaluation of the methods in the task. With the tested configuration, in most cases it seems to be close to the partial match approach but with more moderate results. More experiments are needed to examine more thorough its validity throughout different cases. The field of the evaluation methods in the keyphrase extraction setting is still underdeveloped and a long away from perfect and possibly it will never reach there. Thereby, there should be more work to this direction to at least decrease the multiple cons of the current methods.

In our work, we showed that the selected text excerpt of a document is generally important to extract keyphrases. We explored various text excerpts from different parts of three full-text test datasets on three popular machine learning models. The models are representatives of the statistic-based, graph-based and deep learning categories. For the Bi-LSTM-CRF, we managed to achieve an average increase of 4% over the commonly used title and abstract and a maximum 5.85% increase for the SemEval dataset. For the TfIdf, we achieved an average 2% increase compared to the frequently used full-text and a maximum increase of 3.1%. For the MultipartiteRank, it seems that the best method would be the most easily accessible as there were no significant improvements. Generally that would be either the title and abstract or the full-text. These observations might be extended to the each of their respective categories as a whole, but more experiments with representative algorithms should take place. Lastly, we provided guidelines to choose the most appropriate text excerpt for each of the three machine learning categories.

5.2 Future Work

There are multiple approaches followed in the literature to address this problem. In this work, we focus on solving the keyphrase extraction problem as a classification task. Though, with this method, keyphrases that are not included in the text excerpts cannot be predicted. Thus, this method’s performance caps by the total number of keyphrases that exist in the title and the abstract. Subsequently, instead of extracting keyphrases as part of a classification problem, a more promising approach would be to generate the keyphrases. A step to this direction is
the work of Yuan et al. (2018), in which authors proposed two model variations that allow the user to choose the specific number of generated keyphrases. They also introduced appropriate metrics to evaluate the variable number of keyphrases.

A major work would be to study the capacity of the metrics in which they capture the actual performance of a model as the widely used evaluation method, exact string match, is very strict and it does not represent the actual performance of the models. Such indication comes from works like Papagiannopoulou and Tsoumakas (2020) that have calculated scores both with exact string matching and manually.

As mentioned in the experiments discussion section (4.4.3), an insightful work would be to conduct an in-depth study of the behavior of our methods across different machine learning methods. Further experiments should be carried out with various numbers of retrieved keyphrases and machine learning methods of each of the deep learning, statistic-based and graph-based categories. An interesting approach would be to detect keyphrases with multiple algorithms and methods and combine the sentences that these keyphrases are located in. Then, use one or more keyphrase extraction algorithms to extract the final keyphrase set from the produced text. Another approach would be to simply combine the predicted keyphrase sets generated from different methods and study their performance without considering the duplicates.

Finally, there are some subjects that may require attention but with smaller expected effect on the task. From the results of our experiments, Bi-LSTM-CRF seems to suffer from this as the whole abstract performed worse than if we split the abstract in sentences. Recently, the attention mechanism became popular as there are many works that have successfully utilized it in the spectrum of neural machine translation, image captioning and others. The application of the mechanism on the task of keyphrase extraction would be interesting as it handles the inability of the sequence-to-sequence models to retain past information from longer sequences. Another path that may be worth exploring is to experiment with more summarization techniques, including abstractive methods. Another option would be to train a model on summaries produced by distillated roberta or any other summarization algorithm that produces highly keyphrase dense abstracts. Lastly, one can observe that the title is deemed as an anomaly when paired with the abstract, as it is more keyphrase dense. Subsequently, it may be sensible to employ some technique of handling the title separately.
Appendix A

Extraction, Sequence and Semi-Exact $F_1$-score for All Evaluation Dataset Experiments

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bi-LSTM-CRF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>20.81</td>
<td>34.64</td>
<td>25.69</td>
<td>21.58</td>
</tr>
<tr>
<td>ABSE</td>
<td>28.16</td>
<td>48.10</td>
<td>41.72</td>
<td>24.11</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>17.02</td>
<td>35.42</td>
<td>13.18</td>
<td>13.70</td>
</tr>
<tr>
<td><strong>FS</strong></td>
<td>13.70</td>
<td>30.50</td>
<td>27.11</td>
<td>10.43</td>
</tr>
<tr>
<td><strong>TS</strong></td>
<td>17.12</td>
<td>29.64</td>
<td>20.58</td>
<td>20.12</td>
</tr>
<tr>
<td><strong>AS</strong></td>
<td>20.27</td>
<td>34.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3P220</td>
<td>21.88</td>
<td>35.84</td>
<td>23.79</td>
<td>19.53</td>
</tr>
<tr>
<td>3P400</td>
<td>13.22</td>
<td>24.73</td>
<td>18.31</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bi-LSTM-CRF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>21.68</td>
<td>36.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABSE</td>
<td>28.15</td>
<td>42.85</td>
<td>29.74</td>
<td>30.34</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>31.76</td>
<td>48.45</td>
<td>34.99</td>
<td>34.94</td>
</tr>
<tr>
<td><strong>FS</strong></td>
<td>24.64</td>
<td>34.99</td>
<td>34.99</td>
<td>34.94</td>
</tr>
<tr>
<td><strong>TS</strong></td>
<td>20.49</td>
<td>34.99</td>
<td>34.99</td>
<td>34.94</td>
</tr>
<tr>
<td><strong>AS</strong></td>
<td>24.57</td>
<td>34.99</td>
<td>34.99</td>
<td>34.94</td>
</tr>
<tr>
<td>3P220</td>
<td>27.45</td>
<td>48.45</td>
<td>48.33</td>
<td>48.33</td>
</tr>
</tbody>
</table>

Table A.1: $F_1$ based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bi-LSTM-CRF</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>21.68</td>
<td>36.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABSE</td>
<td>28.15</td>
<td>42.85</td>
<td>29.74</td>
<td>30.34</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>31.76</td>
<td>32.00</td>
<td>26.58</td>
<td>14.91</td>
</tr>
<tr>
<td><strong>FS</strong></td>
<td>24.64</td>
<td>32.00</td>
<td>17.64</td>
<td>11.45</td>
</tr>
<tr>
<td><strong>TS</strong></td>
<td>20.49</td>
<td>32.00</td>
<td>26.70</td>
<td>11.45</td>
</tr>
<tr>
<td><strong>AS</strong></td>
<td>24.57</td>
<td>32.00</td>
<td>30.39</td>
<td>29.95</td>
</tr>
<tr>
<td>3P220</td>
<td>27.45</td>
<td>32.00</td>
<td>33.18</td>
<td>29.50</td>
</tr>
<tr>
<td>3P400</td>
<td>19.67</td>
<td>32.00</td>
<td>23.46</td>
<td>26.30</td>
</tr>
</tbody>
</table>

Table A.2: $F_1$ based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using various input types, i.e., Title+Abstract (TA), abstract in sentences (ABSE), full-text in paragraphs (FP), full-text in sentences (FS), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)
### Appendix A. Extraction, Sequence and Semi-Exact $F_1$-score for All Evaluation Dataset Experiments

#### Table A.3: $F_1@10$ based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
</tr>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
</tr>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
</tr>
<tr>
<td>TA</td>
<td>22.06</td>
<td>42.39</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>15.69</td>
<td>31.00</td>
<td>-</td>
</tr>
<tr>
<td>TS</td>
<td>23.89</td>
<td>43.43</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>21.81</td>
<td>42.79</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>16.05</td>
<td>33.94</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>18.55</td>
<td>38.09</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Table A.4: $F_1@10$ based on Semi-Exact (eSE) and extraction Semi-Exact (eSE) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eSE%</td>
<td>eSE%</td>
<td>eSE%</td>
</tr>
<tr>
<td></td>
<td>eSE%</td>
<td>eSE%</td>
<td>eSE%</td>
</tr>
<tr>
<td></td>
<td>eSE%</td>
<td>eSE%</td>
<td>eSE%</td>
</tr>
<tr>
<td>TA</td>
<td>30.76</td>
<td>36.38</td>
<td>33.37</td>
</tr>
<tr>
<td>F</td>
<td>33.29</td>
<td>33.31</td>
<td><strong>36.53</strong></td>
</tr>
<tr>
<td>TS</td>
<td>33.27</td>
<td>37.68</td>
<td>35.70</td>
</tr>
<tr>
<td>AS</td>
<td>33.07</td>
<td>35.03</td>
<td>33.27</td>
</tr>
<tr>
<td>3P220</td>
<td>27.23</td>
<td>26.08</td>
<td>24.77</td>
</tr>
<tr>
<td>3P400</td>
<td>31.08</td>
<td>29.77</td>
<td>28.77</td>
</tr>
</tbody>
</table>
### Appendix A. Extraction, Sequence and Semi-Exact $F_1$-score for All Evaluation Dataset Experiments

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
</tr>
<tr>
<td><strong>TfIdf</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>17.97</td>
<td>38.79</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>17.07</td>
<td>36.06</td>
<td>-</td>
</tr>
<tr>
<td><strong>TS</strong></td>
<td>20.40</td>
<td>40.69</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>17.33</td>
<td>37.25</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>10.81</td>
<td>26.19</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>13.18</td>
<td>30.12</td>
<td>-</td>
</tr>
</tbody>
</table>

**Table A.5:** $F_1@20$ based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th></th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eSE%</td>
<td>eSE%</td>
<td></td>
</tr>
<tr>
<td><strong>TfIdf</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>28.75</td>
<td>28.64</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>33.35</td>
<td>31.98</td>
<td></td>
</tr>
<tr>
<td><strong>TS</strong></td>
<td>31.17</td>
<td>30.18</td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>28.06</td>
<td>25.98</td>
<td></td>
</tr>
<tr>
<td>3P220</td>
<td>19.31</td>
<td>17.05</td>
<td></td>
</tr>
<tr>
<td>3P400</td>
<td>22.60</td>
<td>20.18</td>
<td></td>
</tr>
</tbody>
</table>

**Table A.6:** $F_1@20$ based on Semi-Exact (eS) and extraction Semi-Exact (eSE) evaluation approach for TfIdf on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)
Appendix A. Extraction, Sequence and Semi-Exact $F_1$-score for All Evaluation Dataset Experiments

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>21.15</td>
<td>42.70</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>15.06</td>
<td>35.93</td>
<td>-</td>
</tr>
<tr>
<td>TS</td>
<td>21.42</td>
<td>43.38</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>20.47</td>
<td>41.67</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>15.31</td>
<td>33.88</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>15.52</td>
<td>35.59</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A.7: $F_1@10$ based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400).

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TA</td>
<td>33.81</td>
<td>39.62</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>36.74</td>
<td>36.62</td>
<td>-</td>
</tr>
<tr>
<td>TS</td>
<td>35.53</td>
<td>40.53</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>35.46</td>
<td>37.44</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>30.42</td>
<td>28.34</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>31.79</td>
<td>29.86</td>
<td>-</td>
</tr>
</tbody>
</table>

Table A.8: $F_1@10$ based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400).
### Table A.9: F$_1$@20 based on extraction Exact (eE), Partial (eP) and Sequence (S) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
</tr>
<tr>
<td>TA</td>
<td>19.53</td>
<td>37.80</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>16.54</td>
<td>38.12</td>
<td>-</td>
</tr>
<tr>
<td>TS</td>
<td>19.46</td>
<td>36.82</td>
<td>-</td>
</tr>
<tr>
<td>AS</td>
<td>17.98</td>
<td>35.90</td>
<td>-</td>
</tr>
<tr>
<td>3P220</td>
<td>12.25</td>
<td>28.48</td>
<td>-</td>
</tr>
<tr>
<td>3P400</td>
<td>12.81</td>
<td>30.51</td>
<td>-</td>
</tr>
</tbody>
</table>

### Table A.10: F$_1$@20 based on Semi-Exact (SE) and extraction Semi-Exact (eSE) evaluation approach for MultipartiteRank on 3 different datasets (SemEval, NUS, ACM) using various input types, i.e., Title+Abstract (TA), full-text (F), Title+Summary (TS), Abstract+Summary (AS), first 3 paragraphs with length 220 (3P220), first 3 paragraphs with length 400 (3P400)

<table>
<thead>
<tr>
<th>MultipartiteRank</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SE%</td>
<td>eSE%</td>
<td>eSE%</td>
</tr>
<tr>
<td>TA</td>
<td>33.10</td>
<td>32.49</td>
<td>32.01</td>
</tr>
<tr>
<td>F</td>
<td>36.52</td>
<td>34.88</td>
<td>35.14</td>
</tr>
<tr>
<td>TS</td>
<td>34.27</td>
<td>32.85</td>
<td>31.44</td>
</tr>
<tr>
<td>AS</td>
<td>31.78</td>
<td>29.14</td>
<td>28.78</td>
</tr>
<tr>
<td>3P220</td>
<td>22.96</td>
<td>19.57</td>
<td>18.66</td>
</tr>
<tr>
<td>3P400</td>
<td>24.34</td>
<td>21.36</td>
<td>19.96</td>
</tr>
</tbody>
</table>
Appendix B

Sentence Bi-LSTM-CRF Results

<table>
<thead>
<tr>
<th>sentence Bi-LSTM-CRF</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E%</td>
<td>P%</td>
<td>E%</td>
<td>P%</td>
</tr>
<tr>
<td>FSE</td>
<td>16.00</td>
<td>33.91</td>
<td>13.68</td>
<td>29.72</td>
</tr>
<tr>
<td>ABSE</td>
<td>10.44</td>
<td>20.86</td>
<td>13.91</td>
<td>25.92</td>
</tr>
</tbody>
</table>

**TABLE B.1:** F$_1$ based on extraction Exact (E) and Partial (P) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input.

<table>
<thead>
<tr>
<th>sentence Bi-LSTM-CRF</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eE%</td>
<td>eP%</td>
<td>S%</td>
<td>eE%</td>
</tr>
<tr>
<td>FSE</td>
<td>17.34</td>
<td>34.60</td>
<td>15.37</td>
<td>14.27</td>
</tr>
<tr>
<td>ABSE</td>
<td>20.91</td>
<td>35.80</td>
<td>27.59</td>
<td>24.48</td>
</tr>
</tbody>
</table>

**TABLE B.2:** F$_1$ based on Exact (eE), Partial (eP) and Sequence (S) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input.

<table>
<thead>
<tr>
<th>sentence Bi-LSTM-CRF</th>
<th>SemEval</th>
<th>NUS</th>
<th>ACM</th>
<th>KP20K-test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SE%</td>
<td>eSE%</td>
<td>SE%</td>
<td>eSE%</td>
</tr>
<tr>
<td>FSE</td>
<td>30.74</td>
<td>31.09</td>
<td>24.92</td>
<td>24.79</td>
</tr>
<tr>
<td>ABSE</td>
<td>21.95</td>
<td>37.12</td>
<td>26.12</td>
<td>39.41</td>
</tr>
</tbody>
</table>

**TABLE B.3:** F$_1$ based on Semi-Exact (SE) and Semi-Exact (eSE) evaluation approach for sentence Bi-LSTM-CRF on 4 different datasets (SemEval, NUS, ACM, KP20K-test) using full-text in sentences (FSE) and abstract in sentences (ABSE) as input.
Bibliography

Alzaidy, Rabah et al. (2019). “Bi-LSTM-CRF sequence labeling for keyphrase extraction from scholarly documents”. In: The world wide web conference, pp. 2551–2557.


Bojanowski, Piotr et al. (2017). “Enriching word vectors with subword information”. In: Transactions of the Association for Computational Linguistics 5, pp. 135–146.


Bougouin, Adrien et al. (2013). “Topicrank: Graph-based topic ranking for keyphrase extraction”. In:


Brin, Sergey and Lawrence Page (1998). “The anatomy of a large-scale hypertextual web search engine”. In:


Gollapalli, Sujatha Das and Cornelia Caragea (2014b). “Extracting Keyphrases from Research Papers Using Citation Networks”. In: AAAI.


Gollapalli, Sujatha Das et al. (2017). “Incorporating expert knowledge into keyphrase extraction”. In: Thirty-first aaai conference on artificial intelligence.


Lafferty, John et al. (2001). “Conditional random fields: Probabilistic models for segmenting and labeling sequence data”. In:


Sterckx, Lucas et al. (2016). “Supervised keyphrase extraction as positive unlabeled learning”. In: *EMNLP2016, the conference on empirical methods in natural language processing*, pp. 1–6.


Bibliography


Won, Miguel et al. (2019). Automatic extraction of relevant keyphrases for the study of issue competition. Tech. rep. EasyChair.

Yang, Min et al. (2018). “Task-oriented keyphrase extraction from social media”. In: Multimedia Tools and Applications 77.3, pp. 3171–3187.


Zhang, Yu et al. (2020). “Keywords extraction with deep neural network model”. In: Neurocomputing 383, pp. 113–121.
