KEYPHRASE EXTRACTION FROM SCIENTIFIC LITERATURE USING JOINT GEOMETRIC GRAPH EMBEDDING MATCHING

by

JUSTIN PAYAN

(Under the Direction of Frederick Maier)

ABSTRACT

Keyphrase extraction is the task of selecting representative words and phrases from a document. Recent research has focused on keyphrase extraction via graph-theoretic approaches, which leverage various graph-based centrality measures to locate important nodes in a constructed keyphrase candidate graph. In this thesis, we propose the use of an inexact graph matching algorithm for keyphrase extraction. We match graphs derived from test documents with graphs that have labeled keyphrases, and label as keyphrases the test nodes matching with known keyphrases. Our graph matching keyphrase extraction algorithm obtains an F-score of 14.6% on the standard SemEval 2010 Task 5 dataset and 37.1% on the well-known Inspec dataset. These scores are in line with time-tested algorithms on both datasets. We therefore conclude that inexact graph matching algorithms can be applied to keyphrase extraction successfully.

INDEX WORDS: Keyphrase extraction, Information extraction, Pattern matching, Graph theory, Natural language processing, Joint geometric graph embedding

KEYPHRASE EXTRACTION FROM SCIENTIFIC LITERATURE USING JOINT GEOMETRIC GRAPH EMBEDDING MATCHING

by

JUSTIN PAYAN

A.B., The University of Georgia, 2017

B.S., The University of Georgia, 2017

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2018

KEYPHRASE EXTRACTION FROM SCIENTIFIC LITERATURE USING JOINT GEOMETRIC GRAPH EMBEDDING MATCHING

by

JUSTIN PAYAN

Major Professor: Frederick Maier

Committee: Khaled Rasheed
Walter Potter

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
December 2018

ACKNOWLEDGMENTS

Many thanks to Dr. Fred Maier for helping me plan out my experiments and thesis writing, and for providing feedback on my drafts. Dr. Rasheed and Dr. Potter, thank you for your patience and support as I worked through numerous iterations of this project. I came up with the initial idea to apply graph matching to keyphrase extraction independently, but inspired by Dr. Rik Sarkar and Alex Hooker at the University of Edinburgh. I would also like to thank my parents John and Jana Payan, as well as my friends Muthukumaran Chandrasekaran, Davis DeRodes, Yazmin Reategui, Matthew Saltz, and Sammy Sbiti for listening to my ideas and complaints throughout the process. Thanks to Maulik Shah for helping me with the logistics of planning my defense, and to Jeffrey Mann and the Goodhue family for letting me stay with them for nearly a month. I am grateful to my former coworkers and managers at Vertica, who were very patient with me as I balanced the demands of my thesis with a full-time job. This whole experience would have been less rewarding and even impossible without the intellectual, emotional, and logistical support of all of these people.

TABLE OF CONTENTS

List of Figures

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM DEFINITION

Keyphrase extraction involves selecting a representative subset of phrases and words from a document. The subset is expected to both fully cover the document's topics as well as distinguish the document from other documents. Keyphrases are very useful for information retrieval, document clustering, automated summarization, opinion mining, and other downstream information extraction tasks [2; 3; 6; 12; 26]. In particular, keyphrase extraction in a scientific context helps with content recommendation and automated reviewer assignment, as well as serving as a key resource for forming a complete understanding of temporal changes in the research community over time [2]. In these tasks, it is rather important to assign keyphrase sets that a human would deem both fully relevant and adequately expansive, as automatically assigned keyphrases serve to position documents within a conceptual space that must be interpretable to humans. Keyphrase extraction can be contrasted with keyphrase assignment, which involves selecting words and phrases present in a external vocabulary as representative of the concepts in the document, regardless of whether or not the words actually appear in the document. The present study focuses entirely on keyphrase extraction.

Most methods for keyphrase extraction belong to one of three major categories - approaches based on linguistic theories of word distributions, supervised machine learning approaches, and unsupervised graph algorithm based approaches. Most recent keyphrase extraction studies use graph algorithm based approaches, though some work draws on advances in neural machine learning such as Word2Vec embeddings and supervised deep learning architectures [5; 32; 37].

1.2 CONTRIBUTIONS

We propose a new supervised graph-based algorithm based on inexact graph matching. The inexact graph matching algorithm we leverage is called Joint Geometric Graph Embedding Based Matching [41], so we call our end-to-end keyphrase extraction algorithm Joint Geometric Graph Embedding Based Keyphrase Extraction (JGGE-KE). To our knowledge, this is the first time that visual pattern recognition technologies have been leveraged for keyphrase extraction.

The major contributions of this work are as follows: 1) we present a novel paradigm for treating keyword extraction as a pattern recognition task, and 2) we combine the two dominant keyword extraction methods (supervised and graph-theoretic) for the first time.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we discuss early theories on term identification and statistical distribution of important keyphrases, feature-based supervised machine learning keyphrase extraction approaches, and unsupervised graph-based approaches. These three categories of prior art are largely organized chronologically; the basic term identification and statistical linguistic theories come from early pioneers of information retrieval, the traditional machine learning work is more recent but fading in relevance, and the unsupervised graph-based approaches are still being researched today. We also discuss the most commonly used metrics for evaluating any of the aforementioned keyphrase extraction techniques.

2.1 STATISTICAL LINGUISTICS FOR KEYPHRASE EXTRACTION

In the most basic sense, keyphrase extraction consists of selecting words and phrases with some specific properties from a document. These properties are meant to accomplish abstract and open-ended tasks, such as "summarizing a document" or "making a document easier to retrieve in a search engine." Researchers began formalizing these properties as they began to consider ways to optimize information search and retrieval.

Before understanding more abstract properties of words and phrases that make them good keyphrases, we must first understand what simple syntactic elements constitute keyphrases. Keyphrases are typically lexical terms, which means that they are single elements of a language's lexicon which convey a single meaning. The meanings of keyphrases are not compositional (composed in any simple fashion from the component words); rather, keyphrases often constitute the smallest possible units of meaning for whatever concepts they convey. In addition, keyphrases

tend to be technical terms which are specific to a small domain. A technical term is defined as a multi-word lexical term. Justeson and Katz describe a theory of technical terms, where they find that meaningful multi-word technical terms are almost exclusively composed of nouns and adjectives, with occasional prepositions [21]. Non-technical noun phrases often contain other syntactic constructions, distinguishing them from technical terms. Most keyphrase extraction algorithms select an initially large list of technical terms as potential keyphrases before filtering out the terms which do not adequately represent the document's topic. In the next section, we shall see a number of early machine learning based keyphrase extraction algorithms. Nearly all studies of such keyphrase extraction algorithms have found that optimal results are obtained when selecting nouns, adjectives, noun phrases, or some combination of these syntactic units as candidate keyphrases.

Despite the existence of this obvious heuristic (keyphrases tend to be technical terms - lexical noun phrases specific to the language of a small technical domain), we still need other metrics on top of this heuristic to further filter technical terms into keyphrase sets that are small enough to be useful in document indexing or other tasks. The most widely known early metric used for keyphrase selection and document indexing was the TFxIDF term weighting scheme [20]. TFxIDF stands for term frequency, weighted by inverse document frequency. This measure can be used for selecting keywords for documents, by computing TFxIDF for a set of candidate keyphrases and selecting the candidates with the highest TFxIDF scores as final keyphrases. Term frequency is computed by dividing the count of the term in the document by the total number of terms in the document. The second term, the inverse document frequency, is computed as

$$idf(t) = log(\frac{N}{df(t)})$$

where N is the total number of documents, and $df(t)$ is the number of documents containing term t.

This metric, and its empirical success, indicate an important feature of the keyphrase extraction task - selecting keyphrases means balancing exhaustiveness with specificity. The best keyphrase sets are exhaustive, in that they adequately cover the full range of topics in a document. However,

they also must describe the concepts in the document in a way that distinguishes the document from others in the collection. Term frequency captures exhaustiveness, while inverse document frequency is a corrective measure capturing specificity.

The keyphrase extraction field owes much to the insights of these initial investigations. Simply using TFxIDF still performs remarkably well on many datasets [15]. It is surprisingly competitive with newer keyphrase extraction algorithms, depending on the dataset. Even on datasets which do not admit TFxIDF or simple technical term identification as useful keyphrase extraction methods, the basic insights of the works discussed in this section provide theoretical justification for more complicated algorithms. Although most of the research of the past decade has relied on either machine learning or graph theoretic frameworks, it has all been heavily influenced by the early linguistic and statistical intuitions developed in these seminal studies.

2.2 Supervised Machine Learning Approaches

In this section we briefly discuss supervised machine learning algorithms that have been applied to the keyword extraction task. All such algorithms identify a set of candidate keyphrases, extract a set of linguistically or statistically motivated features for each candidate in the set, then train a machine learning classifier on positive and negative examples of keyphrases from a training set of documents. The positive and negative examples are generated from the training documents by extracting candidate keyphrases from each document, then comparing these candidate keyphrases to a human-annotated set of keyphrases, often referred to as the "gold-standard" set. The same process is applied to create a set of test documents. Any machine learning classifier can then be trained on the extracted training set and evaluated on the extracted test set. It is clear that these methods can be distinguished by their candidate generation heuristic, their chosen feature set, and their chosen machine learning algorithm. Some approaches introduce additional subtle variations, but for the most part these three decisions fully describe any machine learning based keyphrase extraction approach.

The KEA algorithm was one of the earliest to use supervised machine learning methods to select keywords, and it is still widely known and used today [51]. The algorithm selects all unigrams, bigrams, and trigrams that are not proper nouns and do not begin or end with a stopword as candidates. It then uses TFxIDF and distance from beginning of document as the two features input to a Naïve Bayes classifier. The authors later extended the KEA system into another system called Maui [36]. Candidate selection mirrors the candidate selection in KEA, but they use an expanded set of features and test a bagged decision tree machine learning algorithm. The features used are TFxIDF, index of first usage in the document, distance between first and last usage, keyphraseness (the number of times the phrase appears as a gold standard keyphrase in the training corpus), and phrase length. In addition, they also use some features derived from the Wikipedia page most closely related to the candidate keyphrase - node degree in Wikipedia hyperlink graph, Wikipedia-based keyphraseness (likelihood of being a hyperlink on Wikipedia), a semantic relatedness score [51], and inverse Wikipedia linkage (a number derived from the number of incoming links to the page). They try both a Naïve Bayes model and a bagged decision tree model.

One of the other early advocates of machine learning based keyphrase extraction was the GenEx system [48]. The features they use for each phrase are number of words in the phrase, first occurrence of the phrase, earliest occurrence of any word in the phrase, frequency of the phrase, highest frequency of any word in the phrase, length of the phrase in characters, whether the phrase is a proper noun, whether the phrase ends in an adjective, and whether the phrase contains a common verb. Note that occurrence and frequency are calculated in the context of the single document, not the corpus. They then use these features to train a C4.5 decision tree or a genetic algorithm.

Although highly advanced feature sets seem to improve accuracy of keyphrase extraction algorithms, studies which focus on feature sets fail to adequately explore the effects of candidate generation on the accuracy of the final machine learning algorithms. Anette Hulth was the first author to explore the impact of candidate selection on machine learning algorithm accuracy [18]. She presents three candidate selection methods, which she calls n-gram, NP-chunks, and PoS tag pat-

terns. The n-gram method selects candidates as all unigrams, bigrams, or trigrams that do not start or end with a stopword. The NP-chunks method selects candidates as all noun phrase chunks. For the PoS tag pattern method, Hulth defined the 56 part-of-speech tag patterns that appeared at least ten times among gold standard keyphrases in the training data. On the testing data, any sequence of part-of-speech tagged words following one of these patterns was selected as a candidate keyphrase. For each of the three candidate selection methods, she extracted four features for each candidate - relative position of the first occurrence, within-document frequency, corpus frequency, and part-of-speech tag sequence. The machine learning algorithm used to classify the candidate keyphrases as actual keyphrases is a bagged random forest ensemble classifier, as implemented in Compumine's *Rule Discovery System*. She found that the n-gram method performed best, followed by NP-chunks, then PoS tag patterns. In the current study, we will replicate the finding that candidate selection often has a strong impact on the quality of the final keyphrases. However, we also see that the optimal candidate selection method is often highly dependent on the underlying characteristics of the dataset, as well as the properties of the machine learning algorithm in use.

In the early 2000's, there was a large body of work on supervised machine learning approaches. However, we mention only one additional approach here, as it is the top-performing method on one of the datasets studied later in this work. In addition, it represents a trend in feature-based approaches towards complicated, highly engineered systems which work incredibly well on a single dataset but do not necessarily generalize. The algorithm, called HUMB, is a supervised machine learning approach, so they first convert each candidate keyphrase to a set of features [30]. They have 6 binary features indicating precense or abscence in the title, abstract, introduction, all section titles, conclusion, and the reference or book title. They also use a fairly standard structural feature, relative position of the first occurrence of the phrase. We later find in the current study that many scientific research papers have keyphrases heavily concentrated towards the front of the article. Taking advantage (or failing to take advantage) of this "front-loading" of keyphrases affects the accuracy of keyphrase extraction algorithms to a large degree. They use three "content-based" features - Generalized Dice Coefficient (how much more likely the words are to occur in

the phrase rather than separately), TFxIDF, and the number of times the term was selected as a keyphrase in the training data. Finally, there are three lexical/semantic features - presence in a terminological database the authors previously created called GRISP [31], Wikipedia keyphraseness (the probability that when a term appears in a Wikipedia article, it appears as a hyperlink anchor), and term length in number of words. The authors experimented with many machine learning algorithms, finding bagged decision trees perform best. Finally, after selecting the final list of classified keyphrases, they run a post-processing step, in which they reward candidate keyphrases that cooccur with other candidate keyphrases in the HAL database. HAL is a French repository of 139,000 research articles, many of which are labeled with keyphrases. This method is rather complex, and requires a large amount of hand-engineered features (GRISP and HAL-based features are especially expensive).

In general, feature-based machine learning approaches require extensive feature selection and computation, which hinders their general applicability. In addition, the reliance on costly human-annotated training data limits their ability to transfer to low-resource domains. Although ensemble approaches can reduce some limitations of the machine learning paradigm for keyphrase extraction (such as overfitting to the training data), they have the extremely negative side effect of further decreasing the interpretability, tractability, and ease of implementation of a keyphrase extraction algorithm. Graph-theoretic approaches have enjoyed success in recent years, because they address many of these limitations. They are generally unsupervised, they reduce the need for complicated feature engineering, and they are generally easily implemented and interpreted.

2.3 Unsupervised Graph-based Approaches

Like machine learning approaches, most graph-based approaches select a set of candidate keyphrases, and then classify each candidate as a keyphrase or not using some classification algorithm. However, the classification algorithms in this case typically involve constructing a graph in which the candidate keyphrases are the nodes, then ranking the graph nodes by some

graph-theoretic importance metric. The top ranking nodes are selected as the final keyphrases for the document. The process is illustrated in Figure 2.1.

Node properties that have been considered for ranking include degree (number of incident edges), strength (sum of the weights on incident edges), selectivity (strength divided by degree), neighborhood size, coreness (outermost core number in the k-core graph decomposition), clustering coefficient (edge density in neighborhood), structural diversity index (normalized entropy of adjacent edge weights), PageRank score, HITS score, betweenness (number of shortest paths passing through the node), closeness (reciprocal of sum of distances to all other nodes), eigenvector centrality, and small-worldness (contribution to small-world property of the network) [4; 8; 25; 52; 47; 38; 27; 17; 1; 35; 53; 39].

The earliest example of a graph-based keyphrase extraction algorithm is TextRank [39]. All noun and adjective unigrams in the test document are represented as nodes in a graph. The authors then draw unweighted, undirected edges between vertices representing unigrams that co-occur within an N-word sliding window in the test document. The TextRank algorithm uses an adaptation of Google's PageRank algorithm. Vertices vote for the other vertices which they have edges with, and votes are weighted by the importance of the vertices casting votes. Thus vertices with high degree and vertices adjacent to vertices with high degree will be selected as being most important. The update equation is given as follows:

$$W(V_i) = (1-d) + d * \sum_{V_j \in N(V_i)} \left(\frac{W(E_{i,j})}{\sum_{V_k \in N(V_j)} W(E_{k,j})} * W(V_j) \right)$$

Where d is a damping factor from the original PageRank algorithm, typically set to 0.85. $N(V_i)$ refers to the neighborhood of vertex V_i, the set of vertices connected to V_i by an edge. $W(\cdot)$ refers to the weight of an edge or vertex. The authors experiment with random values between 0 and 10 for the edge weights, and note that setting random edge weights obtains significantly different keyphrase rankings than setting all weights to 1. However, the authors ultimately report scores when weights are set uniformly to 1. In this work, we implement TextRank using edges weighted by the number of co-occurrences within the N-word sliding window, which is a standard varaint

of the TextRank algorithm in later works [49; 28; 9]. All vertices start with a score of 1, and the iterative updates are applied until convergence at a threshold of 0.0001. The authors note convergence usually occurs after 20 to 30 iterations. The top 33% unigrams are selected for a final post-processing step. Because the graph nodes are unigrams, a post-processing step is required. This step consists of considering all of the suggested unigrams and collapsing keywords that appear next to each other in the text into single keyphrases. The TextRank algorithm is interesting in its own right, but it is mostly important because it provided the first example of graph-based keyphrase extraction. Following this paper, the most recent keyphrase extraction algorithms have all proceeded by extracting a graph from a test document, and then using a graph algorithm to rank nodes in the graph. In addition, many extensions have been made to the pre- and post-processing steps, such as extracting whole phrases as candidate keyphrases or using alternative methods to score keyphrases based on the scores of unigram nodes in the graph.

To give a concrete example of extensions of the graph-based keyphrase extraction framework, the most recent graph-based algorithm in the literature is MultipartiteRank [9]. The authors first extract candidate keyphrases as all sequences of nouns and adjectives ending in nouns. Then they sort candidates into topics using a hierarchical agglomerative clustering algorithm proposed by Bourgoin et al. [11], and they draw edges among all candidates which do not share a topic. Finally, they run the standard TextRank iterative algorithm on the resulting graph. Many studies follow this standard formula of extending the TextRank algorithm by changing pre-processing, candidate selection, edge weighting, or post-processing steps around the TextRank iterative node-ranking algorithm. This paper is the most recent and successful of this line of reasoning.

RankUp is another graph-based algorithm that extends the TextRank framework [13]. Their extension is quite literal - the RankUp algorithm begins by computing the TextRank score of each candidate keyphrase. Then they compute the L2 loss of the TextRank scores against the TFxIDF scores of each candidate keyphrase, and apply a gradient descent update. They iterate back and forth between TextRank and the gradient updates until scores converge. They note that this back-propagation based on "error" computation improves upon the baseline of TextRank. However, they

also report scores for their reimplementation of TextRank that are much higher than those reported by the original TextRank study ([39]) for the same dataset. This discrepancy may indicate that some of their improvements in scores can be attributed to differences in pre-processing or evaluation.

Another rather interesting extension of TextRank is SemGraph [34]. They extract candidates from the "title, abstract, introduction, related work, future work, and conclusion" of the papers only, which benefits their algorithm's performance on their evaluation dataset. In addition, they have an alternative edge weighting scheme for their graphs. Rather than weighting edges by cooccurrence within a sliding window inside the evaluation document, they weight edges by the degree to which the nodes' cooccurrence across the entire corpus differs from random chance. They get very good results on their evaluation dataset. However, the authors only evaluate on a single dataset. Unfortunately, it is well-known in the keyphrase extraction community that keyphrase extraction algorithms must be evaluated on more than one dataset, as improvements on one dataset may be related to spurious correlations between the document preprocessing pipeline and the idiosyncratic characteristics of the dataset [15; 10]. If one is interested in the performance of the algorithm in question, these spurious correlations obfuscate the true aim of the study.

One important extension of TextRank is given in [49], which introduces the ExpandRank algorithm. This algorithm locates neighboring documents to the test document by ordering according to the TFxIDF cosine distance, and then executes the PageRank algorithm on the combined graph between the two documents. Graphs are combined by weighting inter-graph edges according to the following equation:

$$affinity(v_i, v_j) = \sum_{d_p \in D} sim(d_0, d_p) * count_{d_p}(v_i, v_j)$$

where d_0 is the test document, and D is the full test set of documents. The similarity function is the cosine similarity between the TFxIDF vectors for the two documents, and $count_{d_p}(v_i, v_j)$ is the number of times that v_i and v_j cooccur within a sliding window in document d_p. In our present study, we will also construct joint graphs by combining the test graph with neighboring graphs. However, ExpandRank connects multiple graphs into one large graph, while we will only connect two at a time. ExpandRank also sets aside a neighborhood size of 0, which they call SingleRank.

SingleRank is almost equivalent to TextRank, but for one major difference. SingleRank also generates graphs from documents by including unigrams as graph nodes. However, in SingleRank, final phrases are selected by ranking each candidate phrase using the sum of the component unigram scores. The TextRank reconstruction scheme is quite brittle; as an example, note that if two words of a three word phrase have very high scores and one word has a medium-high score, the entire phrase may be discarded because of the medium-high scoring word. Yet the SingleRank method does not suffer from this brittleness as severely. In a study from 2010, Hasan and Ng found that after converting SingleRank [49] to use the post-processing procedure used in TextRank, the results suffer greatly [15], indicating that the recombination by sum vastly outperforms the naïve recombination scheme used in the original TextRank paper. We will report a similar finding in this work.

The effect of post-processing (unigram recombination) is more explicitly explored in a recent paper by Florescu and Caragea [14]. They compare 6 graph-based ranking algorithms in conjunction with 3 post-processing (unigram recombination) schemes. In the "sum" recombination scheme, they compute phrasal scores as the sum of the component unigrams' scores. In the "mean" recombination scheme, they compute phrasal scores as the mean of the component scores. Finally in the "mean-tf" scheme they compute phrasal scores as in the "mean" scheme, then multiply by the frequency of the phrase within the document. They find that across 3 datasets, all algorithms attain the highest accuracies when using the "mean-tf" recombination scheme. We demonstrate later that the primacy of the "mean-tf" recombination scheme largely depends on the dataset under consideration.

There are many studies on graph-based keyphrase extraction which have been published in the last decade. Ultimately, the success or failure of various methods is challenging to measure, since performance largely depends on pre- and post-processing procedures and fickle characteristics of the evaluation datasets. It is therefore highly important to control for pre- and post-processing routines by comparing a new algorithm against an old algorithm while using the exact same pre- and post-processing procedures. In addition, algorithms must be compared across multiple datasets in

order to rule out unknown characteristics of any single dataset which fortuitously boost performance of a newly proposed algorithm. More details on evaluation are presented in the next section.

2.4 Evaluation of Keyphrase Extraction Algorithms

It is customary to evaluate keyword extraction algorithms using micro-averaged precision, recall, and F-score on a testing dataset of documents that has been annotated with gold-standard keyphrases. Many datasets are used in the literature. However, there are two particularly popular datasets consisting of technical terms, commonly referred to as SemEval [23] and Inspec [18] respectively. Both datasets consist of text in the domain of computer science research. SemEval contains full conference articles, while Inspec contains only the abstracts of journal articles. Both datasets are freely available online[1]. In addition, some keyphrase extraction algorithms require an external corpus for training unsupervised, distributed vector representations of words (Word2Vec). When studying technical literature, the ArXiv academic preprint repository is often selected for this purpose. We explain these three datasets and the Word2Vec word embedding model in more detail in Sections 4.1 and 3.3.2 respectively.

Precision and recall are computed based on exact full phrase matches. Micro-averaged precision is computed simply as

$$\frac{\sum_{i=1}^{D} \#\ correctly\ extracted\ keyphrases\ for\ doc_i}{\sum_{i=1}^{D} \#\ extracted\ keyphrases\ for\ doc_i}$$

Micro-averaged recall is

$$\frac{\sum_{i=1}^{D} \#\ correctly\ extracted\ keyphrases\ for\ doc_i}{\sum_{i=1}^{D} \#\ gold\ standard\ keyphrases\ for\ doc_i}$$

F-score is computed as

$$2 * \frac{precision * recall}{precision + recall}$$

[1] https://github.com/snkim/AutomaticKeyphraseExtraction

While these metrics seem rather simple, there is an additional complication. In many labeled keyphrase extraction datasets, some of the gold standard keyphrases do not appear in the text of the document. Because of this, the recall of any algorithm on that dataset cannot be greater than the number of phrases which actually appear in the document (assuming the algorithm is extractive rather than abstractive). Some studies report precision, recall, and F-score on the original gold standard, accepting that their recall has a ceiling. However, other studies report precision, recall, and F-score on an adjusted gold standard in which every phrase is present in the document (making perfect recall possible) [29]. These two evaluation schemes are not directly comparable, but some studies in the literature leave it unclear which scheme they are operating under. This confusion has been explicitly noted at least once in the literature, though it is often not addressed explicitly [15].

In addition to the aforementioned point of confusion, there is one other discrepancy in results reported in the literature. Most studies either select multi-word candidate keyphrases or select single-word candidate keywords and recombine them into phrases as a post-processing step. They then evaluate against the gold standard keyphrases in their original forms. However, there is at least one study that extracts single-word candidates and does not recombine them into phrases [45]. They instead split the gold standard keyphrases into their component words, and they evaluate against these keywords rather than against the full keyphrases. The scores resulting from this approach are thus not directly comparable to the scores resulting from any study evaluating on full keyphrases. Although the authors of the mentioned study do explicitly state they focus on keywords only, it is very important for researchers in this domain to pay very close attention to whether or not other works evaluate on keywords or keyphrases, as the results will differ substantially.

Most graph-based keyphrase extraction algorithms produce individual rankings for nodes in the graph, which represent the candidate keyphrases. The size of the final set of keyphrases thus can vary, depending on how final keyphrases are chosen from the ranked list. Some papers select the top p% of ranked candidate keyphrases as final keyphrases, and other papers select the top N ranked candidate keyphrases as final keyphrases. A minority of studies allow an arbitrary number of keyphrases to be chosen as final keyphrases. While this flexibility makes intuitive sense and

often improves precision, recall, and F-score, it also makes direct comparisons between algorithms rather difficult. We therefore fix p% and N in this study. For many datasets in the literature, there is a common setting of p or N that is universally adopted, making direct comparison possible.

Given a ranking of candidate keyphrases, one may also evaluate the quality of the ranking as a whole. In this case, final keyphrases are not selected. Instead, we seek to verify that the candidates are sorted in the correct order, such that correctly chosen candidates appear closer to the top of the list than incorrect keyphrases. Many metrics exist in the document retrieval literature for this very purpose, including Mean Reciprocal Rank, Binary-preference, R-precision, and Mean Average Precision [16]. In addition, the metric of R-precision has been modified specifically for keyphrase extraction, allowing partial matches in addition to exact matches. Finally, metrics from other natural language processing tasks, such as BLEU, METEOR, NIST, and ROUGE, have been applied to keyphrase extraction, though these metrics are not widely adopted [22]. They have been shown to correlate strongly with human preferences, however.

Based on ubiquity in the literature, we choose to evaluate our newly proposed algorithm as well as our reimplementations of existing algorithms on two datasets using precision, recall, and F-score at p% or N. This evaluation procedure is the simplest and most widely accepted procedure, and as such, it allows us to compare our proposed algorithm directly against other algorithms from the literature.

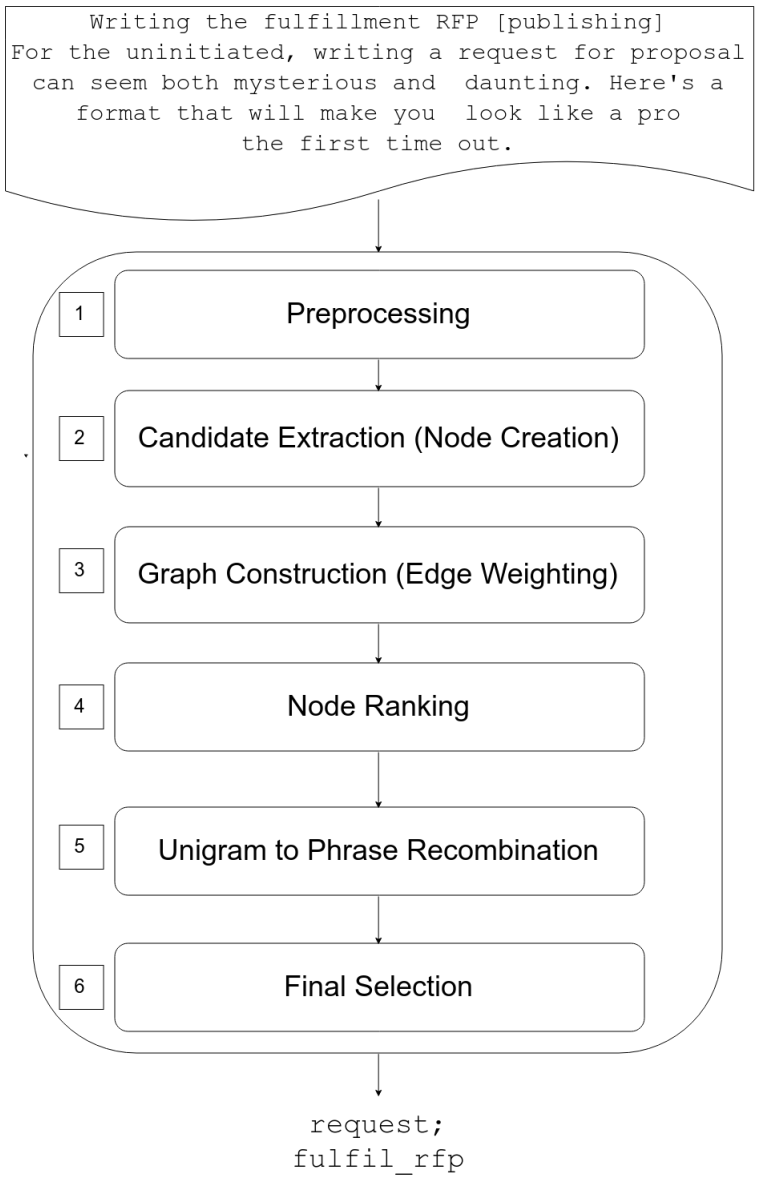

Figure 2.1: Outline of graph-based keyphrase extraction (more details in Chapter 3). There are six steps in the process: preprocessing converts the raw text into stemmed, tagged tokens, candidate extraction selects noun phrases or noun and adjective unigrams to be nodes in the graph, graph construction weights the edges using semantic similarity, node ranking assigns scores to candidate phrases/unigrams, a recombination step converts scored unigrams to scored phrases if required, and the final selection step picks the top p% or top N scored phrases/unigrams as the keyphrases.

CHAPTER 3

KEYPHRASE EXTRACTION AS INEXACT GRAPH MATCHING

Most graph based keyphrase extraction algorithms consist of six parts:

- text preprocessing
- selection of candidate keyphrases, each of which becomes a vertex in the graph
- drawing and weighting edges among candidate keyphrase nodes
- application of a graph algorithm to rank the keyphrases/nodes
- recombination of unigram candidates into full keyphrases if required
- selecting the top p% or top N keyphrase candidates as final keyphrases

The full pipeline is illustrated in Figure 3.1.

In this section, we explain our implementation of each of these six steps (corresponding to the six parts of the visual pipeline). All code was written in Python, and is available on Github[1].

3.1 TEXT PREPROCESSING

We begin by breaking each input document into a list of sentences. We use the Python Natural Language Toolkit (NLTK) Punkt Sentence tokenizer [7]. The Punkt sentence tokenizer builds an unsupervised model of abbreviations, collocations, and words that start sentences to determine the sentences. It considers periods, commas, semicolons, colons, exclamation marks, and question marks as punctuation. This algorithm is explained in detail in the paper by Kiss and Strunk [24].

[1] `https://github.com/justinpayan/KeyphraseExtraction`

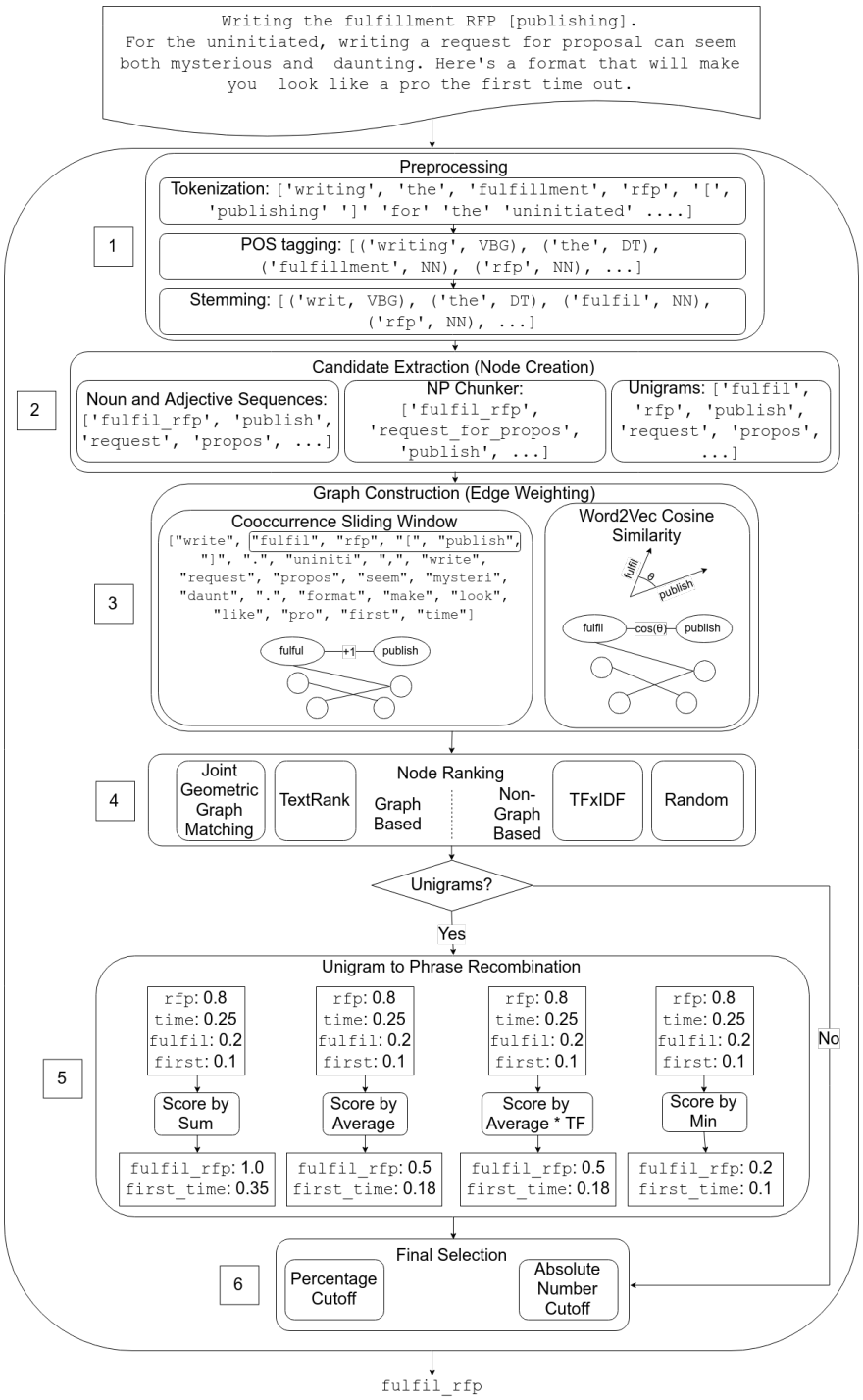

Figure 3.1: The full keyphrase extraction pipeline. There are six steps in the process: pre-processing converts the raw text into stemmed, tagged tokens, candidate extraction selects noun phrases or noun and adjective unigrams to be nodes in the graph, graph construction weights the edges using semantic similarity, node ranking assigns scores to candidate phrases/unigrams, a recombination step converts scored unigrams to scored phrases if required, and the final selection step picks the top p% or top N scored phrases/unigrams as the keyphrases.

The intuition behind the algorithm is that once abbreviations have been labeled, the sentence tokenization task can be solved with high accuracy by simply splitting on punctuation that is known to end sentences. This statement proves relatively correct for many European languages, as shown by the empirically validated superiority of this method on sentence tokenization tasks in 11 European languages.

We then tokenize each sentence with the NLTK TreeBank word tokenizer. The TreeBank word tokenizer breaks contractions, separates most punctuation marks, separates commas and single quotes if followed by whitespace, and separates periods at ends of lines.

We part-of-speech tag each sentence. The part-of-speech tagger is a MaxEnt Tagger trained on the Penn Treebank dataset [33].

Tokens are then also stemmed, but not lemmatized. The stemmer is NLTK's implementation of Porter's Stemmer [43]. Preliminary experiments evaluated both stemmed and unstemmed candidate keyphrases, determining stemmed candidates perform better than unstemmed. At this stage, we also stem the gold standard keyphrases. We want the gold standard keyphrases to represent the general ideas central to the document, and we do not care about matching their exact morphology. Most studies on keyphrase extraction use stemmed gold standard keyphrases for evaluation.

3.2 Candidate Keyphrase Selection

After preprocessing the input text, we must construct a list of words and phrases from the document which may be keywords and keyphrases. We investigate three different methods for selecting such keyphrase candidates: selecting all sequences of nouns and adjectives, using a simple machine-learned noun phrase chunker, or selecting unigrams and combining them into phrases after selecting key-unigrams.

3.2.1 Noun and Adjective Sequences

The first method is to select candidate keyphrases to be all phrases consisting of nouns and adjectives, following the work of Hulth ([18]) and Justeson and Katz ([21]). The method simply looks

for all sequences consisting solely of nouns and adjectives, ending in nouns. The caveat that the noun and adjective sequences must end in nouns is standard in recent keyphrase extraction literature([28; 13; 9]). Because the tagger uses the Penn Treebank tagset, this means that the accepted part of speech tags are the JJ tag and any tag starting with NN.

3.2.2 NAÏVE BAYES NOUN PHRASE CHUNKER

NLTK suggests a simple implementation for a simple machine learning (Naïve Bayes) based noun phrase chunker in chapter 7 of their book *Natural Language Processing with Python* [7]. The intuition behind the noun phrase chunks produced by this method is the same intuition as the Noun and Adjective Sequences approach - extracting non-overlapping sequences of words such that each sequence represents a single entity in the text. However, the approach described here is data-driven rather than being directly derived from a priori part-of-speech information. The Naïve Bayes model is trained on the CONLL-2000 chunked sentences dataset [46]. The training data for this dataset is WSJ sections 15-18 from the Penn Treebank.

The Naïve Bayes chunker uses a set of seven features to identify noun phrases, including part of speech, word, previous part of speech, next part of speech, previous part of speech and current part of speech, next part of speech and current part of speech, and the sequence of parts of speech since the last determiner. Using these features, the Naïve Bayes model classifies each word in the corpus with a chunking tag: NP_B for beginning of a NP, or NP_I for middle of a NP. This chunker is used to find NP chunks, all of which are selected as keyphrase candidates.

3.2.3 UNIGRAMS

The final candidate selection method is to select candidates as all unigrams tagged as a noun or adjective. This method is the most straightforward, and it results in the smallest-sized graphs. This method necessitates an additional post-processing step (step 5 in Figure 3.1), which is discussed in Section 3.5.

In theory, this method would be implemented by simply selecting all nouns and adjectives from the document. In practice, due to the post-processing step, we extract all unigrams which are found in the noun and adjective sequences discussed in Section 3.2.1. This slight deviation allows us to expect every candidate unigram to appear in a full keyphrase during the post-processing recombination step.

3.3 Weighting Graph Edges

Following candidate phrase extraction, each distinct candidate phrase or unigram is represented as a node in the document's graph. We then draw edges between the nodes, weighted by either a cooccurrence-based similarity metric or Word2Vec cosine similarity. In both cases, the edge weights are intended to be a proxy for semantic similarity. In the next two subsections, we discuss these two edge weighting schemes.

3.3.1 Cooccurrence-Based Similarity Metric

The cooccurrence similarity metric is defined pairwise for two phrases or unigrams as follows. Phrases are considered as cooccurring if they appear together within a sliding window of size N. The basic idea is illustrated in Figure 3.2. This edge weighting metric was first proposed in the seminal work on graph-based keyphrase extraction, TextRank [39]. To this day, it is the de-facto edge weighting method. In preliminary experiments comparing all window sizes from $N = 2$ to $N = 15$, we found that window size had very little effect regardless of preprocessing, candidate extraction method, node ranking algorithm, recombination scheme, or final selection criterion. In some cases, very low settings of $N = 2$ or $N = 3$ resulted in 1 or 2% differences in average F-score, so we report scores using $N = 5$ in this work.

We have to do additional processing in order to meaningfully consider cooccurrence of multi-word phrases. After selecting the candidate keyphrases from the document, we replace each instance of a multi-word keyphrase in the document with a single uninterrupted token where spaces have been replaced with underscores. Thus, *The present King of France is bald.* would

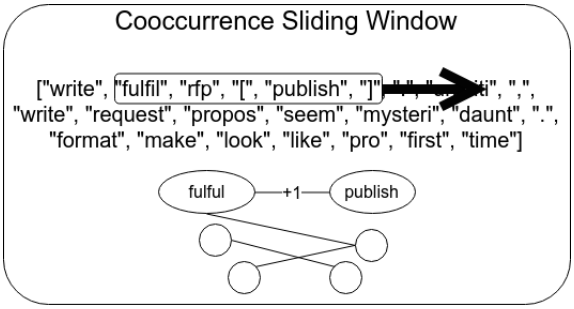

Figure 3.2: The sliding window moves across the processed document, adding a single count to each edge when it encounters a cooccurrence between the edge's incident nodes.

become *The present_King_of_France is bald.* if *present King of France* were identified as the sole multi-word keyphrase in this sentence. As a consequence of this fact, a sliding window of size N includes N *lexical units*, not simply N words. Thus the text sequence *The present_King_of_France is bald.* could be considered to be length $N = 4$, rather than $N = 7$. In addition, we also remove stopwords at this stage of processing, which in our example would result in the text sequence *present_King_of_France bald*. Noun phrases are often separated by many stopwords, so collapsing the text by removing stopwords allows for more highly connected graphs while maintaining the theoretical justification for the edge weights (similar phrases are still close in the text, while dissimilar phrases are still distant in the text).

We draw complete, weighted graphs with this scheme. The different weights are equal to the number of times each pair of phrases cooccurs within a sliding window of length N, after applying the aforementioned processing procedures.

3.3.2 WORD2VEC COSINE SIMILARITY

The Word2Vec semantic word embedding algorithm was originally presented by Mikolov et al. in 2013 [40]. It is a method of converting words into distributed vector representations, with the intention that words that are close to each other in the resulting vector space are close to each other semantically. In addition, the algorithm preserves a degree of compositionality, such that adding

two vectors roughly corresponds to composing the semantic meanings of the two represented words. The Word2Vec training algorithm has two common variants, called the Continuous Bag of Words (CBOW) and Skip-gram models, respectively. In the CBOW model, an artificial neural network is trained to predict the current word given N preceding words and N future words (the "context"). The skip-gram model reverses the CBOW model, training an artificial neural network to predict the context given the current word. Both models are trained using stochastic gradient descent with backpropagation of derivatives. Once the model is trained, the hidden representation of an input word can be treated as the vectorized representation of the word. Other work such as the Sent2Vec algorithm in [42] has searched for more accurate compositional models by extending the original Word2Vec model, but the Word2Vec authors illustrate a few examples of basic compositionality of Word2Vec representations, enough for our purposes in this study.

This neural approach to embedding words and phrases in a semantically motivated vector space has become immensely popular in recent natural language processing work. Indeed, in the last few years, Word2Vec has been applied to keyphrase extraction in a variety of ways. A method called EmbedRank uses the Sent2Vec extension of Word2Vec ([42]) to embed candidate keywords and keyphrases in a semantic embedding space along with a vectorized representation of the document from which the candidates are extracted [5]. They compute the cosine distance between the candidate phrases and the document in this semantic embedding space, and then select the closest words and phrases to the document to be the keyphrases. Cosine distance is defined as:

$$d(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x} \bullet \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}$$

which is derived from the fact that the angle θ between the two vectors $\mathbf{x}$ and $\mathbf{y}$ satisfies the identity

$$cos(\theta) = \frac{\mathbf{x} \bullet \mathbf{y}}{||\mathbf{x}||||\mathbf{y}||}$$

Cosine distance measures the degree of collinearity between two vectors, with parallel vectors in the same direction having a distance of 0, parallel but opposing vectors having a distance of 2, and orthogonal vectors having a distance of 1.

In addition to using cosine distance in Word2Vec embedding space directly for keyphrase extraction, a few recent studies have applied Word2Vec cosine similarity (intuitively, the inverse of cosine distance) to the problem of weighting edges in a document-derived graph, followed by a standard node ranking algorithm such as TextRank. One study uses a weighted linear combination of cooccurrence counts and Word2Vec cosine similarity as the edge weights in the graph of terms [50]. Another recent method, Key2Vec, also uses Word2Vec cosine similarities to weight edges in a graph, followed by using PageRank to sort nodes of the graph [32]. They train their Word2Vec model using papers from various scientific domains on the ArXiv database, as well as the full training and testing sets from their evaluation datasets. They create a graph with candidate keyphrases as nodes and edges drawn between keyphrases that cooccur within a sliding window of length 5. They then weight edges as the product of cosine similarity ($\frac{1}{1-cos(w_i,w_j)}$) and pointwise mutual information of the candidates. Finally, they compute the personalized PageRank score for each candidate keyphrase, where the personalization factor is equal to the cosine similarity between the Word2Vec embedding of the candidate and a "thematic vector". This thematic vector is equal to the sum of all the word vectors in the first N sentences of the document (where N is dependent on the dataset). These previous applications of Word2Vec cosine distances to edge weighting motivate our own usage of this metric for edge weighting.

In this work we use the continuous bag of words model (CBOW) to train word vectors of dimension 100. $\alpha = 0.025$ (α is the initial learning rate for gradient descent, which decreases as training progresses). We combine the context vectors by taking their mean. We use negative sampling, with 5 "noise words" chosen. We train three Word2Vec models, one for each candidate extraction algorithm. The set of computer science articles from ArXiv provides a suitable corpus of scholarly computer science writing for training. We stem every word in the corpus prior to training, and we concatenate phrases by replacing spaces between tokens with underscores (for instance, *don quixote* would become *don_quixote*). Finally, we remove all stopwords and words with greater than 40 characters or less than 1 character from the corpus. We input this final, processed corpus to the Word2Vec training.

We experiment with one additional extension to this basic notion of Word2Vec embeddings. We first compute the cooccurrences of the words and phrases in the document, then we compute the Word2Vec embedding of each node in the graph. At this point, we can simply compute the cosine similarity of two vectors to weight an edge between two nodes. However, we can also update each node's Word2Vec representation by adding to it the vector representation of any words or phrases that cooccur with it within the sliding window size. The summation is partially intended to decrease the sparsity of the resulting adjacency matrices (if a node's unigram/phrase is not contained in the Word2Vec model, cooccurring unigrams/phrases may provide a hint for possible vector representations of the unigram/phrase). One other motivation for this extension is that it makes the distance computation much more similar to the distance computations between points in images. The main question of this work is whether or not a selected inexact graph matching algorithm for shape matching in images can be applied to keyphrase extraction. Therefore, we feel it is natural to attempt to treat document-graphs as being similar to image-graphs. In the particular graph matching algorithm we have adopted, points in images are represented by a vector of features representing the intensities at multiple nearby points [41]. We consider the summation of the Word2Vec embeddings of neighboring words and phrases as an analogue of this inclusion of neighborhood information.

After nodes' vector representations have been finalized, distance between two words or phrases is computed as the cosine distance between the words' or phrases' embedding vectors w_i and w_j:

$$cosine\_dist(w_i, w_j) = 1 - \frac{w_i \cdot w_j}{\|w_i\|\|w_j\|}$$

In the case where one or both of the embedding vectors is the zero vector, we set the cosine distance to a very large random value, drawn from the normal distribution $\mathcal{N}(1000, 100)$. This distribution was arbitrarily chosen, but based on a handful of preliminary experiments we found that varying μ and σ^2 for this distribution do not change the behavior of the algorithm downstream. Finally, we

compute the cosine similarity score as

$$exp(-||cosine\_dist(w_i, w_j)||^2)$$

This equation mirrors a similar equation in the inexact graph matching algorithm we apply [41].

3.4 Joint Geometric Graph Embedding Matching

In this section, we propose a novel node ranking algorithm based on an inexact graph matching algorithm. The inexact graph matching algorithm uses joint geometric graph embeddings to match nodes, hence we call our method Joint Geometric Graph Embedding Based Keyphrase Extraction (JGGE-KE or simply JGGE). We compare this algorithm to a Random node ranking algorithm (nodes are assigned values drawn from the uniform distribution over the unit interval), the TextRank algorithm [39], and TFxIDF scores [20].

Given a test document, the JGGE-KE algorithm proceeds as follows:

1. A graph is extracted from the test document, following the general procedure discussed in the previous section
2. The test graph is combined with a graph extracted from a training document (with the combined graph referred to as the joint graph)
3. The joint graph is embedded in the eigenspace of its adjacency matrix
4. Nodes v in the test subgraph are assigned rankings based on distances from v to nodes w_i labelled as keyphrases in the training subgraph (we explore two methods for assigning scores here, which are illustrated separately in Algorithms 1 and 2)
5. Steps 2-4 may be repeated with multiple different training graphs
6. Each node v in the test graph is ranked as the sum of the rankings derived from the joint graphs

The exact number of times steps 2-4 are repeated is a parameter k to be set by the user. We found that the procedure typically performs well for $k >= 5$, but at approximately $k = 20$ the

algorithm's accuracy levels off. In addition, the algorithm's time complexity is linear in k, so it is wise to limit the number of training graphs to the degree possible. We set $k = 10$ in this work to balance computational effort and accuracy.

There is a natural question: how does one select the small subset of the training set to match against the test graph? We propose using the simple method from [49], whereby TFxIDF document vectors are computed for each document in the training and testing set. One can quickly compute the TFxIDF cosine similarity between a given test document and all of the training documents. The k training documents with the minimum TFxIDF cosine similarities to the testing document are selected to proceed with matching. This procedure is illustrated in more detail in Algorithm 3.

Algorithm 1 Joint Geometric Graph Embedding Based Keyphrase Extraction (Version 1)

procedure JGGE-KE(D_{train},G_{train},d_{test},g_{test},k) ▷
D_{train} = set of training documents
G_{train} = set of training graphs
d_{test} = a test document
g_{test} = the graph for d_{test}
k = number of training documents used per test document
    rank $\leftarrow \mathbf{0} \in \mathbb{R}^{|V_{test}|}$
    for $d_{train} \in closest\_train\_docs(D_{train}, d_{test}, k)$ **do**
        $g_{train} \leftarrow G_{train}[d_{train}]$
        for $v_i \in V_{test}$ **do**
            if $\exists v_j \in gold\_std\_keys(v_{train}) | match(v_i, v_j)$ **then**
                rank$[v_i] += 1$
            else
                unmatched_nodes$+= (v_i, \min_{v_j \in gold\_std\_keys(V_{train})} d_{JGGE}(v_i, v_j))$
            end if
        end for
        for $v_i \in sorted(unmatched\_nodes)$ **do**
            rank$[v_i] += \frac{index[v_i]}{|unmatched\_nodes|}$
        end for
    end for
    return rank
end procedure

The key step of the algorithm is the inexact graph matching algorithm, which can be any inexact graph matching algorithm that runs on weighted, undirected graphs. Inexact graph matching is finding an approximate correspondence between the nodes of two graphs. Correspondences can be

Algorithm 2 Joint Geometric Graph Embedding Based Keyphrase Extraction (Version 2)

procedure JGGE-KE(D_{train},G_{train},d_{test},g_{test},k) ▷
D_{train} = set of training documents
G_{train} = set of training graphs
d_{test} = a test document
g_{test} = the graph for d_{test}
k = number of training documents used per test document
    rank $\leftarrow \mathbf{0} \in \mathbb{R}^{|V_{test}|}$
    for $d_{train} \in closest\_train\_docs(D_{train}, d_{test}, k)$ **do**
        $g_{train} \leftarrow G_{train}[d_{train}]$
        for $v_i, v_j \in cartesianproduct(V_{test}, gold\_std\_keys(v_{train}))$ **do**
            rank$[v_i] += d_{JGGE}(v_i, v_j)$
        end for
    end for
    return rank
end procedure

based on similarity of node attributes, existence of edges between each pair of nodes in the two graphs, or similarity of edge attributes on shared edges. Inexact graph matching algorithms have been historically mostly used in visual object recognition.

There are three reasons why we wish to use inexact graph matching, as opposed to exact graph matching algorithms. One is that for graphs with continuous labels on nodes or edges, exact matches rarely occur. The other is that our graphs are not necessarily the exact same size; so there will not be an exact match by definition. The third reason is that we expect structural patterns to follow general rules, but in a noisy and imprecise way. Thus we wish to match patterns that look similar to each other but not exactly alike.

The most popular type of inexact graph matching algorithm in use today is spectral decomposition algorithms. The basic idea behind these algorithms is taking advantage of properties of the eigenvalues and eigenvectors of the graphs' adjacency matrices. These methods use edge weights instead of node labels, making them particularly well-suited to keyphrase extraction. The feature space for node labels consists of all of the words in the English language. Thus, using the con-

Algorithm 3 Closest Training Documents to a Test Document

procedure CLOSEST_TRAIN_DOCS(D_{train},d_{test},k) ▷
D_{train} = set of training documents
d_{test} = a test document
k = number of training documents to match
    for $d_{train} \in D_{train}$ **do**
        $sim(d_{train}, d_{test}) \leftarrow \frac{\mathbf{tfidf}(d_{train}) \cdot \mathbf{tfidf}(d_{test})}{||\mathbf{tfidf}(d_{train})||||\mathbf{tfidf}(d_{test})||}$
    end for
    return $\{d_{train} | sim(d_{train}, d_{test}) \in topk\}$
end procedure

tinuous information of the edge weights is much more tractable than using the combinatorially explosive node labels. Spectral decompositions take into account real-valued edge weights since those weights can be directly used in the adjacency matrix. In addition, the spectral decomposition of the adjancency matrix yields holistic information about the entire graph structure. Our hypothesis in this work is that sets of keyphrases roughly obey holistic structural patterns in a semantic space, so this interpretation of spectral graph matching algorithms is well-aligned with our thesis.

The graph matching algorithm in our experiments follows an algorithm called Joint Geometric Graph Embedding Based Matching [41]. The authors describe matching graphs derived from images, but we extend the algorithm to the natural language processing domain. This algorithm has a fairly intuitive explanation, and we found it particularly easy to adapt to the keyphrase extraction problem. The algorithm proceeds by projecting the nodes of both graphs into an interpretable joint space, where matches are computed between nodes that are close to each other in the joint space. We will see that we can exploit this joint embedding space in other ways, which will lead both to improved keyphrase extraction scores as well as a deeper understanding of the variable purposes of assigning keyphrases for different datasets.

The Joint Geometric Graph Embedding algorithm for two graphs $G_1 = (V_1, E_1, W_1)$ and $G_2 = (V_2, E_2, W_2)$ is as follows. First, each graph is converted into affinity matrices W_1 and W_2. The affinity matrices can be defined in any fashion, as long as they describe a complete graph, with

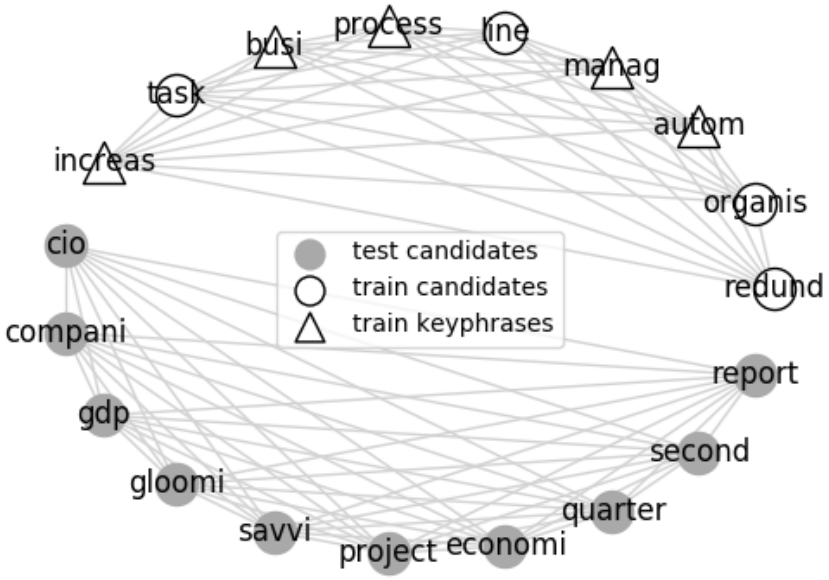

Figure 3.3: Two graphs from the Inspec dataset. The training graph, number 658, is on the top in white. The test graph, number 338, is on the bottom in grey. The graphs are not yet embedded in a joint eigenspace. They are shown in this figure in an arbitrary layout.

edge weights representing similarity between node labels. We set the affinity function to be the Word2Vec cosine similarity function described in the previous section. At this point, the graphs are completely separate, as in Figure 3.3. Then, a matrix must be formed for the inter-graph affinities as well. This matrix, C, is defined as:

$$C_{v_i,v_j} = exp(-||cosine\_dist(v_i, v_j)||^2)$$

where $v_i \in V_1$ and $v_j \in V_2$.

Once the intra-graph affinity matrices W_1 and W_2 and the inter-graph affinity matrix C have been constructed, a joint matrix is created as:

$$W = \begin{bmatrix} W_1 & C \\ C^T & W_2 \end{bmatrix}$$

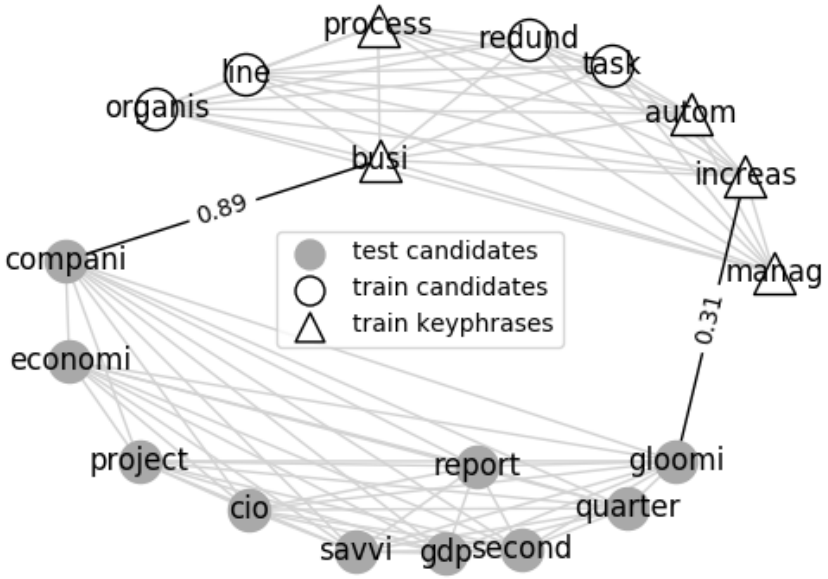

Figure 3.4: Edges are drawn between all white training graph nodes and all grey test graph nodes. These edges are weighted using Word2Vec cosine similarity between the node labels. For example, *compani* and *busi* have a fairly high Word2Vec cosine similarity because they are semantically similar, while *gloomi* and *increas* have a fairly low Word2Vec cosine similarity because they express roughly opposing sentiments.

The graphs have been tied together in a basic sense now. A vizualization of this step in the process is indicated in Figure 3.4. We could theoretically match nodes directly based on the intergraph edge weights. However, we have one additional step, which will incorporate broader structural information into the matching.

We compute the eigenvalues and eigenvectors of this joint matrix. We select the m largest eigenvalues and their associated eigenvectors. In preliminary experiments, we tried multiple values of m. We found that though the value of m did not drastically change any results, setting m to be a quarter of the size of the joint graph was an acceptable setting. Then the Joint Geometric Graph Embedding Distance is defined between two points $v_i \in G_1$ and $v_j \in G_2$ as:

$$d_{JGGE}(v_i, v_j)^2 = \sum_{k=1}^{m} (\phi_k(v_i) - \phi_k(v_j))^2$$

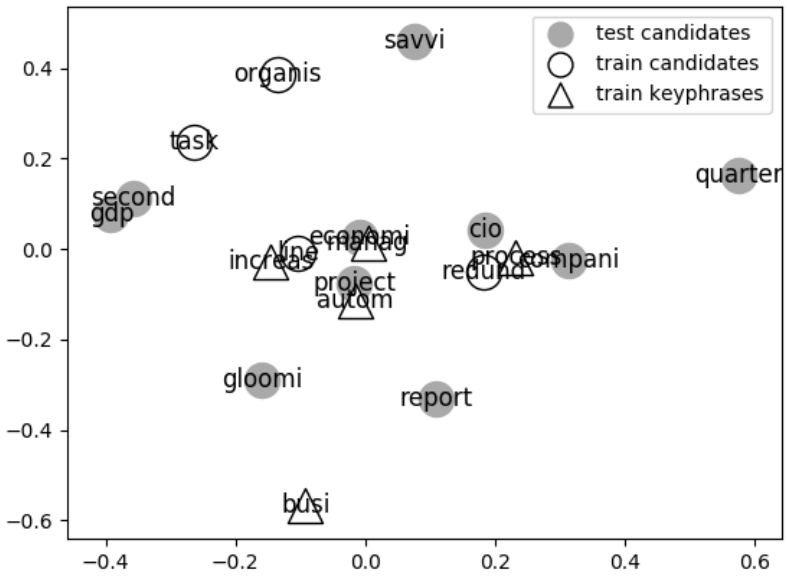

Figure 3.5: Nodes of both graphs are projected into a joint embedding space. Here, the number of eigenvectors (and thus the dimension of the joint embedding space) is $m = 2$ for visualization purposes, and edges are removed for clarity. We look for keyphrases in the test graph (grey nodes) by examining properties of the joint embedding space.

Where ϕ_k is the k^{th} eigenvector of W, and $\phi_k(v_i)$ represents the dimension of ϕ_k corresponding to the node v_i. An intuition for the metric space defined by this distance can be gleaned from observing Figures 3.5 and 3.6.

After this point, there are a number of possibilities for assigning scores to keyphrases using the joint geometric embedding space. We explore two opposing ideas. For the first, we honor the original intuition of using the joint geometric embedding space for matching - we compute matchings by minimizing the distances between each pair of nodes $d_{JGGE}(v_i, v_j)$, enforcing the constraint that matchings must be one-to-one and assigning matches in a greedy fashion. Any node in the test graph matching with a labeled keyphrase in the training graph has its score incremented by 1. In addition, all of the nodes in the test graph which do not obtain an exact match with a labeled keyphrase in the training graph have their scores incremented as well. These leftover nodes v_i are sorted according to $\min_{v_j \in gold\_std\_keys(V_{train})} d_{JGGE}(v_i, v_j)$. Then the nodes v_i that are

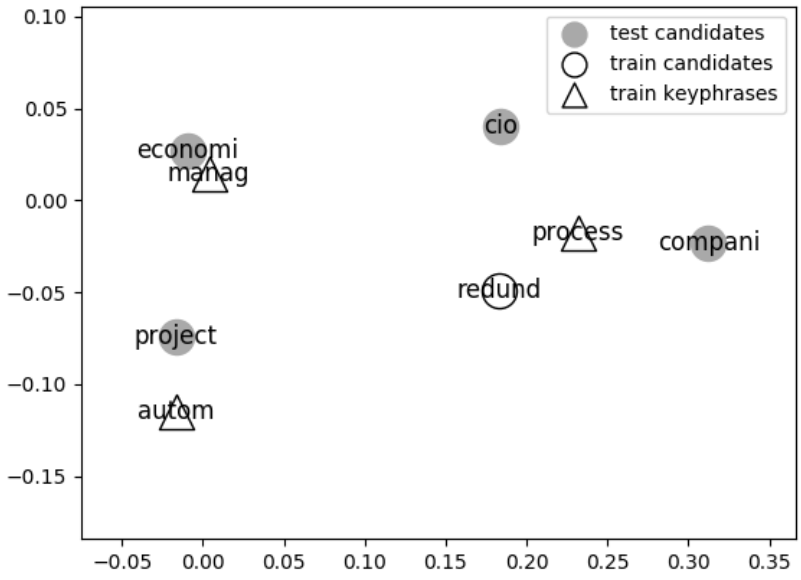

Figure 3.6: We illustrate a zoomed-in portion of the joint embedding space. The only gold standard keyphrase in the test graph is "project," which seems to be quite close to the training keyword "autom" (training keywords are represented by white triangles). Note that this example is not meant to illustrate exactly which properties of the joint embedding space locate keyphrases, but rather to give a simple illustration of how nodes tend to be distributed in the joint embedding space. Specific properties of the joint embedding space will be examined in more detail in Section 4.2.

furthest from a labeled keyphrase v_j receive the smallest score increment, and the nodes v_i that are closest to a labeled keyphrase v_j receive the largest score increment. These score increments for the leftover nodes range uniformly over the interval $(0, 1)$. The pseudocode for this procedure is illustrated by Algorithm 1. The second option for assigining scores to candidates is much simpler. We simply compute a score for each test candidate keyphrase as the sum of the distances between the candidate keyphrase and every labeled training keyphrase in the joint embedding space. The pseudocode for this alternative procedure is given by Algorithm 2.

It should be pointed out that these two ranking orders oppose each other. One rewards candidate keyphrases for being very close to training keyphrases, while the other rewards candidate keyphrases for being very far away from training keyphrases. We will explore this difference in

Section 4.2, and we will also give an indication of which characteristics of datasets lend themselves to which of the two ranking orders.

3.5 Unigram Recombination

This step is only necessary when unigrams are chosen as candidate keyphrases. As previously mentioned, if unigrams are chosen as candidates, they must be recombined into phrases in order to be evaluated against the gold standard set of multiword keyphrases. We investigate four methods of dealing with this discrepancy. For all four methods, once we have obtained unigram scores, we compile a list of candidate phrases in the document. Then, we score phrases by aggregating the scores of their component unigrams. We investigate four methods of aggregation. In Score by Sum, the score of a phrase is the sum of its component unigram scores. This method originates from the SingleRank algorithm in [49], and has been widely used in the literature. In Score by Average, the score equals the average of the component unigram scores, though we compute the average with an extra 1 added to the denominator. The extra 1 rewards longer keyphrases, because it represents a larger portion of the denominator when the number of words in a phrase is smaller [2]. In Score by Average Times Term Frequency, the score is computed as Score by Average, then multiplied by the number of times the phrase appears in the document. This recombination scheme outperforms the Score by Sum and Score by Average schemes in a recent study [14]. In Score by Min, we set the phrase's score equal to the minimum score of any of its component unigrams. This method is intended to mirror the method used in the original TextRank paper ([39]), where first the top p% or top N unigram keywords are selected, and then a given phrase is chosen from the text if and only if it contains only top p% or top N unigrams. The scheme used in the TextRank paper is maximally restrictive, as it requires all of a phrase's component unigrams to have a top p% or top N score. We can approximately replicate the scheme by scoring phrases with their minimum component scores. We do not directly use the original TextRank recombination scheme because it does not allow us to control the exact number of phrases extracted per document, and it is widely ignored in

[2] To take an example, $\frac{0.5}{2} = 0.25$, but $\frac{0.5+0.5}{3} = 0.33$.

the literature. We do not expect the Score by Min recombination method to perform well, but we include it as a point of comparison because it serves to mirror the original recombination method proposed by TextRank.

3.6 Final Selection

Finally, there are two commonly used methods of selecting the top scoring keyphrase candidates as final keyphrases. One can either select a percentage threshold or a absolute threshold, such that the top p% or the top N keyphrases (those with the highest positive scores) are selected. Typically, authors use the top p% method on Inspec, with p=30%. Additionally, SemEval is usually evaluated at N=5, N=10, N=15 or some combination of the three.

CHAPTER 4

EXPERIMENTS AND RESULTS

In this chapter, we evaluate the efficacy of the proposed algorithm, Joint Geometric Graph Embedding Based Keyphrase Extraction. For comparison, we implement TextRank, TFxIDF, and a Random node ranking algorithm. The Random algorithm assigns random scores drawn from the uniform distribution over the unit interval. The other three node ranking algorithms are deterministic in practice. In order to mitigate non-determinism in the scores of the Random algorithm, we run 20 iterations of every experiment with Random and report average F-scores across these 20 iterations.

As previously mentioned, our keyphrase extraction pipeline consists of text preprocessing, candidate extraction, edge weighting, node ranking, unigram recombination if required, and final selection. We keep text preprocessing constant, but experiment with various settings for each other step in the pipeline. For candidate extraction, we consider three options: unigrams, noun and adjective sequences, and the Naïve Bayes noun phrase chunker. For edge weighting, there are two options, cooccurrence sliding window and Word2Vec cosine similarity, yet we determined in preliminary experiments that Word2Vec should only be used in conjunction with JGGE, while cooccurrence sliding window should only be used in conjunction with TextRank. There are four recombination schemes, Score by Sum, Score by Average, Score by Average Times Term Frequency, and Score by Min. We use either a threshold of p% or an absolute limit of N for final selection, though the p% threshold is only applied to Inspec, and the absolute limit of N is only applied to SemEval. Overall, there are 6*4*2=48 experiments in this section - 6 candidate extraction/unigram recombination combinations, 4 node ranking algorithms (two of which do not use

edge weighting, and two of which have had their edge weighting methods fixed beforehand), and 2 datasets.

We first discuss the datasets used for evaluation, then discuss the performance of the four node ranking algorithms when the other elements of the pipeline are selected optimally. We indicate that all else being equal, the Joint Geometric Graph Embedding Based Keyphrase Extraction algorithm attains scores in line with TextRank and TFxIDF, which all strongly outperform the Random method. To investigate further, we indicate dataset characteristics which would lead one to choose Version 1 (in Algorithm 1) or Version 2 (in Algorithm 2) of JGGE. We indicate which subsets of SemEval lead to the best JGGE performance, and why. Finally, we discuss the optimal candidate selection/unigram recombination settings for all datasets and node ranking algorithms.

4.1 DATASETS

We evaluate on two datasets, commonly referred to as SemEval [23] and Inspec [18]. Both datasets consist of text in the domain of computer science research. SemEval contains full conference articles, while Inspec contains only the abstracts of journal articles. Both datasets are freely available online[1].

Summary statistics for both datasets are listed in Table 4.1. This table shows the train/test split for each dataset, the average number of tokens per document, the average number of "acceptable words" per document, the average number of assigned gold standard keyphrases per document, the distribution in gold standard keyphrases among unigrams, bigrams, trigrams, and higher order n-grams, as well as the average length of any single word in any gold standard keyphrase. All statistics are computed across the entire corpus, ignoring the train/test split (except the number of training and testing documents). We define an "acceptable word" as any token with between 1 and 40 characters (inclusive), which has over 50% of its characters defined as a letter in the Unicode

[1] https://github.com/snkim/AutomaticKeyphraseExtraction

Table 4.1: Summary Statistics for Inspec and SemEval Datasets

Statistic	*Inspec*	*SemEval*
Type	Abstracts	Full papers
# Training Documents	1000	144
# Testing Documents	500	100
# Tokens/Document	134	9633
# Acceptable Words/Document	76	4283
Acceptable Words/Tokens (%)	56.7	44.5
# Gold Keyphrases/Document	9.8	15.1
U/B/T/O Distribution in Gold Keyphrases (%)	13/53/25/9	21/53/19/7
Length of Words in Gold Keyphrases	6.1	6.4

Character Database[2], and is not in NLTK's list of English stopwords. This list of stopwords is available in Appendix B.

In order to train the Word2Vec model, we require an external unlabelled dataset. The ArXiv academic preprint repository is suitable for this purpose. We discuss this dataset after introducing the other two.

4.1.1 SemEval 2010 Task 5

The SemEval 2010 Task 5 dataset, compiled by [23], is the most recent and influential dataset under evaluation. There are many recent research articles which use this dataset, the articles are grammatically correct with few spelling errors, and the articles are not too short or long so as to be either trivial or computationally intractable. In addition, the domain (computer science research articles) has a well-defined taxonomy, with terminology that is sufficiently different from everyday speech to require novel models rather than pre-packaged solutions. One final benefit of this dataset is that documents are labeled as belonging to four distinct sub-domains of computer science, enabling further analysis broken down by sub-domain. The training and test sets have an even distribution of

[2] `http://unicode.org/ucd/`

each sub-domain, meaning that an algorithm will not be disproportionately rewarded or penalized for its performance on any single domain.

Each document is an academic conference article from one of four domains: information search and retrieval, distributed systems, multiagent systems, or economics. The creators downloaded the articles from the ACM Digital Library. Thus, the documents were labelled under the 1998 ACM Computing Classification System [3]. Under this system, the four categories in this corpus are labelled H3.3, C2.4, I2.11, and J4, respectively.

The original dataset creators downloaded the articles in PDF format, then converted them to text with the Linux command-line tool pdftotext [4]. They also converted words that had been split across two lines and hyphenated back to their original uninterrupted form. However, they note they may have accidentally converted legitimate hyphenated words into a single string of alphabetical characters during this step.

There are 144 training documents and 100 testing documents. In addition, the authors of this task provided entrants with a 40-document subset of the training set as "trial data". We do not concern ourselves with this set in our study, because it is a proper subset of the training set. The authors ensured an approximately uniform distribution of the four document categories in both the training and testing data. Although the testing data is exactly balanced among the four categories (25 articles per category), the training set is slightly skewed. There are 34 distributed systems papers, 39 information search and retrieval papers, 35 multiagent systems papers, and 36 economics papers.

The papers are annotated with author- and reader-assigned keyphrases. The author-assigned keyphrases may or may not be present in the article text. The author-assigned keyphrases were already present in the PDF versions of the documents at the time of retrieval from the ACM Digital Library. The reader-assigned keyphrases came from 50 Computer Science students at the National University of Singapore, who were hired to annotate 5 papers each. The readers were told to assign keyphrases that were present word-for-word in the article text, headers or captions, not simply

[3] More information on this classification system is available online at `https://www.acm.org/publications/computing-classification-system`.

[4] `https://linux.die.net/man/1/pdftotext`

semantically equivalent phrases. However, the actual percentage of reader-assigned keyphrases present in the article text was 85%, thus the ceiling for recall of any keyphrase extraction system is 85% on this dataset. This percentage is better than the 81% of author-assigned keyphrases that were present in the text. We will use the combined keyphrases (the union of the reader- and author-assigned keyphrases) since they allow us to obtain the maximum possible recall. The original task evaluates systems on both reader-assigned keyphrases and combined keyphrases.

The SemEval creators also discuss the degree of agreement between the reader-assigned keyphrases and the author-assigned keyphrases. If one takes the author-assigned keyphrases as a baseline to compute the precision, recall, and F-score of the reader-assigned keyphrases (a kind of "inter-annotator" precision, recall, and F-score), one obtains a precision of 21.5%, recall of 77.8%, and F-score of 33.6%. These scores are not directly comparable to any algorithm's scores on the combined keyphrase sets. The readers assigned about 12 keyphrases per document on average, while the authors assigned about 4 keyphrases per document on average. This discrepancy between reader- and author-assigned keyphrase set sizes artifically inflates the inter-annotator recall and deflates the inter-annotator precision, making direct comparison to actual keyphrase extraction algorithms impossible. However, the reasonably high precision, recall, and F-score indicate a decent amount of agreement between the readers and the authors.

During data exploration, we found that for each SemEval document, almost all of the gold standard keyphrases are located in the beginning of the document. The histogram of locations of gold standard unigrams in the training set can be seen in Figure 4.1. It is apparent that most gold standard unigrams appear in the first 20% of their respective documents, with a heavy concentration in the first 5-10% of the documents. Indeed, when we gave only the first 6% of each SemEval document to the keyphrase extraction pipeline, we attained much better results on every algorithm. This performance improvement persisted despite the fact that we cut away some gold standard keyphrases when we remove most of the document. Therefore, all of our experiments on SemEval cut out the latter 94% of each training and testing document as part of the preprocessing stage.

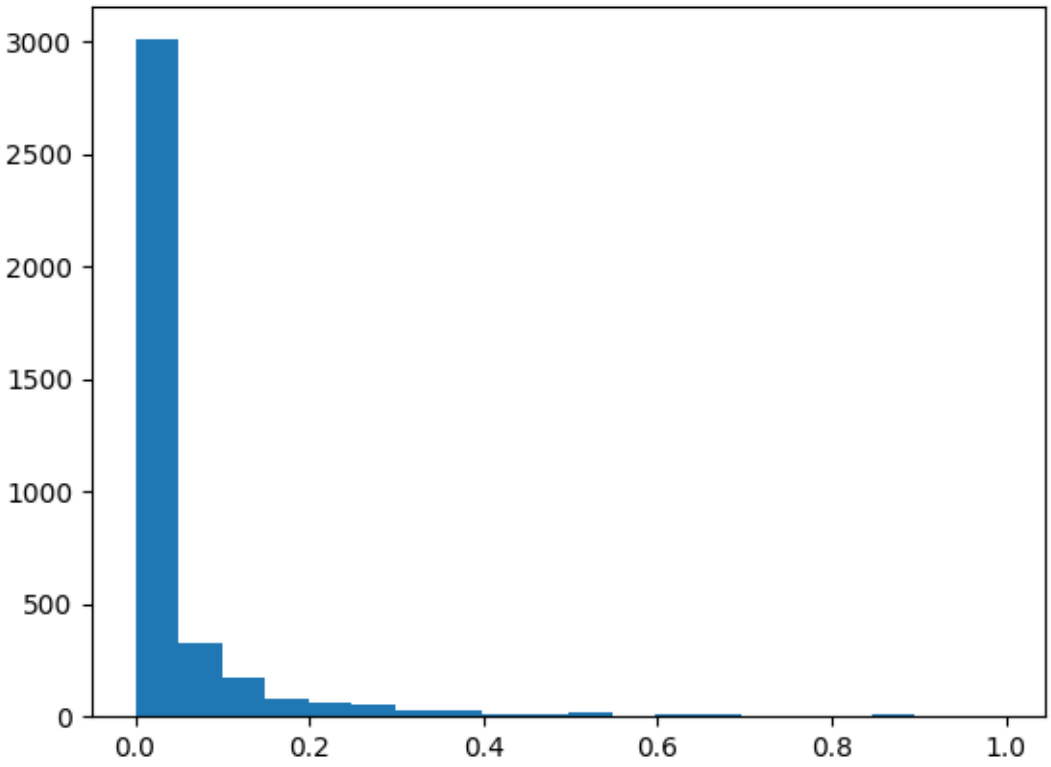

Figure 4.1: Frequencies of gold standard unigrams at various percentages through their respective SemEval documents, binned in increments of 5%.

The performance improvement from cutting out the latter 94% of all documents derives from the fact that keyphrases tend to be concentrated in certain sections of technical literature [44], which we have empirically determined to consist mostly of the abstract and introduction in the SemEval dataset. However, some authors have applied more sophisticated filtering techniques on SemEval, such as using only the title, abstract, introduction, related work, conclusions, and future work [34] or including section-based features in their keyphrase extraction algorithm [30]. In this work, we are mostly concerned with the general applicability of the Joint Geometric Graph Embedding framework across different datasets. Therefore we consider the simpler solution of selecting the first 6% of each document to be sufficient given the focus of this work.

4.1.2 INSPEC

The Inspec corpus was originally introduced by Hulth in 2003, and it has been used in numerous studies since, becoming a classic dataset in keyphrase extraction [18]. We evaluate on this dataset to determine how well different algorithms work on very small documents. This dataset is also very commonly reported in the literature, enabling us to compare our results to others quite readily. Hulth claims that "many journal papers are not available as full-length texts, but as abstracts only, as is the case for example on the Internet." Yet it still would be restrictive for humans to read every abstract for thousands of papers. We must assign keywords to the abstracts to manage and filter them before presenting them to humans interested in specific topics.

The documents in the corpus are composed of only the abstracts of computer science papers from journals in the domains of *Computers and Control* and *Information Technology* during the years 1998 to 2002. There are 2,000 documents, though Hulth has already arbitrarily separated the dataset into 1,000 training documents, 500 validation documents, and 500 testing documents. We follow the same train/test split, though we dispose of the validation documents. Hulth obtained these journal articles from the *Inspec* database, which includes a taxonomy of controlled keyphrases for tagging documents.

Hulth includes with each abstract a set of controlled and uncontrolled keyphrases, both assigned by a professional indexer. The controlled terms were selected from the Inspec thesaurus, while the uncontrolled terms could consist of any phrase in the English language. The indexers had access to the full text of the documents when assigning the terms. Neither the controlled or uncontrolled terms were required to be present in the abstract. Interestingly, 76.2% of the uncontrolled terms are present in the abstract of the document, while only 18.1% of the controlled terms are present in the abstracts. Thus most studies using this dataset use the uncontrolled terms as the ground truth. However, the fact that nearly three-fourths of the uncontrolled terms appear in the abstract despite the ability of the indexers to choose any terms from the entire article is a further indication of the relevance of this dataset. If most of a document's keyphrases appear in the abstract, we can

construct adequate search and retrieval systems for full documents using only keyphrases extracted from abstracts. Therefore, extracting keyphrases from abstracts is a valuable task.

Some studies have additionally taken the liberty of removing any gold standard keyphrases that do not appear in the abstract of the document to which they are assigned [29; 15]. Because such ground truth terms are not extractable from the abstracts, they result in a ceiling on the recall of any keyphrase extraction algorithm. We do not take this corrective step of removing unextractable keyphrases, as we feel it would be misleading to compare the resulting scores to scores which have been computed directly against the original gold standard keyphrases.

4.1.3 ARXIV

Word2Vec requires a large secondary corpus for training. The domain of the documents in the corpus ought to be as similar to the test documents as possible, to maximize the likelihood that specific technical terms will appear in the corpus. One other reason we need our secondary corpus to be in a similar domain to the test corpus is that within a domain rare words may have different meanings than they do in normal contexts. Thus it is important to have them appear in the correct context in the secondary corpus.

Most papers that seek semantic similarity metrics based on external knowledge sources have used either Wikipedia[5] or WordNet[6] as their sources. However, these sources are not useful for this thesis because they are not specific enough to the domains under consideration. However, the online research preprint repository ArXiv[7] does have numerous scientific articles with categories labelled. We use the set of computer science articles on ArXiv as our secondary corpus.

ArXiv labels papers at two levels. The first level is the high-level domain. It includes physics, mathematics, computer science, nonlinear sciences, quantitative biology, quantitative finance, statistics, and electrical engineering and systems science. The second level classifies sub-domain. Computer science articles are provided on ArXiv under a related site called the Computing

[5] `www.wikipedia.com`

[6] `https://wordnet.princeton.edu/`

[7] `https://www.arxiv.org`

Table 4.2: Equivalencies of Computing Research Repository Classes to 1998 ACM Computing Classification System Classes

CoRR	*ACM*
Information Retrieval	H3.0, H3.1, H3.2, **H3.3**, H3.4
Distributed, Parallel, and Cluster Computing	C1.2, C1.4, **C2.4**, D1.3, D4.5, D4.7, E1
Artificial Intelligence	I2.0, I2.1, I2.3, I2.4, I2.8, **I2.11**
Computational Engineering, Finance, and Science	J2, J3, **J4**

Research Repository (CoRR). CoRR lists computer science articles belonging to 40 sub-domains. A full list of all 40 can be found in Appendix C. These sub-domains have been decided upon by the curators of CoRR. In addition to this categorization scheme, CoRR also categorizes articles according to the 1998 ACM Computing Classification System. To compare to the ACM classes included in SemEval 2010 Task 5, we list the equivalencies for the 4 classes used in SemEval in Table 4.2. The ACM classes which appear in the SemEval corpus are bolded.

ArXiv papers are available for bulk download from an Amazon S3 requester-pays bucket[8], in both PDF and source LaTeX format. We downloaded all PDF's from July 1991 to December 2016, then selected exactly the computer science articles from that set of papers. Computer science articles were identified via a stamp added to the side of the PDF which lists the file name and then a tag indicating the subject and subdiscipline as [XX.XX], for example [CS.AI]. We used this information to only select the computer science articles, and to sort them into the 40 subdisciplines. We then converted all PDF's to text using pdftotext [9], a Linux command-line tool. Although the pdftotext converter does not handle equations or figures very elegantly, we suspect that this tool can convert a reasonable fraction of full sentences to text, enabling our semantic model parameters to be calculated to a useful degree of accuracy.

[8] `https://arxiv.org/help/bulk_data_s3`

[9] `https://linux.die.net/man/1/pdftotext`

The amount of articles in each subject is displayed in Figure 4.2. The total number of articles is 76,160. The average length of an article is 11,121 tokens. The average number of "acceptable words" per article is 4,534. Thus, the fraction of "acceptable words" to total tokens is approximately 40.8%, which is comparable to the other 2 datasets. This is a soft indication that the pdfto-text conversion to text was successful, an indication which is considerably strengthened by our subjective evaluation of some random documents.

4.2 Performance of Joint Geometric Graph Embedding Based Keyphrase Extraction

We performed experiments for various candidate extraction methods, edge weighting methods, node ranking algorithms, and unigram recombination procedures. We optimized the keyphrase extraction pipeline for each of four algorithms under consideration: Random, TFxIDF, TextRank, and Joint Geometric Graph Embedding Based Keyphrase Extraction (JGGE-KE or simply JGGE), and fixed the optimal settings of the other steps in the pipeline for each algorithm. We found that across all four algorithms, for SemEval the best candidate extraction algorithm was Unigrams with Score by Average Times Term Frequency recombination, while for Inspec the best candidate extraction algorithm was Unigrams with Score by Sum recombination. In addition, preliminary experiments showed that Word2Vec edge weighting performed rather well when combined with the proposed JGGE keyphrase extraction algorithm, but did not perform very well for TextRank. Therefore, we only report results using cooccurrence-based edge weighting with TextRank and Word2Vec based edge weighting with JGGE. The alternative candidate extraction and unigram recombination schemes are explored in the next section, but here we are only concerned with the performance of each algorithm given the optimal candidate extraction, edge weighting, and unigram recombination scheme. The results of the four algorithms with optimal settings for the rest of the pipeline are indicated in Table 4.3. Overall, we see that the JGGE algorithm performs similarly to TFxIDF and TextRank on SemEval and Inspec, outperforming the other two on Inspec at the

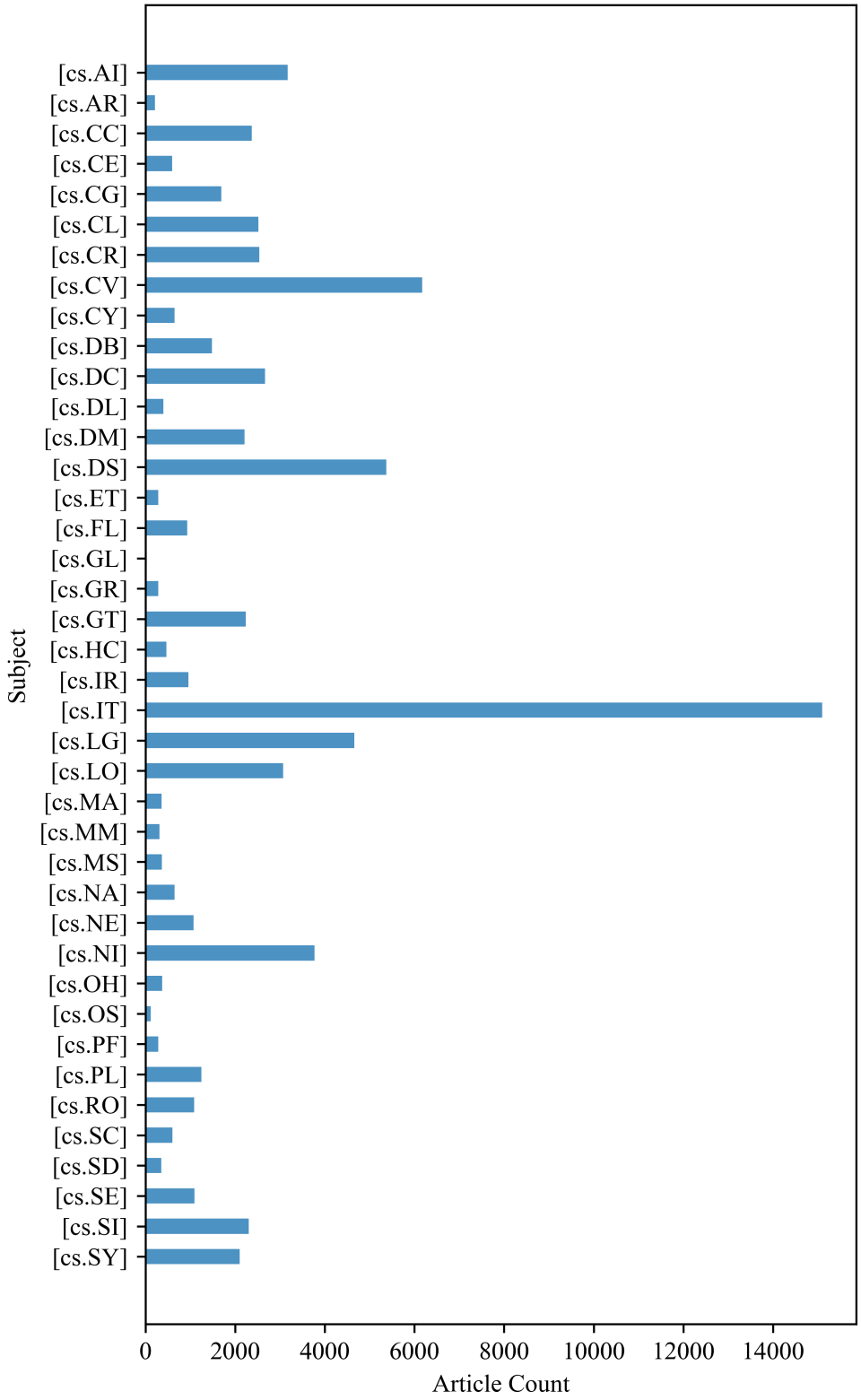

Figure 4.2: Article Counts for ArXiv's CoRR

p=0.05 significance level based on a two-way Wilcoxon signed ranks test with Bonferroni correction. In addition, we found that all three non-Random algorithms outperform the Random algorithm at p=1e-7 significance level, indicating very strongly that the intuition behind these algorithms are correct. In particular, we take this result to mean that the joint geometric embedding space does indeed encode information related to the "keyphraseness" of words in text, and is worthy of further exploration in other NLP tasks.

Table 4.3: F-scores (for Full Phrases) of Random, TFxIDF, TextRank, JGGE, and State-of-the-art Methods on Inspec and SemEval 2010 Task 5. F-scores for Inspec are computed by selecting the top 30% of the ranked candidate keyphrases for each algorithm. F-scores for SemEval are computed by selecting the top 10 and top 15 ranked candidate keyphrases for each algorithm. All non-Random algorithms outperform Random at the p=1e-7 significance level. † indicates significance over other non-Random algorithms at the p=0.05 level. Statistical significance computed using paired Wilcoxon signed ranks tests with Bonferroni correction.

Algorithm	Inspec (p=30%)	SemEval (N=10)	SemEval (N=15)
Random	25.2	8.8	9.1
TFxIDF	34.8	**14.3**	**15.3**
TextRank	35.7	13.8	**15.3**
JGGE	**37.1**†	12.8	14.6
RankUp	46.6	–	–
MultipartiteRank	–	14.5	–
HUMB	–	25.9	27.5
SemGraph	–	–	30.8

Our results are not directly comparable to the four algorithms from the literature without some additional discussion. One difference is the level of supervision. TextRank is completely unsupervised, JGGE-KE requires slightly more supervision, and TFxIDF requires an entire corpus for comparison. HUMB is a supervised method, which means that it requires more resources to train than TextRank or JGGE-KE [30]. RankUp, MultipartiteRank, and SemGraph are unsupervised, but the studies reporting on their effectiveness use different settings of the rest of the keyphrase extraction pipeline than what has been used in our implementations of the four algorithms studied in this work [13; 9; 34].

For example, the RankUp study's reimplementation of the basic TextRank algorithm achieves an F-score of 42.8, so the additional improvement of their novel method over TextRank is 3.8 points in terms of F-score, rather than the 10.9 point difference between our implementation of TextRank and their novel algorithm. In a personal correspondence with the authors, they claimed that the improvement in the score for their implementation of TextRank came from using noun phrases as candidate keyphrases (the original TextRank implementation used unigrams), and setting their thresold p% as a percentage of the *number of words in the document*, rather than the number of candidate keyphrases chosen. They noted that p=10% was selected for Inspec. However, the description of the candidate selection in the RankUp paper appears to be the same as our implementation of the Noun and Adjective Sequences candidate selection, and none of our experiments showed TextRank exceeding an F-score of approximately 36 no matter what the threshold. We relate this discussion here in order to indicate that the substantially higher scores of the RankUp algorithm are likely due in part to optimized settings in other parts of the pipeline - settings which would boost the scores of any node ranking algorithm.

The results of the MultipartiteRank algorithm in [9] are likely directly comparable to the results of the current study. One interesting difference between our keyphrase extraction pipeline and theirs is that we cut off the SemEval documents at 6% mark from the very start of the pipeline, restricting even the candidate extraction step to selecting only words in the first 6% of the documents. In the MultipartiteRank study, the authors use the entire SemEval documents, but add a factor to each edge weight in the graph inversely related to the position of the incident nodes in the document. Thus words appearing early in the document have higher in-degree than words appearing later in the document, which leads to a higher score for the early words in the documents since MultipartiteRank's core algorithm is the PageRank centrality computation. We do not explore this more subtle approach to incorporating position information on SemEval, as our main aim in this study is to demonstrate the basic utility of the joint geometric embedding space in the keyphrase extraction task. However, the weighting approach is an obvious extension which could increase SemEval recall of all four algorithms under consideration in this study (if the position-

biased weighting were applied after node ranking). The study presenting SemGraph indicates a similar benefit, and their usage of position information on SemEval is even more advanced [34]. They use a conditional random field to label sections of the documents, then extract candidates only from the "title, abstract, introduction, related work, future work, and conclusion" of the papers [34]. Indeed, they report a TextRank F-score of 22.1 on SemEval, compared to the score of 15.3 indicated by our own implementation. We suspect therefore that some of their improvement in F-score derives from their choice to select candidates from very specific portions of the documents, which we have not done in as much detail.

4.2.1 Maximizing or Minimizing JGGE Distance

Interestingly, we found that when we compute JGGE scores as the sum of distances to keyphrases (Version 2 as shown in Algorithm 2 in Section 3.4), the scores in Inspec are maximized. However, on SemEval, the F-score is maximized by computing JGGE scores so as to minimize distance to training keyphrases (Version 1 as shown in Algorithm 1 in Section 3.4). We hypothesize that SemEval gold standard keyphrases serve more to broadly cover the document topics (they emphasize within-document diversity of keyphrases) while Inspec gold standard keyphrases serve more to differentiate documents from each other (emphasizing cross-document differentiation). Emphasis on within-document diversity intuitively corresponds with minimizing the JGGE distance, while emphasis on cross-document differentiation corresponds with maximizing the JGGE distance. This difference in emphasis results in the difference in optimal ordering behavior on the two datasets.

To further explore this hypothesis, we use the metric of lexical diversity to measure the diversity of gold standard keywords for Inspec and SemEval. Lexical diversity can be computed as the number of types divided by the number of tokens (the so-called type-token ratio). However, it has been found that, ceteris paribus, longer sequences of text have lower type-token ratios [19]. Broadly speaking, when authors use more tokens, they are forced to repeat the most important tokens. Thus we must measure lexical diversity as type-token ratio corrected for differences in total number of tokens. We can compute both the "cross-document" lexical diversity, as well as

the "within-document" lexical diversity. We expect that Inspec gold standard keyphrases will have higher cross-document diversity (favoring greater JGGE distance), while SemEval will have higher within-document diversity (favoring lesser JGGE distance).

To compute "cross-document" lexical diversity for a dataset, we randomly select 3,000 tokens from the list of unigrams contained in all training documents' gold standard keyphrase sets (we choose 3,000 because there are approximately 5,000 tokens across SemEval's training set and approximately 23,000 tokens across Inspec's training set, and we would like to choose a large number of tokens while still allowing for randomness on both datasets). We then can compute the number of unique types across these 3,000 tokens, and divide by 3,000. We repeat this procedure 100 times for both datasets, obtaining a list of 100 cross-document lexical diversity measurements each. We find the average lexical diversity on SemEval to be 30.50% with 0.002% variance, and the average lexical diversity on Inspec is 47.50% with 0.006% variance. The two lexical diversity distributions differ to a degree that is statistically significant at $p<0.001$. The gold standard keyphrases seem to be more diverse on Inspec's training set, indicating that the keyphrases might be a bit more focused on differentiating the documents than in SemEval.

Conversely, we can compute the within-document lexical diversity for each document of both training sets, then compare the distributions of lexical diversities. In this case, we must control for differing numbers of unigrams in the gold standard sets for individual documents. We compute the average number of gold standard tokens for a single document to be approximately 18 in the Inspec training set, and approximately 25 on SemEval. Therefore, we consider only documents that have at least 18 tokens in their gold standard keyphrases. For each document, we randomly select 18 tokens and compute the type-token ratio over those 18 tokens. On Inspec's training set, we find within-document lexical diversity computed in this manner to have an average of 67.2%, and a variance of 1.20%. SemEval's training set documents have an average within-document lexical diversity of 66.4% with a variance of 1.07%. Using a paired t-test, we determine these 2 sets of lexical diversities to be not significantly different at the $p=0.1$ level. The implication is

that neither dataset has more lexical diversity within documents, indicating that topic coverage is roughly equally important in both datasets.

We conclude that one half of our original hypothesis is statistically validated - that is, Inspec annotators focused more on differentiating documents than SemEval annotators did. Annotators of both datasets focused on broad topic coverage. As a result, the JGGE keyphrase extraction algorithm based on maximizing distance between documents boosted scores on Inspec, but maximizing distance between documents had much less positive influence on SemEval. On SemEval, the dominant effect is that of emulating common patterns in the joint geometric embedding space (minimizing distance in the embedding space).

4.2.2 JGGE F-scores on the 4 Subsets of SemEval

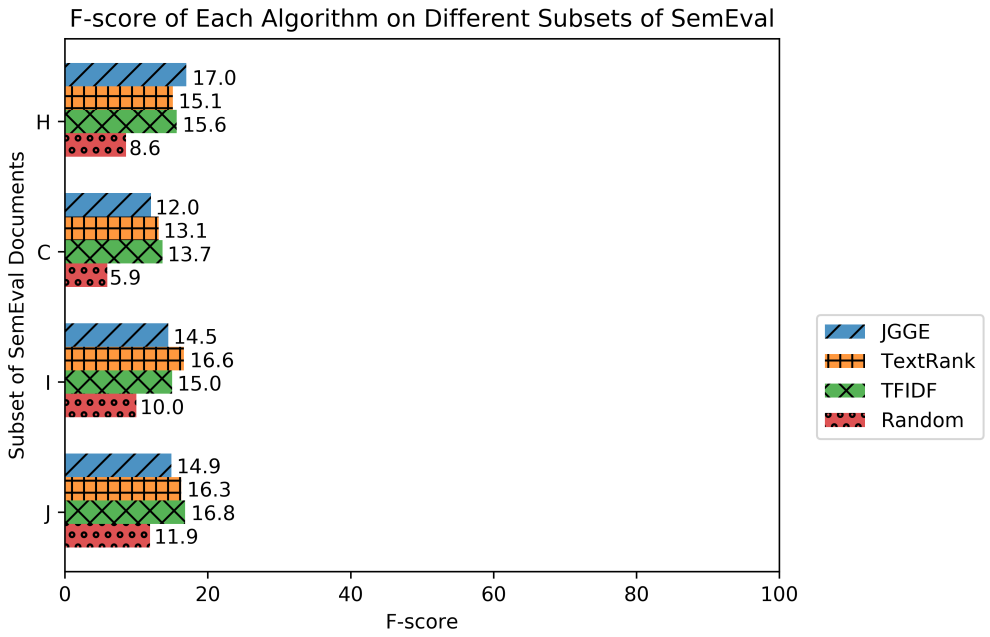

Figure 4.3: F-scores on Different Subsets of SemEval 2010 Task 5 Combined Keyphrases (N=15). The subsets are labeled with their ACM 1995 Classification System abbreviations: Information Retrieval (H), Distributed Systems (C), Artificial Intelligence (I), and Computational Finance/Economics (J).

In order to better understand the performance of JGGE, we can also consider its performance across multiple known domains. In Figure 4.3, we see that the JGGE algorithm performs rather

well on the Information Retrieval subset of the SemEval dataset, while performing adequately but less well than TextRank and TFxIDF on the other three subsets. We suspect this is due to the Word2Vec edge weighting model having more training data on the vocabulary used in Information Retrieval research. If we briefly examine the Word2Vec model, we find that 5.7% (179/2973) of the candidate unigrams in the Information Retrieval (H) test set are not found in the Word2Vec model. In comparison, the Distributed Systems (C) subset has 6.4% (200/2946) of its unigrams out-of-model, the Artificial Intelligence (I) subset has 6.2% (203/3090) of its unigrams out-of-model, and the Computational Finance/Economics (J) subset has 6.0% (222/3484) of its unigrams out-of-model. These out-of-model fractions line up perfectly with the differences in JGGE performance on the subsets - H has the highest F-score for JGGE, and the lowest out-of-model fraction. J, I, and C see increasing out-of-model fractions, and correspondingly decreasing JGGE F-scores. Though not conclusive proof that the differences in JGGE F-scores are expressly caused by differences in the out-of-model fraction for each subset, this analysis does implicate lack of coverage in the Word2Vec model as a primary suspect for differentiating JGGE F-scores.

4.3 Variation of Candidate Keyphrase Selection

It is rather informative to consider the effect of the various candidate selection methods. In this section, we look at the top scoring candidate selection methods across the two datasets and the four node ranking algorithms. As mentioned previously, preliminary experiments showed that Word2Vec edge weighting performed well with JGGE, but did not perform very well for TextRank, so we use only cooccurrence-based edge weighting with TextRank and Word2Vec based edge weighting with JGGE.

Figure 4.4 indicates the F-scores attained using the four node-ranking algorithms when combined with the three candidate extraction methods on Inspec. In the case of extraction of unigram candidates, there are four schemes for recombination of unigrams into full phrases. It is plain to see that on the Inspec data, the Unigrams with Score by Sum Recombination scheme maximizes F-scores. We suspect that this is likely due to the fact that this recombination promotes longer

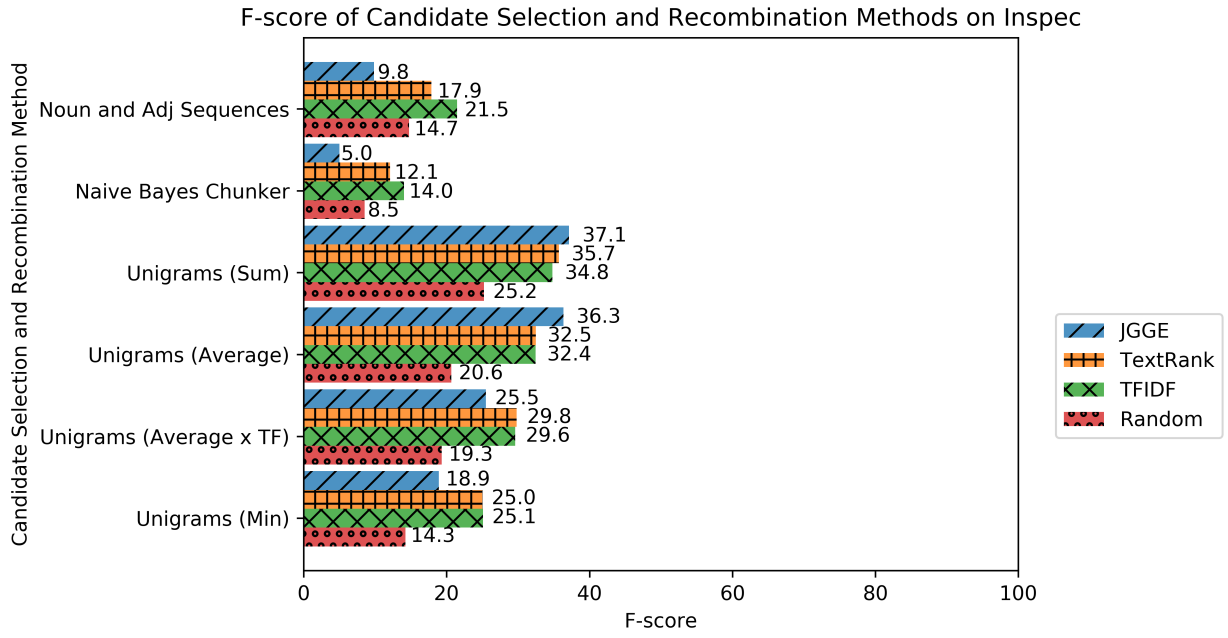

Figure 4.4: F-scores with Various Candidate Selection Methods and Node Ranking Algorithms on Inspec Reader-Assigned Keyphrases (p=30%)

candidate keyphrases. If we consider the percentages of unigrams, bigrams, trigrams, and higher order n-grams in Inspec's gold standard, we see approximately 13% unigrams, 53% bigrams, 25% trigrams, and 9% higher order n-grams. However, the candidate keyphrases (the Noun and Adjective Sequences, which are the phrases which the unigram methods recombine into) have approximately 43% unigrams, 37% bigrams, 14% trigrams, and 6% higher order n-grams. Thus, it appears that any keyphrase extraction pipeline which ultimately promotes longer keyphrases will serve to convert the candidate distribution, which is more heavily centered around unigrams and bigrams, into a distribution more heavily centered around bigrams and trigrams such as the gold standard keyphrases. The Unigrams with Score by Average Recombination scheme also promotes longer keyphrases to a lesser degree, due to the addition of an extra 1 to the denominator when computing the average. Thus, this scheme seems to be the second best of the 4. The Score by Average Times Term Frequency recombination method is neutral in this regard (it neither promotes nor demotes

longer keyphrase candidates), while the Score by Min recombination method actually punishes longer keyphrases (more component unigrams means more opportunity for a low score, all else being equal).

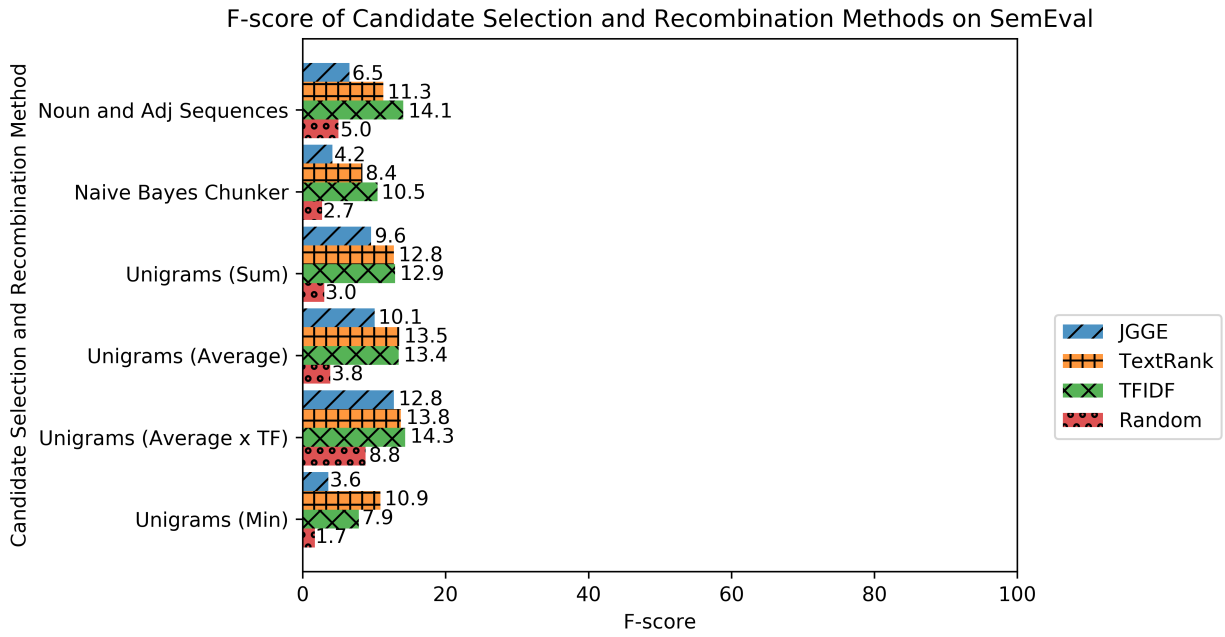

Figure 4.5: F-scores with Various Candidate Selection Methods on SemEval 2010 Task 5 Combined Keyphrases (N=15)

Figure 4.5 demonstrates the effectiveness of the various candidate extraction and node-ranking algorithm combinations on the SemEval dataset. We find that the Score by Average Times Term Frequency recombination method outperforms the other candidate selection methods. However, the Score by Sum, Average, and Average Times Term Frequency methods all seem to perform rather well on this dataset, and we do not see the same pronounced differences that we do on the Inspec dataset. Again, we see that the Noun and Adjective Sequences, Naïve Bayes noun phrase chunker, and Unigrams with Recombination by Min methods perform comparatively poorly, which is to be expected.

In both the Inspec and SemEval experiments, we see that although JGGE performs quite well for the Unigrams with Recombination candidate selection scheme, performance of JGGE is disproportionately poor (compared to TextRank and TFxIDF) when the Naïve Bayes noun phrase

chunker or the Noun and Adjective Sequences candidate extraction schemes are used. We contend that this is due to poor quality of the Word2Vec models used for those two candidate extraction schemes. Recall we trained three Word2Vec models on the ArXiv corpus, one model where we identified phrases using the Naïve Bayes noun phrase chunker, one with phrases identified using the Noun and Adjective Sequences method, and one where we simply trained directly against the unigrams. We computed the number of test document candidate keyphrases in each Word2Vec model vs. the number of test document candidate keyphrases not contained in each Word2Vec model. We found that for unigram candidates, the Word2Vec model has vectors for 19,983 candidate unigrams, while it is missing only 609 (out-of-model fraction is 2.0%). In the case of the Naïve Bayes noun phrase chunker, the Word2Vec model has vectors for 6,419 candidate keyphrases, but is missing 10,267 candidate keyphrases (out-of-model fraction is 62.5%). Likewise, for Noun and Adjective Sequences, the Word2Vec model has vectors for 9,564 candidate keyphrases, and it is missing 5,051 candidate keyphrases (out-of-model fraction is 34.6%). The discrepancy in Word2Vec coverage is only natural - as phrases get longer, their frequencies decrease drastically. Therefore, it is reasonable to expect that a Word2Vec model trained on longer phrases would have lower recall than a Word2Vec model trained on the same corpus with only unigrams. We take this result to mean that the Naïve Bayes chunker and Noun and Adjective Sequences candidate extraction schemes are poor choices as preprocessing steps, rather than an indication that our JGGE algorithm performs poorly. This interpretation is supported by the fact that TFxIDF and TextRank also perform relatively poorly when paired with these two candidate extraction methods.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

Overall, we find that embedding training and testing graphs in a joint spectral embedding space encodes vital semantic and graph theoretic information for keyphrase extraction. However, the quality of the embedding space is highly dependent on the quality of the edge weighting used during graph construction. Though we have indicated that joint geometric graph embedding can be successfully used in keyphrase extraction, more research is necessary to develop improved edge weighting methods.

In light of the success of spectral graph embedding on this problem, we suggest that other problems be cast as object recognition in weighted graphs (the traditional domain in which inexact graph matching is applied). Document summarization is obvious because the transition from keyphrase extraction to document summarization is already done in many previous research papers (most notably, TextRank [39]). It may also be quite useful for relation extraction. If two nodes match with nodes in some ordered relation in a certain proportion of the training set, they may be said to hold that ordered relationship with each other.

Despite any limitations of the current study, we have demonstrated that there is much to gain in recasting keyphrase extraction as a structural pattern recognition problem. We hope that this research will spur further applications of structural pattern recognition algorithms to areas of information extraction.

BIBLIOGRAPHY

[1] W. D. Abilhoa and L. N. De Castro. A keyword extraction method from twitter messages represented as graphs. *Applied Mathematics and Computation*, 240:308–325, 2014.

[2] I. Augenstein, M. Das, S. Riedel, L. Vikraman, and A. McCallum. Semeval 2017 task 10: Scienceie-extracting keyphrases and relations from scientific publications. *arXiv preprint arXiv:1704.02853*, 2017.

[3] R. Barzilay and M. Elhadad. Using lexical chains for text summarization. 1997.

[4] S. Beliga, A. Meštrović, and S. Martinčcić-Ipšić. Toward selectivity based keyword extraction for croatian news. *arXiv preprint arXiv:1407.4723*, 2014.

[5] K. Bennani-Smires, C. Musat, M. Jaggi, A. Hossmann, and M. Baeriswyl. Embedrank: Unsupervised keyphrase extraction using sentence embeddings. *CoRR*, abs/1801.04470, 2018. URL `http://arxiv.org/abs/1801.04470`.

[6] G. Berend. Opinion expression mining by exploiting keyphrase extraction. 2011.

[7] S. Bird, E. Loper, and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[8] F. Boudin. A comparison of centrality measures for graph-based keyphrase extraction. In *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 834–838, 2013.

[9] F. Boudin. Unsupervised keyphrase extraction with multipartite graphs. *CoRR*, abs/1803.08721, 2018. URL `http://arxiv.org/abs/1803.08721`.

[10] F. Boudin, H. Mougard, and D. Cram. How document pre-processing affects keyphrase extraction performance. In *COLING 2016 Workshop on Noisy User-generated Text (WNUT)*, 2016.

[11] A. Bougouin, F. Boudin, and B. Daille. Topicrank: Graph-based topic ranking for keyphrase extraction. In *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 543–551, 2013.

[12] E. DâĂŹAvanzo and B. Magnini. A keyphrase-based approach to summarization: the lake system at duc-2005. In *Proceedings of DUC*, 2005.

[13] G. Figueroa, P.-C. Chen, and Y.-S. Chen. Rankup: Enhancing graph-based keyphrase extraction methods with error-feedback propagation. *Computer Speech Language*, 47: 112 – 131, 2018. ISSN 0885-2308. doi: https://doi.org/10.1016/j.csl.2017.07.004. URL http://www.sciencedirect.com/science/article/pii/S0885230816300377.

[14] C. Florescu and C. Caragea. A new scheme for scoring phrases in unsupervised keyphrase extraction. In J. M. Jose, C. Hauff, I. S. Altıngovde, D. Song, D. Albakour, S. Watt, and J. Tait, editors, *Advances in Information Retrieval*, pages 477–483, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56608-5.

[15] K. S. Hasan and V. Ng. Conundrums in unsupervised keyphrase extraction: Making sense of the state-of-the-art. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 365–373, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1944566.1944608.

[16] K. S. Hasan and V. Ng. Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1262–1273, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P14-1119.

[17] C. Huang, Y. Tian, Z. Zhou, C. X. Ling, and T. Huang. Keyphrase extraction using semantic networks structure analysis. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 275–284. IEEE, 2006.

[18] A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 216–223, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1119355.1119383. URL `http://dx.doi.org/10.3115/1119355.1119383`.

[19] V. Johansson. Lexical diversity and lexical density in speech and writing: A developmental perspective. *Working Papers in Linguistics*, 53:61–79, 2009.

[20] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[21] J. S. Justeson and S. M. Katz. Technical terminology: Some linguistic properties and an algorithm for identification in text. *Natural Language Engineering*, 1(1):9–27, 1995.

[22] S. N. Kim, T. Baldwin, and M.-Y. Kan. Evaluating n-gram based evaluation metrics for automatic keyphrase extraction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 572–580. Association for Computational Linguistics, 2010.

[23] S. N. Kim, O. Medelyan, M.-Y. Kan, and T. Baldwin. Semeval-2010 task 5: Automatic keyphrase extraction from scientific articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 21–26, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1859664.1859668`.

[24] T. Kiss and J. Strunk. Unsupervised multilingual sentence boundary detection. *Comput. Linguist.*, 32(4):485–525, Dec. 2006. ISSN 0891-2017. doi: 10.1162/coli.2006.32.4.485. URL `http://dx.doi.org/10.1162/coli.2006.32.4.485`.

[25] S. Lahiri, S. R. Choudhury, and C. Caragea. Keyword and keyphrase extraction using centrality measures on collocation networks. *arXiv preprint arXiv:1401.6571*, 2014.

[26] D. Lawrie, W. B. Croft, and A. Rosenberg. Finding topic words for hierarchical summarization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 349–357. ACM, 2001.

[27] M. Litvak and M. Last. Graph-based keyword extraction for single-document summarization. In *Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24. Association for Computational Linguistics, 2008.

[28] Z. Liu, P. Li, Y. Zheng, and M. Sun. Clustering to find exemplar terms for keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1*, EMNLP '09, pages 257–266, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. ISBN 978-1-932432-59-6. URL `http://dl.acm.org/citation.cfm?id=1699510.1699544`.

[29] Z. Liu, W. Huang, Y. Zheng, and M. Sun. Automatic keyphrase extraction via topic decomposition. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 366–376, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1870658.1870694`.

[30] P. Lopez and L. Romary. Humb: Automatic key term extraction from scientific articles in grobid. In *SemEval@ACL*, 2010.

[31] P. Lopez and L. Romary. Grisp: A massive multilingual terminological database for scientific and technical domains. In *LREC 2010*, 2010.

[32] D. Mahata, J. Kuriakose, R. R. Shah, and R. Zimmermann. Key2vec: Automatic ranked keyphrase extraction from scientific articles using phrase embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, Volume 2 (Short Papers), volume 2, pages 634–639, 2018.

[33] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[34] J. Martinez-Romo, L. Araujo, and A. D. Fernandez. Semgraph: Extracting keyphrases following a novel semantic graph-based approach. *JASIST*, 67:71–82, 2016.

[35] Y. Matsuo, Y. Ohsawa, and M. Ishizuka. Keyworld: extracting keywords from a document as a small world. In *Discovery Science*, volume 2226, pages 271–281. Springer, 2001.

[36] O. Medelyan, E. Frank, and I. H. Witten. Human-competitive tagging using automatic keyphrase extraction. In *EMNLP*, 2009.

[37] R. Meng, S. Zhao, S. Han, D. He, P. Brusilovsky, and Y. Chi. Deep keyphrase generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 582–592, 2017.

[38] R. Mihalcea. Graph-based ranking algorithms for sentence extraction, applied to text summarization. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 20. Association for Computational Linguistics, 2004.

[39] R. Mihalcea and P. Tarau. Textrank: Bringing order into texts. In *Conference on Empirical Methods in Natural Language Processing*, EMNLP '04, 2004.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

[41] A. Mukhopadhyay, A. C. S. Kumar, and S. M. Bhandarkar. Joint geometric graph embedding for partial shape matching in images. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, March 2016. doi: 10.1109/WACV.2016.7477728.

[42] M. Pagliardini, P. Gupta, and M. Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 528–540, 2018.

[43] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[44] S. Teufel and M. Moens. Sentence extraction and rhetorical classification for flexible abstracts. In *AAAI Spring Symposium on Intelligent Text summarization*, pages 89–97, 1998.

[45] A. J.-P. Tixier, F. D. Malliaros, and M. Vazirgiannis. A graph degeneracy-based approach to keyword extraction. In *EMNLP*, 2016.

[46] E. F. Tjong Kim Sang and S. Buchholz. Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, pages 127–132, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. doi: 10.3115/1117601.1117631. URL `http://dx.doi.org/10.3115/1117601.1117631`.

[47] G. Tsatsaronis, I. Varlamis, and K. Nørvåg. Semanticrank: Ranking keywords and sentences using semantic graphs. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1074–1082, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1873781.1873902`.

[48] P. D. Turney. Learning to extract keyphrases from text. *CoRR*, cs.LG/0212013, 2002. URL `http://arxiv.org/abs/cs.LG/0212013`.

[49] X. Wan and J. Xiao. Single document keyphrase extraction using neighborhood knowledge. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, AAAI'08, pages 855–860. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL `http://dl.acm.org/citation.cfm?id=1620163.1620205`.

[50] Y. Wen, H. Yuan, and P. Zhang. Research on keyword extraction based on word2vec weighted textrank. In *Computer and Communications (ICCC), 2016 2nd IEEE International Conference on*, pages 2109–2113. IEEE, 2016.

[51] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. Kea: Practical automatic keyphrase extraction. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, DL '99, pages 254–255, New York, NY, USA, 1999. ACM. ISBN 1-58113-145-3. doi: 10.1145/313238.313437. URL `http://doi.acm.org/10.1145/313238.313437`.

[52] Z. Xie. Centrality measures in text mining: prediction of noun phrases that appear in abstracts. In *Proceedings of the ACL student research workshop*, pages 103–108. Association for Computational Linguistics, 2005.

[53] Z. Zhou, X. Zou, X. Lv, and J. Hu. Research on weighted complex network based keywords extraction. In *Workshop on Chinese Lexical Semantics*, pages 442–452. Springer, 2013.

APPENDIX A

EXAMPLE DOCUMENTS FROM EACH DATASET

A.1 INSPEC

```
Sensorless control of induction motor drives.
```

```
Controlled induction motor drives without mechanical speed sensors at the motor
       shaft have the attractions of low cost and high reliability. To replace
       the sensor the information on the rotor speed is extracted from
       measured stator voltages and currents at the motor terminals.
       Vector-controlled drives require estimating the magnitude and spatial
       orientation of the fundamental magnetic flux waves in the stator or in
       the rotor. Open-loop estimators or closed-loop observers are used for
       this purpose. They differ with respect to accuracy, robustness, and
       sensitivity against model parameter variations. Dynamic performance and
       steady-state speed accuracy in the low-speed range can be achieved by
       exploiting parasitic effects of the machine. The overview in this paper
       uses signal flow graphs of complex space vector quantities to provide
       an insightful description of the systems used in sensorless control of
       induction motors.
```

The gold-standard keyphrases are:

sensorless control	reliability
induction motor drives	stator voltages

stator currents	robustness
vector-controlled drives	sensitivity
magnitude	model parameter variations
spatial orientation	steady-state speed accuracy
fundamental magnetic flux waves	parasitic effects
open-loop estimators	signal flow graphs
closed-loop observers	space vector quantities

A.2 SEMEVAL

The following is one page of a document in the training partition of the SemEval corpus.

```
    A Framework for Agent-Based Distributed Machine
Learning and Data Mining
Jan Tozicka
Gerstner Laboratory
Czech Technical University
TechnickÂńa 2, Prague, 166 27
Czech Republic
tozicka@labe.felk.cvut.cz
Michael Rovatsos
School of Informatics
The University of Edinburgh
Edinburgh EH8 9LE
United Kingdom
mrovatso@inf.ed.ac.uk
Michal Pechoucek
Gerstner Laboratory
Czech Technical University
TechnickÂńa 2, Prague, 166 27
```

Czech Republic

pechouc@labe.felk.cvut.cz

ABSTRACT

This paper proposes a framework for agent-based distributed machine learning and data mining based on (i) the exchange of meta-level descriptions of individual learning processes among agents and (ii) online reasoning about learning success and learning progress by learning agents. We present an abstract architecture that enables agents to exchange models of their local learning processes and introduces a number of different methods for integrating these processes. This allows us to apply existing agent interaction mechanisms to distributed machine learning tasks, thus leveraging the powerful coordination methods available in agent-based computing, and enables agents to engage in meta-reasoning about their own learning decisions. We apply this architecture to a real-world distributed clustering application to illustrate how the conceptual framework can be used in practical systems in which different learners may be using different datasets, hypotheses and learning algorithms. We report on experimental results obtained using this system, review related work on the subject, and discuss potential future extensions to the framework.

General Terms

Theory

Categories and Subject Descriptors

I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence-Multiagent Systems

1. INTRODUCTION

In the areas of machine learning and data mining (cf. [14, 17] for overviews), it has long been recognised that parallelisation and distribution can be used to improve learning performance. Various techniques have been suggested in this respect, ranging from the low-level integration of independently derived learning hypotheses (e.g. combining different classifiers to make optimal classification decisions [4, 7], model averaging of Bayesian classifiers [8], or consensusbased methods for integrating different clusterings [11]), to the high-level combination of learning results obtained by heterogeneous learning agents using meta-learning (e.g. [3, 10, 21]).

All of these approaches assume homogeneity of agent design (all agents apply the same learning algorithm) and/or agent objectives (all agents are trying to cooperatively solve a single, global learning problem). Therefore, the techniques they suggest are not applicable in societies of autonomous learners interacting in open systems. In such systems, learners (agents) may not be able to integrate their datasets or learning results (because of different data formats and representations, learning algorithms, or legal restrictions that prohibit such integration [11]) and cannot always be guaranteed to interact in a strictly cooperative fashion (discovered knowledge and collected data might be economic assets that should only be shared when this is deemed profitable; malicious agents might attempt to adversely influence others" learning results, etc.).

Examples for applications of this kind abound. Many distributed learning domains involve the use of sensitive data

and prohibit the exchange of this data (e.g. exchange of patient data in distributed brain tumour diagnosis [2]) - however, they may permit the exchange of local learning hypotheses among different learners. In other areas, training data might be commercially valuable, so that agents would only make it available to others if those agents could provide something in return (e.g. in remote ship surveillance and tracking, where the different agencies involved are commercial service providers [1]). Furthermore, agents might have a vested interest in negatively affecting other agents" learning performance. An example for this is that of fraudulent agents on eBay which may try to prevent reputationlearning agents from the construction of useful models for detecting fraud.

Viewing learners as autonomous, self-directed agents is the only appropriate view one can take in modelling these distributed learning environments: the agent metaphor becomes a necessity as oppossed to preferences for scalability, dynamic data selection, interactivity [13], which can also be achieved through (non-agent) distribution and parallelisation in principle.

Despite the autonomy and self-directedness of learning agents, many of these systems exhibit a sufficient overlap in terms of individual learning goals so that beneficial cooperation might be possible if a model for flexible interaction between autonomous learners was available that allowed agents to

1. exchange information about different aspects of their own learning mechanism at different levels of detail

without being forced to reveal private information that
should not be disclosed,
2. decide to what extent they want to share information
about their own learning processes and utilise
information provided by other learners, and
3. reason about how this information can best be used to
improve their own learning performance.
Our model is based on the simple idea that autonomous
learners should maintain meta-descriptions of their own
learning processes (see also [3]) in order to be able to
exchange information and reason about them in a rational way
(i.e. with the overall objective of improving their own
learning results). Our hypothesis is a very simple one:
If we can devise a sufficiently general, abstract
view of describing learning processes, we will be
able to utilise the whole range of methods for (i)
rational reasoning and (ii) communication and
coordination offered by agent technology so as to
build effective autonomous learning agents.
To test this hypothesis, we introduce such an abstract
architecture (section 2) and implement a simple, concrete
instance of it in a real-world domain (section 3). We report
on empirical results obtained with this implemented system
that demonstrate the viability of our approach (section 4).
Finally, we review related work (section 5) and conclude
with a summary, discussion of our approach and outlook to
future work on the subject (section 6).
2. ABSTRACT ARCHITECTURE
Our framework is based on providing formal (meta-level)

descriptions of learning processes, i.e. representations of all
relevant components of the learning machinery used by a
learning agent, together with information about the state of
the learning process.
To ensure that this framework is sufficiently general, we
consider the following general description of a learning
problem:
Given data D âŁĘ D taken from an instance space
D, a hypothesis space H and an (unknown)
target function c âĹĹ H1
, derive a function h âĹĹ H that
approximates c as well as possible according to
some performance measure g : H âĘŠ Q where Q
is a set of possible levels of learning performance.

These are the gold-standard keyphrases:

agent
machine learning
datum mining
individual learning process
meta-reasoning
distributed clustering application
frameworks and architecture
unsupervised clustering
bayesian classifier
consensusbased method
communication and coordination
autonomous learning agent
historical information

APPENDIX B

LIST OF ENGLISH STOPWORDS ACCORDING TO NLTK

a	being	for	if	no	same
about	below	from	in	nor	shan
above	between	further	into	not	she
after	both	had	is	now	should
again	but	hadn	isn	o	shouldn
against	by	has	it	of	so
ain	can	hasn	its	off	some
all	couldn	have	itself	on	such
am	d	haven	just	once	t
an	did	having	ll	only	than
and	didn	he	m	or	that
any	do	her	ma	other	the
are	does	here	me	our	their
aren	doesn	hers	mightn	ours	theirs
as	doing	herself	more	ourselves	them
at	don	him	most	out	themselves
be	down	himself	mustn	over	then
because	during	his	my	own	there
been	each	how	myself	re	these
before	few	i	needn	s	they

this	until	we	which	with	yours
those	up	were	while	won	yourself
through	ve	weren	who	wouldn	yourselves
to	very	what	whom	y	
too	was	when	why	you	
under	wasn	where	will	your	

APPENDIX C

LIST OF SUBJECTS IN ARXIV COMPUTING RESEARCH REPOSITORY

Artificial Intelligence (cs.AI)

Computation and Language (cs.CL)

Computational Complexity (cs.CC)

Computational Engineering, Finance, and Science (cs.CE)

Computational Geometry (cs.CG)

Computer Science and Game Theory (cs.GT)

Computer Vision and Pattern Recognition (cs.CV)

Computers and Society (cs.CY)

Cryptography and Security (cs.CR)

Data Structures and Algorithms (cs.DS)

Databases (cs.DB)

Digital Libraries (cs.DL)

Discrete Mathematics (cs.DM)

Distributed, Parallel, and Cluster Computing (cs.DC)

Emerging Technologies (cs.ET)

Formal Languages and Automata Theory (cs.FL)

General Literature (cs.GL)

Graphics (cs.GR)

Hardware Architecture (cs.AR)

Human-Computer Interaction (cs.HC)

Information Retrieval (cs.IR)

Information Theory (cs.IT)

Machine Learning (cs.LG)

Logic in Computer Science (cs.LO)

Mathematical Software (cs.MS)

Multiagent Systems (cs.MA)

Multimedia (cs.MM)

Networking and Internet Architecture (cs.NI)

Neural and Evolutionary Computing (cs.NE)

Numerical Analysis (cs.NA)

Operating Systems (cs.OS)

Other (cs.OH)

Performance (cs.PF)

Programming Languages (cs.PL)

Robotics (cs.RO)

Social and Information Networks (cs.SI)

Software Engineering (cs.SE)

Sound (cs.SD)

Symbolic Computation (cs.SC)

Systems and Control (cs.SY)