# Leveraging the Pattern-Relation Duality for Domain-specific Keyphrase Mining

**Dipro Ray**
Department of Computer Science
University of Illinois at Urbana-Champaign
dipror2@illinois.edu

Advised by Professor **Kevin C.C. Chang**

## Abstract

Phrase mining over large text corpora has been a widely studied area. However, the closely related problem of domain-specific keyphrase mining has not been researched. Domain-specific keyphrase mining has wide applications in information extraction and retrieval, topic modelling and taxonomy construction - quite often, we are concerned with corpora phrases with respect to a particular domain (instead of generic domain-independent phrases). In this thesis, we leverage the pattern-relation duality in building a novel, low-complexity approach to domain-specific keyphrase mining. We adapt a previous framework for tuple extraction over the web (PRDualRank) to our problem, then extend it using a public knowledge base (Wikipedia) and a phrasal segmentation system. We further improve performance by incorporating the public knowledge base signals within PRDualRank's inference framework. We evaluate our frameworks on detecting "computer science" keyphrases, given a collection of "computer science" publications. Experiments reveal that our frameworks perform substantially better than previous state-of-the-art phrase detection systems in both phrase mining and domain-specific keyphrase mining tasks - particularly, we improve $F$-score by $34\%$ for phrase mining and by $39\%$ for domain-specific keyphrase mining.

## 1 Introduction

With the rapidly increasing generation of textual data on the web and in social media networks, various forms of text summarization have become highly important. One such key technique is keyphrase mining, that extracts relevant keyphrases from a given collection of documents. However, in summarizing textual data, we are often motivated by summarizing it with respect to some topic - a domain. For example, consider the task of summarizing the research interests of a computer science professor. Here, we are only concerned with mining keyphrases from the professor's publications that are relevant to computer science (and not generic domain-independent keyphrases). Such a task of finding the keyphrases occurring within a documents collection is the domain-specific keyphrase mining task.

We present solutions[1] to the task of domain-specific keyphrase mining: given a collection $C$ of documents about a particular domain $d$, detect high quality keyphrases that are relevant to domain $d$. For example, "machine learning" is a high quality keyphrase with respect to the "computer science" domain.

[1]Implementations of our frameworks can be found at https://github.com/harrywsh/phrase-detection

**What's a good quality keyphrase?** Han et al. [2; 4] stated that a phrase needs to meet four criteria to be considered good: Popularity, Concordance, Informativeness, Completeness. We adopt similar criteria in identifying a good quality keyphrase, with modifications to Informativeness to be domain-specific:

1. *Popularity*: Good keyphrases occur substantially in our corpus (collection of documents).
2. *Concordance*: The collocation of tokens in good keyphrases is substantially higher than that expected due to chance.
3. *Domainness*: Good keyphrases convey information about a specific topic or concept that are of substantial relevance to the domain.
4. *Completeness*: Good quality keyphrases must be interpretable as complete semantic units in some given context.

Note that the threshold for popularity for a good quality keyphrase can be quite low, provided other criteria are significantly satisfied. In an experiment on a corpus of 915 computer science publications' titles and abstracts, our framework exhibited about 90% Recall with gold high quality keyphrases that occurred $\geq 2$ times in our corpus.

The above domain-specific keyphrase mining task is difficult as we must figure out the correct subset of occurring keyphrases that are relevant to the domain. In order to do so, we must have signals and a ranking framework that not only tells us whether a keyphrase is of good quality, but also if it is related to the domain (and how strongly related it is).

We present three novel frameworks to the domain-specific keyphrase mining task - PRDR-Detect, PRDR-Detect-PostRank and MultiSource-PRDR-Detect - which leverage the pattern-relation duality, a public knowledge base (Wikipedia) and a phrasal segmentation system in ranking candidate keyphrases. Below, we describe the expected input to the framework. Further discussion behind the design of the three frameworks can be found in Section 3.2. Design.

**What's supplied to the framework?**

- Corpus: We expect a collection $C$ of documents, on material specific to the concerned domain, to be provided.
- Seed Keyphrases $S$: A set of few (5 - 10) good quality keyphrases, of the concerned domain, need to be provided to initialize the framework.

To clarify the above, consider the example of task of obtaining high-quality computer science keyphrases. Example input to our framework would be a collection of computer science publications (which are documents containing material relevant to computer science) and {"machine learning", "neural networks", "database system", "finite state machine", "neural dependency parser"} (which is a set of good quality CS keyphrases).

We evaluate our three models against the current state-of-the-art phrase detection system AutoPhrase on a collection of computer science publications. Our models exhibit improved performance over AutoPhrase in both the phrase detection and domain-specific keyphrase detection tasks.

This thesis makes the following contributions:

- We adapt the PRDualRank tuple-extraction framework to over domain-specific keyphrase mining problem.
- We introduce novel methods to model domainness and completeness signals using a public knowledge base (Wikipedia) and a POS-guided phrasal segmentation system respectively.
- We modify PRDualRank's dual inference framework for precision and recall to incorporate domainness signals. Particularly, we formally introduce domainness precision and domainness recall for inference, interpreted as random walks over the Wikipedia article graph.

- We present four evaluation metrics for our models, and develop algorithms to compute each of those metrics. Our models perform substantially better than AutoPhrase, with improvement of $F$-score by 34% for phrase mining and by 39% for domain-specific keyphrase mining.

The rest of the thesis is organized as follows. Section 2 positions our solutions relative to currently existing solutions. Section 3 describes the motivation, design and implementation of all three frameworks in detail. Experiments, evaluation metrics and algorithms, and their results are covered in Section 4. Section 5 summarizes our work and presents future directions. The appendix presents miscellaneous relevant information. conclude the study in Section 7.

## 2 Related Work

**In terms of problem** Identifying high quality phrases within a corpus has been extensively studied. The natural language processing (NLP) community has conducted research on automatic term recognition, for extracting technical phrases [5; 6; 7]. The information retrieval (IR) community has also studied the same problem, as better documents indexing terms can improve search engine performance. Such indexing algorithms [8; 9; 10] use supervised-learning methods to figure out POS tags-based rules that can identify candidate phrases. Other works build up on this idea by incorporating more advanced NLP concepts and textual statistical features [7; 11; 12].

However, due the dependence of such methods on linguistic analyzers and domain-specific rules, the IR community has presented new methods that require massive text corpora to estimate phrase quality, using statistical features [2; 4; 13; 14; 15]. AutoPhrase [2] presents a positive-only distant training method for phrase quality estimation, and uses POS-guided phrasal segmentation to adjust statistical features for accurate quality estimation. The domain-specific keyphrase extraction task has been researched for specific domains or on specific corpora [16; 17; 18].

**In terms of techniques** The PRDualRank was presented in [1], and it built upon work in [3]. The framework was built for application to tuple extraction. To our knowledge, there do no exist domain-specific keyphrase detection models that use the pattern-relation duality in implementation.

Usage of Wikipedia, particularly the Wikipedia Category Graph, has been prevalent in taxonomy derivation, query classification and word semantic relatedness [19; 20].

## 3 Solutions

### 3.1 Motivation

Our approach to the domain-based keyphrase detection problem is primarily inspired by the PRDualRank framework [1], which was originally developed for the task of tuple extraction on the Web using the pattern-relation duality. Though we address a problem with significant differences, the core idea of patterns and tuples reinforcing each other still applies.

**Patterns** Since good keyphrases need not occur very frequently in the corpus, probability-based phrase detection approaches do not reveal all the good keyphrases in a corpus.[2] However, the context in which good keyphrases occur sheds light on where other keyphrases will occur. The context around a good keyphrase can be interpreted as a pattern - a pattern of POS tags, prepositions and words. For example, in the sentence “Applications of machine learning in bioinformatics are numerous.”, the pattern that represents the context around “machine learning” (a good keyphrase) is [“Applications”, “of”, NN, NN, “in”]. Using patterns that capture the contexts of known good keyphrases in our corpus, we can retrieve new good keyphrases from our corpus, at other occurrences of the same patterns. Continuing the previous example, if our corpus had another sentence “Applications of neural networks in dependency parsing . . . ”, we would identify “neural networks” as a potential good keyphrase, based on its occurrence within the same pattern (context). In this manner, we can use patterns to identify candidate good keyphrases.

[2]Hereafter, we shall use *keyphrase* to denote a domain-specific keyphrase, and *phrase* to denote an ordinary (domain-independent) phrase.

**Tuples** Since it is the keyphrases that occur with the patterns, the keyphrases will be the (unary) tuples in our scenario. Knowing some good keyphrases in our corpus allows us to extract the patterns (contexts) with which they occur. In our running example, if we knew "machine learning" was a good keyphrase, we would extract its context ["Applications", "of", NN, NN, "in"] (and then, use this context to find more potential keyphrases). So, just as patterns identify potential tuples (good keyphrases), tuples help identify potential patterns (good keyphrases' contexts).

**Pattern Tuple Reinforcement** As mentioned above, the patterns (good keyphrases' contexts) and the tuples (good keyphrases) reinforce each other, as good patterns help find good keyphrases and vice-versa. This allows us to start with a corpus and a few known good keyphrases, execute this reinforcement process iteratively, and end up with many good keyphrases detected from the corpus. The PRDualRank framework helps execute this process, with precision and recall based ranking of patterns and tuples at each step to prevent snowballing. We discuss the implementation further in 3.2 Design and 3.3 Framework.

## 3.2 Design

Let us discuss the design of *our approach*. We wish to create a system that detects keyphrases with the above criteria (to ensure they are of high quality).

**Popularity and Concordance** The PRDualRank framework, through leveraging the pattern-relation duality (as described previously) and precision and recall based ranking, outputs candidate good keyphrases that meet the Popularity, Concordance and (to some extent) Completeness criteria.

**Domainness and Completeness** To be able to detect candidate good keyphrases that meet the Domainness and Completeness criteria substantially, we propose novel modifications to the PRDualRank framework to incorporate external signals.

1. Using Wikipedia: In modelling the domainness of a keyphrase, we leverage Wikipedia, a public knowledge base. A majority of articles on Wikipedia use phrases in their titles. We express the domainness of a keyword in terms of precision and recall, through random walks on the Wikipedia articles graph between the article of the keyword and the article of the domain. We use the Wikipedia category graph to approximate the Wikipedia articles graph in computing precision and recall for domainness. These precision and recall metrics are embedded within the modified PRDualRank framework.

2. Using Phrasal Segmentation: In modelling the completeness of a keyphrase, we use AutoPhrase's POS-guided phrasal segmentation system [2]. AutoPhrase's system provides a list of phrasal segments which are leveraged within our framework to model completeness. Note that this data-point is quite sparse, as phrasal segmentation systems are probability-based (so they only identify keyphrases that occur very frequently). Further, domain-specific keyphrases can include "modern" terms which might not be identified as phrasal segments correctly.

For ease of future discussion, we provide below a description of *three key framework variants* we developed, implemented and experimented with, that incorporate the signals discussed above with gradually increasing granularity.

1. PRDR-Detect: This model adapts the PRDualRank framework to the domain-specific keyphrase detection problem. It does not incorporate domainness or completeness signals.

2. PRDR-Detect-PostRank: This model is similar to PRDR-Detect, but, at each iteration, it incorporates a final ranking step that uses domainness and completeness signals.

3. MultiSource-PRDR-Detect: Instead of a final ranking step as in PRDR-Detect-PostRank, this model embeds domainness signals within the PRDualRank precision and recall computation framework. It incorporates completeness signals through a final ranking step (at each iteration).

| 1) **QuestP:** ***Quest Backward for Precision Inference*** | 2) **QuestR:** ***Quest Forward for Recall Inference*** |
|---|---|
| ***P1***: $\mathcal{P}(p) = \sum_{t_i \in \tau(p)} \mathcal{P}(t_i) \cdot \frac{\lvert I_{t_i p} \rvert}{\lvert I_p \rvert}$ | ***R1***: $\mathcal{R}(p) = \sum_{t_i \in \tau(p)} \frac{\lvert I_{t_i p} \rvert}{\lvert I_{t_i} \rvert} \mathcal{R}(t_i)$ |
| ***P2***: $\mathcal{P}(t) = \begin{cases} \mathcal{P}_0(t) & \text{if } t \in T_0; \\ \mathcal{P}(t) = \sum_{p_i \in \pi(t)} \mathcal{P}(p_i) \cdot \frac{\lvert I_{t p_i} \rvert}{\lvert I_t \rvert} & \text{otherwise.} \end{cases}$ | ***R2***: $\mathcal{R}(t) = \sum_{p_i \in \pi(t)} \frac{\lvert I_{t p_i} \rvert}{\lvert I_{p_i} \rvert} \mathcal{R}(p_i)$ |

Figure 1: The dual inference framework of PRDualRank: QuestP and QuestR

**Algorithm 1** PRDualRank
Inputs: $G = (T, C, P)$ (the context graph), $T_o \subset T$ (the seed tuples)
Outputs: $P$ and $T$ ranked by precision and recall

```
1: for each $t \in T_o$ do
2:     $\mathcal{P}(t) \leftarrow 1$
3:     $\mathcal{R}(t) \leftarrow \frac{1}{|T_o|}$
4: end for
5: QuestP: update $\mathcal{P}$ till convergence by rules P1, P2;
6: QuestR: update $\mathcal{R}$ till convergence by rules R1, R2;
7: return $P$, $T$ ranked with $\mathcal{P}$ and $\mathcal{R}$ scores;
```

### 3.3 Framework

#### 3.3.1 PRDR-Detect

PRDR-Detect adapts **PRD**ual**R**ank for keyphrase **detect**ion. It does so by using PRDualRank as a blackbox.

**Algorithm 2** PRDR-Detect
Inputs: $K_{seed}$ (the seed keyphrases), $C$ (the collection of documents), $N_p$ (the number of top patterns to remember), $N_k$ (the number of top keyphrases to remember), $N_{iter}$ (the number of PRDualRank iterations)
Outputs: $L$, a list of detected keyphrases ranked by $F$-score

```
 1: Initialize $L \leftarrow []$, $L_p \leftarrow []$
 2: for each $k \in K_{seed}$ do
 3:     Add $k$ to $L$
 4: end for
 5: for iter = $1, 2, \ldots, N_{iter}$ do
 6:     for each keyword $k \in L$ do
 7:         Add to $L_p$ the (new) patterns that $k$ occurs in
 8:     end for
 9:     Initialize $K_{iter} \leftarrow []$
10:     for each pattern $p \in L_p$ do
11:         Add to $K_{iter}$ the keyphrases that occur with $p$
12:     end for
13:     $X \leftarrow L_p$
14:     $Y \leftarrow \{$Keywords in $L\} \cup K_{iter}$
15:     Construct context graph $G$ using $X$ and $Y$
16:     $P, T \leftarrow$ PRDualRank$(G, K_{seed})$
17:     Update $L \leftarrow \{$Top $N_k$ Keyphrases based on $F$-score from $T\}$
18:     Update $L_p \leftarrow \{$Top $N_p$ Keyphrases based on $F$-score from $P\}$
19: end for
20: return $L$ after sorting based on $F$-score
```

The PRDualRank function is shown in Algorithm 1. It initializes precision and recall values for seed tuples before running the inference process. As explained in [1], dual inference for precision and recall computation for PRDualRank is based on QuestP and QuestR framework. The framework is presented in Figure 1 above.

The PRDR-Detect function (Algorithm 2) uses PRDualRank as a blacbox helper function. PRDR-Detect maintains a running list of good patterns and good keyphrases. PRDualRank() is called once in each iteration. Before it is called, the current set of good keywords is used to extract new candidate patterns (by analyzing the text using NLP tools). These new patterns, along with our running set of patterns, is used to extract new candidate keyphrases that occur with those patterns. All patterns and keyphrases so far are used to construct the context graph which is then passed into PRDualRank() with the seed keyphrases. PRDualRank() returns the patterns and keyphrases ranked by precision and recall. Their $F$-scores are then computed, and they're ranked, to only keep a user-set number of patterns and candidates (to prevent snowballing). This process is then repeated for a user-set number of iterations.

### 3.3.2 PRDR-Detect-PostRank

PRDR-Detect-PostRank builds up on **PRDR-Detect** - in each iteration, **post** execution of PRDual-Rank(), an extra **rank**ing step is performed for keyphrases . The purpose of keyphrase (re)ranking is to incorporate domainness and completeness signals within the ranking score assigned to each keyphrase. Algorithm 3 describes the PRDR-Detect-PostRank function. It differs from PRDR-Detect in lines 17 through 20. All other operations and the PRDualRank function remain the same as in PRDR-Detect.

**Algorithm 3** PRDR-Detect-PostRank
Inputs: $K_{seed}$ (the seed keyphrases), $C$ (the collection of documents), $N_p$ (the number of top patterns to remember), $N_k$ (the number of top keyphrases to remember), $N_{iter}$ (the number of PRDualRank iterations)
Outputs: $L$, a list of detected keyphrases ranked by $F$-score

```
 1: Initialize $L \leftarrow []$, $L_p \leftarrow []$
 2: for each $k \in K_{seed}$ do
 3:     Add $k$ to $L$
 4: end for
 5: for iter = $1, 2, \ldots, N_{iter}$ do
 6:     for each keyword $k \in L$ do
 7:         Add to $L_p$ the (new) patterns that $k$ occurs in
 8:     end for
 9:     Initialize $K_{iter} \leftarrow []$
10:     for each pattern $p \in L_p$ do
11:         Add to $K_{iter}$ the keyphrases that occur with $p$
12:     end for
13:     $X \leftarrow L_p$
14:     $Y \leftarrow$ {Keywords in $L$} $\cup K_{iter}$
15:     Construct context graph $G$ using $X$ and $Y$
16:     $P, T \leftarrow$ PRDualRank$(G, K_{seed})$
17:     for each $k \in Y$ do
18:         Compute and store PostRankScore($k$)
19:     end for
20:     Update $L \leftarrow \{$Top $N_k$ Keyphrases based on the computed PostRankScores$\}$
21:     Update $L_p \leftarrow \{$Top $N_p$ Keyphrases based on $F$-score from $P\}$
22: end for
23: return $L$ after sorting based on $F$-score
```

Keyphrase (re)ranking after running PRDualRank() (in each iteration) uses the following formula:

$$\text{For a keyword } k, \text{PostRankScore}(k) = e^{\mathcal{F}(k) \times \mathcal{W}(k) \times \mathcal{C}(k)}$$

where $\mathcal{F}, \mathcal{W}, \mathcal{C}$ are as follows:

- $\mathcal{F}(k)$: The $F$-score of a keyphrase $k$ in the latest execution of PRDualRank()

- $\mathcal{W}(k)$: The "wikiscore" function, used to model the domainness of a keyphrase, using Wikipedia.

$$\mathcal{W}(k) = \frac{\text{Number of top } \alpha \text{ retrieved articles, for } k\text{, that are domain-relevant}}{\alpha}$$

  Wikiscore is based on the rationale that a word that is very relevant to a particular domain will retrieve many Wikipedia articles that are known to be relevant to that domain. In the above function, we examine top $\alpha$ retrieved documents for each keyword $k$ and check how many of those articles are relevant to the domain.

  Alternatively stated, wikiscore depicts a random walk on the Wikipedia article graph - it captures the probability that, starting from an article about keyphrase $k$, we will reach the article about the domain itself. This rationale is discussed further in the next section on MultiSource-PRDR-Detect.

- $\mathcal{C}(k)$: The "segscore" function, used to model the completeness of a keyphrase, using Autophrase's phrasal segmentation system.

$$\mathcal{C}(k) = \begin{cases} c_o & k \text{ is not identified as a phrase} \\ \frac{\text{Number of phrase occurrences of } k}{\text{Max. number of occurrences of a particular phrase}} & k \text{ is identified as a phrase} \end{cases}$$

  The AutoPhrase phrasal segmentation returns a list of phrases it has identified. For any keyword $k$:

  1. If $k$ occurs as one of the identified phrases: We count the number of times it has been identified as a phrase (as the phrase detection is context-dependent). This is the normalized by dividing this number by the maximum number of occurrences of any particular phrase within the above list.
  2. If $k$ did not occur as one of the identified phrases: We set $\mathcal{C}(k)$ to a small constant $c_o$.

  As segscore captures the probability a keyphrase $k$ was identified as a phrase, it is a good indicator of the completeness criterion. When a keyphrase isn't identified as a phrase, we set its segscore to a small constant. This is useful because when segscore exists, it adds good signal for a keyphrase $k$ (which means it'll be ranked higher), and when it does not, it defaults to a small constant that doesn't affect the other three signals.

Experimentation with various combinations of $\mathcal{F}, \mathcal{W}, \mathcal{C}$ (sum, product, exponent, etc.) revealed that the exponential function worked best for our problem. This can be explained by the fact that since the exponential function is a faster growing function than a sum or product, candidate keyphrases that have large values for all three functions end up having a quite large PostRankScore, causing those keyphrases to be ranked very high up in the final result.

### 3.3.3 MultiSource-PRDR-Detect[3]

MultiSource-PRDR-Detect takes **PRDR-Detect**-PostRank's incorporation of domainness and completeness signals even further, by embedding domainness within the dual inference framework of PRDualRank, creating a **multi**ple **source** dual inference framework. It also applies completeness signal in an extra ranking step after PRDualRank() in each iteration.

**Multiple Source Dual Inference Framework** In the original PRDualRank framework, the dual inference framework was based on one source of truth - the set of relevant contexts. However, we also use another source of truth - Wikipedia. Incorporating both sources within the inference framework would allow for a more fine-tuned ranking of patterns and tuples. Incorporating Wikipedia then requires defining precision and recall measures for it.

Parallel to the intuition of precision and recall for the set of relevant contexts as random walks on the PRDualRank context graph, we perceive the Wikipedia precision and recall as random walks on the Wikipedia article graph.

For a keyphrase $k$,

[3]This framework is still under development, and is subject to change. All experiment metrics may not yet exist for it yet (as it's not finalized).

- Wikiprecision ($\mathcal{P}_{wiki}$): This can be interpreted as a random walk from an article about $k$ to the article about the domain. In practice, we use the weighted average of the probability of random walks for top $K$ articles' titles retrieved by $k$ on Wikipedia.
- Wikirecall ($\mathcal{R}_{wiki}$): This can be interpreted as a random walk to an article about $k$ from the article about the domain. In practice, we use the weighted average of the probability of random walks for top $K$ articles' titles retrieved by $k$ on Wikipedia.

For example, consider a keyphrase "machine learning". To find its wikiprecision and wikirecall with respect to the "computer science" domain, we look at random walks between the top $K$ articles on "machine learning" and the article on "computer science".

We use a weighted average of top $K$ retrieved articles for any keyphrase $k$ to get a generalized picture of its wikiprecision and wikirecall. In implementation, we also approximate the Wikipedia article graph using the Wikipedia category graph. Notes about its implementation can be found in the Appendix.

Inspired by the QueST framework [3], we combine both sources (Wikipedia and the set of relevant contexts) by applying weights that add up to 1. Therefore, the modified dual inference framework is as follows:

1. MultiSource-QuestP:

$$\textbf{P1: } \mathcal{P}(p) = \sum_{t_i \in \tau(p)} \mathcal{P}(t_i) \times \frac{|I_{t_i p}|}{|I_p|}$$

$$\textbf{P2: } \mathcal{P}(t) = \begin{cases} \beta \times \mathcal{P}_o(t) + (1-\beta) \times \mathcal{P}_{wiki}(t) & \text{if } t \in T_o; \\ \beta \times \sum_{p_i \in \pi(t)} \mathcal{P}(t_i) \frac{|I_{p_i t}|}{|I_t|} + (1-\beta) \times \mathcal{P}_{wiki}(t) & \text{otherwise} \end{cases}$$

2. MultiSource-QuestR:

$$\textbf{R1: } \mathcal{R}(p) = \sum_{t_i \in \tau(p)} \mathcal{R}(t_i) \times \frac{|I_{t_i p}|}{|I_{t_i}|}$$

$$\textbf{R2: } \mathcal{R}(t) = \beta \times \sum_{p_i \in \pi(t)} \mathcal{R}(t_i) \frac{|I_{p_i t}|}{|I_{p_i}|} + (1-\beta) \times \mathcal{R}_{wiki}(t)$$

**Post Ranking** In each iteration, after PRDualRank() with the multiple source dual inference framework is computed, a final (re)ranking step is applied. Along the lines of the ranking formula for PRDR-Detect-PostRank, the ranking formula here is given by:

$$\text{For a keyword } k, \text{MultiSource-PostRankScore}(k) = e^{\mathcal{F}'(k) \times \mathcal{C}(k)}$$

where $\mathcal{F}', \mathcal{C}$ are as follows:

- $\mathcal{F}'(k)$: The $F$-score of a keyphrase $k$ in the latest execution of multiple source PRDualRank()
- $\mathcal{C}(k)$: The same "segscore" function, as in PRDR-Detect-PostRank, used to model the completeness of a keyphrase using Autophrase's phrasal segmentation system.

**Algorithm 4** MultiSource-PRDualRank
Inputs: $G = (T, C, P)$ (the context graph), $T_o \subset T$ (the seed tuples)
Outputs: $P$ and $T$ ranked by precision and recall

```
1: for each $t \in T_o$ do
2:     $\mathcal{P}(t) \leftarrow 1$
3:     $\mathcal{R}(t) \leftarrow \frac{1}{|T_o|}$
4: end for
5: MultiSource-QuestP: update $\mathcal{P}$ till convergence by rules P1, P2;
6: MultiSource-QuestR: update $\mathcal{R}$ till convergence by rules R1, R2;
7: return $P$, $T$ ranked with $\mathcal{P}$ and $\mathcal{R}$ scores;
```

Algorithm 4, called the MultiSource-PRDualRank, describes the PRDualRank function updated to use the multiple source dual inference framework. the Algorithm 5 describes the MultiSource-PRDR-Detect function, which uses the MultiSource-PRDualRank() function as a helper function.

**Algorithm 5** MultiSource-PRDR-Detect

Inputs: $K_{seed}$ (the seed keyphrases), $C$ (the collection of documents), $N_p$ (the number of top patterns to remember), $N_k$ (the number of top keyphrases to remember), $N_{iter}$ (the number of PRDualRank iterations)

Outputs: $L$, a list of detected keyphrases ranked by $F$-score

```
1: Initialize $L \leftarrow []$, $L_p \leftarrow []$
2: for each $k \in K_{seed}$ do
3:     Add $k$ to $L$
4: end for
5: for iter = $1, 2, \ldots, N_{iter}$ do
6:     for each keyword $k \in L$ do
7:         Add to $L_p$ the (new) patterns that $k$ occurs in
8:     end for
9:     Initialize $K_{iter} \leftarrow []$
10:    for each pattern $p \in L_p$ do
11:        Add to $K_{iter}$ the keyphrases that occur with $p$
12:    end for
13:    $X \leftarrow L_p$
14:    $Y \leftarrow$ {Keywords in $L$} $\cup K_{iter}$
15:    Construct context graph $G$ using $X$ and $Y$
16:    $P, T \leftarrow$ MultiSource-PRDualRank$(G, K_{seed})$
17:    for each $k \in Y$ do
18:        Compute and store MultiSource-PostRankScore$(k)$
19:    end for
20:    Update $L \leftarrow \{$Top $N_k$ Keyphrases based on the computed MultiSource-PostRankScores$\}$
21:    Update $L_p \leftarrow \{$Top $N_p$ Keyphrases based on $F$-score from $P\}$
22: end for
23: return $L$ after sorting based on $F$-score
```

### 3.3.4 Solutions Summary

We adapt the PRDualRank framework to our task.

- Algorithm 1 presents the PRDualRank function, which is given a context graph and seed set as input. It iteratively executes the rules of the dual inference framework until convergence of precision and recall values of patterns and tuples.
- Algorithm 4 presents the MultiSource-PRDualRank function, which is a variant of the PRDualRank function. Particularly, it differs in lines 5-6, when it utilizes MultiSource-QuestP and MultiSource-QuestR (the dual inference framework that also incorporates Wikipedia) rules instead of the original QuestP and QuestR rules.

We present three models:

- PRDR-Detect (Algorithm 2): In each iteration, it performs a search for new candidate patterns with the existing keywords (parallel to PatternSearch in the original PRDualRank paper), and uses these new candidate patterns to find new candidate keyphrases (parallel to TupleSearch in the original PRDualRank paper). Then, it constructs the context graph with all patterns and keyphrases so far, and executes PRDualRank. It then keeps a user-set number of top patterns (ranked by $F$-score) and top keyphrases (ranked by $F$-score).
- PRDR-Detect-PostRank (Algorithm 3): In each iteration, it performs a search for new candidate patterns with the existing keywords (parallel to PatternSearch in the original PRDualRank paper), and uses these new candidate patterns to find new candidate keyphrases (parallel to TupleSearch in the original PRDualRank paper). Then, it constructs the context graph with all patterns and keyphrases so far, and executes PRDualRank. Afterwards, a reranking is done using PostRankScore$(k) = e^{\mathcal{F}(k) \times \mathcal{W}(k) \times \mathcal{C}(k)}$ (to incorporate domainness and completeness signals). It then keeps a user-set number of top patterns (ranked by $F$-score) and top keyphrases (ranked by PostRankScore).

  - For use in PRDR-Detect-PostRank, we present definitions for $\mathcal{W}(k)$ (the "wikiscore" function - the fraction of categories of articles about $k$ that are domain-relevant) and $\mathcal{C}(k)$ (the "segscore" function - the probability that $k$ is a complete phrasal segment).

- MultiSource-PRDR-Detect (Algorithm 5): In each iteration, it performs a search for new candidate patterns with the existing keywords (parallel to PatternSearch in the original PRDualRank paper), and uses these new candidate patterns to find new candidate keyphrases (parallel to TupleSearch in the original PRDualRank paper). Then, it constructs the context graph with all patterns and keyphrases so far, and executes MultiSource-PRDualRank. Afterwards, a reranking is done using MultiSourcePostRankScore$(k) = e^{\mathcal{F}'(k) \times \mathcal{C}(k)}$ (to incorporate completeness signal). It then keeps a user-set number of top patterns (ranked by $F$-score) and top keyphrases (ranked by MultiSource-PostRankScore).
  - For use in MultiSource-PRDR-Detect, we present definitions for Wikiprecision and Wikirecall (as random walks on the Wikipedia article graph).

## 4 Experiments

### 4.1 Methodology

**Which models are we experimenting with?** We compare our three models (1) PRDR-Detect, (2) PRDR-Detect-PostRank and (3) MultiSource-PRDR-Detect against the current state-of-the-art phrase detection system (4) AutoPhrase. Since our models utilize Wikipedia, to allow for fair comparison, we also compare our results against (5) AutoPhrase-WikiSort, which are the AutoPhrase results reranked using RankScore$(k)$ = $0.5\times$ AutoPhrase-Score$(k)$ + $0.5\times$ Wikiscore$(k)$.

**How are we experimenting?** Our experimental task is detecting keyphrases for the "computer science" domain, from a collection of "computer science" documents. We utilize a set of 915 documents, each including the title and abstract of a computer science publication, as our computer science documents collection.[4] For a fair comparison, we evaluate all five models performance with respect to: (1) phrase precision (2) phrase recall (3) keyword precision (4) keyword recall.

(1) Phrase precision: Phrase precision measures the fraction of result keyphrases that are good phrases. Since it's impossible to know all good phrases within a given collection of documents, we created a three-step FindGoodPhrases algorithm. FindGoodPhrases (presented in Algorithm 6)

**Algorithm 6** FindGoodPhrases
Inputs: $L$ (a list of keyphrases), $\tau$ (fuzzy string similarity threshold)
Output: $L_p$ (a subset of $L$ which has keyphrases that are good domain-independent phrases)

```
 1: Initialize $L_p \leftarrow []$
 2: for each $k \in L$ do
 3:     Check whether a Wikipedia page exists for $k$
 4:     If Yes, move $k$ from $L$ to $L_p$
 5:     If No, try singular/plural forms of $k$
 6:     If that works, move $k$ from $L$ to $L_p$
 7: end for
 8: for each $k \in L$ do
 9:     for each $q_k \in$ Top 10 Suggested Wikipedia Queries for $k$ do
10:         If FuzzyStringSimilarity$(k, q_k) > \tau$, move $k$ from $L$ to $L_p$, and Break
11:     end for
12: end for
13: return $L_p$
14: for each $k \in L$ do
15:     Manually review, and move $k$ from $L$ to $L_p$ if $k$ is a good phrase
16: end for
17: return $L_p$
```

[4]The utilized documents collection is a subset of those available at `https://www.kaggle.com/neelshah18/arxivdataset`

takes as input a list of keyphrases and returns a list of input keyphrases that are good phrases.

- For each keyword, it tries to check whether there exists a Wikipedia page with the same name. This is based on the intuition that Wikipedia article titles are good phrases. Singular/plural forms of the keyword are also checked in the same manner. (E.g.: If "neural networks" doesn't give me a Wikipedia page, the script tries "neural network" as well.)
- For the remaining keywords, it pulls the top 10 suggested queries for each keyword by Wikipedia. Then, it checks fuzzy string similarity for each suggested query with the associated keyword. If the similarity is higher than a user-set threshold, it is classified as a good phrase. (This helps tackle a keyphrase like "Branch and Price Algorithm" which does not have a Wikipedia article of the same name but instead has an article with title "Branch and Price".)
- A manual review step. This helps ensure we do have any false negatives.

Using the FindGoodPhrases algorithm, precision is computed as follows:

For a model $m$, $\text{Precision@Top}K = \dfrac{|\textsf{FindGoodPhrases}(\text{Top } K \text{ keyphrases returned by model } m)|}{K}$

(2) Phrase recall: Similar to the problem in Phrase precision, we do not have a ground truth set of phrases that appear in our document. Therefore, we use the goodphrases from the union of the results of all frameworks @Top$K$ as the ground truth set in recall computation. The formula for phrase recall is then as follows:

Given models $m_1, m_2, \ldots, m_n$, for a model $m$,

$$\text{Recall@Top}K = \frac{|\textsf{FindGoodPhrases}(\text{Top } K \text{ keyphrases returned by model } m)|}{|\textsf{FindGoodPhrases}(\bigcup_{i=1}^{n} \text{Top } K \text{ keyphrases returned by model } m_i)|}$$

(3) Keyphrase Precision: Since it is impossible to know for sure whether a keyphrase is a good keyphrase, we leverage our "wikiscore" is here. For any keyphrase $k$, if its wikiscore $> \tau_{wiki}$ (a wikiscore threshold), $k$ is considered to be domain-relevant and is classified as a good keyphrase. This is described in Algorithm 7, as the FindGoodKeyPhrases function. Then, the keyphrase precision is defined by:

For a model $m$,

$$\text{KeyphrasePrecision@Top}K = \frac{|\textsf{FindGoodKeyPhrases}(\text{Top } K \text{ keyphrases returned by model } m)|}{K}$$

(4) Keyword Recall: For our ground truth dataset, we use a subset of "computer science" keywords

**Algorithm 7** FindGoodKeyPhrases
Inputs: $L$ (a list of keyphrases), $\tau_{wiki}$ (wikiscore threshold)
Output: $L_p$ (a subset of $L$ which has keyphrases that are good domain-relevant keyphrases)

```
1: Initialize $L_p \leftarrow []$
2: for each $k \in L$ do
3:     Compute the wikiscore of $k$, $\mathcal{W}(k)$
4:     If $\mathcal{W}(k) > \tau_{wiki}$, move $k$ from $L$ to $L_p$
5: end for
6: return $L_p$
```

collected from Wikipedia that occur $\geq 2$ times within our collection of documents. Then, for each result keyphrase $k$, we use fuzzy string similarity against our ground truth dataset keyphrases. If $\exists k' \in$ GroundTruthDataset such that FuzzyStringSimilarity$(k, k') > \tau_{fuzzy}$, add $k$ as a keyphrase that matches a keyphrase in the ground truth. This is depicted in Algorithm 8 as the FindMatchedKeyPhrases function. Therefore, the keyphrase recall is given by: For a model $m$, and ground truth set $G$,

$$\text{KeyphraseRecall@Top}K = \frac{|\textsf{FindMatchedKeyPhrases}(\text{Top } K \text{ keyphrases returned by model } m, G)|}{|G|}$$

**Algorithm 8** FindMatchedKeyPhrases

Inputs: $L$ (a list of keyphrases), $\tau_{fuzzy}$ (fuzzy string similarity threshold), $G$ (ground truth set of keyphrases)

Output: $L_p$ (a subset of $L$ which has keyphrases that exist in the ground truth set)

```
1: Initialize $L_p \leftarrow []$
2: for each $k \in L$ do
3:     for each $k' \in G$ do
4:         Compute the fuzzy string similarity of $k$ and $k'$, $\mathcal{S}(k, k')$
5:         If $\mathcal{S}(k, k') > \tau_{fuzzy}$, move $k$ from $L$ to $L_p$
6:     end for
7: end for
8: return $L_p$
```

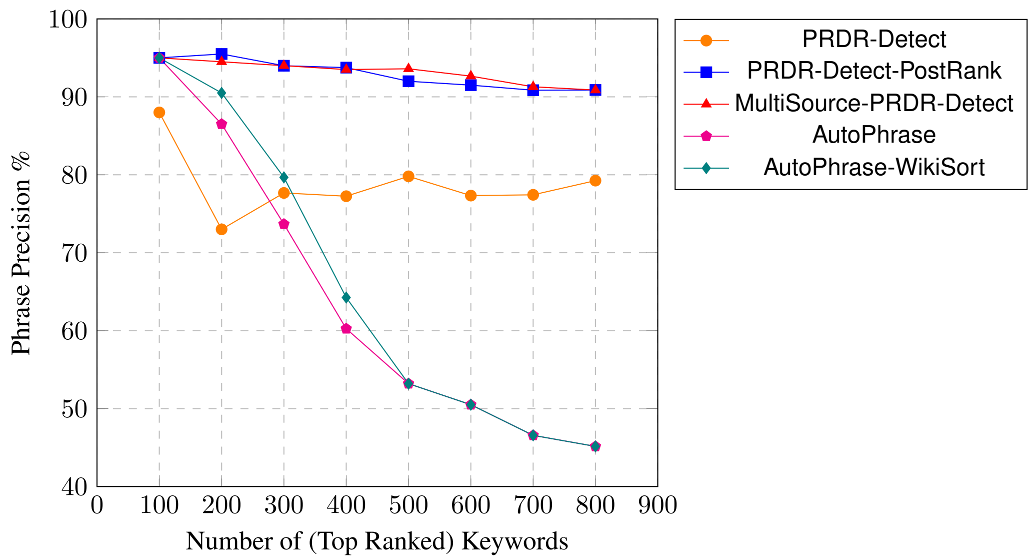
### 4.2 Results

Below, we present the results obtained for the models with respect to each of the four evaluation measures.

#### 4.2.1 Phrase Precision

Results shown below in Plot 1 and Table 1. The best performer is MultiSource-PRDR-Detect. A close second is PRDR-Detect-PostRank. PRDR-Detect performs worse relatively, but all three of our models perform much better than AutoPhrase and AutoPhrase-WikiSort.

Plot 1: Phrase Precision Performance of Different Frameworks



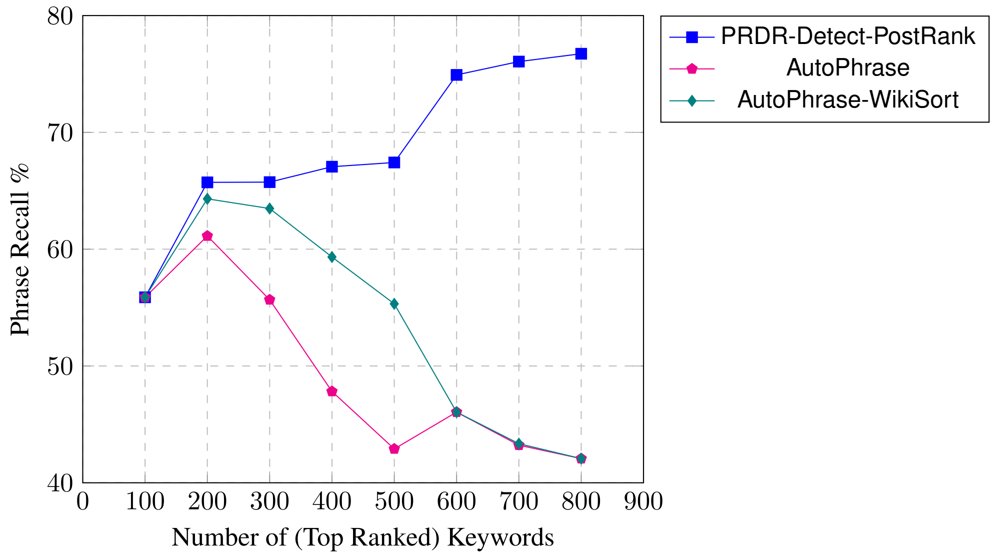
| | Top 100 | Top 400 | Top 800 |
|---|---|---|---|
| PRDR-Detect | 88% | 77.25% | 79.25% |
| PRDR-Detect-PostRank | 95% | 93.75% | 90.88% |
| MultiSource-PRDR-Detect | 95% | 93.5% | 90.87% |
| AutoPhrase | 95% | 60.25% | 45.15% |
| AutoPhrase-WikiSort | 95% | 64.25% | 45.15% |

Table 1: Phrase Precision Performance

### 4.2.2 Phrase Recall

Results shown below in Plot 2 and Table 2. PRDR-Detect-PostRank performs best here, with 76.73% recall for all its results (Top 800).

Plot 2: Phrase Recall Performance of Different Frameworks



| | Top 100 | Top 400 | Top 800 |
|---|---|---|---|
| PRDR-Detect-PostRank | 55.88% | 67.06% | 76.73% |
| AutoPhrase | 55.88% | 47.82% | 42.07% |
| AutoPhrase-WikiSort | 95% | 64.25% | 45.15% |

Table 2: Phrase Recall Performance

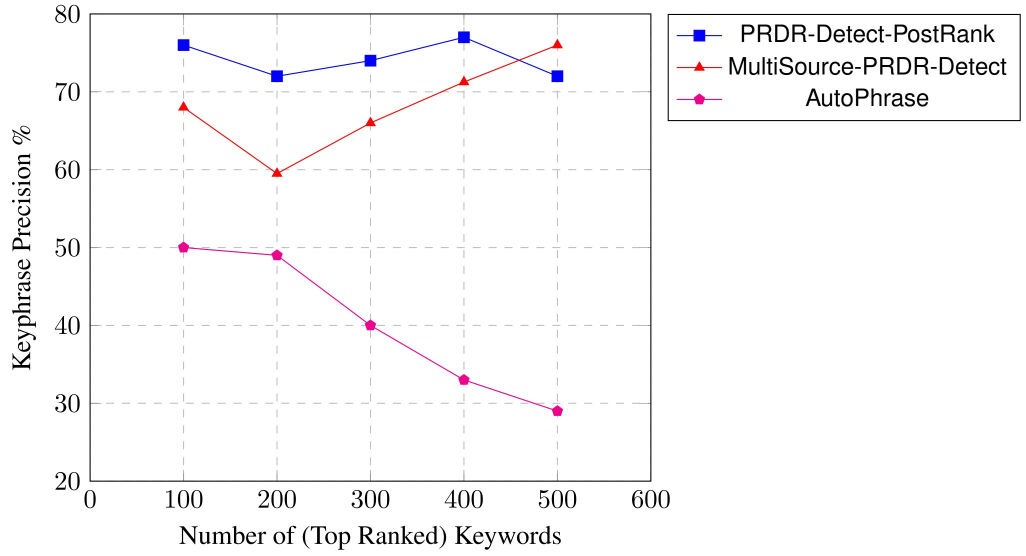
### 4.2.3 Keyphrase Precision

Results are shown in Table 3 and Plot 3. PRDR-Detect-PostRank performs best, with keyphrase precision averaging around 75% across all $K$'s.

| | Top 100 | Top 300 | Top 500 |
|---|---|---|---|
| PRDR-Detect-PostRank | 76% | 74% | 72% |
| MultiSource-PRDR-Detect | 68% | 66% | 76% |
| AutoPhrase | 50% | 40% | 29% |

Table 3: Keyphrase Precision Performance

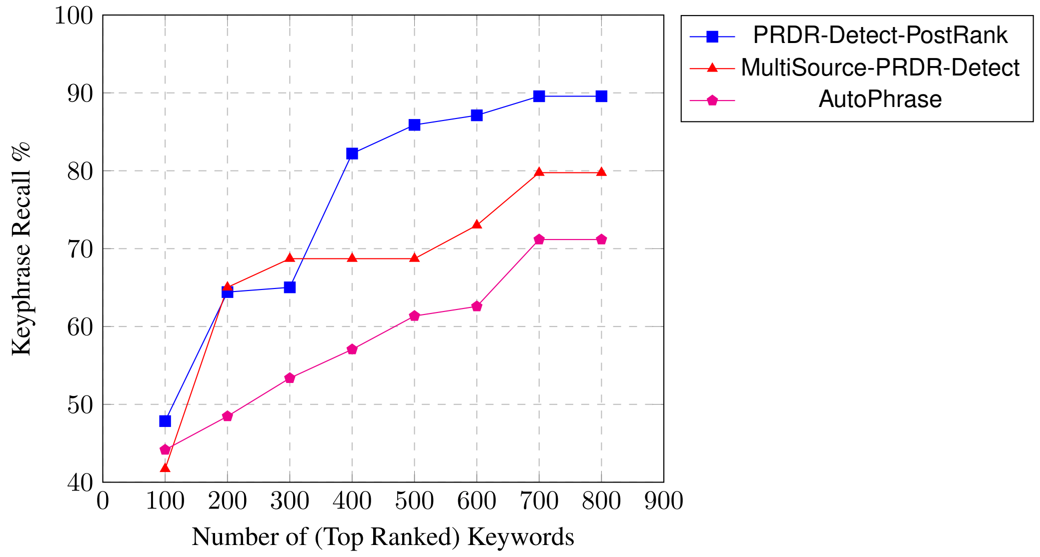
Plot 3: Keyphrase Precision Performance of Different Frameworks



### 4.2.4 Keyphrase Recall

Results are shown in Table 4 and Plot 4. PRDR-Detect-PostRank performs best, with keyphrase recall @ Top 800 being 20% higher than that of AutoPhrase's keyphrase recall @ Top 800.

Plot 4: Keyphrase Recall Performance of Different Frameworks



| | Top 100 | Top 400 | Top 800 |
|---|---|---|---|
| PRDR-Detect-PostRank | 47.85% | 82.21% | 89.57% |
| MultiSource-PRDR-Detect | 41.72% | 68.71% | 79.75% |
| AutoPhrase | 44.17% | 57.06% | 71.17% |

Table 4: Keyphrase Recall Performance

## 5 Conclusion and Future Work

### 5.1 Conclusion

In this paper, we present a novel solution to the domain-specific keyphrase mining problem, with the pattern-relation duality. We use work on pattern-relation duality from a previous paper, and develop three models - PRDR-Detect, PRDR-Detect-PostRank, MultiSource-PRDR-Detect - the latter two of which incorporate using Wikipedia and AutoPhrase's phrasal segmentation system. We evaluate our models with five measures - phrase precision, phrase recall, keyphrase precision, keyphrase recall.

### 5.2 Future Work

1. Further Development of MultiSource-PRDR-Detect: Although the model in theory should perform better than PRDR-Detect-PostRank (because it embeds Wikipedia precision and Wikipedia recall within the dual inference framework), this expectation is not met in our experiments. Analysis ofthe model is required to figure out the issue behind this.
2. Reducing the execution time of our models: Currently, our models take a while to execute (approx. 6 hours on our dataset), the bottleneck being the iterations till convergence required for the dual inference framework.
    - To combat this, we can possibly try a bootstrapping approach to finding candidate patterns using distributional semantics. We could pre-compute and store all patterns that occur in our documents collection. Then, each word segment $w$ can be represented using a vector $v(w)$ where $v(w)_i = 0$ or $1$ depending on whether $w$ occurs with the $i$th pattern. Once we precompute $v$'s for all possible word segments, we can use vector similarity approaches to quickly find candidate keyphrases from the seed keyphrases. This would eliminate the need for multiple global iterations of a framework (to iteratively find new patterns and keyphrases). We would only need to execute PRDualRank once, after we have all the candidate keyphrases (and associated patterns).
    - For pattern-matching within our document collection, we can develop an Aho-Corasick automaton based algorithm, which would execute in time linear in the length of the text.

## Acknowledgements

Thanks to Professor Chang, who guided and mentored me during this project. I really appreciate the weekly project meetings and one-on-one discussions that were instrumental in furthering my progress in the project, espcially when I hit roadblocks.

Thanks to Shuhan Wang, who helped implement the pattern-searching and tuple-searching part of the PRDualRank code. I am thankful for Shuhan's knowledge on the spaCy NLP library and graph visualization libraries that helped us progress quickly.

## References

[1] Y. Fang, K. Chang. Searching Patterns for Relation Extraction over the Web: Rediscovering the Pattern-Relation Duality. WSDM 2011.

[2] J. Shang et al. Automated Phrase Mining from Massive Text Corpora. IEEE Transactions on Knowledge and Data Engineering 2018. 30 (10), 1825–1837.

[3] G. Agarwal, G. Kabra, and K. Chang. Towards rich query interpretation: Walking back and forth for mining query templates. WWW 2010. Pages 1 - 10.

[4] J. Liu, J. Shang, C. Wang, X. Ren, and J. Han. Mining quality phrases from massive text corpora. ACM SIGMOD 2015.

[5] K. Frantzi, S. Ananiadou, and H. Mima. Automatic recognition of multi-word terms:. the c-value/nc-value method. JODL 2000. 3(2):115–130.

[6] Y. Park, R. J. Byrd, and B. K. Boguraev. Automatic glossary extraction: beyond terminology identification. COLING 2002.

[7] Z. Zhang, J. Iria, C. A. Brewster, and F. Ciravegna. A comparative evaluation of term recognition algorithms. LREC 2008.

[8] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. NIPS, 2001.

[9] E. Xun, C. Huang, and M. Zhou. A unified statistical model for the identification of english basenp. ACL 2000.

[10] K.-h. Chen and H.-H. Chen. Extracting noun phrases from large-scale texts: A hybrid approach and its automatic evaluation. ACL 1994.

[11] T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. ACL-HLT 2008.

[12] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. EMNLP 2005.

[13] P. Deane. A nonparametric method for extraction of candidate phrasal terms. ACL 2005.

[14] A. Parameswaran, H. Garcia-Molina, and A. Rajaraman. Towards the web of concepts: Extracting concepts from large datasets. VLDB 2010.

[15] A. El-Kishky, Y. Song, C. Wang, C. R. Voss, and J. Han. Scalable topical phrase mining from text corpora. VLDB 2015.

[16] I V S Venugopal, D Lalitha Bhaskari, M N Seetaramanath. A Domain Specific Key Phrase Extraction Framework for Email Corpuses. International Journal of Information Technology and Computer Science (IJITCS) 2018.

[17] Q. Pan, P. Guo, X. Xin and J. Liu. Extracting Company-Specific Keyphrases from News Media. International Conference on Computational Intelligence and Security (CIS) 2017.

[18] Qi Zhang, Yang Wang, Yeyun Gong, Xuanjing Huang. Keyphrase Extraction Using Deep Recurrent Neural Networks on Twitter. ACL 2016.

[19] A. Aouicha, M. Taieb, M. Ezzedine. Derivation of "is a" taxonomy from Wikipedia Category Graph. Engineering Applications of Artifical Intelligence 2016.

[20] Mohamed Ali Hadj TaiebMohamed Ben AouichaMohamed TmarAbdelmajid Ben Hamadou. Wikipedia Category Graph and New Intrinsic Information Content Metric for Word Semantic Relatedness Measuring. ICDKE 2012.

## Appendix

### Wikiprecision and Wikirecall Computation

Below, we describe how Wikiprecision and Wikirecall are approximated using the Wikipedia category graph.

### Preliminaries

**The Wikipedia Category Graph** Wikipedia contains categories, each of which pertain to a particular topic (for example, "computer science" is a category). Each category contains a list of articles which directly pertain to that category/topic (for example, the Wikipedia article on "computer science"). Each category also contains sub-categories under it. Each of these sub-categories are a Wikipedia category, which further contain directly related articles and their sub-categories. This results in the Wikipedia category graph. Categorization within the Wikipedia Category Graph is based on the linkes between Wikipedia articles.

**Wikipedia Article w.r.t. the Category Graph** Each Wikipedia article contains a list of categories $\mathcal{L}$ that it directly pertains to. (That is, each category in article $A$'s list of categories, contains $A$ as an article that directly pertains to it.)

### Algorithm for Wikiprecision and Wikirecall Computation

**Wikiprecision** Wikiprecision of $k$ represents the probability that starting from an article about $k$, we end up at the article about the domain (through a random walk).

First, "articles about $k$" are found using the Wikipedia API search function. It returns the top 500 results that directly pertain to $k$. In practice, we use the top 100 or 200 articles about $k$ (the relevance of the articles declines after the top 200 mark).

For each $a \in A_k$ (the top articles about $k$), we need to find the probability that we end up at the article about the domain $d$. We use $\mathcal{L}_a$ here, as follows:

$$\mathcal{X}_a = \frac{\text{Categories in } \mathcal{L}_a \text{ that are descendant categories of } d\text{'s category in the Wikipedia Category Graph}}{|\mathcal{L}_a|}$$

We then use a weighted average to calculate the Wikiprecision for $k$. If $A_k = \{a_1, a_2, \ldots, a_N\}$

$$\text{Wikiprecision}(k) = \sum_{i=1}^{N} \beta_i \mathcal{X}_{a_i}$$

$$\text{where } \beta_i > \beta_j \text{ if } i < j, \text{ and } \sum_{i=1}^{N} \beta_i = 1$$

**Wikirecall** Wikirecall of $k$ represents the probability that starting from the article about the domain, we end up at an article about $k$ (through a random walk).

The algorithm to compute Wikirecall is nearly the same as Wikiprecision, except the denominator of $\mathcal{X}_a$ changes:

$$\mathcal{Y}_a = \frac{\text{Categories in } \mathcal{L}_a \text{ that are descendant categoriess of } d\text{'s category in the Wikipedia Category Graph}}{\text{Number of descendant categories of } d\text{'s category in the Wikipedia Category Graph}}$$

In practice, since computing the denominator can be computationally expensive, we substitute the denominator with a (large) constant $C$, which is greater than any $\mathcal{L}_a$ (this works, since we only care about relative quantities here).

$$\mathcal{Y}_a \approx \frac{\text{Categories in } \mathcal{L}_a \text{ that are descendant categoriess of } d\text{'s category in the Wikipedia Category Graph}}{C}$$

Again, we use a weighted average to calculate the Wikirecall for $k$. If $A_k = \{a_1, a_2, \ldots, a_N\}$

$$\text{Wikirecall}(k) = \sum_{i=1}^{N} \beta_i \mathcal{Y}_{a_i}$$

$$\text{where } \beta_i > \beta_j \text{ if } i < j\text{, and } \sum_{i=1}^{N} \beta_i = 1$$

In practice, the first run of a framework is expensive because we need to compute Wikiprecision and Wikirecall for many candidates. By storing these computed values (as these do not change that quickly) within a file, subsequent runs of the framework require drastically less time.