SEERLAB: A System for Extracting Keyphrases from Scholarly Documents

Pucktada Treeratpituk[1] **Pradeep Teregowda**[2] **Jian Huang**[1] **C. Lee Giles**[1,2]
[1] Information Sciences and Technology
[2] Computer Science and Engineering
Pennsylvania State University, University Park, PA, USA

Abstract

We describe the SEERLAB system that participated in the SemEval 2010's Keyphrase Extraction Task. SEERLAB utilizes the DBLP corpus for generating a set of candidate keyphrases from a document. Random Forest, a supervised ensemble classifier, is then used to select the top keyphrases from the candidate set. SEERLAB achieved a 0.24 F-score in generating the top 15 keyphrases, which places it sixth among 19 participating systems. Additionally, SEERLAB performed particularly well in generating the top 5 keyphrases with an F-score that ranked third.

1 Introduction

Keyphrases are phrases that represent the important topics of a document. There are two types of keyphrases associated with scholarly publications: author-assigned ones and reader-assigned ones. In the Keyphrase Extraction Task (Kim et al., 2010), each system receives two set of scientific papers from the ACM digital library; a training set and a testing set. The author-assigned keyphrases and reader-assigned keyphrases are given for each paper in the training set. The objective is to produce the keyphrases for each article in the testing set.

This paper is organized as followed. First, We describe our keyphrase extraction system, SEERLAB. We then discuss its performance in SemEval 2010. Lastly, we analyze the effectiveness of each feature used by SEERLAB, and provide a summary of our findings.

2 System Description

SEERLAB consists of three main components: a section parser, a candidate keyphrase extractor, and a keyphrase ranker. To generate keyphrases for a paper, the section parser first segments the document into pre-defined generic section types. Secondly, the candidate keyphrase extractor generates a list of candidate phrases based on the document content. Then, the keyphrase ranker ranks each candidate according to the likelihood that it is a keyphrase. The top candidates are selected as keyphrases of the paper.

2.1 Section Parser

The goal of the section parser is to parse each document into the same set of pre-defined sections. However, segmenting a scientific article into pre-defined section types is not trivial. While scholarly publications generally contains similar sections (such as *Abstract* and *Conclusion*), a section's exact header description and the order in which it appears can vary from document to document. For example, the "*Related Work*" section is sometimes referred to as "*Previous Research*" or "*Previous Work*." Also, while the "*Related Work*" section often appears right after the introduction, it could also appear near the end of a paper.

(Nguyen and Kan, 2007) had success in using a maximum entropy (ME) classifier to classify sections into 14 generic section types including those such as *Motivation, Acknowledgement, References*. However, their approach requires annotated training data, which is not always available. Instead, SEERLAB uses regular expressions to parse each document into 6 generic section types: *Title, Abstract, Introduction, Related Work, Methodology + Experiments, and Conclusion + Future Work*. We decided to go with the smaller number of section types (only 6), unlike previous work in (Nguyen and Kan, 2007), because we believed that many sections, such as *Acknowledgement*, are irrelevant to the task.

Proceedings of the 5th International Workshop on Semantic Evaluation, ACL 2010, pages 182–185,
Uppsala, Sweden, 15-16 July 2010.

2.2 Extracting Candidate Keyphrases

In this section, we describe how SEERLAB derives a set of candidate keyphrases for a given document. The goal of the candidate extractor is to include as many actual keyphrases in the candidate set as possible, while keeping the number of candidates small. The performance of the candidate extractor determines the maximum achievable Recall of the whole system. The more correct candidates extracted at this step, the higher the possible Recall. But a bigger candidate set potentially could lower Precision. In our implementation, we decided to ignore the *Methodology + Experiments* sections to limit the size of candidate sets.

First, SEERLAB extracts a list of bigrams, trigrams and quadgrams that appear at least 3 times in titles of papers in DBLP[1], ignoring those that contain stopwords. Prepositions such as "of", "for", "to" are allowed to be present in the ngrams. From 2,144,390 titles in DBLP, there are 376,577 of such ngrams. It then constructs a trie (a prefix-tree) of all ngrams so that it can later perform the longest-prefix matching lookup efficiently.

To generate candidates from a body of text, we start the cursor at the beginning of the text. The DBLP trie is then used to find the longest-prefix match. If no match is found, the cursor is moved to the next word in the text. If a match is found, the matched phrase is extracted and added to the candidate set, while the cursor is moved to the end of the matched phrase. The process is repeated until the cursor reaches the end of the text.

However, the trie constructed as described above can only produce non-unigram candidates that appear in the DBLP corpus. For example, it is incapable of generating candidates such as *"preference elicitation problem,"* which does not appear in DBLP, and *"bet,"* which is an unigram. To remedy such limitations, for each document we also include its top 30 most frequent unigrams, its top 30 non-unigram ngrams and the acronyms found in the document as candidates.

Our method of extracting candidate keyphrases differs from most previous work. Previous work (Kim and Kan, 2009; Nguyen and Kan, 2007) uses hand-crafted regular expressions for candidate extractions. Many of these rules also require POS (part of speech) inputs. In contrast, our method is corpus-driven and requires no additional input from the POS tagger. Additionally, our approach allows us to effectively include phrases that appear only once in the document as candidates, as long as they appear more than twice in the DBLP data.

[1] http://www.informatik.uni-trier.de/ ley/db/index.html

2.3 Ranking Keyphrases

We train a supervised Random Forest (RF) classifier to identify keyphrases from a candidate set. A Random Forest is a collection of decision trees, where its prediction is simply the aggregated votes of each tree. Thus, for each candidate phrase, the number of votes that it receives is used as its fitness score. Candidates with the top fitness scores are then chosen as keyphrases. The detail of the Random Forest algorithm and the features used in the model are given below.

2.3.1 Features

We represent each candidate as a vector of features. There are the total of 11 features.

N: The length of the keyphrase.

ACRO: A binary feature indicating whether the keyphrase appears as an acronym in the document.

TF$_{doc}$: The number of times that the keyphrase appears in the document.

DF: The document frequency. This is computed based on the DBLP data. For document-specific candidates (unigrams and those not found in DBLP), their DFs are set to 1.

TFIDF: The TFIDF weight of the keyphrase, computed using TF$_{doc}$ and DF.

TF$_{headers}$: The number of occurrences that the keyphrase appears in any section headers and subsection headers.

TF$_{section_i}$: The number of occurrences that the keyphrase appears in the $section_i$, where $section_i \in \{$*Title, Abstract, Introduction, Related Work, Conclusion*$\}$. These accounted for the total of 5 features.

2.3.2 Random Forest

Since a random forest (RF) is an ensemble classifier combining multiple decision trees (Breiman, 2001), it makes predictions by aggregating votes of each of the trees. To built a random forest, multiple bootstrap samples are drawn from the original training data, and an unpruned decision tree is built from each bootstrap sample. At each node in a tree, when selecting a feature to split, the selection is done not on the full feature set but on a randomly selected subset of features instead. The

Gini index[2], which measures the class dispersion within a node, is used to determine the best splits.

RFs have been successfully applied to various classification problems with comparable results to other state-of-the-art classifiers such as SVM (Breiman, 2001; Treeratpituk and Giles, 2009). It achieves high accuracy by keeping a low bias of decision trees while reducing the variance through the introduction of randomness.

One concern in training Random Forests for identifying keyphrases is the data imbalanced problem. On average, 130 candidates are extracted per document but only 8 out of 130 are correct keyphrases (positive examples). Since the training data is highly imbalanced, the resulting RF classifier tends to be biased towards the negative class examples. There are two methods for dealing with imbalanced data in Random Forests (Chen et al., 2004). The first approach is to incorporate class weights into the algorithm, giving higher weights to the minority classes, so that misclassifying a minority class is penalized more. The other approach is to adjust the sampling strategy by down-sampling the majority class so that each tree is grown on a more balanced data. In SEERLAB, we employ the down-sampling strategy to correct the imbalanced data problem (See Section 3).

3 Results

In this section, we discuss the performance and the implementation detail of our system in the Keyphrase Extraction Task. Each model in the experiment is trained on the training data, containing 144 documents, and is evaluated on a separate data set of 100 documents. The performance of each model is measured using Precision (P), Recall (R) and F-measure (F) for the top 5, 10 and 15 candidates. A keyphrase is considered correct if and only if it exactly matches one of the answer keys. No partial credit is given.

Three baseline systems were provided by the organizer: *TF.IDF*, *NB* and *ME*. All baselines use the simple unigrams, bigrams and trigrams as candidates and *TFIDF* as features. *TF.IDF* is an unsupervised method that ranks each candidate based on *TFIDF* scores. NB and ME are supervised Naive Bayes and Maximum Entropy respectively.

We use the randomForest package in R for our keyphrase ranker (Liaw and Wiener, 2002). All RF models are built with the following parameters: the number of trees = 200 and the number of features considered at each split = 3. The average training and testing time are around 15s and 5s.

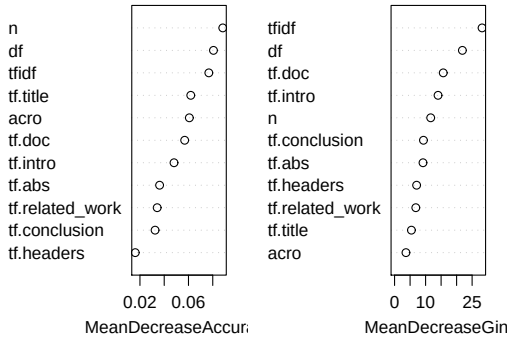

Figure 1: Variable importance for each feature

Table 1. compares the performance of three different SEERLAB models against the baselines. RF_0 is the basic model, where the training data is imbalanced. For $RF_{1:1}$, the negative examples are down-sampled to make the data balanced. For $RF_{1:7}$, the negative examples are down-sampled to where its ratio with the positive examples is 7 to 1. All three models significantly outperform the baselines. The $RF_{1:7}$ model has the highest performance, while the $RF_{1:1}$ model performs slightly worse than the basic model RF_0. This shows that while the sampling strategy helps, overdoing it can hurt the performance. The optimal sampling ratio ($RF_{1:7}$) is chosen according to a 10-fold cross-validation on the training data. For the top 15 candidates, $RF_{1:7}$'s F-score (C) ranks sixth among the 19 participants with a 24.34% F-score approximately 1% lower than the third place team. We also observed that SEERLAB performs quite well for the top 5 candidates with 39% Precision (C). Its F-scores at the top 5, 19.84% (C) and 18.19% (R), place SEERLAB third and second respectively among other participants.

Figure 1. shows two variable importance indicators for each feature: *mean decrease accuracy (MDA)* and *mean decrease Gini (MDG)*. Both indicators measure each feature's contribution in identifying whether a candidate phrase is a keyphrase. The *MDA* of a feature is computed by randomly permuting the value of that feature in the training data and then measuring the decrease in prediction accuracy. If the permuted feature is

[2]For a set S of data with K classes, its Gini index is defined as: $Gini(S) = \sum_{j=1}^{K} p_j^2$, where p_i denotes the probability of observing class i in S.

System	by	top 5 candidates			top 10 candidates			top 15 candidates		
		P	R	F	P	R	F	P	R	F
TF.IDF	R	17.80	7.39	10.44	13.90	11.54	12.61	11.60	14.45	12.87
	C	22.00	7.50	11.19	17.70	12.07	14.35	14.93	15.28	15.10
NB	R	16.80	6.98	9.86	13.30	11.05	12.07	11.40	14.20	12.65
	C	21.40	7.30	10.89	17.30	11.80	14.03	14.53	14.87	14.70
ME	R	16.80	6.98	9.86	13.30	11.05	12.07	11.40	14.20	12.65
	C	21.40	7.30	10.89	17.30	11.80	14.03	14.53	14.87	14.70
SEERLAB (RF_0)	R	29.00	12.04	17.02	22.50	18.69	20.42	18.20	22.67	20.19
	C	36.00	12.28	18.31	28.20	19.24	22.87	22.53	23.06	22.79
SEERLAB ($RF_{1:1}$)	R	26.00	10.80	15.26	20.80	17.28	18.88	17.40	21.68	19.31
	C	32.00	10.91	16.27	26.00	17.74	21.09	21.93	22.44	22.18
SEERLAB ($RF_{1:7}$)	R	31.00	12.87	18.19	24.10	20.02	21.87	19.33	24.09	21.45
	C	**39.00**	**13.30**	**19.84**	**29.70**	**20.26**	**24.09**	**24.07**	**24.62**	**24.34**

Table 1: Performance (%) comparison for the Keyphrase Extraction Task. R (Reader) indicates that the reader-assigned keyword is used as the gold-standard and C (Combined) means that both author-assigned and reader-assigned keyword sets are used.

a very good predictor, then the prediction accuracy should decrease substantially from the original model. The *MDG* of a feature implies that average Gini decreases for the nodes in the forest that use that feature as the splitting criteria.

TFIDF and *DF* are good indicators of performance according to both *MDA* and *MDG*. Both are very effective when used as splitting criteria, and the prediction accuracy is very sensitive to them. Surprisingly, the length of the phrase (*N*) also has high importance. Also, TF_{title} and *ACRO* have high *MDA* but low *MDG*. They have high *MDA* because if a candidate phrase is an acronym or appears in the title, it is highly likely that it is a keyphrase. However, most keyphrases are not acronyms and do not appear in titles. Thus, on average as splitting criteria, they do not decrease Gini index by much, resulting in a low *MDG*. Also, $TF_{related\_work}$ and $TF_{headers}$ have lower *MDA* and *MDG* than TF of other sections (TF_{intro}, TF_{abs}, and $TF_{conclusion}$). This might suggest that the occurrences in the "*Related Work*" section or section headers are not strong indicators of being a keyphrase as the occurrences in the sections "*Introduction*," "*Abstract*" and "*Conclusion*."

4 Conclusion

We have described our SEERLAB system that participated in the Keyphrase Extraction Task. SEERLAB combines unsupervised corpus-based approach with Random Forests to identify keyphrases. The experimental results show that our system performs well in the Keyphrase Extraction Task, especially on the top 5 key phrase candidates. We also show that the down-sampling strategy can be used to enhance our performance.

References

Leo Breiman. 2001. Random forests. *Machine Learning*, Jan.

Chao Chen, Andy Liaw, and Leo Breiman. 2004. Using random forest to learn imbalanced data. *Technical Report, University of California, Berkeley*.

Su Nam Kim and Min-Yen Kan. 2009. Re-examining automatic keyphrase extraction approaches in scientific articles. *Proceedings of the Workshop on Multiword Expressions, ACL-IJCNLP*, Jan.

Su Nam Kim, Olena Medelyan, Min-Yen Kan, and Timothy Baldwin. 2010. Semeval-2010 task 5: Automatic keyphrase extraction from scienctific article. *ACL workshop on Semantic Evaluations (SemEval 2010)*.

Andy Liaw and Matthew Wiener. 2002. Classification and regression by randomforest. *R News*.

Thuy Dung Nguyen and Min-Yen Kan. 2007. Keyphrase extraction in scientific publications. *Proceedings of International Conference on Asian Digital Libraries (ICADL'07)*, Jan.

Pucktada Treeratpituk and C Lee Giles. 2009. Disambiguating authors in academic publications using random forests. *In Proceedings of the Joint Conference on Digital Libraries (JCDL'09)*, Jan.