A System for Mapping Text to Specific Reviewers

Master-Thesis von Xinyu Liu aus Darmstadt
Tag der Einreichung:

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

A System for Mapping Text to Specific Reviewers

Vorgelegte Master-Thesis von Xinyu Liu aus Darmstadt

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Eneldo Loza

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den June 8, 2018

_____

(Xinyu Liu)

Abbreviations

BoW Bag-of-words

DBLP Digital Bibliography and Library Project

IR Information Retrieval

NLTK Nature Language Toolkit

POS Tagging Part–Of–Speech Tagging

RMS Reviewer Mapping System

TF-IDF Term frequency-inverse document frequency

VSM Vector-space model

w2v Word2Vec

WE Word Embeddings

Nomenclature

A The administrator of the Reviewer Mapping System

D The document collection of all reviewers

d_i The document file of the i-th reviewer

p_i The profile of the i-th reviewer

q A query

R The set of reviewers

r_i The i-th reviewer

U The user collection

u A user

List of Figures

List of Tables

Abstract

Automatic reviewer assignment is a common and crucial task faced in either academia or daily life. For example, for program chairs of conference or scientific journal editors, assigning a submitted paper to the most appropriate reviewers is a basic process in order to enable the author to get a fair and high-quality assessment. However, it is not a simple task, as the numbers of both papers and reviewers are fairly large, manually assigning thousands of papers to thousands of reviewers is significantly beyond the ability of one person. Furthermore, another constraint exists: Usually, if a new reviewer need to be added into the review system, and the program chair is not familiar with him or her, a lot of time has to be spent studying the expertise of this reviewer, which is really time-consuming. Selecting appropriate reviewers must certainly includes some consideration regarding a reviewer's expertise or research interest.

This paper provides an overview of the web interface Reviewer Mapping System(RMS) developed by the author. Its features include modifying the set of reviewers in the system (adding and deleting reviewers), providing the suggested URLs of a reviewer's pages (in this system, is the reviewer's home page in DBLP computer science bibliography [5]), characterizing a reviewer's profile by analyzing their published papers, and matching the abstracts of submitted papers to the appropriate reviewers. Nowadays, it is common to convert the task of matching abstracts to appropriate reviewers into an information retrieval task. In this thesis, two approaches are used to implement information retrieval: one is the bag-of-words model, vector-space model is a typical algorithm, and the other one is the word-embedding model, a typical approach is word2vec. Both approaches will be evaluated in order to determine which model is more appropriate for the Reviewer Mapping System.

Contents

1 Introduction

Peer review is a process where experts make comments and give feedback about the quality of works of others. Specifically, peer review of written works is a main constituent of modern academic research; in peer review, appropriate experts review submitted papers and perform an assessment of their quality. Nowadays, more and more research subtopics are emerging. For example, under the topic of computer science, there are plenty of subtopics, e.g., machine learning, operating systems, computer networks, etc. Under the topic of machine learning, for instance, other more refined subtopics exist as well, e.g., rule learning, reinforcement learning, deep learning, etc.. Different experts have their own individual topics in which they are most interested. Even if two reviewers are experts in the field of computer science, and even the field of machine learning, one may be better at reinforcement learning while the other one may do great research about deep learning. Although the expert in reinforcement learning might also have knowledge about deep learning, it would be better to assign a paper about deep learning to the expert who actually does a lot of research in this field, so that the paper can get a more professional, fair evaluation and more appropriate feedback. Vice versa, it is also possible that some new and constructive research results and ideas presented by a high-quality paper may further support the research of this expert. Hence, accurately matching papers with reviewers is a job that is of great significance.

1.1 Motivation

The task of expert assignment and recommendation is a common task faced in both academia and daily life. It requires the program chair or editor to know enough about the expertise and interest of the experts at their disposal. Taking an example from our daily life, a job recruiting process normally involves hundreds of applicants applying for a position. Human Resources (HR) need to look at hundreds of resumes, learn about the applicants' expertise by looking at their previous work experience, their educational background, and their skills, and identify their strengths in order to pick applicants who meet the requirements of a particular job. This process is fairly time-consuming, and manually performing this task is very difficult and ineffective. Hence, it is necessary to develop a system that automatically recognizes the expertise of each applicant.

Another example is the situation at a scientific conference. Prior to the submission phase, the conference chair lists a series of research topics. All the reviewers then need to select the topics that correspond to their research interests and expertise. During the submission process, the authors are required to explicitly describe which topics their papers belong to. One possible problem is that the topics selected by the authors sometimes do not perfectly describe the actual topics of their paper, which is always misleading to a certain degree. In addition, some conferences ask the reviewers to select the submitted papers by means of bidding. Normally, the papers selected by a reviewer should refer to their expertise and competence. However, sometimes reviewers have certain preferences or bias, due to their own taste or because they are curious. This usually causes the situation that some papers are very popular and many reviewers are willing to review them, while other papers may be received less enthusiastically. This is not fair, because sometimes a paper cannot get an objective assessment. In order to make this process more objective, a system should be developed for automatically assigning papers to appropriate reviewers.

1.2 Contributions

The main contribution of this thesis is the web interface Reviewer Mapping System (RMS) that was developed. The interface is able to implement the following functions:

- There are two groups of users: an administrator A, and several general reviewers R.
- There is a finite set of reviewers R. It should be possible for A to interactively modify this set, e.g., to add or delete people from the set.
- For each reviewer r_i in R, the system should suggest pages (e.g., the personal page in DBLP, their Google Scholar pages, their home pages, etc.) so that A is able to check their published papers and documents, in this thesis we use DBLP page.
- For each reviewer r_i in R: documents d_i need to be provided that characterize the research profile of p_i.
- From each reviewer's DBLP page suggested by the system, a "reviewer profile" must be extracted according to their published papers. This can be in the form of some sort of "average bag-of-words (BoW)" or in the form of word embeddings that make semantic sense, such as word2vec (w2v).
- Every user u in U should then be able to use the Web interface and post a query q consisting of a longer text (e.g., the abstract of a research paper). This will then be processed in the same way as the documents for each reviewer have been processed (using either a BoW or a w2v) to represent. The system will then return a ranked list of the people showing which people best fit to the query.

In this thesis, both a bag-of-words model and a word embeddings model are used to represent the "user profile" and the "reviewer profile". The task of finding reviewers is an application of the task of information retrieval (IR). The submitted abstract is regarded as a query, and the reviewers' profiles are seen as documents. Then these documents are ranked according to the similarity between query and document. Hence, two approaches are adopted to implement the task of information retrieval:

- One approach is based on the statistical language model, using a bag-of-words vector to represent the "user profile" and the "reviewer profile" and then ranking the list of reviewers by means of the vector-space model.
- The other approach is based on word embeddings, using a word2vec to represent the "user profile" and "reviewer profile" and then ranking the list of reviewers by measuring the distance between "user profile" and "reviewer profile". The shorter the distance, the higher the similarity between the "user profile" and this "reviewer profile".

Finally, an evaluation will be performed for both approaches. To do so, 80 reviewers are entered into the system, two abstracts are taken from about 33 of the reviewers, and it is checked for how many of them the correct author is identified. This allows drawing conclusions on which approach is more appropriate for the RMS. Figure 1 shows the workflow of the reviewer assignment process.

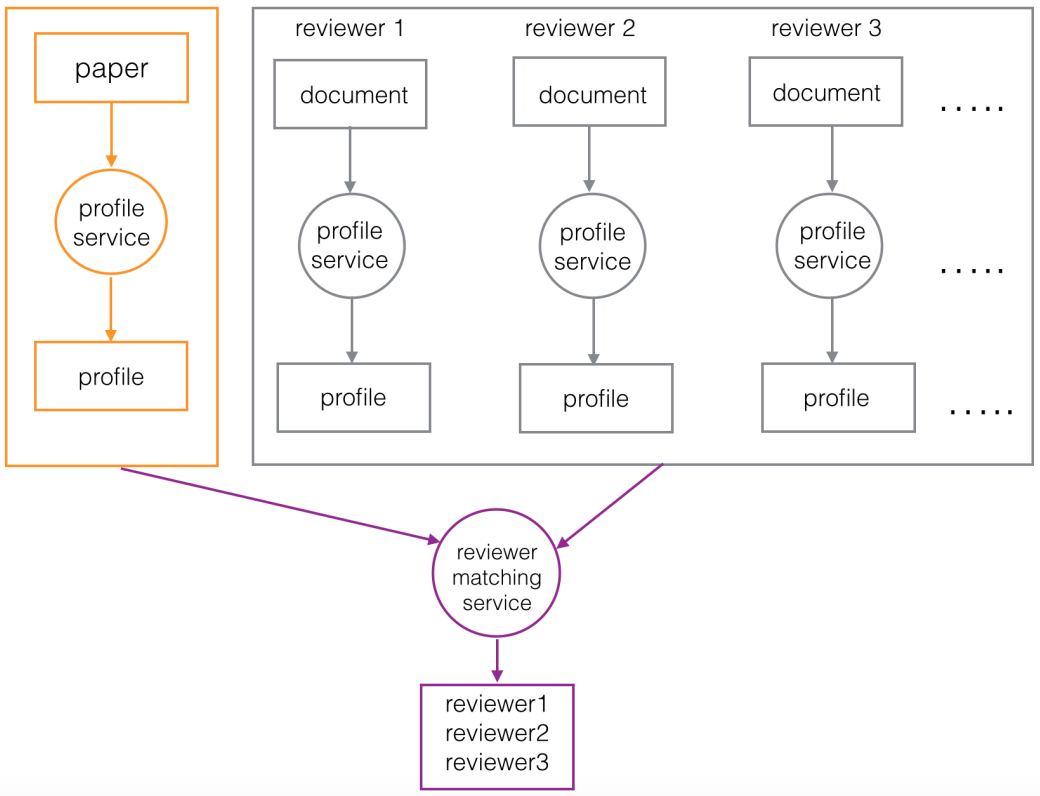

Figure 1: Work flow of the reviewer assignment process

1.3 Related Works

To date, a lot of research has been performed already about reviewer assignment. Most of it addresses this task by using machine learning and information retrieval approaches. For example, the popular Toronto Paper Matching System TPMS [16] uses Latent Dirichlet Allocation (LDA) to model the topics of the reviewers and the variants of a vector-space model to rank the reviewers. It also adopts the bidding strategy, meaning reviewers are asked to select the papers in which they are interested. The recent work in [2] Hettich et al. used term frequency-inverse document frequency (TF-IDF) to measure the similarity between papers and reviewers. Similarly, Basu et al. [1] looked for the abstracts of a reviewer's published papers on the web and also utilized the TF-IDF model to rank the reviewers. In [8], Mimno et al. extracted the topics of the reviewers and those of the submitted papers, then measured the similarity of the topics. Rodriguez and Bollen [10] utilized a graph model connecting co-authorship and the assigned paper to rank the reviewers. Price et al. [9] used a vector-space model to determine the appropriate reviewers. As a complement, Zhai et al. [3] proposed that the reviewer assignment problem should not only consider the main topics presented in a paper, but also the subtopics. In the work of [6], Liu et al. propose that solving the reviewer assignment problem not only involves the aspect of expertise, but should also take into account other aspects, e.g., authority and diversity. Based on a graph model, they determined the most appropriate reviewer for a given paper.

Most existing papers solve the task of matching a given paper to the appropriate reviewers by means of a bag-of-words approach. Originally, researchers used the TF-IDF approach to model a paper and each reviewer, then ranked the reviewers by measuring the similarity. An

obvious shortcoming of this approach proved to be that the semantic relations among words were not taken into account. A typical example is a query like: “This paper is about using the k-means cluster algorithm to extract the topics of the given document.” Assume two reviewers: the “profile” of reviewer 1 includes “unsupervised learning”, “page rank algorithm”, and “nature language processing”; that of reviewer 2 includes “supervised learning” and “image cluster”. The “k-means cluster algorithm” is a sort of unsupervised learning cluster approach. Besides, this query is obviously about natural language processing. Hence, in theory, reviewer 1 would be the more appropriate reviewer; however, as the term “cluster” occurs in the profile of reviewer 2, the TF-IDF model thinks that reviewer 2 is a better match for the query and thus assigns the query paper to reviewer 2. This is a common problem for the TF-IDF model. In later research, although the BoW model continued to be used to solve the reviewer assignment problem, it was rather based on words extracted from the topics of both reviewers’ documents and submitted papers, then comparing the similarity between those topics, the LDA or LSA (Latent Semantic Analysis) topic model were applied to solve this problem. Using this approach makes it possible to overcome the shortcomings of the TF-IDF model, as it considers the effect of semantics. So it can be regarded as an improvement.

In addition to the BoW model, this thesis proposes using the Word Embeddings model to solve the reviewer assignment problem, using word2vec to represent each word. By measuring the distance between two words, we can determine the degree of similarity between these two words. Hence, this model also considers the semantic relations.

1.4 Structure of this Thesis

In chapter 2, all natural language processing (NLP) and information retrieval (IR) methods used in this thesis will be introduced, including the TF-IDF model, the vector-space model, the Word Mover’s Distance, and the word embeddings model (word2vec). In chapter 3, the reviewer assignment problem will be explained in more detail. In chapter 4, the two methods used to solve the reviewer assignment problem will be introduced: the BoW model and the word embeddings model. In addition, our solution, the RMS, will be presented, where a reviewer profile is built, abstracts are matched to reviewers, and the implementation is shown. In chapter 5, the two approaches will be evaluated and compared in order to find the most appropriate approach for the RMS. In chapter 6, the usage of the RMS will be presented, including its database design and GUI explanation. Finally, in chapter 7, conclusions will be drawn, the main contributions of this thesis will be summarized, and some future work for the RMS will be sketched.

2 Fundamentals

2.1 Introduction to Information Retrieval (IR)

With the advent of the Internet and Big Data, we are surrounded by more and more information. Selecting only relevant information from a large collection of data is becoming a really urgent task in order to significantly improve our work efficiency. The area of Information Retrieval (IR) emerged already in the 1950s, but only in the last 40 years has IR technology been widely used and gradually matured. A typical example of IR is a search engine, which finds the most relevant documents when a query is entered.

In earlier stages, IR systems used Boolean mechanisms, which ask users to use complex Boolean expressions (ANDs, ORs, NOTs) to specify their query (e.g., Messi AND Ronaldo NOT Neymar). The IR system then returns all the documents that satisfy the Boolean expressions. Table 1 is a term-document incidence matrix. Each document is represented as bag-of-terms. If the query term occurs in the document, the entry is 1; otherwise it is 0. Let us assume the query is Messi AND Ronaldo NOT Neymar. By taking the vectors for Messi, Ronaldo, and Neymar, and using a complement for the vector of Neymar, then doing a bitwise calculation on the three vectors, $1100 \ \ and \ \ 1101 \ \ and \ \ 1011 \ \ = \ \ 1000$, the system finds that document 1 meets the query and is returned. Obviously, a Boolean IR system has several shortcomings: (1) The Boolean expression query is too complicated for common users; (2) as the number of documents increases, the matrix becomes extremely sparse; (3) a Boolean IR system returns the documents without any ranking. Especially the third shortcoming means that such a system is unable to meet the requirements of today's users. Most users expect to get ranked documents in order to be able to assess the usefulness of a document. Hence, most IR systems assign a numeric score to each document and rank it according to this score. In this thesis, two IR models will mainly be introduced, the vector-space model and the word embeddings model.

	doc 1	doc 2	doc 3	doc 4
Messi	1	1	0	0
Ronaldo	1	1	0	1
Neymar	0	1	0	0
Result	1	0	0	0

Table 1: Example of a Boolean IR system

2.2 The TF-IDF Model

The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times the term occurs in this document. Intuitively, a term i, $tf_i = 10$ has a greater chance of being a key word than a term j, $tf_j = 1$. But it is hard to say that term i is ten times as important as term j, meaning the importance does not increase proportionally with the frequency of a term. Hence, instead of simply using tf to represent term frequency, log term frequency will be used, which is defined as follow

$$w_{t,d} = \begin{cases} 1 + log_{10} tf_{t,d}, & \text{if } tf_{t,d} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Besides term frequency in a document, the frequency of a term in a collection of documents is also non-negligible because some rare terms may need greater focus. Consider a term such as arachnocentric, which may very rarely occur in a collection, but it is more informative than frequently occurring terms. If this term is contained in a document, it is very likely a keyword. In our case, this means that it is highly probable that it is a reviewer's research topic. Therefore, the weight of terms like arachnocentric should be high. To sum up, we want high weight for rare terms and relatively low weight for frequent words such as structure or algorithm. So document frequency, df_t should also be introduced, which is defined as the number of documents in which a term occurs. df_t is an inverse measure of the informativeness of term t, so we define the idf_t weight of term t as follows:

$$idf_t = log_{10}(\frac{N}{df_t})$$

where N is the number of documents in the collection and idf_t is a measure of the informativeness of the term. Like tf_t we also use $log \frac{N}{df_t}$ instead of $\frac{N}{df_t}$ to decrease the influence of idf_t.

In conclusion, the TF-IDF weight of a term is the product of its tf weight and its idf weight, that is

$$(1 + log_{10} tf_{t,d}) * log_{10} \frac{N}{df_t}$$

2.3 The Vector-Space Model

In the vector-space model, each document is represented as a vector of terms; with terms being phrases or words. If a term occurs in a document, it becomes a dimension in a super high-dimensional vector space. Then any document is able to be represented by a vector in the high-dimensional vector space. Provided a term occurs in a document, it obtain a non-zero value along the corresponding dimension; otherwise the value is zero. Because of the limited number of words in a document, document vectors are really sparse.

To assign a score to a document for a given query, the vector-space model measures the similarity between query vector and document vector, as a query is also represented as a vector in the high-dimensional vector space. Typically, there are two approaches for measuring the similarity between vectors:

- Cosine of the angle: If the cosine value is 1.0, both vectors are identical. If the cosine value is 0.0, both vectors are orthogonal.
- Inner-product between two vectors: If all the vectors are unit length, the inner-product between two vectors is the same as its cosine of the angle.

Next, we will use a formula to describe this. Query and documents are represented as the real-valued vector of the TF-IDF weights $\in R^{|V|}$, where V is the set of unique terms that form a V-dimensional real-valued vector space. A vector is normalized by dividing each of its components by its length
- here using L2 norm:

$$\left|\left|x\right|\right|_2 = \sqrt{\sum_i x_i^2}$$

As a result, longer documents and shorter documents have weights of the same order of magnitude. Hence, the similarity between two vectors is described as Equation 1:

$$Similarity(\overrightarrow{q}, \overrightarrow{d}) = cos(\overrightarrow{q}, \overrightarrow{d}) = \frac{\overrightarrow{q} \cdot \overrightarrow{d}}{|\overrightarrow{q}| \cdot |\overrightarrow{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}} \quad (1)$$

- q_i is the TF-IDF weight of term i in the query;
- d_i is the TF-IDF weight of term i in the document;
- $|\overrightarrow{q}|$ and $|\overrightarrow{d}|$ are the lengths of $\overrightarrow{q}$ and $\overrightarrow{d}$.

2.4 Word Mover's Distance

The Word Mover's Distance [4] is able to measure the similarity between two documents by "traveling" through all the embedded words in one document to reach the embedded words in the other document. In other words, all the non-stop words of two documents are embedded into a word2vec space. The Word Mover's Distance between two documents is the minimum accumulative distance matching all the words in a document to the other document. Matt J. Kusner and Yu Sun discussed this approach in [4]. They also illustrated this approach as depicted in Figure 2.

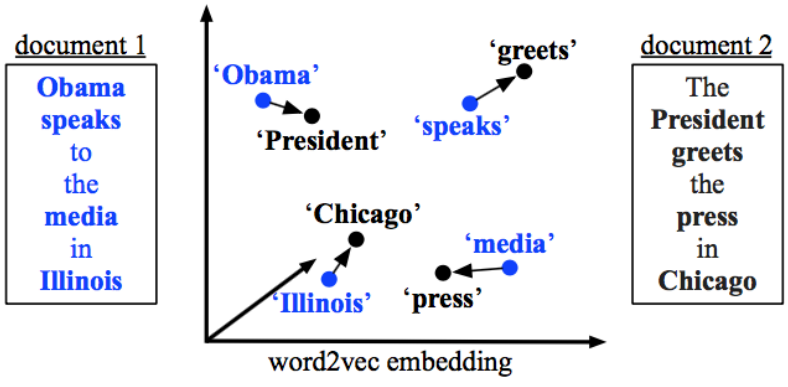

Figure 2: Example of the Word Mover's Distance example

Assume there are two documents: document 1: Obama speaks to the media in Illinois; document 2: The President greets the press in Chicago. After removing the stop-words, all the

remaining words of the two documents are denoted in the word2vec space. Blue dots represent document 1, while black dots represent document 2. Finding the closest word of document 2 for each word in document 1 results in matches such as: “Obama” to “President”, “speaks” to “greets”, “Illinois” to “Chicago” and “media” to “press”, as shown in Equation 2.

$$\begin{aligned} dist\{doc1, doc2\} = dist\{\text{‘Obama’, ‘President’}\} + dist\{\text{‘speaks’, ‘greets’}\} + \\ dist\{\text{‘Illinois’, ‘Chicago’}\} + dist\{\text{‘media’, ‘press’}\} \end{aligned} \tag{2}$$

Hence, the similarity between the two documents consists of a series of semantic similarities between individual word pairs (e.g., ‘speaks’ and ‘greets’). The distance between the words in a pair is called word travel cost. One measure for representing the word travel cost is their Euclidean distance in the word2vec space (Equation 3).

$$c(i, j) = ||\mathbf{x}_i - \mathbf{x}_j||_2 \tag{3}$$

Assuming the number of words in document 1 is n, and in document 2 it is m. Then the distance between two documents is calculated as shown in Equation 4:

$$dist(doc1, doc2) = \sum_{i=1, j=1}^{i=n, j=m} c(i, j) \tag{4}$$

2.5 Word Embeddings

Much research has been done to find the semantic relation between two different words. In 2013, Mikolov [7] introduced a word embeddings approach, namely word2vec, where each word is represented as a vector. The w2v model was trained using a neural network model. Specifically, a skip-gram model was introduced as a network architecture, including an input layer, a projection layer, and an output layer, to predict the words in the neighbor. Here is an example of the skip-gram model: ( )( )for( )( ). The words in parentheses are the neighboring words of the central word “for”. A skip-gram model is used to predict the neighboring words given a central word, as illustrated in Figure 3. Each word vector is trained to maximize the likelihood that the neighboring words will occur, as shown in Equation 5; i.e., assuming a series of words, the central word is represented as w, while its neighboring words are denoted as c:

$$\begin{aligned} & argmax_\theta \prod_{w \in Text} \prod_{c \in C(w)} p(c|w; \theta) \\ = & argmax_\theta \sum_{(w, c \in Text)} logp(c|w; \theta) \end{aligned} \tag{5}$$

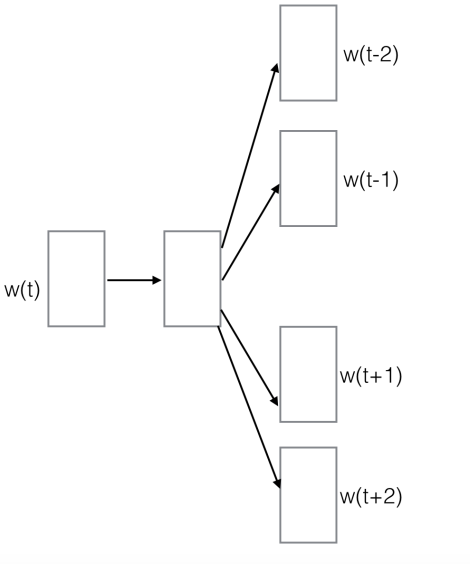

Figure 3: Skip-gram model

Due to the use of some training tricks like negative sample, this architecture is able to effectively train more than a billion words an hour. Due to its efficiency, it is able to train a large corpus so that it can identify the semantic relations between words, e.g., $king - man + woman = queen$. The word2vec training process is unsupervised, so we can use it to train our corpus. In this thesis, we will train the corpus on the topic of computer science.

3 Problem Desription

Reviewer assignment is a common task for a lot of people, e.g., academic leaders, conference program chairs, and journal editors. Normally, when research papers are submitted, e.g., to a conference, the conference program chair has to assign them to certain experts for review, then give the author(s) high-quality feedback and a fair assessment. The assignment must be based on prior knowledge about the expertise of the reviewers and the content of the submitted paper. Nowadays, most of this task is done manually by the conference program chairs. As the number of research papers is constantly increasing, program chairs are realizing that they can no longer do this task on their own, as it is very time-consuming. Hence, they proposed a new idea, asking reviewers to bid on papers. This means that reviewers are invited to bid on papers according to their own research interests and expertise. In this way, program chairs ease their own burden and shift the task to the reviewers. As mentioned above, the bidding system is still not able to solve the problem created by the reviewer assignment task, as it remains very time-consuming. Furthermore, it is also likely to lead to the problem of unfair assignment. For example, some high-quality papers dealing with currently hot topics always get more attention, and many reviewers are willing to assess them. On the other hand, papers with less quality are not welcome. This unfair and low-quality assignment does not fulfill our requirements. Hence, a high-quality reviewer assignment system that performs this task automatically should be built. Three tasks are important when building an automatic reviewer assignment system: (1) adding reviewers to the system, (2) extracting "reviewer profiles" according to their research works (e.g., published papers), their expertise and research interests, and (3) ranking the reviewers for a given paper according to similarity. In the following, these three tasks will be presented in more detail as part of the RMS developed in this thesis.

3.1 Adding Reviewers to the System

The first step in building the RMS is to add reviewers to the database. This requires two problems to be solved:

- There are many experts with the same name, but they do research in different areas. Hence, actually specifying who is the exact expert we want to add is a key point. During the process of adding reviewers, name disambiguation is a task to be solved.
- Only adding an expert is not enough. In order to identify the expert's specific expertise ("Reviewer Profile"), the documents d that can serve to characterize the research profile need to be created as well.

We use the DBLP [5], a computer science bibliography website that collects all the experts in the area of computer science and all their published papers to disambiguate the reviewers and create the documents d for each reviewer by crawling the DBLP page.

3.2 Building "Reviewer Profiles"

A reviewer's profile is a text representation describing his or her expertise and research interests. A profile can be used to get a list of unique terms (key phrases or keywords). It can also be compared with the "profiles" of submitted abstracts in order to produce a ranked list of reviewers according to similarity. A general approach for creating "reviewer profiles" is to automatically

extract key phrases or keywords from a reviewer's document d. Normally the "profile" of a reviewer means the most significant features that is able to character the reviewer. For example, reviewer 1 does a lot of research about "machine learning" and "rule learning", while reviewer 2's profile is "machine learning" and "neural networks". Because both reviewers' profiles contain the term "machine learning", "machine learning" should not be regarded as an important profile feature for characterizing both reviewers; in other words, "machine learning" should not be regarded as an important profile taht was assigned a high weight. Vice versa, "rule learning" can be seen as a crucial "profile" of reviewer 1 and "neural networks" as a crucial "profile" of reviewer 2. Hence, based on all reviewer documents $[d_1, d_2, ..., d_n]$ forming a document collection, we use the TF-IDF model to extract the important terms that can specifically characterize a particular reviewer.

3.3 Reviewers Ranking

Assume the given query is the abstract of a paper, and the abstract is processed in the same way as the documents for each reviewer have been processed to extract the "profile", and let $R = \{r_1, r_2, ..., r_n\}$ denote n reviewers in the system. In order to accomplish the task of reviewer assignment, we need to find the most appropriate reviewer from the set R, so we rank the reviewers according to the similarity between "reviewer profile" and submitted "paper profile". Let us use an example with one submitted abstract and two reviewers $r1$, $r2$:

Abstract profile : "rule learning", "decision tree"
r1 profile : "machine learning", "rule learning"
r2 profile: "machine learning", "neural networks"

Significantly, $r1$ has a common research interest shared with the paper, so the system should assign this paper to $r1$.

4 Solution

The task of reviewer assignment is usually converted into an Information Retrieval (IR) task. In this chapter, both of the approaches used to build the Reviewer Mapping System will be introduced: bag-of-words (BoW) and Word Embeddings. As mentioned in chapter 3, adding reviewers into system, "reviewer profile" building and reviewer ranking are three crucial tasks. Hence, the concrete way the three tasks were implemented using the BoW model, and the word embeddings model, will be introduced separately.

Each reviewer r_i has their own document d_i, which consists of a list of titles of all their published papers obtained by crawling their DBLP page. The documents of all the reviewers $[d_1, d_2, ..., d_n]$ compose the corpus D.

4.1 Adding Reviewers to System

The primary function that RMS implement is the administrator is able to add or remove the reviewer in the system. But it is possible that two or more reviewers have the same name. So RMS need to provide the function to identify the specific reviewer that administrator want to add. Furthermore, for each reviewer we need to create a documents d that can serve to characterize the research profile need to be created as well.

4.1.1 Disambiguate the reviewers with the same name

In each person's page, DBLP provides the tag "Other persons with the same name", Figure 4 shows this tag in the DBLP page. It includes all the persons with this name and the affiliation of each of them. It is rare to occur that two different reviewers with the same name even have the same affiliation. So with the help of affiliation information it is possible to disambiguate the reviewers.

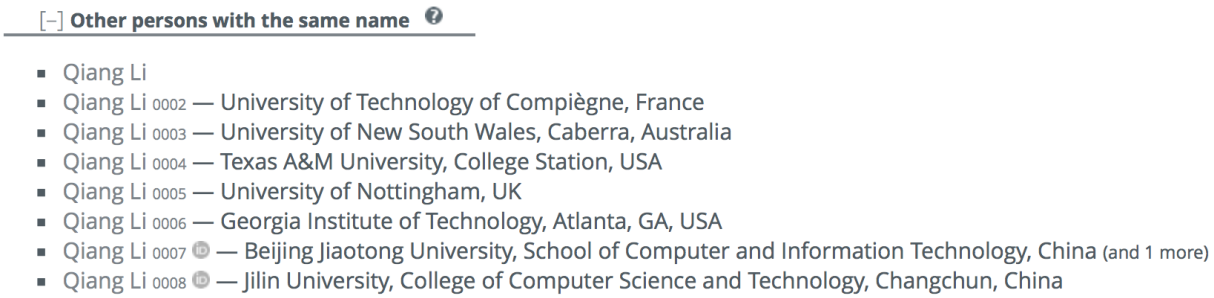

Figure 4: DBLP page tag for finding the persons with same name

4.1.2 Creating a document d_i for each reviewer

For each reviewer r_i: We need to provide documents d_i that characterize the research profile p_i. The best choice for this document are a reviewer's published papers. Because the web interface RMS is mainly aimed at users in the field of computer science, the bibliography website DBLP is used, which collects all computer science experts and their published papers. On account of the limit of authority [5], it is only possible to crawl the titles of published papers on each expert's DBLP page; abstracts are not allowed. Hence, we crawl the titles to compose the document d_i. Figure 5 shows an example of the HTML structure for title tags in a DBLP page.

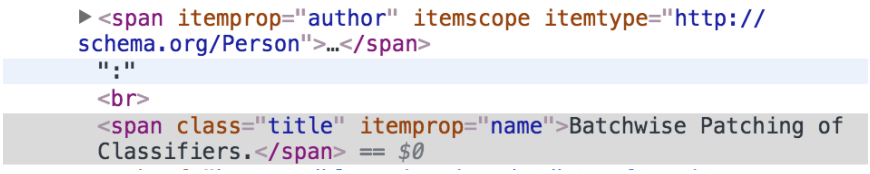

Figure 5: A example of HTML structure for title tag in DBLP page

- **Code explanation**:
  Two packages, urllib and BeautifulSoup in Python, are used to crawl HTML pages. Then crawl all titles of papers in each reviewer's DBLP page.

- **Code implementation**:

```
def parser_title(url):
"""
parser all the titles from a person's DBLP page
:param url(String): a person's DBLP page
:return: titles(list),  a list of titles of all the person's
   published papers
"""
# import two package
import urllib
from bs4 import BeautifulSoup

titles = list()   # create a list to store titles
#create a soup
response = urllib.urlopen(url)
content = response.read()
soup = BeautifulSoup(content, 'html.parser')

# get all titles of the published paper
title_tags = soup.find_all('span', class_='title')

for title_tag : title_tags:
  titles.append(title_tag.string)
return titles
```

Listing 1: Crawling titles from person's DBLP page

Before creating the reviewer document, we need to perform pre-processing for the document.

- Remove stop words.
- Lemmatize all the tokens.

Because stop words like “of”, “for”, and punctuation marks make no sense in this context, these terms should be removed to reduce the noise. The NLTK toolkit already contains all the stop words in its corpus. Hence, every title should first be tokenized; then it should be check there are any stop words. In addition to removing stop words and punctuation marks, we also need to lemmatize the tokens in order to reduce the influence of the same terms with different forms. After pre-processing, we write the title into a document file. The detailed code is as follows:

- **Code explanation**:
  Nature Language Toolkit (NLTK) package is used, which is able to implement the functions of tokenization, Part–Of–Speech Tagging (POS Tagging) and lemmatization. In addition, it collects all the english stopwords.

- **Code implementation**:

```
def pre_processing (title):
"""
Lemmatization of a title, removing stopwords and punctuation from
    the title.
: param : title(String): the title of one paper
: return : filtered_title(String), a title without stopwords and
   punctuation and be lemmatized
"""
import nltk
from nltk import pos_tag , word_tokenize
from nltk.stem. wordnet import WordNetLemmatizer
wnl = WordNetLemmatizer()

"""
Firstly tokenize the title to a list of tokens along with its POS
   -Tag, then accoring to different classes of tags(e.g.
   Adjective, Noun, Verb) to lemmatize each token and remove
   those tokens belong to stopwords and punctuations.
"""
filtered_title_tokens =
[wnl.lemmatize(token, pos_tag[0].lower())
  if pos_tag[0].lower() in ['a', 'n', 'v']
  and token not in stopwords.words('english')
  and token not in string.punctuation
  else wnl.lemmatize(token)
  for token, pos_tag in pos_tag(word_tokenize(title))
]

filtered_title = ' '.join(filtered_title_tokens)
return filtered_title
```

Listing 2: Pre-processing of title

4.2 The Bag-of-Words (BoW) Model

The vector-space model, which is based on BoW, is a mainstream model for solving IR problems. Each document d_i in corpus D is represented as a bag-of-words; in other words, each document d_i is represented as a multiset of terms that occur in this document. All these distinct terms in corpus D define an N-dimensional vector space, where N means the number of distinct terms. Each document d_i is represented as a vector $\boldsymbol{d_i}$ in the vector space. The query is also represented as a vector $\boldsymbol{q}$ in the same space. By measuring the angle θ between the two vectors $\boldsymbol{d_i}$ and $\boldsymbol{q}$ it can be determined how similar document d_i is to this query q. The smaller the angle θ, the more similar they are.

4.2.1 Creating a “reviewer profile”

Each “Reviewer profile” p_i is represented as a list of key phrases of document d_i in descending order according to its TF-IDF weight. Hence, we convert the problem of building the “reviewer profile” into a problem of extracting key phrases. The first phrase is able to represent the reviewer best, while the last one plays the least important role. A basic approach for extracting key phrases is the TF-IDF model. The workflow for the extraction of key phrases is shown in Figure 6.

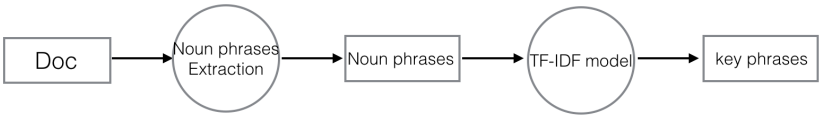

Figure 6: Workflow for the extraction of key phrases

- **Step 1: Extracting noun phrases**
  Generally, the key phrase is a noun phrase which consist of a key noun word and maybe along with some adjectives words to modify it, so we directly extract all the noun phrases from document d_i. Then we divide the document into chunks using the grammar shown below to extract all the noun phrases:

```
1  grammar = "NP: {<JJ>* <NN.*>+}"
```

- **Step 2: Ranking noun phrases**
  After the noun phrases have been extracted, they should be sorted in descending order according to their TF-IDF score. Here , the gensim package of Python is used. The relative implementation method is shown below.

  – **Code explanation**:
  Gensim package of Python is mainly designed for Machine Learning model. It includes TF-IDF model as well. Hence, we can use it to implement the function of extracting key phrases. Based on a list of documents $[d_1, d_2, ..., d_n]$, where d_i is the document of reviewer r_i, to train tf-idf model, so that the tf-idf score of each phrase is obtained. According to the tf-idf score to determine which noun phrases are more important, those important phrases prove to be key phrases.

– **Code implementation**:

```
def score_keyphrases_by_tfidf(documents, candidates='chunks'):
  """
  use tf-idf method to score key phrases
  :param 1 documents(List of document): contains all reviewers' documents
  :param 2 candidates: either chunks(extracting key phrases) or words(extracting keyword)

  :return 1 corpus_tfidf: a corpus contains all the unique noun phrases in the texts, and along with tf-idf score
  :return 2 dictionary: {noun phrase id: noun phrase}
  """
  import gensim
  # extract all the noun phrases from each text in texts which are regarded as candidate key phrases
  boc_texts = [extract_candidate_chunks(text) for text : texts]

  # make gensim dictionary and corpus
  dictionary = gensim.corpora.Dictionary(boc_texts)
  corpus = [dictionary.doc2bow(boc_text) for boc_text in boc_texts]

  # based on the corpus to train the tf-idf model
  tfidf_model = gensim.models.TfidfModel(corpus)

  # get the tf-idf score of each unique term in the corpus
  tfidf_score = tfidf_model[corpus]  # tfidf_score: a list of 2-tuple (id, tf-idf score)
return tfidf_score, dictionary
```

Listing 3: Creating a "reviewer profile"

– **Results**:
Figure 7 illustrate a example of one reviewer's profile. All noun phrases are ranked with a descending order in regard to TF-IDF score.

```
Key phrase                                    TF-IDF
-----------------------------------------------------
preference learning.....................0.354
rule learning...........................0.315
label ranking...........................0.157
comparison..............................0.118
efficient voting prediction.............0.118
evaluation..............................0.118
label preferences.......................0.118
large-scale problems....................0.118
legal domain............................0.118
multi-label classification..............0.118
classification..........................0.118
machine learning........................0.087
costs...................................0.079
coverage................................0.079
decision list...........................0.078
decision lists..........................0.078
decision tree...........................0.078
decision trees..........................0.078
discovery science.......................0.078
dynamic reduction.......................0.078
event-based clustering..................0.077
```

Figure 7: Example of a "reviewer profile"

4.2.2 Matching abstract to reviewers

Matching a given abstract to relevant reviewers is an important part of the RMS. This task can be converted into an IR task, i.e., finding the most relevant information for the given query. A widely applied approach to IR is the vector-space model. In our case, each reviewer r_i has their own document d_i, The corpus D is composed of all the reviewers' documents $[d_1, d_2, ..., d_n]$. Then the vector-space model is applied to this corpus to match the given abstract to the most relevant reviewer.

- **Step 1: Creating tf-idf model**
  Based on all the reviewers' documents $[d_1, d_2, ..., d_n]$, a corpus D is created using the TF-IDF model in order to accurately match abstracts to reviewers, the process of implementation is the same as the step 2 of 4.2.1.

  Table 2 illustrate a simple example of a vector-space model. Each reviewer is represented by a sparse vector and each term in the vector is weighted by its TF-IDF score. Significantly, Reviewer 1 has more bias to "Decision tree", Reviewer 2 have more research about the topic of "computer vision", while Reviewer 3 is more good at "Rule learning".

Reviewer \ Term	Rule learning	Decision tree	Computer vision
Reviewer 1	0.17	0.21	0.01
Reviewer 2	0.23	0.05	0.25
Reviewer 3	0.19	0.09	0.13

Table 2: An example of a TF-IDF model

- **Step 2: Matching abstract to reviewers**
  So far we have constructed a TF-IDF model for the corpus making from reviewers' profiles. So in order to find out the most appropriate reviewers for a given abstract, firstly the query also should be converted to a sparse vector in the same vector space and calculate TF-IDF score of each unique term, then comparing with the vector of each reviewer using cosine similarity algorithm to rank reviewers matching this abstract.

  - **Code explanation**:
    Gensim package provide a class called "similarities" that is able to be used to calculate the similarity between two texts. Hier using the trained TF-IDF model to get the tf-idf score of each term in the query. Based on this, it can be compared with the reviewers' profiles to get the similarities.
  - **Code implementation**:

```
from gensim import similarities

# create a similarity object in the package of gensim
similarity = similarities.Similarity('Similarity-tfidf-index',
    corpus_tfidf, num_features=len(dictionary))
query = "Input a query"

#extract Noun Phrases from this query
candidate_phrases = extract_candidate_chunks(query)

# represent the query as a sparse vector based on our trained
   corpus
vec_bow = dictionary.doc2bow(candidate_phrases)

# calculate tf-idf score for each term in query
query_tfidf = tfidf_model[vec_bow]

# get all the similarity score comparing with all reviewers
similarity[query_tfidf]
```

– **Results**:
Figure 8 show a example for the result. Left side show a submitted abstract, and the right side (reviewer name, similarity score) show the top 12 reviewers matching the abstract most.

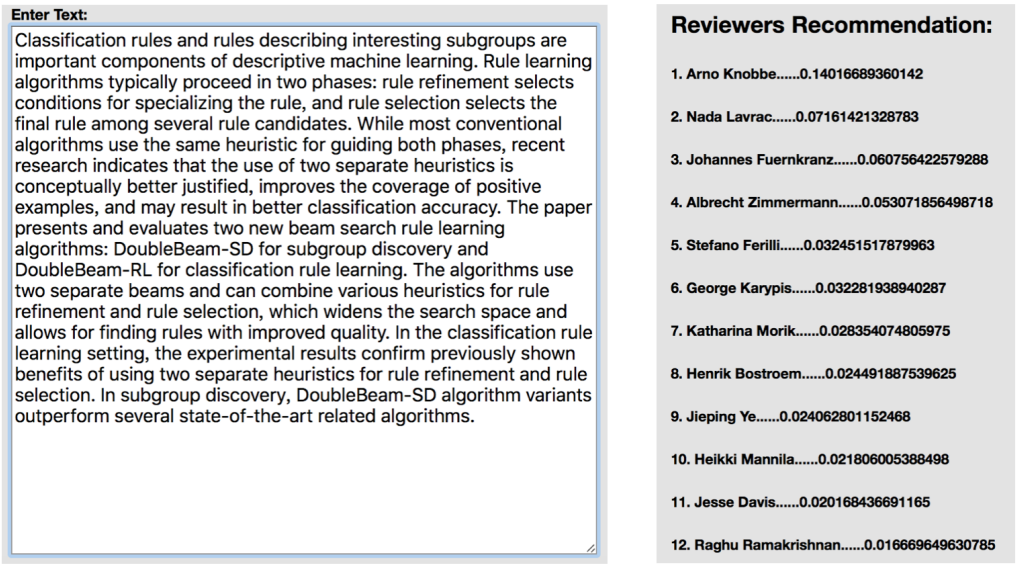

Figure 8: Reviewer matching example

4.3 Word Embeddings Model for Implementing IR

Obviously the vector-space model only considers the static information of the documents without considering any semantic relationships between terms. This is likely to cause the problem that no matching reviewer might be found for a given abstract if none of terms in the abstract is a constituent of the vector space. But our demand is that, in spite of the number of documents is not very large, the RMS can still be used as long as there are reviewers who research analogous topics.

A word embeddings model (e.g., word2vec) is a good approach for overcoming this shortcoming. In such a model, each word is represented as a vector; accordingly, each document is represented as a bag-of-vectors rather than a bag-of-words. An important advantage of a word embeddings model is that it consider the semantic relations between terms. The more similar the two words are, the shorter the distance between two vectors is. In terms of a method for measuring the similarity between two documents, the most basic approach is to use an averaged vector **d** to represent a document and an averaged vector **a** to represent the given abstract, then calculate the Euclidean distance between **a** and **d**. Obviously this approach only scales for those documents with a small number of terms. If, however, a document consists of hundreds of terms, an averaged vector cannot perfectly describe this document because it will likely consist of several topics. So in this thesis, the Word Mover's Distance [4] is used to measure the distance between two documents instead of creating an averaged vector, and to find the minimal accumulated distance when transferring each term of one document to the closest term in the other document. This approach is considered to make most sense for our case. In the RMS, noun

phrases are extracted from both a given abstract q and a reviewer document d_i as the profile. Hence, the "paper profile" is first compared with each "reviewer profile", and then these reviewers are ranked. The smaller the distance, the more similarity exists between the abstract and the reviewer.

In terms of creating a "profile", a vector-space model extracts key phrases using the TF-IDF model to represent a "reviewer profile". A word embeddings model, on the other hand, directly extracts the noun phrases as the "reviewer profile" because here we only consider how far "transfering" the abstract to each "reviewer profile".

In the following, the process for creating a "reviewer profile" and matching an abstract to reviewers will be described.

4.3.1 Creating a directory and a file for each reviewer

Before the implementation of information retrieval using a word embeddings model, four file directories (**Documents**, **Profiles**, **Profiles_Vector** and **Research_Interests**) are created.

- **Documents Directory**:
  We create a document file for each reviewer that is stored in the Documents Directory. Each document file d_i includes the titles of all the papers published by a reviewer. So the number of files in the directory equals the number of reviewers. These documents are used to train the word2vec model.
- **Profiles Directory**:
  Each reviewer has a profile file that is kept in the Profiles Directory. Accordingly, there are n files in the Profiles Directory.
- **Profiles_Vector Directory**:
  In the Profiles Directory, we have already created a profile for each reviewer. In the Profiles_Vector Directory, each profile is represented as a set of vectors, and each noun phrase is represented as a vector (word2vec). Therefore, each reviewer has a set of vectors used to describe their profile, which are stored in Profiles_Vector Directory.

Figure 9 illustrate the instance of file directory structure.

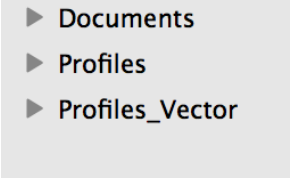

Figure 9: File directory structure

4.3.2 Creating the document and profile file for each reviewer

The document file stores the titles of the papers published by a reviewer in order to train the word2vec model, whereas the profile file contains the profile information for each reviewer.

- Before creating the document file, we need to perform pre-processing for our titles data in the same way as in the bag-of-words model. How to crawl the titles in the DBLP system was already described in 4.1.2, so it is not repeated here.
- In terms of creating a "profile", a vector-space model extracts key phrases using the TF-IDF model from the reviewer document d_i to represent the "reviewer profile". A word embeddings model, on the other hand, directly extracts the noun phrases from d_i as the "paper profile" because here we only consider the Word Mover's Distance between the "paper profile" and the "reviewer profile" to compare how similar the both "profile" are, as for which noun phrases in "reviewer profile" should get a high weight makes no sense for this model.

4.3.3 Training the word2vec model

We have already created a document file for each reviewer r_i, where each line represents a title. Now the document files of all reviewers are used as training data. The gensim package contains the word2vec model, so the word2vec can be trained more effectively. The code below presents the implementation of training the model.

- **Code explanation**:
  The gensim package contains the word2vec model, which is based on the theory of Mikolov namely via skip-gram and CBOW models and using softmax or negative sampling to train the w2v model.

  In the code below:
  - **size** – dimensionality of the feature vectors. Here we trained a 200-dimensional model.
  - **window** – the maximum distance between the current and the predicted word within a sentence.
  - **min_count** – ignores all words with a total frequency lower than this. Because after pre-processing all the terms in the document file refer to critical information, no term is ignored here.

- **Code implementation**:

```
def train_data(train_data):
"""
train w2v model
:param train_data(list of list): tokennized sentence, e.g. [['first', 'sentence'], ['second', 'sentence'], ...]
"""

from gensim.models import Word2Vec

# train a model 200-dimension
model = Word2Vec(train_data, size=200, window=5, min_count=1, workers=4)
```

- **Results**:
  Example: The word “data” can be represented as a 200-dimensional vector.

$$data : 0.683588, 0.282057, -0.642279, -0.645473, 0.043452, -0.338720...$$

  Tables 3 and 4 show the test for our trained word2vec model that found the most similar words for the words “neural” and “rule”, respectively.

Word	*Similarity*
prediction	0.99992311
interaction	0.99991936
Bayesian	0.99991751
predict	0.99990982
improve	0.99990875
adaptive	0.99990547

Table 3: The most similar words with the word "neural"

Word	*Similarity*
mine	0.99985170
discover	0.99983281
spatial	0.99983168
pattern	0.99982178
algorithm	0.99981844

Table 4: The most similar words with the word "rule"

The results show that the trained w2v model worked well. It included 8078 unique words; each word could be represented as a 200-dimensional vector. If administrator A adds more reviewers to the system through the web interface (clicking on the button: **"UPDATE"**), *A* can easily train a new word2vec model based on the document data of both "old" and "new" reviewers. The trained model is saved in a file (./python_code/w2v_train_model/word2vec.txt) so that it can be loaded quickly when necessary. The function provided by the Word2Vec class of the gensim package is used to save the model, as shown below:

```
from gensim.models import Word2Vec
model.wv.save_word2vec_format(path, binary=False)
```

4.3.4 Obtaining the vector representation of a profile

The word-form profile has been saved in the Profiles Directory and the word2vec model is also available. In order to measure the similarity between a reviewer and a submitted abstract, we need to measure their distance. Hence, the profiles need to be represented as vectors. The detailed implementation is shown below:

- **Code explanation**:
  As each reviewer's profile is made up of a series of noun phrases, using average vector approach to get the vector of each phrase. For example, $phrase_1$ include three words, namely $word_1$, $word_2$, $word_3$ respectively, their w2v are $w2v_1$, $w2v_2$, $w2v_3$, the average vector of $w2v_1$, $w2v_2$, $w2v_3$ are adopted as a vector of $phrase_1$. Hier Numpy package of Python is used to calculate the average vector; The class "KeyedVectors" is used to load the already trained w2v model.

- **Code implementation**:

```
def w2v_transfer(profile):
"""
represent each phrase in profile as w2v
:param profile(list):  a list of phrases [phrase1, phrase2,...]
:return: profile_w2v(list),  a list of vectors that represent
   profile
"""

import numpy as np
from gensim.models import KeyedVectors

# load the already trained w2v model
w2v_model = KeyedVectors.load_word2vec_format('../python_code/
   w2v_train_model/word2vec.txt', binary=False)
profile_w2v = []  #transfer each phrase in profile to a
   corresponding vector

for p in phrases:
  phrase_wv = np.zeros(200)   # the vector is 200 dimension
  count = 0

  # get the w2v for a phrase, using average vector approach
  for word in p.split(' '):
    if word in w2v_model.vocab:
    count += 1
    phrase_wv += w2v_model[word]
  if(count != 0):
    phrase_wv /= count

  profile_w2v.append(phrase_wv)
return profile_w2v
```

After the vector form of all the profiles has been obtained, they are saved to the Profile_Vector Directory.

4.3.5 Representing a query as a vector

In our case, a query is normally an abstract of a paper. In order to measure the distance to the profile of each reviewer to find the "nearest" reviewer, the query should also be represented in the form of a word2vec. This process is the same as when we transferred a reviewer's document d_i to a vector format.

4.3.6 Matching to reviewers for the given abstract

If the query is a news article, it usually contains only one topic or a few topics. In such a case, using an averaged vector or a set of clustered mean vectors to represent the query makes sense.

However, in our case, the query is an abstract of an academic paper, so the number of topics cannot be easily estimated because each abstract has a different number of topics. Hence, we cannot simply assume the specific number of topics in an abstract, meaning we cannot, e.g., use a K-means algorithm to cluster the abstract into, let's say five clusters, and use these five mean vectors to represent the abstract. Therefore, in this thesis we use the Word Mover's Distance to measure the similarity between an abstract and the profile of a reviewer as a better choice.

The idea behind this is to extract a set of noun phrases as "profiles" of the given abstract, using the trained word2vec model saved in the file (./python_code/w2v_train_model/word2vec.txt) to acquire the vector representation of each noun phrase. For each reviewer, the distance between the "abstract profile" and the "reviewer profile" is measured with the Word Mover's Distance. The smaller the distance, the more similarity exists between this reviewer and the abstract. The detailed implementation is shown below. After the distance between the abstract and each reviewer was calculated the decision can be made who is the most appropriate reviewer for the abstract.

- **Code explanation for the process of reviewer recommandation**:
  The submitted abstract need to be processed like the profile of a reviewer, namely using a list of vectors to represent it, where each vector is a vector representation of a key phrase. Because the vector representation of the reviewers' profiles are stored in the directory of **Profiles_Vector**, the "pickle" package is used to load the data. Then the distance between the abstract with each reviewer is calculated, so that the appropriate reviewers for the abstract is able to be selected.

- **Code implementation**:

```
def recommand_person(query):
  """
  sort all the reviewers according to the similarities with the
   query
  :param:query(string), a given abstract
  :return: sorted_dists(list), list of distance between query
   with each reviewer
  """
  import pickle
  from os import listdir
  from os.path import isfile, join

  # step 1 : get vector representation for query
  query2vec = query2vec(query)

  # step 2:  read Profiles_Vector directory, for each reviewer
   profile file is stored as a dictionary structure, like: {phase
    1: vector 1, phase 2: vector 2, ...}

  # load all the reviewer profile data.
  dic_profiles = {} #dic of dic, {file_name_11:profile2vec_1{
   phrase: vector}, file_name_2:profile2vec_2{phrase: vector},
   ...}
  base_path = './Profiles_Vector/'
  files = [f for f in listdir(base_path) if isfile(join(base_path
   , f)) ]

  for file_name in files:   #file_name: discriminate the
   different reviewers
  f_path = join(base_path, file_name)
    with open(f_path, 'rb') as f:
      dic_profiles[file_name] = pickle.load(f)

  # step 3:  get distance between query to each profile
  dists = list()
  for file_name, profile2vec in dic_profiles.items():
    # get the distance between the query and a profile of
   reviewer
    distance = calculate_distance(query_vectors, profile2vec)
    dists.append([file_name, distance])

  # step 4:  sorting the result list
  sorted_res = sorted(dists, key=lambda x: x[1])
  return sorted_dists
```

- **Code explanation for the process of distance calculating**:
  Find the nearest noun phrases in each reviewer's profile to every noun phrases in abstract, and calculate the distance between them, at last sum it up to get the distance between abstract and reviewer's profile. The Numpy package, which implement the function of scientific computing, is used. Besides, the euclidean_distance function in the class cluster of package NLTK are used to obtain the distance between two vectors.

- **Code implementation**:

```
def calculate_distance(query2vec, profile2vec):
  """
  Calculating the accumulated  Word Mover's Distance(WMD) between
    query and reviewer's profile.

  :param query2vec(list): vector representation for the query
  :param profile2vec(list): vector representation for the profile
  :return: distance(float) the accumulated  Word Mover's Distance
    between query and reviewer's profile.
  """
  import sys
  import numpy as np
  from nltk.cluster import euclidean_distance

  wmds = [] #store the Word Mover's Distance of each term in the
   query

  # calculating the Word Mover's Distance of each term in the
   query
  for qv in query2vec:
    min_dist = sys.float_info.max
    for pv in profile2vec:
      wmd = euclidean_distance(qv, pv)
      if wmd < min_dist:
        min_dist = wmd
    dists.append(min_dist)

  distance = np.sum(np.array(wmds))
  return distance
```

- **Results**: Figure 10 shows an example of how the submitted abstract is matched to a set of reviewers ranked according to similarity (the numbers on the right side represent the distance between the abstract and each reviewer.).

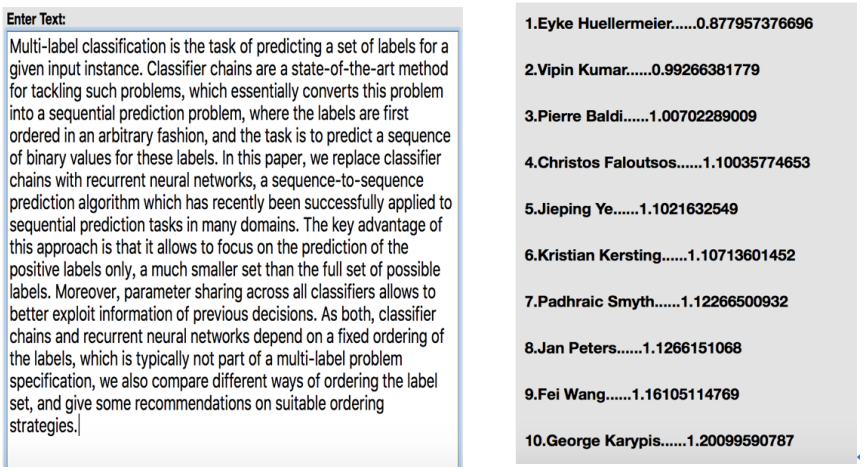

Figure 10: Example of matching a query to reviewers

5 Evaluation

5.1 Evaluation Results

Because the RMS is a web interface, accurately finding the most appropriate reviewers for a given abstract is a very crucial issue. In this thesis, a vector-space model and a word embeddings approach are applied. The evaluation is performed separately for the two models.

80 reviewers have already been added to the system; the list of reviewers can be found on the website:
`http://www.springer.com/computer/database+management+%26+informationretrieval/journal/10618/PSE?detailsPage=editorialBoard`

About 33 of these reviewers were randomly sampled for the evaluation, and two new abstracts are used for each of these reviewers. Finally, the rank of the author in the system will be reported, and a scoring approach will be used to decide which model performed better. The results are categorized into ten groups.

Category1: If the reviewer's rank is between 0% and 10%, namely 1 and 8, 1 point is assigned.
Category2: If the reviewer's rank is between 10% and 20%, namely 9 and 16, 2 point is assigned.
Category3: If the reviewer's rank is between 20% and 30%, namely 17 and 24, 3 point is assigned.
Category4: If the reviewer's rank is between 30% and 40%, namely 25 and 32, 4 point is assigned.
Category5: If the reviewer's rank is between 40% and 50%, namely 33 and 40, 5 point is assigned.
Category6: If the reviewer's rank is between 50% and 60%, namely 41 and 48, 6 point is assigned.
Category1: If the reviewer's rank is between 60% and 70%, namely 49 and 56, 7 point is assigned.
Category8: If the reviewer's rank is between 70% and 80%, namely 57 and 64, 8 point is assigned.
Category9: If the reviewer's rank is between 80% and 90%, namely 65 and 72, 9 point is assigned.
Category10: If the reviewer's rank is between 90% and 100%, namely 72 and 80, 10 point is assigned.

The points of each reviewer are then summed up for the two approaches, respectively. The lower the number of points that an approach gets, the more suitable this approach is for the RMS. The source of the evaluation data come from each person's DBLP page, as showed in table 6, the entries represent the paper ID which is denoted in each person's DBLP page. Table 5 shows the evaluation results.

VSM: Vector-space model
WE: Word embeddings
The number of reviewers are evaluated: 33

Reviewer	Abstract 1				Abstract2			
	VSM		WE		VSM		WE	
	rank	score	rank	score	rank	score	rank	score
Johannes Fürnkranz	3	1	21	3	1	1	32	4
Rakesh Agrawal	1	1	1	1	5	1	7	1
Christos Faloutsos	4	1	1	1	1	1	11	2
Heikki Mannila	31	4	24	3	1	1	1	1
Raghu Ramakrishnan	5	1	3	1	7	1	7	1
Padhraic Smyth	22	3	13	2	4	1	4	1
Geoff Webb	1	1	14	2	6	1	25	4
Pierre Baldi	1	1	2	1	1	1	1	1
Albert Bifet	1	1	23	3	1	1	49	7
Hendrik Blockeel	1	1	5	1	1	1	5	1
Toon Calders	13	2	34	5	2	1	38	5
Ian Davidson	1	1	1	1	2	1	23	3
Aristides Gionis	1	1	2	1	1	1	13	2
Bart Goethals	25	4	49	7	12	2	42	6
George Karypis	5	1	24	3	4	1	4	1
Eamonn Keogh	2	1	2	1	3	1	1	1
Kristian Kersting	1	1	12	2	3	1	9	2
Donato Malerba	2	1	5	1	2	1	11	2
Pauli Miettinen	3	1	60	8	1	1	56	7
Panagiotis Papapetrou	1	1	4	1	7	1	56	6
Hanghang Tong	1	1	43	6	1	1	24	3
Jieping Ye	39	5	4	1	15	2	6	1
Anthony Bagnall	7	1	48	6	2	1	32	4
Henrik Bostroem	1	1	5	1	1	1	6	1
Wray Buntine	29	4	32	4	6	1	34	5
Michelangelo Ceci	3	1	16	2	2	1	35	5
Sanjay Chawla	24	3	20	3	9	2	10	2
Jesse Davis	1	1	60	8	2	1	14	2
Saso Dzeroski	5	1	1	1	2	1	3	1
Stefano Ferilli	3	1	2	1	1	1	16	2
Eibe Frank	4	1	38	5	1	1	36	5
Jingrui He	1	1	39	5	1	1	38	5
Eyke Huellermeier	6	1	4	1	3	1	1	1
Total		**51**		**92**		**36**		**95**
Average		**1.55**		**2.79**		**1.09**		**2.88**

Table 5: Evaluation results

Reviewer	Data Source	
	Abstract 1	Abstract 2
Johannes Fürnkranz	j35	i5
Rakesh Agrawal	j55	c165
Christos Faloutsos	j141	j139
Heikki Mannila	j54	c130
Raghu Ramakrishnan	c168	c160
Padhraic Smyth	c112	c107
Geoff Webb	j53	i10
Pierre Baldi	j117	j116
Albert Bifet	j19	j18
Hendrik Blockeel	i18	i17
Toon Calders	j21	c63
Ian Davidson	j18	j17
Aristides Gionis	j43	c123
Bart Goethals	j22	j21
George Karypis	c148	i2
Eamonn Keogh	j56	j55
Kristian Kersting	j39	i27
Donato Malerba	j54	j53
Pauli Miettinen	j9	c30
Panagiotis Papapetrou	j17	j16
Hanghang Tong	j42	c124
Jieping Ye	j86	i35
Anthony Bagnall	j19	j18
Henrik Bostroem	j18	j17
Wray Buntine	c63	i19
Michelangelo Ceci	j26	1p6
Sanjay Chawla	j22	j21
Jesse Davis	c53	c52
Saso Dzeroski	j76	j73
Stefano Ferilli	j24	c164
Eibe Frank	j18	c61
Jingrui He	j11	j10
Eyke Huellermeier	j97	j96

Table 6: Data source of evaluation results

Score of the vector-space model : $\frac{1.55+1.09}{2} = 1.32$
Score of the word embeddings model : $\frac{2.79+2.88}{2} = 2.84$

Hence, we can draw the conclusion that the vector-space model has a nearly perfect performance for the RMS and performed better than the word embeddings model. Therefore, we still regard the vector-space model as the perfect approach for the RMS. In section 5.2, an analysis of the evaluation results will be performed.

5.2 Evaluation Analysis

As the evaluation results show, the vector-space model performed really well for the RMS and was better than the word embeddings approach. In this section, some examples will be presented to analyze the reasons for these results.

5.2.1 Evaluation analysis for the vector-space model

In order to analyze the implementation results of the vector-space model, an example will be used to illustrate its implementation process. Query 1 is the abstract of the paper [11] from the author Prof. Dr. Johannes Fürnkranz.

Query 1: “Classification rules and rules describing interesting subgroups are important components of descriptive machine learning. Rule learning algorithms typically proceed in two phases: rule refinement selects conditions for specializing the rule, and rule selection selects the final rule among several rule candidates. While most conventional algorithms use the same heuristic for guiding both phases, recent research indicates that the use of two separate heuristics is conceptually better justified, improves the coverage of positive examples, and may result in better classification accuracy. The paper presents and evaluates two new beam search rule learning algorithms: DoubleBeam-SD for subgroup discovery and DoubleBeam-RL for classification rule learning. The algorithms use two separate beams and can combine various heuristics for rule refinement and rule selection, which widens the search space and allows for finding rules with improved quality. In the classification rule learning setting, the experimental results confirm previously shown benefits of using two separate heuristics for rule refinement and rule selection. In subgroup discovery, DoubleBeam-SD algorithm variants outperform several state-of-the-art related algorithms”.

Three representative reviewers were selected for the analysis. After the abstract was submitted, Arno Knobbe came out as the best matched reviewer, ranking first; this abstract’s author, Prof. Johannes Fürnkranz, ranked third, while Rakesh Agrawal’s ranking was relatively low regarding this abstract, being number 25. In Table 7, the TF-IDF weights of the terms in the query for the three different reviewers are compared; the column Query shows the terms’ TF-IDF weight for this query.

In table 7:
N: The corresponding term is not found in TF-IDF corpus.

query terms	TF-IDF Weight			
	query	Johannes Fürnkranz	Arno Knobbe	Rakesh Agrawal
phase	0.54	N	N	N
rule	0.43	0.037	N	N
classification rule	0.41	0.035	N	N
subgroup discovery	0.37	0.032	0.3765	N
experimental result	0.27	N	N	N
coverage	0.23	0.078	N	N
positive example	0.23	N	N	N
set	0.13	N	N	0.02
algorithm	0.11	0.01	N	0.025

Table 7: TF-IDF weights of terms in Query 1

It is beyond any doubt that Rakesh Agrawal should rank lower than Prof. Johannes Fürnkranz and Arno Knobbe since only two relatively unimportant terms for the abstract,"set", and "algorithm", occur in his profile as well; all the other important terms are not found.

However, for Prof. Johannes Fürnkranz and Arno Knobbe, apparently Prof. Johannes Fürnkranz matches the abstract best because most of the terms occurring in the abstract can be found in Prof. Johannes Fürnkranz's profile, while Arno Knobbe's profile only contains the term "subgroup discovery". The reason why the system ranked Arno Knobbe higher than Prof. Johannes Fürnkranz, however, is that although only one term ("subgroup discovery") exists in his profile, this term is really important, with a weight 0.3765. Although most of the terms exist in the profile of Prof. Johannes Fürnkranz, these terms do not have a high weight for his profile.

$$0.37 \cdot 0.3765 = 0.14 > 0.43 \cdot 0.037 + 0.41 \cdot 0.035 + 0.37 \cdot 0.032 + 0.23 \cdot 0.078 + 0.11 \cdot 0.01 = 0.06$$

In conclusion, it can be observed for the vector-space model that, although sometimes the first reviewer recommended by the system is not the best matching reviewer for the abstract, the best matching reviewer for the abstract is normally ranked in the top 10%, i.e., relatively high.

5.2.2 Evaluation analysis for the word embeddings model

The same evaluation analysis process as for the vector-space model was also used for the word embeddings model, and query 1 again serves as an example. Given this abstract, the RMS ranked Bing Liu in first place, with a distance of 1.06: The abstract's author, Prof. Johannes Fürnkranz, rank 22nd, with a distance of 1.49. For the comparison, the least relative reviewer for this abstract, Charu Aggarwal, was selected, who ranked 80th, with a distance of 20.03. The idea of calculating the similarity was used for analyzing the results of the word embeddings approach. As a recap, the idea behind the use of the word embeddings approach is to calculate the accumulated Word Mover's Distance between the profile of the abstract and the profile of the reviewers; in other words, the closest terms in the profile of a reviewer are identified and then the sum of

all the distances is the distance between the abstract and the reviewer. In Table 8, we present the similarity calculation for Prof. Johannes Fürnkranz, Bing Li, and Charu Aggarwal, respectively.

Query terms	The nearest terms in different reviewer's profile, distance			
	Best match	Bing Liu	Johannes Fürnkranz	Jan Ramon
rule refinement	rule chain, 0.02	semi-structured data, 0.05	text source, 0.04	subgraph pattern, 0.043
important component	warming scenario, 0.016	unique key, 0.02	error-correcting output code, 0.108	validation, 0.027
search rule	exploratory search, 0.026	internet search, 0.032	rule learning, 0.036	subgraph mining, 0.047
classification accuracy	multi-target classification, 0.025	generation use, 0.037	global modeling use, 0.035	representation, 0.058
improve quality	semantic similarity, 0.021	semi-structured, data, 0.037	binary decomposition method, 0.031	comparison, 0.0406
subgroup discovery	subgroup discovery, 0.0	domain knowledge, 0.08	subgroup discovery, 0.0	reinforcemen learn, 0.15
set	pruned set, 0.0	set, 0.0	regression, 0.083	distribute algorithm, 0.03
phase	phase, 0.0	phrase, 0.031	event-related microposts ,0.026	malicious adversar, 0.03
separate heuristic	conjunctive normal form, 0.019	progressive sampling, 0.021	error-correcting output code, 0.103	validation, 0.026
classification rule	classification rule, 0.0	classification rule, 0.0	classification rule, 0.0	framework, 0.093
search space	efficient online evaluation, 0.025	multiple set, 0.036	event sequence, 0.03	dependency network, 0.061
rule	rule, 0.0	discovered rule, 0.0	rule, 0.0	tree search, 0.09
positive example	positive example, 0.0	extract resource term, 0.03	predict train failure, 0.036	relevancy zone, 0.0387
experimental result	experimental result, 0.0	chemical key algorithm, 0.023	policy iteration 7 algorithm, 0.02	relation, 0.03
algorithm	algorithm, 0.0	localized algorithm, 0.0	algorithm, 0.0	learn, 0.183
rule selection	rule selection, technique, 0.022	multiple set, 0.05	probabilistic rule, 0.035	active learning, 0.069
coverage	coverage, 0.0	interval merger, 0.028	coverage, 0.0	pca, 0.030
classification rule learn	learn classification rule, 0.0	search, 0.071	search, 0.071	dynamic data analysis, 0.062
machine learning	mine interest knowledge, 0.069	mine interest knowledge, 0.069	knowledge retrieval system, 0.060	subgraph mining, 0.082

Table 8: Word embeddings analysis for query and reviewers

From this comparison table, we can identify the reasons why the system did not perform very well with this approach and decided to recommend the abstract to Bing Liu rather than to Prof. Johannes Fürnkranz.

- Because for word embeddings approach, we find the nearest term for each term occurring in the query, the result has a lot of noise. A not important term such as “important component” matches “unique key” in Bing Liu’s profile as its nearest term, with a distance of 0.02; On the other hand, it matches the term “error-correcting output code” in Prof. Johannes Fürnkranz’s profile as its closest term, with a distance of 0.108. Hence, these noise terms may have a great influence on the system’s performance. In comparison with the vector-space model, more terms with a high TF-IDF weight are considered; these terms are normally meaningful.
- The document collection used to train the word embeddings model includes 80 reviewers’ documents, and although it includes more than 8000 terms, the training data is still not enough because for the term “rule refinement”, if we look for the closest term for all reviewers’ profiles, it matches the term “rule chain”, with a distance of 0.02, but the closest term in the profiles of Bing Liu, Prof. Johannes Fürnkranz, and Jan Ramon seems to make no sense. The reason for this result might be that the amount of training data was not enough.

In conclusion, for RMS, the task of IR using vector space model perform better than using word embeddings approach. Word embeddings approach is able to cover the shortage of the vector space model that not taking the semantic relation between two terms, but it still bring in a lot of noise to affect the performance of the system. Nevertheless, I think there is space to improvement for word embeddings approach.

6 Usage of RMS and Presentation of the Results

As an initial model of the RMS, an Apache Web Server was used. The URL are:
`http://localhost/rms_vsm/index.php` for vector-space model;
`http://localhost/rms_we/index.php` for word embeddings model;
A MySQL database was used to store the information about the reviewers.

6.1 Database Design

In the RMS, a MySQL database was used as a back-end database. The name of the table is "reviewers". A reviewer ID in the form of a single primary key is used to identify the unique reviewer. The reason for not using the reviewer's name as the primary key was that there might be duplicate names for multiple reviewers. The table contains five columns: **id, name, dblp_url, affiliation, and file_name**, respectively, as illustrated in Figure 11.

id	name	dblp_url	affiliation	file_name
75	Rakesh Agrawal	http://dblp.uni-trier.de/pers/hd/a/Agrawal_0001:Ra...	Data Insights Laboratories	Rakesh Agrawal5ae1f0a1c0ea65.62471625.txt
76	Christos Faloutsos	http://dblp.uni-trier.de/pers/hd/f/Faloutsos:Chris...	Carnegie Mellon University	Christos Faloutsos5ae1f0d8ade9b0.02841687.txt
77	Heikki Mannila	http://dblp.uni-trier.de/pers/hd/m/Mannila:Heikki	Academy of Finland	Heikki Mannila5ae1f0fd843c38.73484245.txt
78	Raghu Ramakrishnan	http://dblp.uni-trier.de/pers/hd/r/Ramakrishnan:Ra...	University of Wisconsin	Raghu Ramakrishnan5ae1f11ce2aa00.58515064.txt
79	Padhraic Smyth	http://dblp.uni-trier.de/pers/hd/s/Smyth:Padhraic	University of California	Padhraic Smyth5ae1f136bb6d86.59698917.txt

Figure 11: Database example

dblp_url is the dblp page of the reviewer; **affiliation** is the work place of this reviewer. This was included in order to make it easier for an administrator who wants to add a reviewer to the RMS to select the desired reviewer if there are duplicate names in the DBLP system. An example of this will be illustrated later.

The detail SQL code is following:

```
CREATE TABLE reviewers (
id  int(11)  not null PRIMARY key AUTO_INCREMENT,
name  varchar(50)  not null,
dblp_url  varchar(100),
affiliation  varchar(50),
file_name  varchar(300)
);
```

Every reviewer has a document file for storing their profile. If reviewers with the same name occur in this system, this could easily cause duplicate file names for these reviewers. To avoid this, the name plus a series of random numbers are used as the file name of a reviewer. The code below shows how the database table was created.

6.2 Adding a Reviewer to the Systemm

The process of adding a reviewer to the RMS is divided into three steps.

- **Step 1: Enter reviewer's name.**
  Enter the reviewer's name and submit it to search for persons with this name in the DBLP database. Click on the button **Submit**. Let us take a person called Qiang Li as an example. Figure 12 shows the GUI for the first step.

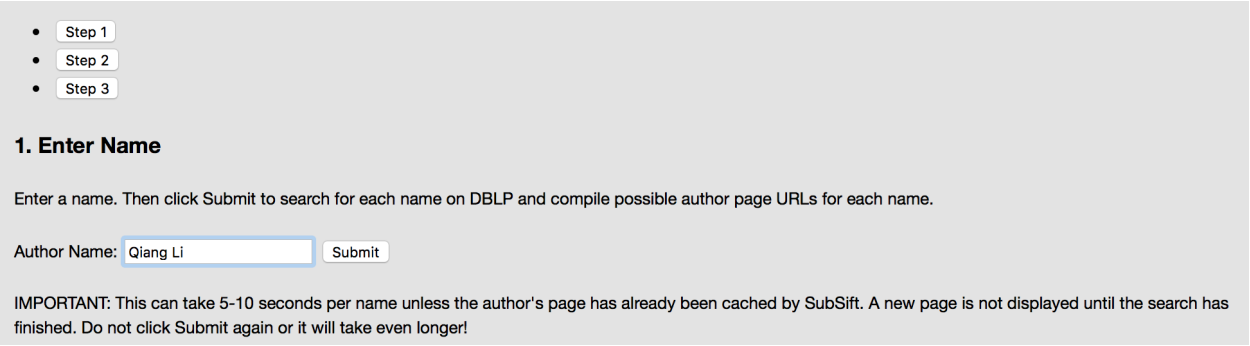

Figure 12: GUI for Step 1

- **Step 2: Disambiguate the reviewers with the same name.**
  After the name of the reviewer has been entered, the system will parse all persons with this name in the DBLP database. As the DBLP page provides the tag "Other persons with the same name", it is able to do this task. The output includes all the persons with this name and the affiliation of each of them to disambiguate the reviewers more easily, which is shown in Figure 13. Then the desired person can be selected directly and **'go!'** can be clicked.

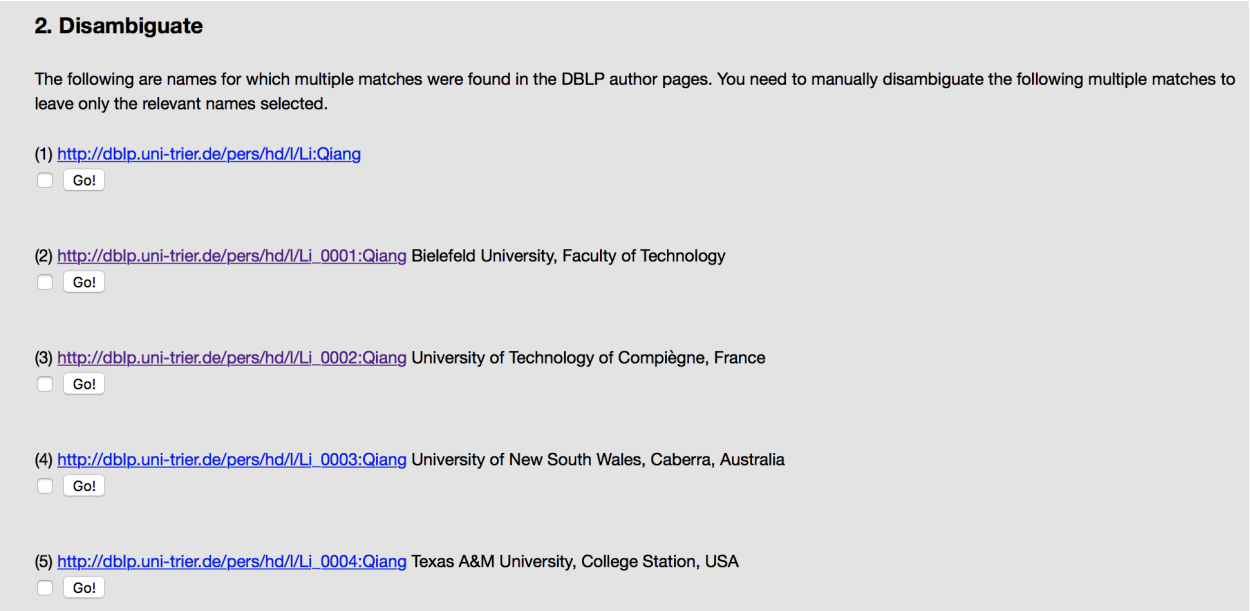

Figure 13: GUI for Step 2

- **Step 3: Adding the reviewer to the database.**
  In step 2, we selected the reviewer we want to add. In step 3, the system adds this reviewer to the database and creates a file name for the profile file of each reviewer. Besides this, in this step the basic information for this reviewer is done, e.g., the titles of all his or her

published papers are crawled to form the reviewer's document file, and the candidate key phrases will be extracted from the titles as a preparation to build the reviewer's profile.

6.3 Creating the Profile of New Added Reviewer

After the reviewer has been added in database, the document file d (titles of all published papers) has been created. Besides, the noun phrases of these titles are extracted which are regarded as candidate key phrases. After **UPDATE** Button is clicked, RMS will create the profile of the new added reviewer and update all the other reviewers' profiles automatically.

6.4 Checking the Information of a Reviewer

Check All Reviewers button enable users to check the information of all reviewers, including affiliation, DBLP Page and Profiles. Figure 14 show a example of the GUI for one reviewer.

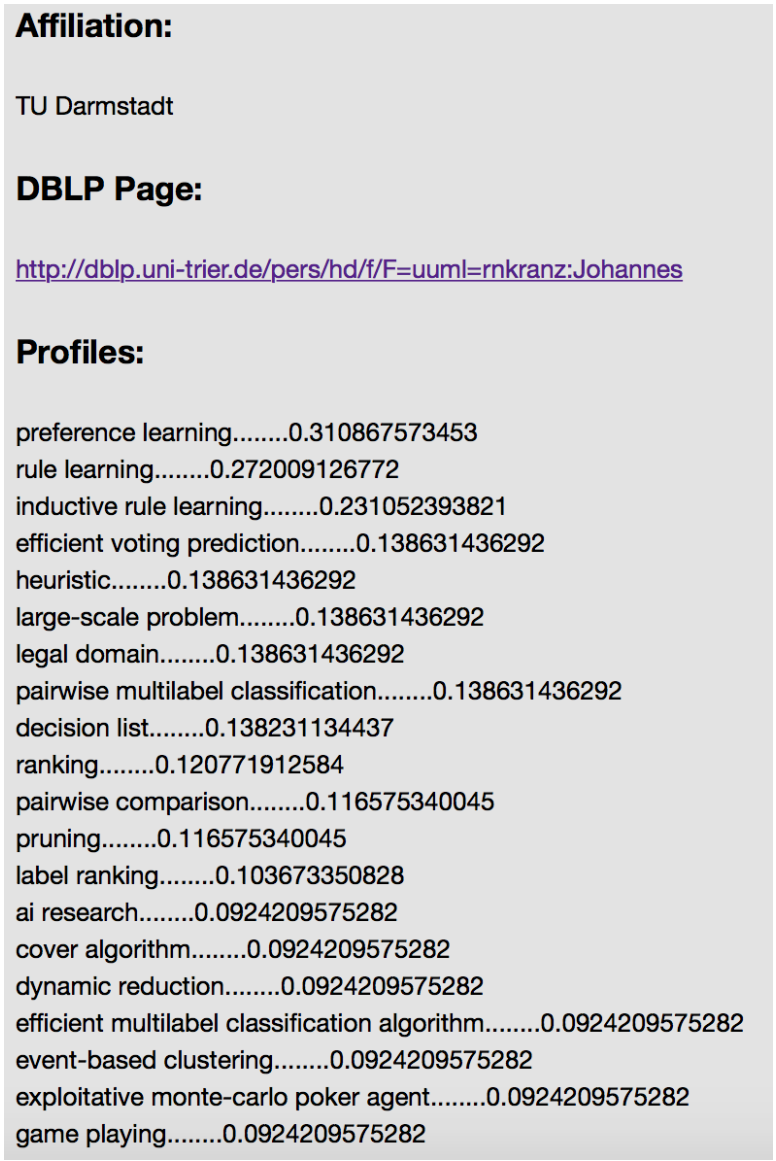

Figure 14: GUI for checking reviewer

6.5 Matching the Submitted Abstract to Reviewers

After clicking on the button **Reviewer mapping**, we get access to a reviewer matching page. Entering the abstract of a paper in the text box, then clicking on **"Submit"** will lead to the reviewers who match this submitted abstract being shown on the right side. These reviewers will be ranked according to the similarity score with the abstract. As an example, we enter the abstract from the paper [11]. Prof. Dr. Johannes Fürnkranz is the author of this paper. Figure 15 illustrate this process.

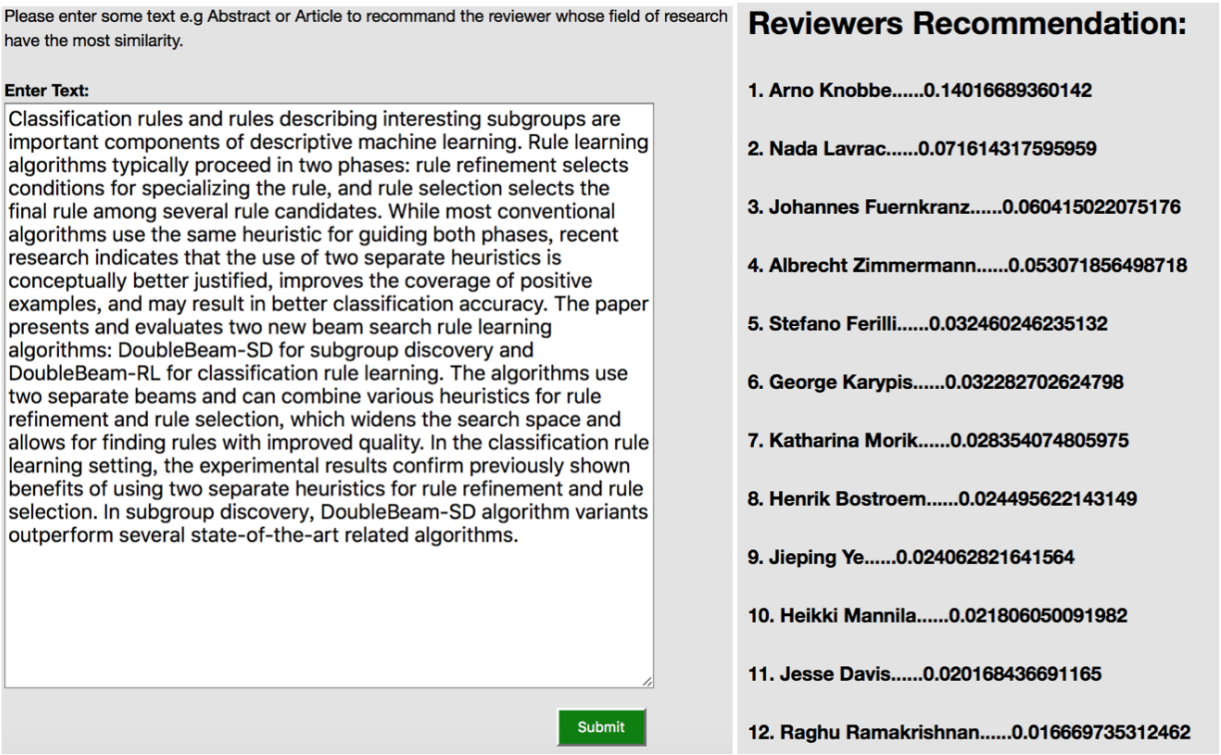

Figure 15: Example of submitted abstract and matched reviewers

7 Conclusions and Future Work

7.1 Conclusions

Reviewer assignment is a crucial but time-consuming task. Creating an automatic reviewer assignment system is appealing for many reasons, including improving the effectiveness and the quality of the assignment. Hence, this thesis introduced the newly developed web interface Reviewer Mapping System. One of its functions is to enable an administrator (A) to modify the set of reviewers (R). Hence, the RMS should allow A to add new reviewers to the system. Because of the possibility that several reviewers might have the same name, name disambiguation is a major issue. With the help of DBLP, the RMS provides a function that enables users to crawl all persons with the same name and see the affiliation of each of them. The RMS then allows the administrator to add a specific reviewer according to their affiliation. In order to solve the problem of reviewer assignment, knowing about the expertise and research interests of each reviewer plays an important role. The RMS extracts the titles of each reviewer's published papers as their character document. Key phrases extracted from this document to form the profile of the reviewer. Hence, during the process of adding a new reviewer, the RMS crawl the titles from this reviewer's DBLP page and then creates the profile. As is commonly done, the RMS also converts the reviewer assignment task into an IR task. The query is the abstract of the submitted paper, while the documents are the "reviewer profile". In this thesis, two models (bag-of-words model and word embeddings model) were used to solve the IR problem and compared to determine which is the better model for RMS. After the evaluation, the most common model, the bag-of-words model, achieved extremely good results. So we finally adopted the bag-of-words model to implement the reviewer assignment task.

7.2 Future Works

Although the bag-of-words model achieved great results, a non-trivial shortcoming of this model is that it does not consider the semantic relations among words. So we propose using the word embeddings model to overcome this shortcoming. In spite of the evaluation of the word embeddings model, which showed that it did not perform well, there still exists a lot of room for improvement.

- Idea 1: It should be possible to improve the approach of creating the "reviewer profile". In this thesis, we assumed that all the noun phrases of the reviewer's document are components of the "reviewer profile". But actually, some words or phrases is not able to reflect a research topic, which causes a problem for the word embeddings model. When using the Word Mover's Distance to measure the distance between the abstract and the "reviewer profile", those noun phrases that are not a research topic will cause great noise, which is not acceptable for us. So not every noun phrase in the "reviewer profile" can be regarded as an area of expertise. Because the RMS is aimed at users in the areas of computer science, the suggestion is hence for the administrator to list all the relevant research topics, then filter the "reviewer profile" to get exactly these research topics. This is expected to greatly improve the performance of the word embeddings model.
- Idea 2: Instead of using the Word Mover's Distance, another approach could be used to match an abstract to reviewers. In the Word Mover's Distance approach, for each "reviewer

profile" the closest word to each term in the "query" is found, then the accumulative distance between the "query" and each "reviewer profile" is calculated. The problem is that for a single "reviewer profile", the number of terms is limited, so usually the closest term in the "reviewer profile" does not semantically correspond to the term in the "query". For example, in Table 8, the closest term in Bing Liu's "reviewer profile" to the term "rule refinement" in the query is "semi-structured data". Obviously, the two terms do not have any semantic relation. So the distance between these two terms makes no sense to us. However, we notice in Table 7 that, if we enlarge the scope to all unique terms from all "reviewer profiles", the closest term for "rule refinement" is "rule chain". This is a good result. Hence, another approach could be used to match an abstract to reviewers: We denote all unique terms for all "reviewer profiles" as T. For each term in the "query", we find the closest term (t_r) in T; if t_r occurs in a "reviewer profile" , the corresponding reviewer scores 1 point. At the end, the reviewer with the highest score is considered the most appropriate reviewer for the submitted query.

- Idea 3: Some research has revealed that a supervised learning method can also be used to extract the "reviewer profile". The two features are the place of the first occurrence of the term and the TF-IDF weight of the term. The supervised learning method can achieve better performance than an unsupervised learning method (e.g., the TF-IDF model). In our case, because the query is a paper's abstract, these two features cannot reflect the whole paper. So after testing this approach, the result was not found to be any better than the TF-IDF model. If we use a whole paper as the query, this method is ideal, but the price is a decrease in the speed with which the result is obtained.

References

[1] Chumki Basu, Haym Hirsh, William W Cohen, and Craig Nevill-Manning. Recommending papers by mining the web. 1999.

[2] Seth Hettich and Michael J Pazzani. Mining for proposal reviewers: lessons learned at the national science foundation. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 862–871. ACM, 2006.

[3] Maryam Karimzadehgan and ChengXiang Zhai. Constrained multi-aspect expertise matching for committee review assignment. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1697–1700. ACM, 2009.

[4] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[5] Michael Ley. Die trierer informatik-bibliographie dblp. In *Informatik'97 Informatik als Innovationsmotor*, pages 257–266. Springer, 1997.

[6] Xiang Liu, Torsten Suel, and Nasir Memon. A robust model for paper reviewer assignment. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 25–32. ACM, 2014.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[8] David Mimno and Andrew McCallum. Expertise modeling for matching papers with reviewers. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 500–509. ACM, 2007.

[9] Simon Price, Peter A Flach, Sebastian Spiegler, Christopher Bailey, and Nikki Rogers. Subsift web services and workflows for profiling and comparing scientists and their published works. *Future Generation Computer Systems*, 29(2):569–581, 2013.

[10] Marko A Rodriguez and Johan Bollen. An algorithm to determine peer-reviewers. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 319–328. ACM, 2008.

[11] Anita Valmarska, Nada Lavrac, Johannes Fürnkranz, and Marko Robnik-Sikonja. Refinement and selection heuristics in subgroup discovery and classification rule learning. *Expert Systems with Applications*, 81:147–162, 2017.