# PERSONALIZED SEARCH: AN INTERACTIVE AND ITERATIVE APPROACH

by

Haiming Wang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

# Abstract

In the face of an overwhelmingly information intensive Internet, searching has become the most important way to locate information efficiently. Current searching techniques are able to retrieve relevant data, however, personalization techniques are still needed to better identify different user requirements. This thesis proposes an interactive and iterative approach to infer a user's intentions implicitly, and adapt to changing user requirements. We gather relevance feedback from the user, and classify items in the query result set into different groups based on the feedback for each item. We rerank the original result set according to the user's interest towards each group. The group of the user's interest is ranked higher. We illustrate the approach using a personalized academic paper searching application and evaluate it with real users. The experimental results show improvements after applying our approach. The system design is extensible and potentially applicable to other search domains.

# Preface

This thesis is an original work by Haiming Wang. The experimentation part of this thesis involves a user study. The user study received Research Ethics Board approval from the University of Alberta Research Ethics Office. The application name is ”Searching for papers with different requirements, an interactive and iterative approach”, with ID Pro00047273, an approval date of April 8, 2014, and expiration date of April 6, 2015.

This thesis is a continued work of our previous publication by Haiming Wang and Kenny Wong, Personalized search: an interactive and iterative approach, in *Proceedings of the 2nd International Workshop on Personalized Web Tasking at the 10th World Congress on Services*, IEEE, 2014. I was responsible for coming up with the research idea, system design, implementation, literature review, and manuscript composition. Dr. Wong was the supervisory author and contributed to the brainstorming, manuscript revision, and gave the presentation. This thesis further developed the published paper in Chapters 1, 2, 3, 5, and 6 with broader coverage, more details, and more in-depth analysis. This thesis also evaluated the approach with real users. During this phase, I was responsible for enhancing the existing system, designing and conducting the experiment, analyzing the experimental result, and improving the manuscript. Dr. Wong was the supervisor and was involved with discussions and manuscript revision.

# Acknowledgements

I would first love to thank my supervisor, professor Kenny Wong for his generous support and careful supervision for my master's degree. Dr. Wong gave lots of both high-level and detailed suggestion during the research, and was especially involved during the composition of the thesis manuscript, which helped a lot.

I would also love to express my thanks to my committee chair, professor Eleni Stroulia, for helping me with the defense documents and sending me feedback in a very timely manner. I also want to thank professor Osmar R. Zaiane for his careful review of my thesis and coming up with brilliant questions and suggestions during the defense.

Thanks go to my friends for their support and encouragement during my hardest times. It is my lifetime treasure. Last but not the least, thanks to my beloved family for their continued love and patience. I can never do enough to pay this back.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Searching is a fast and efficient way to locate information on the Web. Data generation has been growing at an increasing rate, creating large amounts of data that need to be filtered effectively. This happens in all areas of the Web, making the need for search engines a common necessity in e-commerce websites, digital libraries, social networks, *etc*.

Current search engines are able to retrieve and rank items by relevancy to users in most cases. However, there are still special individual needs that cannot be addressed effectively. This problem is caused by the resource-centric essence of current search engines [39]. The characteristics of items are considered more, while user preferences are considered less. Ignoring different requirements between users and changing user requirements will cause search engines to not locate and rank information of actual user interests appropriately. Thus personalization techniques are needed to make the searching more individualized.

In this thesis, we propose an interactive and iterative approach to personalize searching. Different from existing personalization approaches, our system infers a user's intentions implicitly through their iterative selections of items of particular interest, and reranks items of the user's interest to the top of the search results. Compared with the original order, the new ranking is more personalized towards the user.

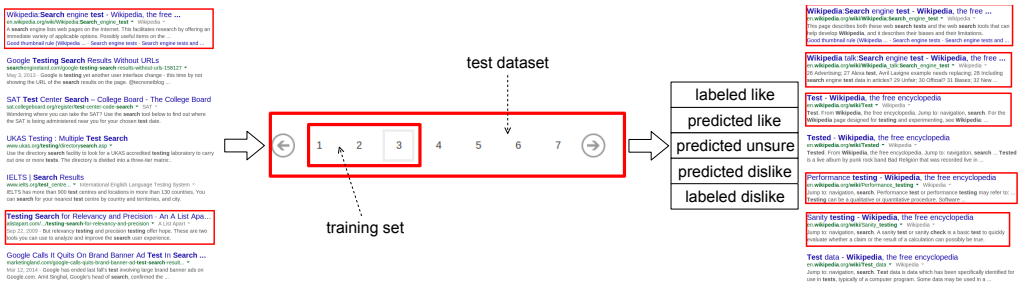



Figure 1.1: Approach Introduction

We analyze user feedback. First, when the user submits a query string and gets the original result list, he/she may be interested in only a few items, as highlighted in the rectangles in the left part of Figure 1.1. In our approach, we ask users to provide explicit feedback on each item in the original list. The feedback can be like, dislike, or unsure (Figure 1.2). Second, we regard personalization as a classification problem (middle part in Figure 1.1). The items that the user has viewed so far and labeled via the feedback are considered the training set. Items in the original result list comprise the test set, and we classify these items into 3 classes: *like*, *dislike*, and *unsure*. Third, we rerank the items according to their labels. Labeled liked items are ranked on top, followed by predicted liked items, then unsure items, predicted disliked items, and labeled disliked items. Thus the items remain the same, yet their ranking changes according to the user's selections. Higher ranked items will be more of the user's interest (right part in Figure 1.1). The user can iterate by further providing on the reranked list and rerank again. If the user wants to start a new search task, he/she can clear out all the previous selections.


Figure 1.2: User Feedback

In this way, each individual's preferences towards items are captured by training classifiers. Changing the selections will change the classifier subsequently. Since our system focuses on ad-hoc queries that requires little about a user's historical profiles, our system is adaptive to changing user requirements. Compared with the

initial order, the personalized ranking focuses more on the user's current requirements.

Our thesis statement is that with an acceptable amount of effort expended, our personalization approach is capable of ranking search results so that they are more of interest to a user.

The contributions of this thesis are as follows.

1. We propose an interactive and iterative approach to improve the results of existing search engines by analyzing user feedback with machine learning techniques.

2. We apply the approach in the academic search domain, and explore a variety of techniques to build the feature space for papers.

3. We evaluate the personalized academic search with real users, and collect their ratings towards the original order of results and our personalized reranking in terms of several properties. The findings show that users favor personalized search.

4. The implementation follows separation of concerns[1] design principles, and our system can easily incorporate different algorithms to construct features and perform machine learning operations, making it very extensible and potentially applicable to different search domains.

The structure of the rest of the thesis is as follows. In Chapter 2, we will introduce the high-level system design, system components, and working procedures. In Chapter 3, we implement a personalized paper search application using our proposed system. In Chapter 4, we conduct an experiment to evaluate the performance of personalized search in terms of several properties. In Chapter 5, we review existing search personalization techniques related to our approach. In Chapter 6, we conclude this thesis and outline possible future directions.

[1] Separation of concerns is a design principle for separating a computer program into different parts, each of which addresses a specific concern, such that the change of one part does not impact other parts. [23]

# Chapter 2

# System Design and Methodologies

When we personalize the initial order of the search results, we aim for the reranked list to reflect the current requirements of the user. We do this through an interactive and iterative process. Firstly, the user needs to tell our system his/her preferences in terms of like, dislike, or unsure on items he/she views in the initial list. Secondly, we use the above explicit user feedback to analyze the user requirements implicitly, then change the ranking of the item set for presentation to the user. Thirdly, the user can further interact with the reranked list by specifying more preferences, and let the system rerank according to the updated preferences. This process goes iteratively until the user is done with the searching task.

The second step is our key to personalize the initial result. In order to analyze user requirements implicitly based on the explicit user feedback in the first place, we need to be able to represent possible user requirements. In our approach, we build a feature space for the items, thus the user's requirements towards items are represented by the features. Moreover, we need to understand what each individual's current requirements are. We treat this as a classification problem. Items that the user has viewed so far and labeled via the feedback are the training set, and the user's preferences are the labels. The items' features, which represent the candidate requirements, are the feature set. The test set contains all items in the query result set. By training a classifier based on the user's selections, we can infer the possible requirements implicitly. By classifying the test set and reordering the result according to the predicted labels, we present a personalized reranking of the items to the user.

In order to leverage data from different sources, we use a data adapter to wrap different kinds of data to make the analysis easier in the downstream steps.

In this chapter, we will first give a high level description of the system architecture, then describe the system activities.

## 2.1 System Components

Our system consists of four basic components: Data Acquisition, Feature Construction, User Interaction, and Machine Learning components. The overview of the system can be seen in Figure 2.1. The arrows in this figure are data flows. There are possibilities of adding new components, which will be discussed later in this thesis.

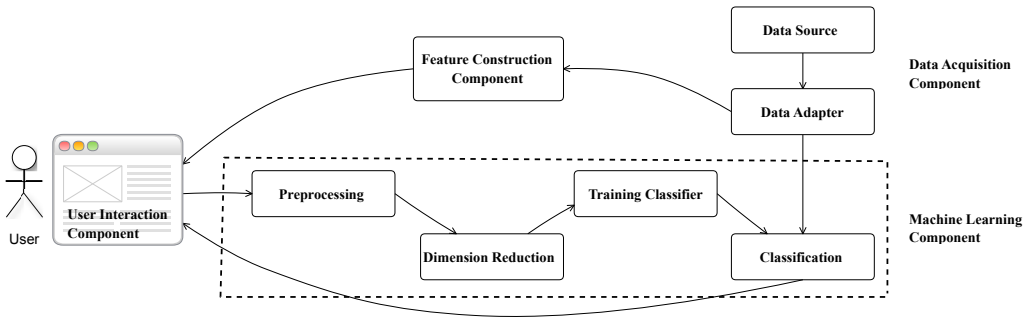



Figure 2.1: System Design

### 2.1.1 Data Acquisition

The Data Acquisition component consists of data sources and data adapters. Depending on the search domain, data sources range from less structured crawled webpages, various kinds of raw datasets, to more structured open APIs and databases. The data source can be textual, or non-textual (such as social networks, image, or audio).

Data adapters are designed to get raw data from the data sources, and transform them to the format that our system can consume. The outputs of data adapters provide datasets for the other components. By using adapters, we can easily leverage multiple, different data sources without changing other code.

### 2.1.2 Feature Construction

The Feature Construction component constructs possible features for the Machine Learning component from the formatted data source(s). Depending on the type of the data, we use different techniques. For example, for textual data, we use *tf-idf* [32] to vectorize the content, or use topic modeling techniques [13] to indicate the semantics. For relational data, social network analysis techniques can be used to get the graph metrics. And for images or audio data, signal processing techniques can be adopted. Each technique serves as a candidate algorithm for the system pipeline, meaning that only when a technique is appropriate for an application area, it is used. After feature construction, the raw data is represented as numerical values to be consumed in later steps.

The Feature Construction component is very extensible, *i.e.*, a new algorithm can be easily added to the algorithm pool, and be incorporated into the pipeline.

### 2.1.3 User Interaction

The User Interaction component allows a user to give inputs to the system and view related items. The inputs are search queries, preferences, filtering criteria, and other service requests such as reranking and starting over. After getting the query results, a user can specify preferences in terms of like, dislike, or unsure on items he/she views, which can be regarded as a manual labeling process. We will use this labeling information in our Machine Learning components.

Moreover, after getting user labeling information, our backend system will re-generate results accordingly upon request, returning a personalized result to the user for further interaction.

### 2.1.4 Machine Learning

The Machine Learning component consists of four modules: preprocessing, dimensionality reduction, training, and classification.

The preprocessing module preprocesses the constructed features of the data viewed by the users. The goal of preprocessing is to make the feature set more

suitable to the downstream machine learning algorithms. Common preprocessing jobs include mean removal, variance scaling, normalization, imputation of missing values [7].

Dimensionality reduction is used to reduce high-dimensional features to a space of fewer dimensions before the datasets are passed to learning algorithms to avoid the problem of *the curses of dimensionality* [12]. Common dimensionality reduction techniques include *principal component analysis* [25], *linear discriminant analysis* [24], and *canonical correlation analysis* [19]. After dimensionality reduction, only features of interest will be retained among all features, which improves the performance of the learning algorithms by focusing only on these discriminating features.

The training of a classifier is the core of the system. By utilizing the labels specified by a user, *i.e.*, items of interest to the user, and preprocessed features, we can train a classifier that reflects the preferences of the user towards features matching their interests. There are a variety of machine learning algorithms to choose from, *e.g.*, *logistic regression* [22], *support vector machine (SVM)* [41], *decision trees* [31], *Bayesian networks* [29]. However, different learning algorithms may perform distinctly in different user scenarios. For example, in some cases the user requirements may be better expressed by a linear kernel SVM, while the quadratic kernel SVM may not perform well. Thus a learning algorithm selection process is needed to find the best match. We use cross-validation to calculate the accuracy of each candidate algorithm and choose one with the highest score to perform classification. This is done dynamically such that a user's preference can be expressed by the best classifier available.

In the classification module, we classify the items that the user has not viewed yet, with the selected classifier. Items classified to be of user interest will be ranked higher when the results are presented to the user.

The Machine Learning component is also extensible in that new algorithms can be easily added to the algorithm pool, and incorporated into the pipeline if appropriate.

## 2.2 Working Procedure

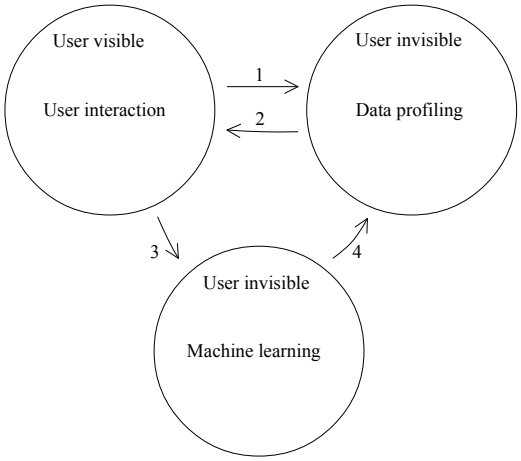



Figure 2.2: System Activities

As can be seen from Figure 2.2, there are mainly three activities when working with this system: User Interaction, Data Profiling, and Machine Learning.

### 2.2.1 User Interaction

When a user begins to use the system, he/she automatically starts in the User Interaction activity. To begin with, the user needs to give inputs to the system, such as query strings, which will make the system go to the Data Profiling activity (step 1). After getting back to the User Interaction activity (step 2), the user can view a list of query results and try to select ones of his/her interests. When the user is done with the selection, the system will go to the Machine Learning activity (step 3).

In the User Interaction activity, the inputs are empty (when a user has not yet entered anything) or a list of items (when a user searches for or regenerates new items), and the outputs are user input queries and selections. The User Interaction component is involved in this activity.

### 2.2.2 Machine Learning

When the user submits his/her selections of items, the system goes to the Machine Learning activity (step 3). In this activity, the Machine Learning component will first use the feature set and user labels to train a classifier. More specifically, it first

preprocesses the features, then performs dimensionality reduction, and trains and selects a classifier of the highest accuracy.

After the training step, we use the classifier to perform classification over additional query results, which are produced in the Data Acquisition component and processed by the Feature Construction component. Data classified to be of user interest will be ranked higher in the new list of items returned. After that, the system will go to the Data Profiling activity again (step 4).

In the Machine Learning activity, the inputs are feature sets and user selections, and the outputs are the regenerated list of items. The Machine Learning, Data Acquisition and Feature Construction components are involved in this activity.

### 2.2.3 Data Profiling

When the user submits some inputs to the system to get relevant items (step 1), or when the Machine Learning component generates a new list of items (step 4), the system goes into the Data Profiling activity. The difference between step 1 and step 4 is that, in step 1, the data is acquired through data sources using the Data Acquisition component, while in step 4, the data is sent directly from the Machine Learning component. If this activity is entered from step 1, the Feature Construction component will construct features of the item set. If this activity is entered from step 4, no feature constructions are needed since the item set remains the same and only the ranking changes. After getting the list of items, the user interface will be populated with the new data, and the system will go to the User Interaction activity (step 2).

In the Data Profiling activity, the inputs are user queries (when the previous activity is User Interaction) or a list of items (when the previous activity is Machine Learning), and the outputs are a list of items and feature set of the items. Data Acquisition and Feature Construction components are involved in this activity.

Among the three activities, the User Interaction activity is visible to users, *i.e.*, users can view the effects or have control over the system, while Machine Learning and Data Profiling activities are invisible to the user. All three activities happen online, *i.e.*, all the calculations and interactions are done in real time.

# Chapter 3

# Application

In this chapter, we apply our proposed personalized search system to find relevant academic papers to show the operations of our system. We motivate this application domain, describe the system architecture, outline the application-specific components, and explain how to use the system.

## 3.1 Motivating Story

Paper searching is an important activity in academia. Researchers search for different reasons. For example, a new graduate student may want to find an authoritative survey paper to get familiar with a new area. A researcher may want to find papers related to a certain topic with emphasis on a specific aspect. An author may want to find references from a specific conference or people. A scholar may want to get rid of irrelevant papers that accidentally match his/her query string. These diverse requirements emphasize different aspects of a paper, *e.g.*, the topic coverage, the influence in terms of citations. The relevant features are difficult to infer, yet are important for a particular user.

However, major academic search engines mainly focus on keyword matching, ignoring specific needs of different users. Even with advanced searching functionalities, such as filtering or sorting by some criterion, flexibility is still limited because some requirements are hard to represent. This issue results in users refining their queries by trial and error, or flipping through many pages of results, which is time consuming and may miss what the user wants.

In order to address this problem, we apply our proposed personalized search system to allow users to provide feedback. Users indicate which initial paper results seem more relevant. The system discovers the features of interest across the selected papers, using the feedback to produce improved results. The interaction happens iteratively to better converge to the user's requirements. The system is extensible, thus we can easily add other kinds of features to address various user requirements.

## 3.2 Architecture

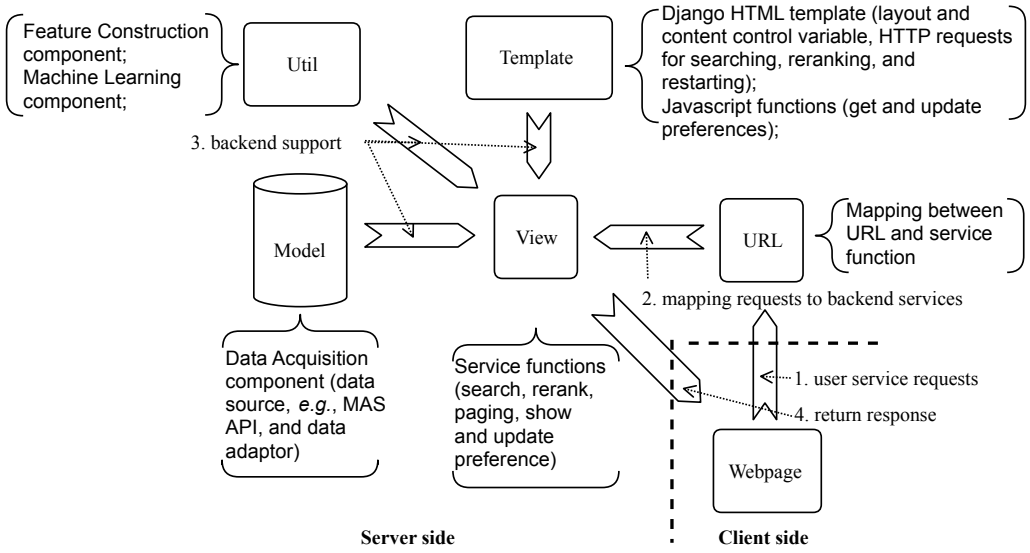



Figure 3.1: System Architecture

We use the open source Django framework to implement this system. Django is a high-level Python Web framework which uses a Model-Template-View structure to provide a clean, pragmatic design to support rapid development [2].

We show the system architecture in Figure 3.1, where the directed edges denote data flows. The text in the brackets explain the content of major components. More detailed comments and the implementation can be found in [8].

The following is a walkthrough of the working procedure of this Django application, combined with the system components and activities.

### 3.2.1 User Service Requests

When a user requests services from the system, the web browser will send HTTP requests to a target URL of the backend server. The services include searching for papers, flipping through pages, selecting interesting papers, and generating new papers of interest. The requests can be either synchronous or asynchronous (such as Ajax calls). The User Interaction component and the User Interaction activity is involved.

### 3.2.2 Mapping Requests to Backend Services

The type of requested service is identified by a specified URL to which the request is sent. A URL module in Django maps a URL to a service function in the View module. Thus, the service request can be handled by a corresponding service function.

### 3.2.3 Backend Support

When a service function in the View module is called, it can utilize a variety of data sources and algorithms from the backend to serve the incoming requests. In our application, the backend support includes data acquisition, feature construction, and machine learning. After getting the desired results, the service function can either render them to a specific HTML template or package them into JSON[1] objects.

Since the service functions are separated from the controller (URL module), data model, and presentation tier (HTML templates), we can easily add new algorithms to the application. The data acquisition, feature construction, and Machine Learning components, as well as Data Profiling and Machine Learning activities are involved in the backend support.

### 3.2.4 Return Response

When the rendered HTML webpages or JSON data are returned to the browser, client-side scripts can manipulate the response before presenting it to the user.

[1] http://www.json.org/

## 3.3 Application-Specific System Components

Since we are applying our personalized search system to paper searching, we need to customize some of the components to the application area. The major customization exists in the Data Acquisition, Feature Construction, and Machine Learning components.

### 3.3.1 Customization of Data Acquisition

The goal of data acquisition is to get raw data from data sources, and use data adapters to transform them to the format that our system can consume. In this application, we want to customize the data sources of paper dataset providers, fetch paper information, and store that in our system. The dataset provider should be legally usable, rich in data, and provide necessary functionalities.

There are many paper dataset providers, such as ArnetMiner[2] and arXiv[3]. However, arXiv significantly lacks publication information in certain computing science research areas, such as software engineering. ArnetMiner only provides limited information of the publications. While Google Scholar[4] is very popular in academia, it does not provide any public interfaces that developers can use.

Microsoft Academic Search (MAS) also provides relatively good search services for academia. Moreover, it provides an API that allows developers to build applications by leveraging the data and functions of MAS. The API enables applications to [5]:

- send a free text query to retrieve relevant objects (with object types including publication, author, conference, journal, organization, keyword and domain);
- get the detailed information for a given object;
- explore the relationships between objects, such as *Reference* and *Citation*.

The MAS API provides powerful search functionalities. For example, the developers can specify different conditions such as title matching and author matching,

[2] http://arnetminer.org/
[3] http://arxiv.org/
[4] http://scholar.google.com/

and the returned objects will be the requested object type and meet the intersection of all the conditions, which enables the developers to implement all functionalities of MAS itself. The returned objects of MAS API are in pure JSON format, which is easy to manipulate.

Thus we use the MAS API as our data source. We focus on using the publication search function of the MAS API. When we issue a valid publication search request, the returned results are sets of relevant publications containing information about paper titles, authors, abstracts, citation count, conference or journal, keywords, and URL link.

These rich fields of publication information can be used to construct paper features. We can potentially use the information of other object types, such as author or conference, to apply the personalized search in other search aspects.

### 3.3.2 Customization of Feature Construction

The goal of customizing the Feature Construction component is to allow it to address different requirements as mentioned in Chapter 1. Thus we need to identify possible user requirements and analyze the information available to address as many requirements as we can.

For example, some users may be interested in papers from some groups of researchers. We can address this requirement by mining the community of the coauthorship graph, *i.e.*, clustering authors who work together frequently into the same community and analyze whether the user is interested in some communities of authors. Using the *Louvain Method*[5] [15], we can quickly find the community of authors in a coauthor graph, which is constructed from the authors of the current query result set. Figure 3.2 shows the result of mining the coauthorship graph of a query result set. Authors in the same community are in the same color. Communities containing less than five nodes are hidden in the graph. This graph is visualized with Meerkat[6], a social network analysis tool.

[5]Louvain Method is a community mining algorithm that finds the structure of a network and divides it into clusters based on the connections between the nodes.

[6]http://www.aicml.ca/?q=node/41

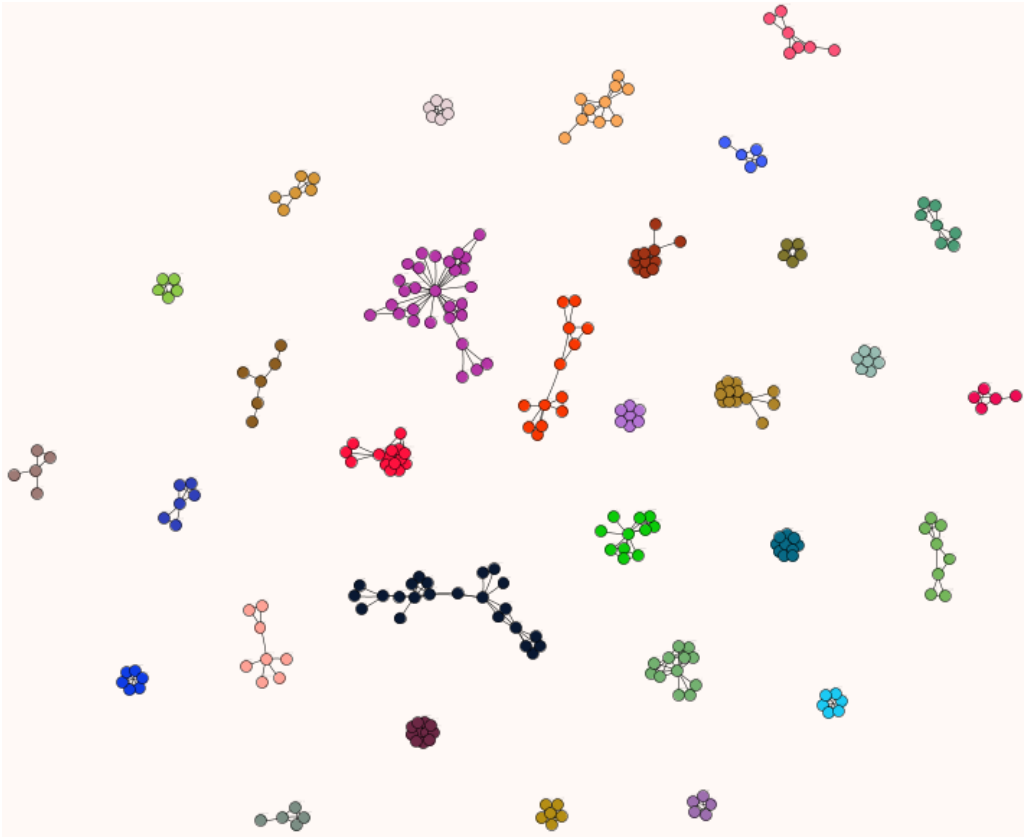


Figure 3.2: Community Mining of Coauthorship Graph

Textual features also play a very important role in academic search. Using *tf-idf* is a good way to search keywords; however, in our application, there is limited training data compared with the dimensionality of *tf-idf* features, which is in the thousands. Using *tf-idf* can make the feature space significantly greater than the number of data points, which will downgrade the performance of some classifiers. For example, the decision tree model will overfit the data if the feature dimensionality is much greater than the number of samples [7]. Moreover, since the query results provided by the MAS API are already related to the query words, the need for matching query words is reduced. Thus, we use the Latent Dirichlet Allocation (*LDA*[7]) [13] language model instead. *LDA* can help to create topic distributions, which significantly reduces dimensions while keeping semantics of items as a feature. We first train a *LDA* language model from the concatenated texts of title and

[7]LDA is a language modeling technique that models topics as distributions of words and documents as distributions of topics. It can greatly reduce the dimensionality of the feature space for the documents.

abstract of each paper in the current query result set. The language model contains ten topics, and each topic is a distribution of words in the current query result set. Each paper has a distribution of the ten topics, denoting its semantics. There is a maximum of 1000 papers in each query result set. If we set the number of topics too large, the distinction between topics becomes very small so that papers cannot be easily differentiated by their topic distributions. If we set the number of topics too small, there will not be enough topics to differentiate the papers. We observed the data and tried with different numbers of topics and found that 10 topics is a good balance. On average, each topic is supported by 100 paper entries (where each entry is a concatenation of title and abstract that are around 100 words), thus the topics are well built. The papers can be differentiated by their distributions of different topics instead of just one topic. Moreover, since the papers are already related to the query words, the differences between papers are not as vast as papers randomly chosen from the whole paper dataset. Thus we do not need to use that many topics to differentiate the papers.

There are also other features that a user may be interested in. For example, the user may be interested in what conference the paper is published in, how many citations the paper gets, how many papers that paper cites, and how recent the paper is. We can easily get this information using the MAS API.

Different types of data will produce different types of features, which are constructed with different algorithms. Table 3.1 summarizes the implemented list of features, feature types, and their corresponding feature construction techniques in the publication search area. Please note that this list can be extended easily after identifying appropriate new features on the dataset. We do so by inserting the new feature construction techniques into the Feature Construction component, and adding the features they construct to the whole feature set. This is documented in the ProcessFeature module in [8].

### 3.3.3 Customization of Machine Learning

The Machine Learning component is the core of the system. Based on the preferences given by the User Interaction component and the feature set given by the Fea-

Table 3.1: Paper Feature List

| Feature Name | Feature Type | Technique |
|---|---|---|
| Community number of first author in coauthorship graph | integer | Community Mining using *Louvain Method* [15] |
| Topic distribution of title and abstract | list of float | *Latent Dirichlet Allocation* [13] |
| Conference ID | integer | from MAS |
| Citation count | integer | from MAS |
| Reference count | integer | from MAS |
| Recency of publication | integer | from MAS |

ture Construction component, the Machine Learning component trains classifiers that can capture a user's preferences, which helps to reach our goal of personalization.

Since there are different search domains, the feature sets may vary vastly in terms of dimensionality, and differ in value type, *i.e.*, categorical or numerical. And different users are likely to focus on different features. Thus it is necessary to include a wide range of classification algorithms, train different types of algorithms, and choose one of the best.

Also, it is important to do dimensionality reduction to avoid overfitting [20] and increase time efficiency when there are too many features. Increasing time efficiency is important because we want the personalization to be interactive and more responsive to save the user's time.

Moreover, since some classification algorithms hold certain assumptions of the feature values, *e.g.*, zero mean, unit variance, or Gaussian distribution, it is required to preprocess the features to meet these requirements first.

Lastly, when users are interacting with the publication items, they mainly have three different attitudes towards each item:

- *Like*. The user is clear that this item meets his/her criteria.
- *Dislike*. The user is clear that this item does not meet his/her criteria at all.
- *Unsure*. The user does not have good confidence in either *Like* or *Dislike*, maybe because the item is not interesting enough, or the user cannot understand it well.

Thus instead of having two classes, we have three classes representing each item. The benefits of having three classes is not only letting users avoid making hard decisions, but also giving papers with uncertainty an appropriate place. Thus the classification algorithm should support multi-class classification.

Considering the above requirements, we choose the open source Python machine learning library, *scikit-learn*, to help implement the Machine Learning components.

**Classification algorithms**

When constructing the classification algorithm pool, we are mainly concerned about the ability to process every feature in the feature set, thus we need to make the algorithm pool as complete as possible. However, we also need to consider the efficiency tradeoff when there are too many classification algorithms, since each algorithm will be trained and evaluated online.

Consider the following issues.

- Our paper search feature set contains both categorical and numerical values.
- Most of the features are independent of each other. (Naive Bayes Classifier assumes that all features are independent of each other.)
- A user's requirements on some of the features may be linear or non-linear.
- A user's requirements may evolve during the search.

One classification algorithm may not perform well at all times, thus we construct the following algorithm pool that can address these issues.

1. *Support Vector Classifier (SVC). SVC* trains a hyper surface (decision function) using kernel functions (such as linear kernel) and some key training data points (support vectors). The label of a test data point is determined by its position relative to the hyper surface. *SVC* is very effective in high dimensional space, even when the dimensionality is greater than the class size. It is also memory efficient, because it only uses a few support vectors to train the classifier. More importantly, it is very versatile since different kernel functions

can be specified for the decision function. The decision function is the hyper surface that determines whether a data point belongs to one class or another. The shape of decision function is determined by the kernel functions. We can customize the decision function using different kernel functions such as the linear kernel, polynomial kernel and radial basis function (RBF) kernel. Thus both linear and non-linear requirements can be addressed. However, we should note that *SVC* is not scale invariant, thus we preprocess the feature set to make features in the same scale to avoid one feature being weighted more importantly than another. [7]

2. *Decision Tree*. *Decision Tree* creates a model that predicts the label of a test data by learning simple if-then-else rules from the training dataset. *Decision Tree* is a very useful candidate because of its ability to handle both numerical and categorical values. Besides, it requires little data preprocessing. Moreover, the decision process is quite interpretable and can be easily visualized. However, we should note that *Decision Trees* may create a complex model that does not generalize to the whole dataset. It may also create biased trees if one class dominates. In order to address these problems, we perform dimensionality reduction to avoid overfitting. [7]

3. *Naive Bayes Classifier (NBC)*. *NBC* is based on Bayes theorem with the "naive" assumption of independence between every pair of features. The probability of test data belonging to one class can be decoupled to the product of the probability of each independent feature's presence in that class. *NBC* has a simple assumption that all features are independent of each other. Despite this simple assumption, *NBC* works decently well and is extremely fast compared with other more sophisticated classifiers because of the decoupling, which also helps to alleviate problems caused by the curse of dimensionality. [7]

4. *Logistic Regression Classifier (LRC)*. *LRC* is a linear model classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function. The *LRC* model also assumes that the

features are independent of each other. [7]

5. *Nearest Neighbors Classifier (NN)*. *NN* can also serve as a classifier by predicting the label of a data point to be the most frequent label of its nearest N neighbors. [7]

There are also a lot of other classifiers that can be easily added. We can do this by inserting the new classification algorithm to the Machine Learning component, and calling it in the classifier comparison function. This is documented in the clf module in [8]. However, if too many classifiers are added, each requiring some training and tuning time, the time efficiency may be reduced, making the users wait longer. Thus we only use the above classifiers, which are able to address the above issues without sacrificing too much time efficiency.

Each classifier has its own set of parameters that should be tuned for the best performance, *e.g.*, to make the classifier gain highest accuracy on the current set of training data. And the best parameters may change when the training set changes. Thus it is important to get the best set of parameters for each type of classifier for each classification task.

We use *k-fold cross validation* to find the best set of parameters. Firstly, we use the *StratifiedKFold* module provided by *scikit-learn* to split the training set into *k* folds, where each fold has approximately the same ratio of different classes. We use k-1 folds as the training set, and the remaining 1 fold as the test set. By running it k times, we can get the average performance of a specific classifier under a set of parameters. Secondly, we prepare candidate values for some parameters of classifiers as needed (Table 3.2). For example, in *SVC*, we can choose different kernels for the decision function such as the linear kernel and RBF kernel. Also, the penalty parameter C is a tradeoff between training error and the generalization ability of the model. If C is too small, the training error will be large. However, if C is too large, the model overfits the training data and loses the generalization property. Thus we prepare a set of candidate values for C, [1, 10, 100, 1000], which ranges from a small penalty to a very large penalty. Thirdly, for each classifier, we use the *GridSearchCV* module of *scikit-learn* to score each parameter combination

prepared earlier. In the above example, we have 2 kernel candidates and 4 penalty parameter candidates, thus there are 8 ($2 \times 4$) combinations. The best parameter combination for that classifier is retained until the training data changes, when the user changes his/her selections and requests to rerank. This step once again reminds us that, if too many classifiers are added to the algorithm pool, the search efficiency would be reduced significantly since each classifier will be trained $k \times m$ times, where $k$ is the folds number, while $m$ is the number of parameter combinations.

Table 3.2: Parameter Grid of Classifiers

| **Classifier** | **Parameters and value sets** |
|---|---|
| *SVC* | kernel: [RBF, linear]; $\gamma$: [$10^{-3}$, $10^{-4}$]; C: [1, 10, 100, 1000] |
| *NN* | n-neighbors: [3, 5]; algorithm: auto; weights: [uniform, distance] |

After each classifier gets its best parameters, we further compare the scores of different classifiers, and use the highest scoring one to classify the test data set.

There are many scoring techniques when comparing classifier performance, *e.g.* accuracy, recall, and precision. When we are choosing the score, we should mainly consider the application being built. Taking paper searching as an example, if a disliked item is classified as liked (False Positive), it will be ranked higher; a truly liked item (True Positive) will be consequently ranked lower, which the user may miss. However, if a liked item is classified as disliked (False Negative), the other truly liked items will still be ranked higher. Thus, users will be more tolerant with False Negative than False Positive. In this situation, we choose accuracy as our scoring criterion.

### Preprocessing

As mentioned in Section 3.3.3, some classification algorithms require preprocessing, thus we preprocess the feature set before training classifiers. According to the classifier requirements, we process the data to have zero mean, unit variance, and same scale among different features before the dataset is applied to the classification algorithms. Although some classification algorithms (such as the decision trees) do not require preprocessing, it does no harm to standardize the data for them. Sharing

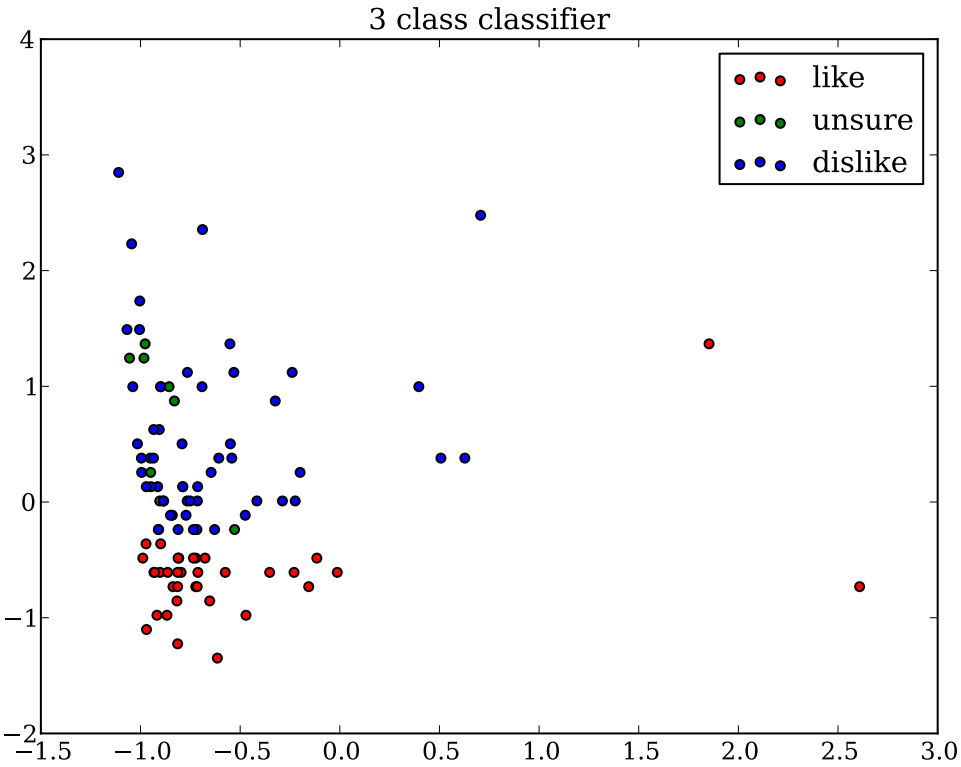
the same preprocessing step for all the classification algorithms can also save the time of preprocessing for each algorithm that needs preprocessing separately.

### Dimensionality reduction

Performing dimensionality reduction is a good way to increase time efficiency as well as avoid overfitting. We use tree based estimators (*ExtraTreeClassifier* [7] provided by *scikit-learn*) to compute feature importance, which can be used to discard irrelevant features.

Figure 3.3 is an example of a three-class classifier, which has gone through the preprocessing, dimensionality reduction, and parameter tuning stages. And it has been compared against several other classifiers and selected as the best one on the result dataset for the query terms of *software engineering*.

From the Figure 3.3, we can see that after dimensionality reduction, there are only 3 features left, which means only 3 features are considered to be significant to discriminate papers belonging to three different classes, *i.e.*, like, dislike, and unsure. The figure plots the data points according to only the first two dimensions, yet we still can see that data points from the like class are well separated from the other two classes. However, this is not always the case, especially when the features are not discriminative and when the user has unclear or over-complex requirements. In this example, decision tree classifier has the highest accuracy and is selected as the classifier to classify the test data.




Query:software engineering
Classifier:DecisionTreeClassifier(compute_importances=None, criterion=gini,
max_depth=None, max_features=None, min_density=None,
min_samples_leaf=1, min_samples_split=2, random_state=None,
splitter=best)
Item number:100
Feature dimension:3

Figure 3.3: 3 Class Prediction and Corresponding Classifier

After customizing these components, our personalized search system can work for a paper search application. The implementation can be found in our Bitbucket host [8].

## 3.4 Example Interaction Scenario

In this section, we will show how the frontend works in the personalized paper search engine.

The frontend consists of four sections: input, commands, selections, and results (Figure 3.4). The input section allows a user to input new queries, refine queries, and search. The commands section has three commands for the user. The first

command is *Search*, which users can click to request the backend server to search relevant items according to the search query. The second command is *Rerank*, which means the user asks to rerank the items from the current query results based on the current preferences. The third command is *Start Over*, which means the user can empty the current selections and start a new searching task. The selections section shows the user's current liked and disliked items. If a user selects new liked or disliked items, they will be appended to the selections section. The selections section will only be emptied when the user clicks the *Start Over* button. The results section shows the set of relevant items when the user clicks *Search* button, or the personalized set of interesting items when the user clicks the *Rerank* button.

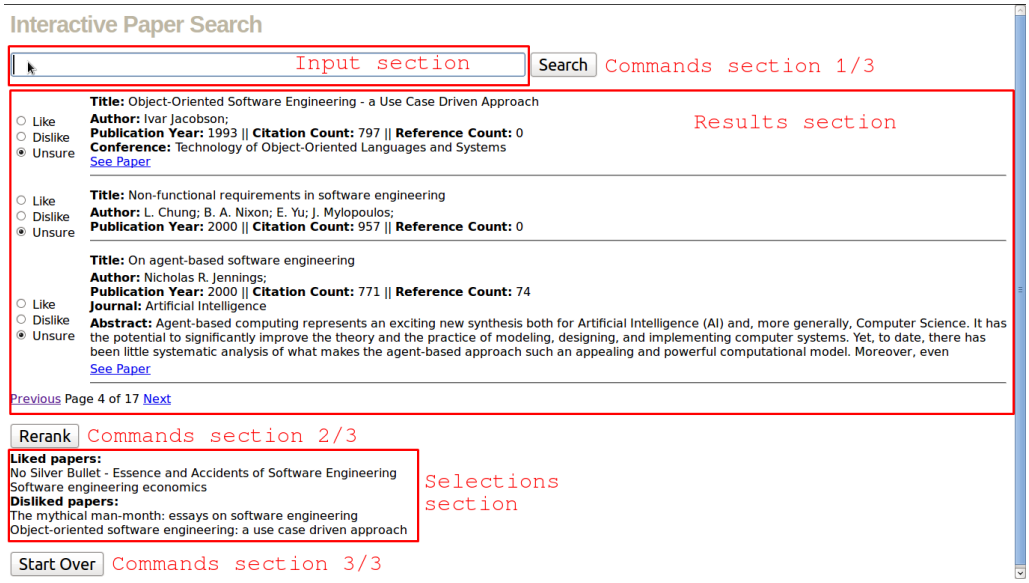



Figure 3.4: Frontend Layout

The following pages show a step-by-step user interaction of the personalized paper searching application.

- Search for publications using queries

    1. As shown in Figure 3.5, a user can input a query string and search just like an ordinary search engine.

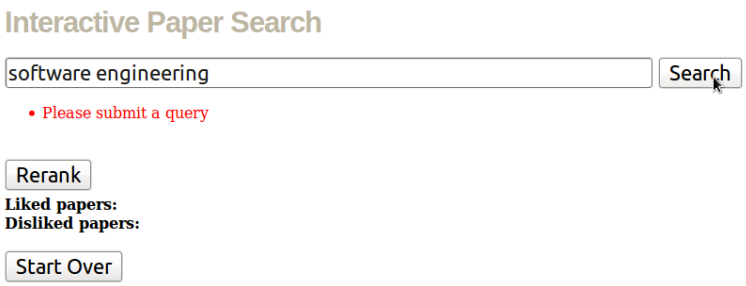



Figure 3.5: User Interaction: input query

2. After the query string is submitted, the backend Data Acquisition component fetches data from MAS then sends it to the frontend. After that, the user gets a list of publications containing information such as title, authors, publication year, citation count, reference count, venue, abstract, keywords, and its link (as shown in Figure 3.6). Meanwhile, in the backend, the Feature Construction component constructs the features set in another thread.

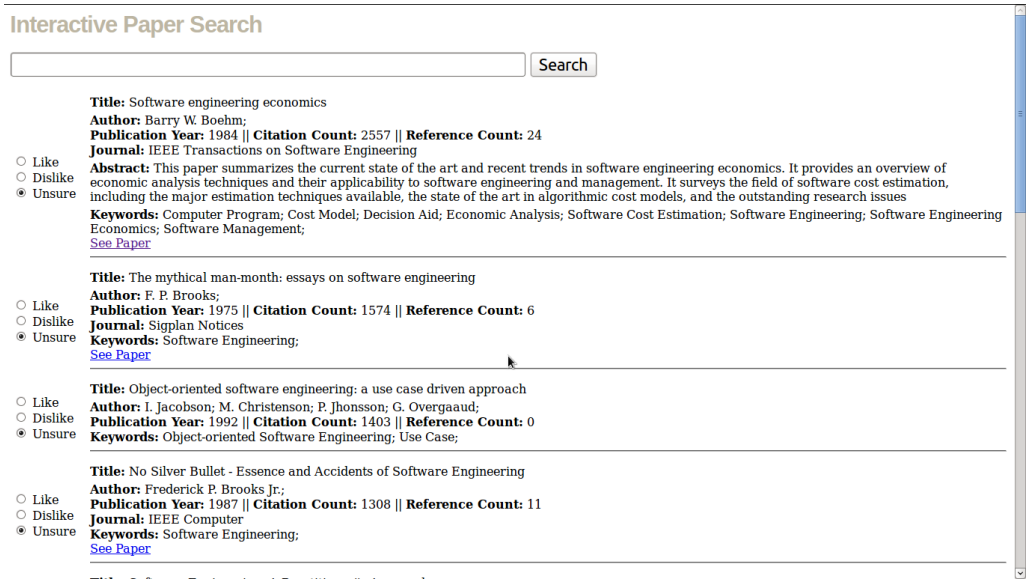



Figure 3.6: User Interaction: get initial results

- Personalize the search results

3. On the left of each publication item, there is a group of radio buttons indicating the user preference of that publication. It is *Unsure* by default.

Users can click on one of the choices to specify the preference as shown in Figure 3.7.

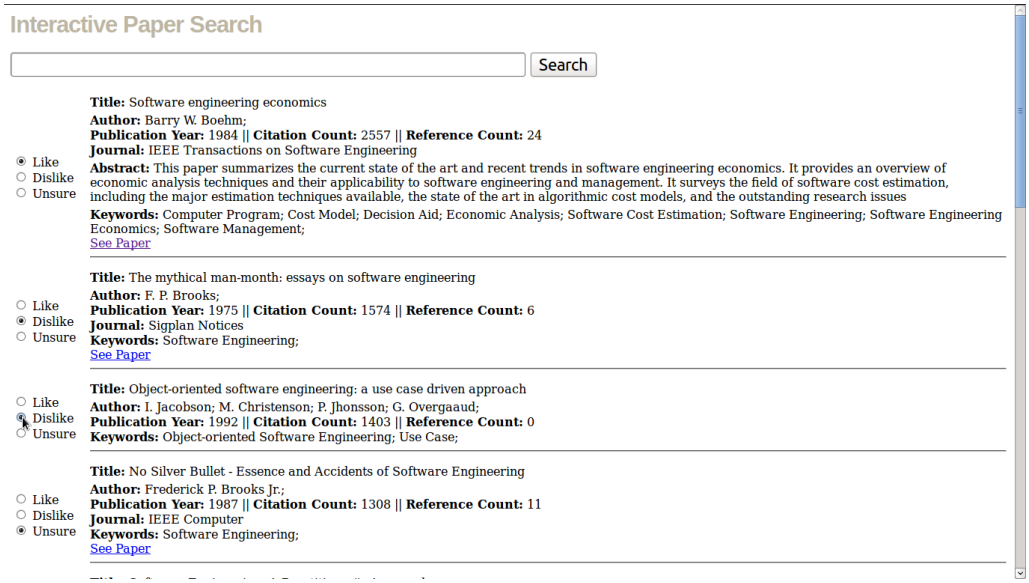



Figure 3.7: User Interaction: specify preferences

4. At the bottom of the page, the user can see his/her personalized selection results. He/She can also browse through pages as shown in Figure 3.8.

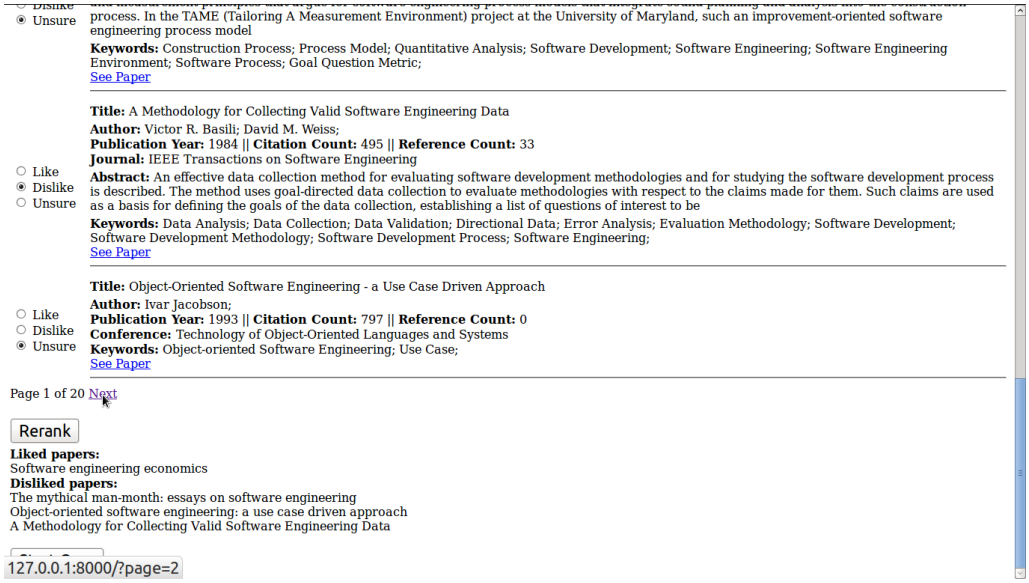



Figure 3.8: User Interaction: view next page

5. After browsing the publication list and specifying preferences, the user

can click on the *Rerank* button to rerank the current results based on the selections as shown in Figure 3.9. At this point, features of all the paper entries viewed so far by the user and papers in the selection list (liked and disliked papers) are the training dataset, while the user specified preferences, *i.e.*, like, dislike, and unsure, serve as the labels. The features of all the papers in the query result set are the test dataset. The Machine Learning component trains classifiers, predicts and reranks items with the best classifier. After the reranking, the labeled liked papers are listed on the top, predicted liked papers come after, predicted unsure papers in the middle, predicted disliked papers after, labeled disliked papers at the bottom. Figure 3.10 shows the reranked list of publications.

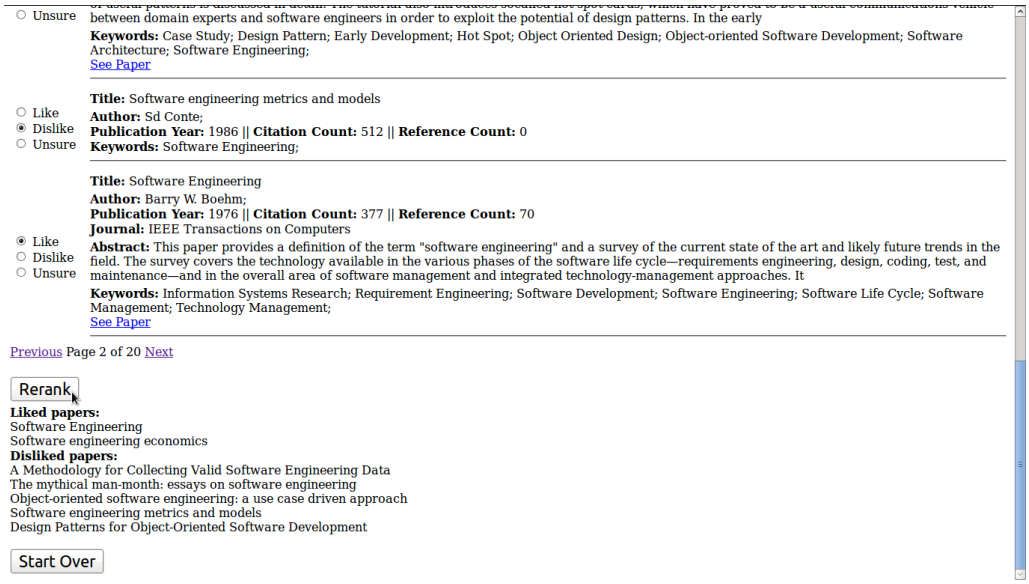



Figure 3.9: User Interaction: rerank

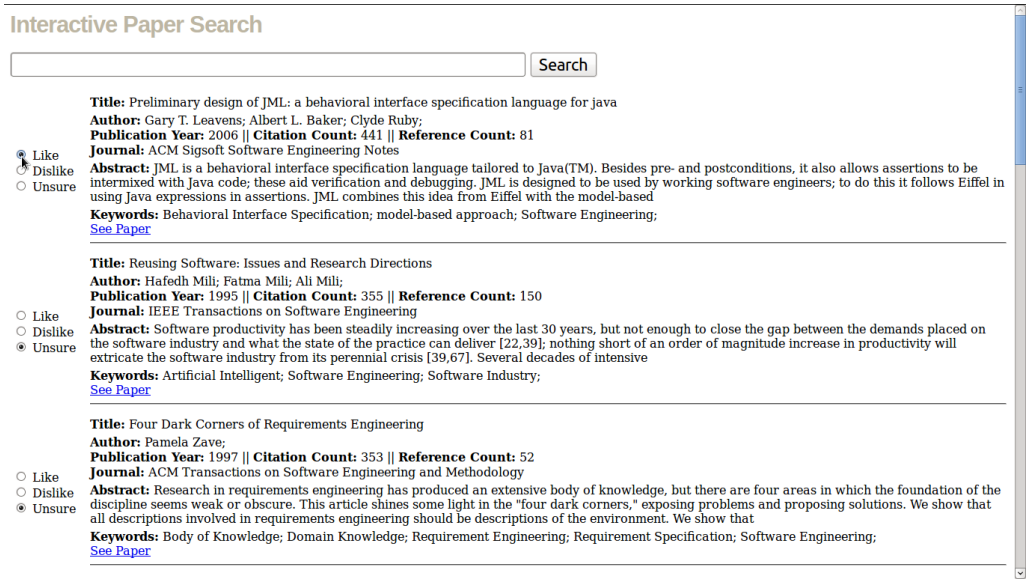



Figure 3.10: User Interaction: get reranked results

6. If the user is not satisfied with the initial rerank, he/she can just browse the first page and click on the *Rerank* button again. This is called *Successive Reranking*, which takes advantage of existing selection results, thus can save effort. Since the viewed papers change, the training dataset will also change, resulting in an updated classifier. The second rerank will generate a different ranking accordingly. This is shown in Figure 3.11.

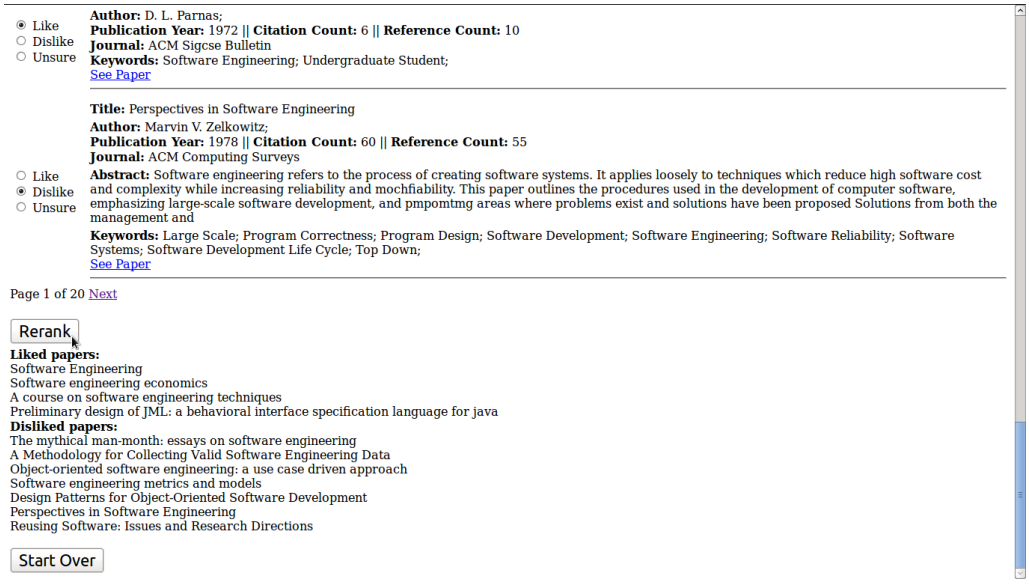



Figure 3.11: User Interaction: successive reranking

- Revise query terms

7. During the process of searching for papers, the users may come up with better search terms for the query task. If so, the user can simply input new query terms in the search box and submit the query (Figure 3.12). All his/her previous selections are kept. A new list of search results is returned (Figure 3.13).

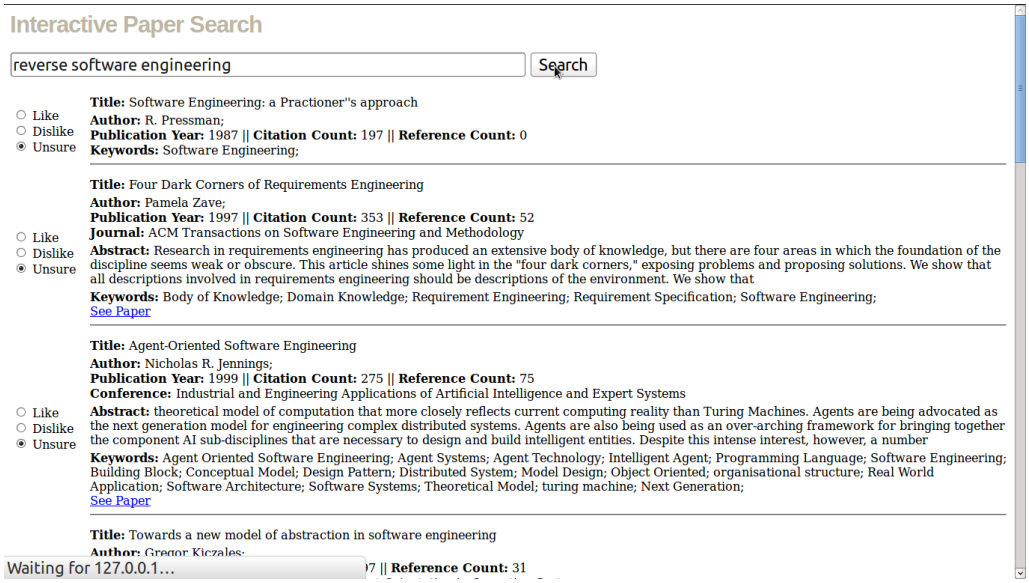



Figure 3.12: User Interaction: revise query terms

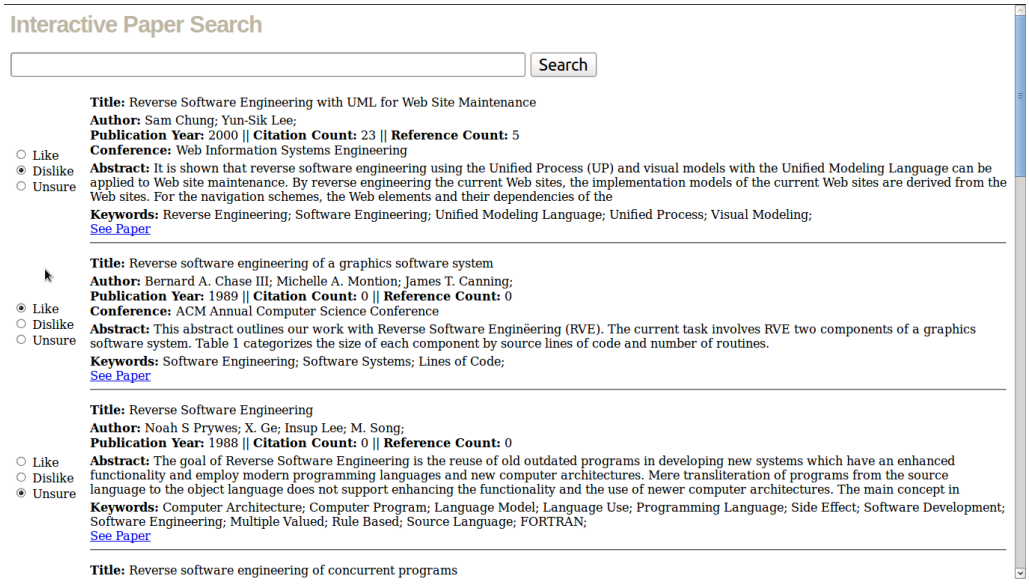



Figure 3.13: User Interaction: get new results

8. The user can also specify personal preferences on the new query result set, and rerank the new results. His/her previous selections for the same task stay the same (Figure 3.14).

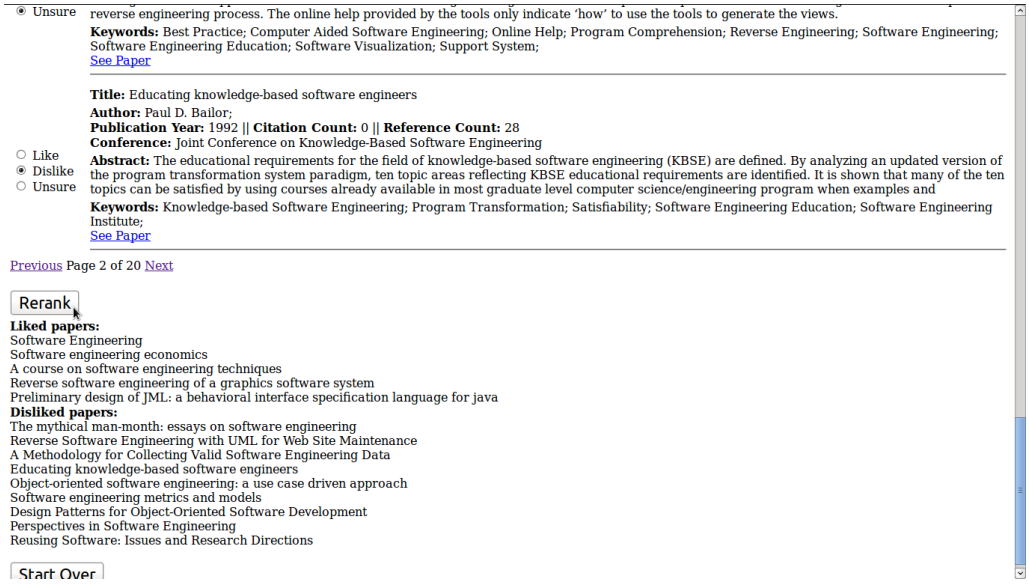



Figure 3.14: User Interaction: rerank on new results

- Start Over

9. Click on the *Start Over* button to clean all the internal states and start a

new task (Figure 3.15).

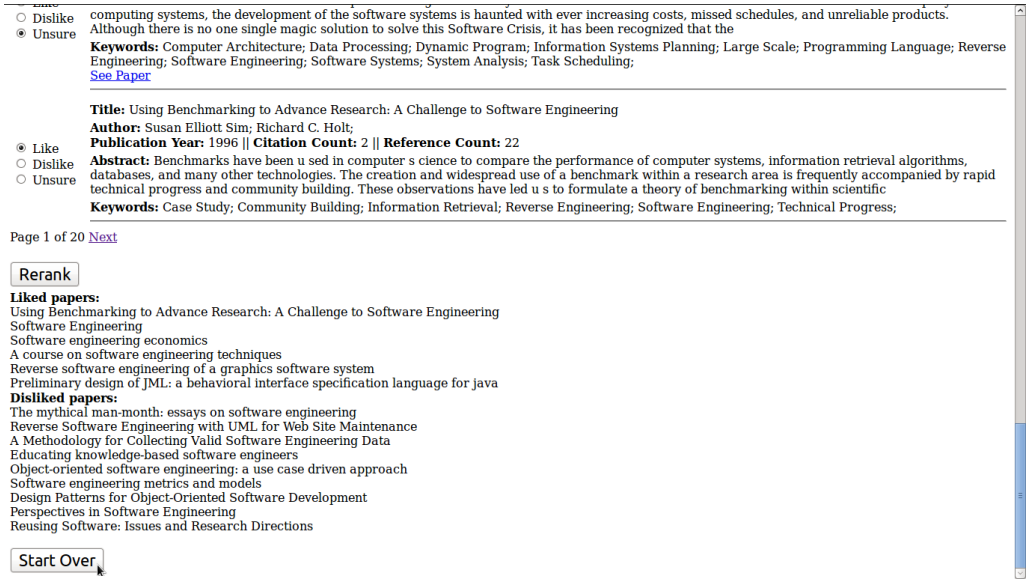



Figure 3.15: User Interaction: start a new search task

# Chapter 4

# Experiment

Personalized search focuses on the specific needs of each unique user, thus a user's rating is very important when evaluating the system [36]. As a continued work for our previous research [40], this thesis evaluates the approach with real users. We arrange the real users to use our personalized search engine (system A) as well as an ordinary search engine (system B). Each user performs the same searching task on both search engines, and rates their performance.

When the user is performing a searching task, he/she inputs a query string to get items, and holds certain criteria when judging whether an item is of his/her interest or not. We define items of interest as items meeting the criteria.

However, not all the returned items are relevant to the query according to the user. And these query-irrelevant items are not of the user's interest, because they are regarded as irrelevant by the user and therefore they do not meet the user's criteria. Also, among the query-relevant items, some are of the user's interest, while some are not of the user's interest. The user gives *like* as the relevance feedback to items of interest, and *dislike/unsure* as the relevance feedback to items not of interest.

For example, a user wants to search for papers about applications of *principle component analysis*. He/she uses *PCA*, the abbreviation, as the query string. However, there are papers about *patent certificate application* returned, because the search engine thinks it is relevant to the query. In this scenario, the papers about *patent certificate application* are query-irrelevant items according to the user. Even if the user is interested in patent applications, these papers are still categorized as items not of interest because they do not meet the current search criteria. Also,

among the *principle component analysis*-relevant papers, some focus on applications, and some focus on theories. According to our definition of items of interest, application-relevant papers are items of interest, while theory-relevant papers are items not of interest.

The categories of items can be visualized in Figure 4.1.

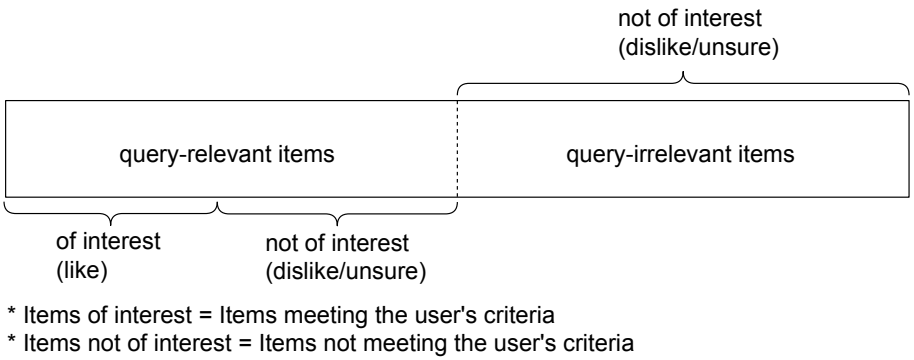



Figure 4.1: Items Categories

We focus on the following properties when rating search engine performance:

- *Accuracy*. Accuracy focuses on whether items of interest will be ranked higher.
- *Coverage*. Coverage focuses on whether query-irrelevant items will be ranked lower, leaving room for query-relevant items to be ranked higher.
- *Serendipity*. There are some items that are query-relevant but does not meet the user's criteria (not of interest), however, the user may find these items potentially useful. We define these items as serendipity. And the serendipity property focuses on whether these items will be ranked higher.
- *Effort*. Good accuracy may come with the cost of extra user effort. Effort is a measure of user specific tolerance of extra effort, indicating the user's subjective perception instead of objective measurement such as time consumption.
- *Overall*. Taking all properties into consideration, overall is the general tendency of a search engine being preferred.

We hypothesize that our personalized search engine (system A) will outperform ordinary search engine (system B) in *Accuracy*, *Coverage*, and *Serendipity* properties.

## 4.1 Experiment Design

### 4.1.1 Participants

In this experiment, we called for participants, and 10 graduate students agreed to participate. All of the graduate students were in their second year or above from the Department of Computing Science. The choice of second year or above is because this group is more familiar with their research. Their research background varied in different research topics of computing science, such as theoretical algorithms, software engineering, machine learning, computer vision, visualization, natural language processing, and wireless sensor networks.

### 4.1.2 System Setting

The experiment was run on a laptop, a quad core Intel®Core™ i3-M370 CPU 2.40GHz and 3 GB of memory. The required disk space for this experiment is 20 MB, while our available disk space is 200 GB. This configuration was far beyond the needs of this experiment.

Both system A, the personalized search engine, and benchmark system B, the ordinary search engine, used the MAS API. The maximum number of results retrieved was 1000 for a query, which took around 10 seconds for both systems. System A used exactly the same set of features and classifiers as described in Chapter 3, and the implementation is in [8] (commit id: a67721f).

### 4.1.3 Experimental Goal

In this experiment, our goal was to compare the 5 properties (as mentioned above) of both systems, and validate our hypothesis.

### 4.1.4 Procedure

The experiment was conducted in person. In this way, we could answer the participant's questions and have a better chance to observe user behavior and ask user requirements. For each participant, we first prepared him/her with an introduction and hands-on practice. Then we let the user choose and perform his/her search tasks. After a search task was finished, the participant filled in a questionnaire (Appendix D) and began the next task. There are 2 tasks for each participant.

**Preparation and warming up**

1. Participant signed a consent form (Appendix A).
2. We introduced the features and usage of system A (personalized search engine) and benchmark system B (ordinary search engine) to the participant.
3. We explained to participant about the questionnaire comparison properties, *i.e.*, *accuracy*, *coverage*, *serendipity*, *effort*, and *overall*.

**Choose searching tasks**

Each searching task contained some query words and certain criteria. We provided searching tasks from Appendix D as examples. For instance, one searching task was searching with the query word *Hbase*, with the criteria of finding papers with emphasis on the application and industrial side. Since participants had different background and areas of interest, they could instead create a searching task on their own. The differences in participants-created searching tasks did not impact our goal of collecting their property ratings. On the contrary, if a participant was interested in a searching task, his/her ratings of properties could reflect his/her preferences more accurately. Each participant chose two searching tasks.

**Searching and rating in each task**

Each participant would fulfill the two searching tasks he/she had chosen. In the first task, the participant would first use system A, then system B. In the second task, the participant would first use system B, then system A. In this way, the potential bias

caused by the order of use can be decreased. A questionnaire was filled after each task.

During one task, the participant entered the same query words to both A and B. Since A and B shared the same data source (*MSAS API*), they would return exactly the same initial result set and ranking. If the participant was not satisfied with the result set, he/she can change the query words, but the input to both systems stayed the same.

A participant was expected to use system A and B in the following steps.

**Using system A** After getting the initial result set, the participant scanned through the list of items. For each item, he/she had three options, *i.e.*, *like*, *dislike*, and *unsure*. The default option was *unsure*. The participant needed to select at least one liked or disliked paper. If the participant did not see enough liked papers from the first page, he/she could go to the second page. After specifying the preferences, the participant could click on the *Rerank* button. After the classification, a reranked list was returned. The labeled liked items were listed on the top, predicted liked items after, predicted unsure items in the middle, predicted disliked items after, and labeled disliked items at the bottom.

If the participant was not satisfied with the reranking, he/she could click on the *Rerank* button again after making a few adjustments such as selecting more liked items. The participant did not need to be nervous about making false judgments since there would not be any information lost. He/she could even re-click on the *Rerank* button without specifying any new preferences.

During the experiment, we observed that the participant often browsed 2 pages (where each page contained 20 items) before reranking. And the participant usually labeled 5 to 15 items for each of the like and dislike class. The total number of labeled items was usually 10 to 15. The labeling was subject to the result set and the participant's criteria.

Figure 4.2 shows the user interface of system A.

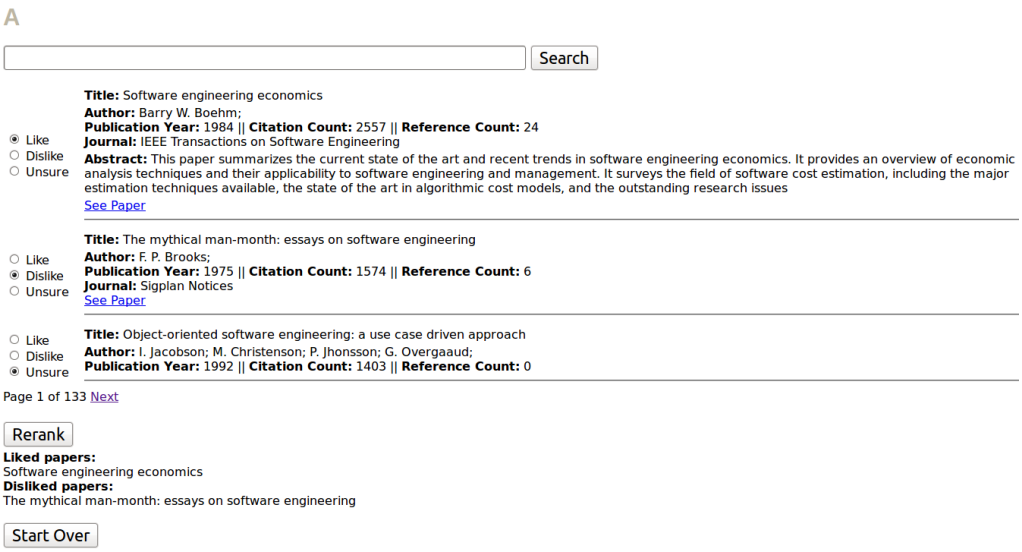



Figure 4.2: System A Layout

**Using system B** After getting the result set, the participant reviewed the default list of items. Participants were asked to keep their criteria consistent throughout the experiment.

Figure 4.3 shows the user interface of system B. The only difference of the user interface between A and B is that B did not support the reranking function. Though B also provided radio buttons, they were designed to remind participants of their paper preferences.

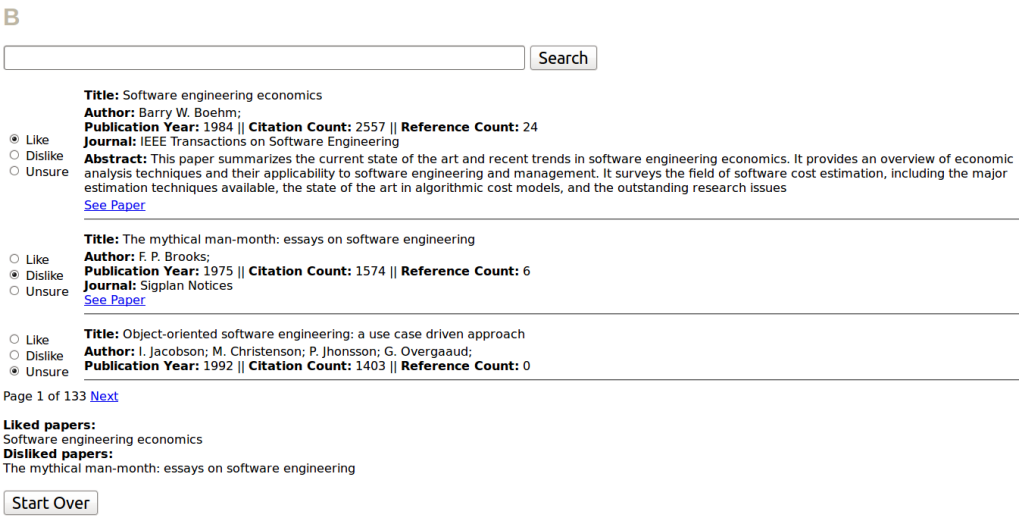



Figure 4.3: System B Layout

**Comparison between A and B** After getting the reranked list in system A and the default list in system B, the participant compared the rankings from the two lists. In order to guarantee a fair comparison, we asked the participants to ignore items that the participant had already labeled as *like* or *dislike* in both systems and only compare items that were not labeled by the participant.

In order to validate our hypothesis, we asked the participant to focus on three properties, *i.e.*, *accuracy*, *coverage*, and *serendipity*. When focusing on the *accuracy* property, the participant compared the rankings of items of interest in both systems. For example, if there were more items of interest among higher ranked items in system A, then the accuracy score for A would be higher. When focusing on the *coverage* property, the participant compared the rankings of query-irrelevant items in both systems. For example, if there were more higher ranked query-irrelevant items in system B, then the coverage score for A would be higher. When focusing on the *serendipity* property, the participant compared the rankings of serendipitous items in both systems. For example, if there were more higher ranked serendipitous items in system A, then the serendipity score for A would be higher.

Since different participants had a different definition of *higher ranked items*, we allowed them to decide on the number of items to be considered as *higher ranked* as long as the number was no greater than 100 (*i.e.*, 5 pages).

We also asked the participant to estimate his/her perceived effort for completing the task in each system, and give an overall score for each system.

**Rating properties** Based on the previous comparison, the participants rated the 5 properties of both systems, *i.e.*, *accuracy*, *coverage*, *serendipity*, *effort*, and *overall*. The values were in the scale of 1 to 5, where 1 meant worst, 3 meant neutral, and 5 meant best.

The participant recorded the scores in the questionnaire. For each property, they could also write down comments, *e.g.*, reasons why he/she thought A or B was good or bad in property *accuracy*. We also recorded the query task (query string and criteria) and information about each participant's familiarity with the searched topic.

When the first task was finished, the participant began with the second task. Each task took between 15 to 20 minutes.

## 4.2 Experimental Result and Analysis

We collected 20 questionnaires in total. Each questionnaire contained a searching task and 10 property scores (where each system received 5 property scores). Participants created 17 of the searching tasks. We found that these tasks covered a wide range of topics (see Table 4.1). The last 3 rows are experimenter created tasks adopted by participants.

Table 4.1: Searching Tasks

| Query | Criteria |
|---|---|
| deblurring | recent publications about motion deblurring |
| large scale data visualization | general application of large scale data visualization, large scale data rendering solution |
| vegetation signal propagation | attenuation microwave, RET model |
| object recognition | approaches of 2 dimension image object recognition |
| NP hard | algorithms for solving NP hard problems, not identifying NP hard problems |
| smoothed analysis | recent development |
| principle component analysis | image related application or theoretical method focused |
| natural language processing | fundamental works, theories, not application |
| approximation algorithms about TSP | recent algorithms |
| combinatorial game | paper with high citation |
| resource mobile ad hoc networks | resource allocation, grid, mobility models, distributed |
| bloom filter network | application of bloom filter in network protocols |
| natural language processing | has statistic as a keyword |
| spatio-temporal data visualization | applications of different spatio-temporal data visualization methods under different domains |
| affiliation extraction from research papers | extract meta-data from pdf research papers |
| document cluster labeling | has labeling as keyword |
| simultaneously localizing and mapping (SLAM) | visual SLAM |
| Hbase | application side of Hbase |
| machine learning | machine learning publication with high citation and reference |
| image processing | new approaches to processing images |

### 4.2.1 Preliminary Processing of Result

For each system, we got its score distribution for each property.

Also, we define the relative score of property $X$ (relative property $X$ score) as follows: In each questionnaire, the relative score of property $X$ equals to the difference between the scores of system A and B in terms of property $X$. A positive relative score of property $X$ means system A outperforms system B and a negative relative score means system B outperforms system A (in terms of property $X$). We got the distribution of relative score for each property.

For example, if one participant's score of accuracy for system A in a questionnaire was 4, and his/her rating of accuracy for system B in the same questionnaire was 2, then his/her relative score of accuracy was 2 $(4-2)$. Furthermore, if the relative score of accuracy equaled to 2 in 4 out of the 20 questionnaires, then the percentage of the relative accuracy score of 2 was 20%.

### 4.2.2 Result Analysis

We analyzed the correlation between different properties, compared the score distribution between the 2 systems for each property, and showed the distribution of relative scores for each property. Lastly, we compared the average and standard deviation of the 5 properties between the 2 systems.

#### Pearson correlation coefficient

We calculated the Pearson correlation coefficient (PCC) of relative scores of different properties as follows:

$$r = \frac{cov(X,Y)}{\sqrt{var(X)}\sqrt{var(Y)}}$$

PCC is a measure of a linear relationship between two variables X and Y [10]. The value ranges from -1 to 1, where the greater the absolute PCC value, the higher the correlation. A positive PCC value means a positive correlation, and a negative value means a negative correlation.

We calculated the PCC for the relative scores in every pair of properties (Table 4.2). For example, if the relative score of accuracy from questionnaire 1 to 20 is X=(accuracy$_1$, accuracy$_2$, ..., accuracy$_{20}$), and the relative score of overall from questionnaire 1 to 20 is Y=(overall$_1$, overall$_2$, ..., overall$_{20}$), then we can use the above equation to calculate the value of PCC for relative accuracy and relative overall.

Table 4.2: PCC Value for Relative Scores of Each Pair of Properties

| **Variate X** | **Variate Y** | **PCC** |
|---|---|---|
| relative accuracy | relative coverage | **0.8227** |
| relative accuracy | relative serendipity | 0.5234 |
| relative accuracy | relative effort | 0.4156 |
| relative accuracy | relative overall | **0.9045** |
| relative coverage | relative serendipity | **0.6908** |
| relative coverage | relative effort | 0.3981 |
| relative coverage | relative overall | **0.7263** |
| relative serendipity | relative effort | 0.2892 |
| relative serendipity | relative overall | 0.5048 |
| relative effort | relative overall | 0.3276 |

We can observe high correlations between some of the properties.

The PCC value of 0.8227 between relative accuracy and relative coverage shows a high correlation between the relative scores of accuracy and coverage. The reason is that if items of interest are ranked higher, items not of interest (which included query-irrelevant items) will be ranked relatively lower. The PCC value of 0.9045 between relative accuracy and relative overall shows a high correlation between the relative scores of accuracy and overall. This result shows that if system A generates higher ranks for items of interest, participants will generally prefer system A. The PCC value of 0.6908 between relative coverage and relative serendipity shows a high correlation between relative coverage and relative serendipity. The reason is that if query-irrelevant items are ranked lower, query-relevant items (which include serendipitous items) will be ranked relatively higher. The PCC value of 0.7263 between relative coverage and relative overall shows a high correlation between relative coverage and relative overall. This result shows that if system A generates lower ranks for query-irrelevant items, participants will generally prefer system A.

### Accuracy

The accuracy property focuses on whether items of interest will be ranked higher. A score of 3 means neutral, and a score greater than 3 means good. We show the distribution of the accuracy scores as a bar chart in Figure 4.4, which allows for better comparison of score numbers. We show the distribution of relative scores of accuracy as a pie chart in Figure 4.5, which allows for intuitive comparison of score percentages. The same goes for other properties. As can be seen from Figure 4.4, the number of score 4 for system A is 11, which means, in more than half of the querying tasks, people feel good about the *accuracy* property of system A, while only in 4 of the querying tasks, people feel good about the *accuracy* property of system B. In Figure 4.5, more than half of the relative scores of accuracy are positive, which means, system A is helpful in ranking items of the user's interest higher in more than half of the search tasks.

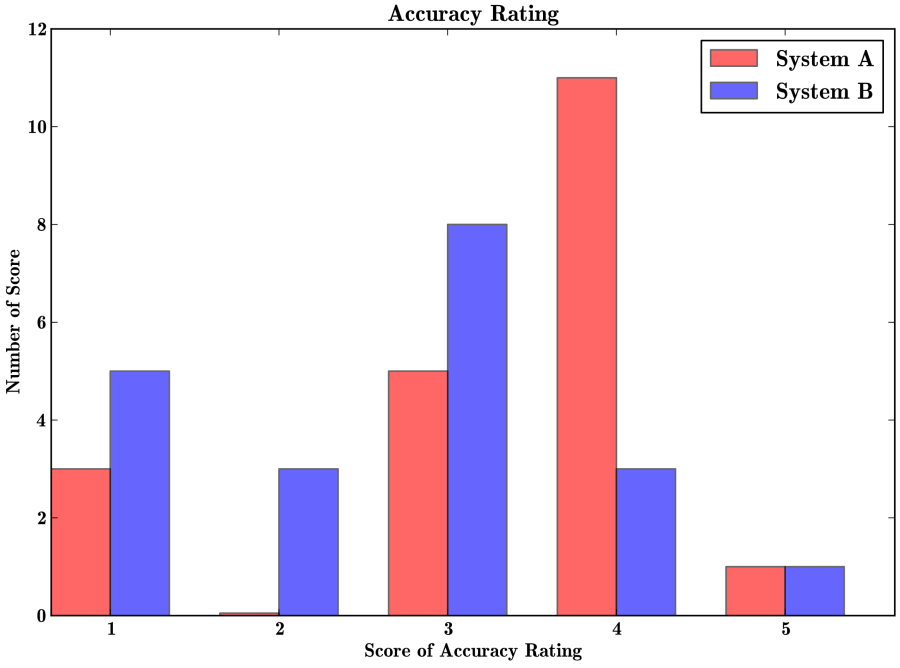



Figure 4.4: Distribution of Accuracy Score

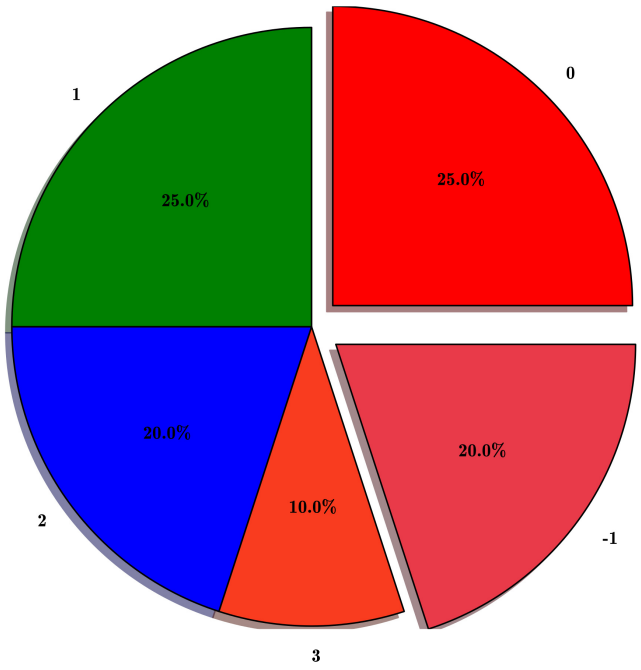



Figure 4.5: Distribution of Relative Score of Accuracy *(pie sector label: relative score)*

However, we also can notice that about 25% of the relative accuracy scores have value 0 and 20% have value -1. This means, in nearly half of the search tasks, system A does not help much or is worse than system B when ranking items of interest.

We investigated into the questionnaires, and found that there were mainly three causes:

- First, poor accuracy happened when the original results were poor. For example, we had seen a participant input a query, but only 70 items returned, most of which were apparently irrelevant. In this case, the classifier could not get enough data for the *liked* class, thus it could not create an effective classifier.

- Second, even if the original results were relevant to the query string, the criteria might constrain the liked items to only a few candidates, which would also make the *unsure* and *disliked* classes dominate, resulting in a poor performing classifier. The classifier might mis-classify the already nearly extinct good candidates and rank them lower, which might make the user unhappy.

- Third, even if the original results were good, and even if there were many liked items, when the criteria could not be represented by the system well, the classifier might not be able to know enough specifics of the user. The reasons varied for this phenomena. For example, if the Feature Construction component failed to represent the right feature of the liked items, the Machine Learning component might make false judgments. We had seen a participant looking for papers with relevancy to microwaves; however, the system mistakenly recommended frequency related papers.

Based on these observations, we conclude that the performance of the reranking is dependent on the original dataset, the user's feedback of *liked* or *disliked* items, and the system's ability to represent the criteria. In order to represent more kinds of user requirements, there should be more features. However, with too many features and too few specified items, the performance of classification may be poor, even with the presence of the dimensionality reduction module.

**Coverage**

Coverage focuses on whether query-irrelevant items will be ranked lower, leaving room for query-relevant items to be ranked higher. In Figure 4.6, we can see that after reranking, in more than half of the questionnaires, users are happy about the coverage property. In Figure 4.7, we can see that in 45% of the questionnaires, the coverage property does not change after the reranking. In 40% of the questionnaires, reranking helps to rank query-irrelevant items lower, while in 15% of the questionnaires, reranking mistakenly ranks query-irrelevant items higher.

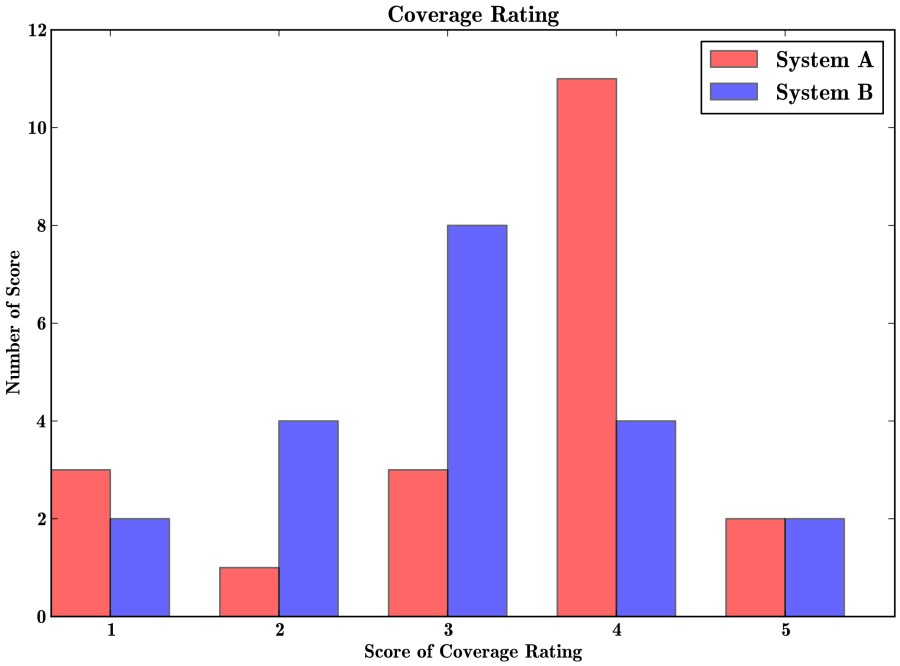



Figure 4.6: Distribution of Coverage Score

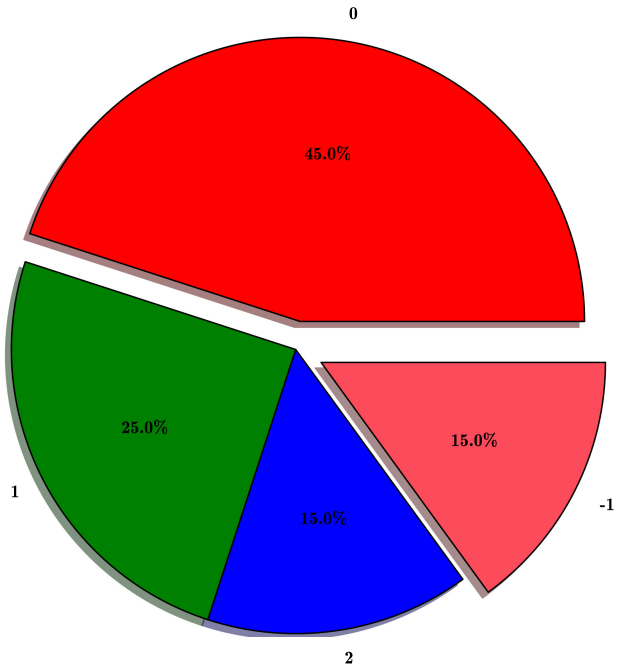



Figure 4.7: Distribution of Relative Score of Coverage *(pie sector label: relative score)*

### Serendipity

The serendipity property focuses on whether serendipitous items will be ranked higher. It is observed that in Figure 4.8, the score of 3 is the most frequent one for both systems. This is not hard to imagine because of the infrequent nature of good surprises. In Figure 4.9, we can also observe that in 45% of the questionnaires, reranking does not help with the serendipity performance. However, we notice that the serendipity score of 4 and 5 for system A is notably better than for system B.

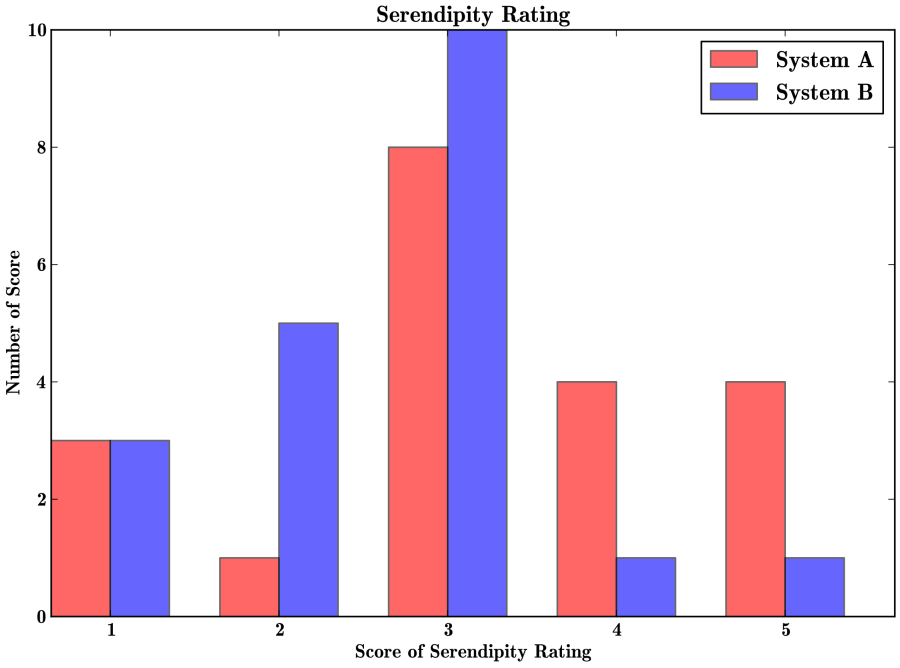



Figure 4.8: Distribution of Serendipity Score

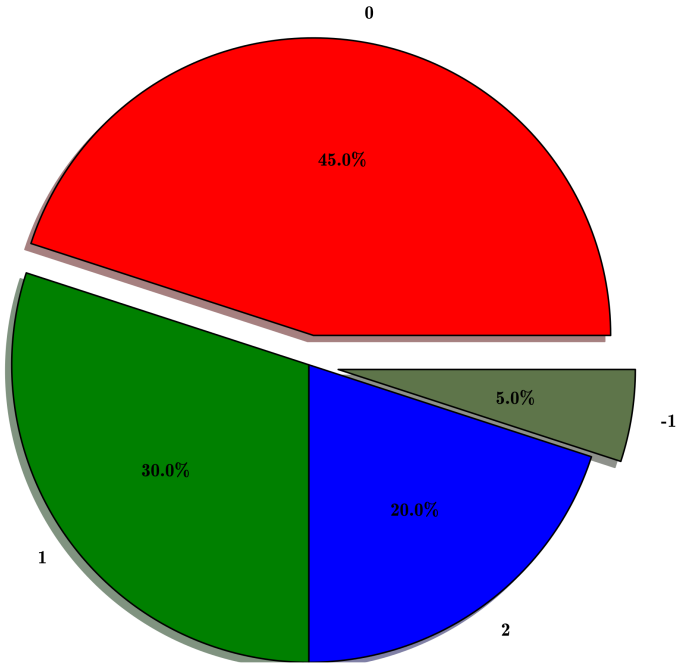



Figure 4.9: Distribution of Relative Score of Serendipity *(pie sector label: relative score)*

### Effort

Effort is a measure of user specific tolerance of extra effort, indicating the users subjective perception instead of objective measurement such as time consumption. It is our expectation that the effort property score will be relatively low for system A, because the user simply has to do more work for a better reranked list. The low values of PCC between the relative effort score and relative scores of other properties confirm this point. However, the effort is less than we have expected. According to Figure 4.10, in more than half of the questionnaires, users give good scores on this property. In Figure 4.11, 75% of participants report no increased effort with system A. We see comments saying that reranking is not hard, that the good reranking is worth the effort, and that it actually feels good to know the system tries to learn from the user.

In real world academic search, users tend to click on the items of interest, which can be seen as a manual labeling process. Though our system requires users to click on the item explicitly to specify a preference, it does not require particularly extra

work, since they are doing something similar anyway. They will potentially get a better ranked list from the effort. A typical user will view 3 to 5 pages of search results and select around 10 papers of interest or not of interest before reranking.

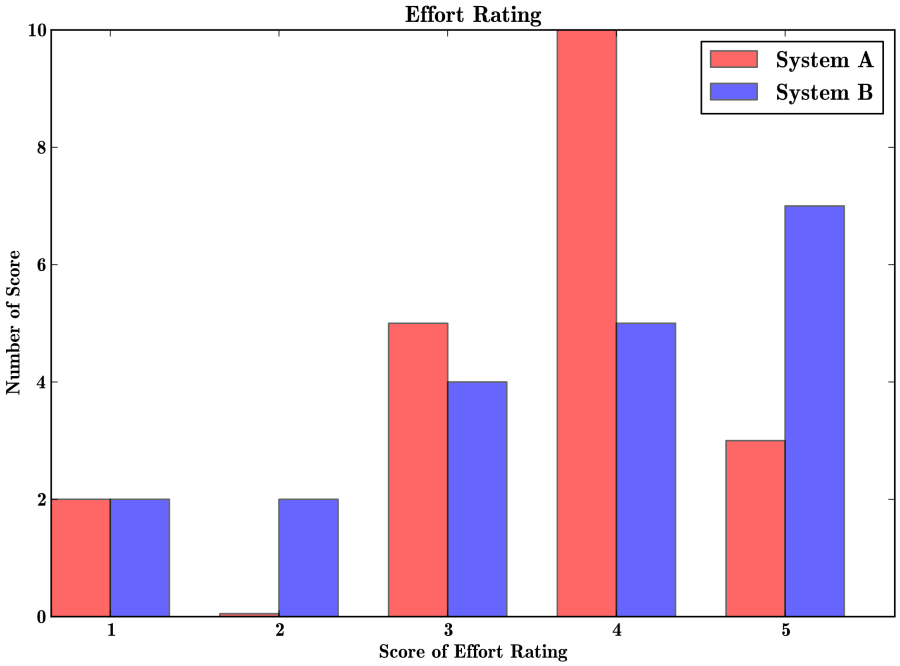



Figure 4.10: Distribution of Effort Score

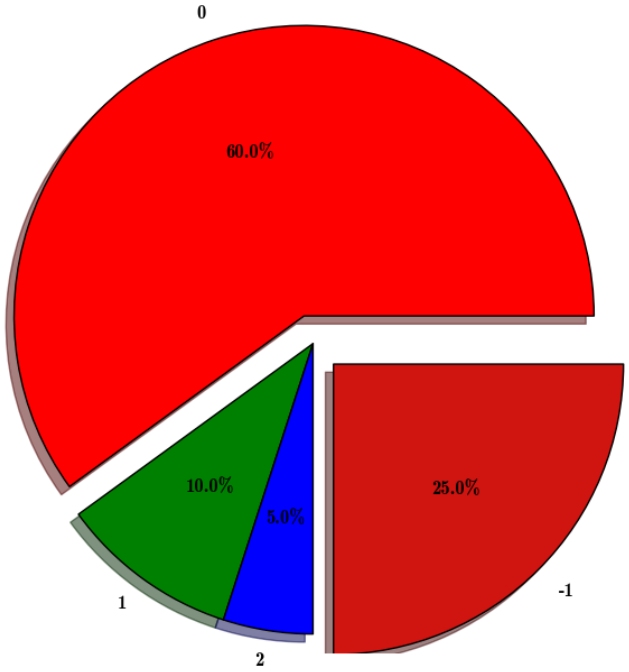



Figure 4.11: Distribution of Relative Score of Effort *(pie sector label: relative score)*

### Overall

We received very encouraging overall ratings. As seen in Figure 4.12 and 4.13, In 60% of questionnaires, participants prefer system A over system B. As mentioned previously, we can see a high correlation between overall and accuracy and coverage, which implies that a better ranking from system A can make people prefer system A. This is especially true in areas where people would like to devote effort to search for something interesting. The reranking functionality ranks higher items of interest that would otherwise be hidden in further pages.

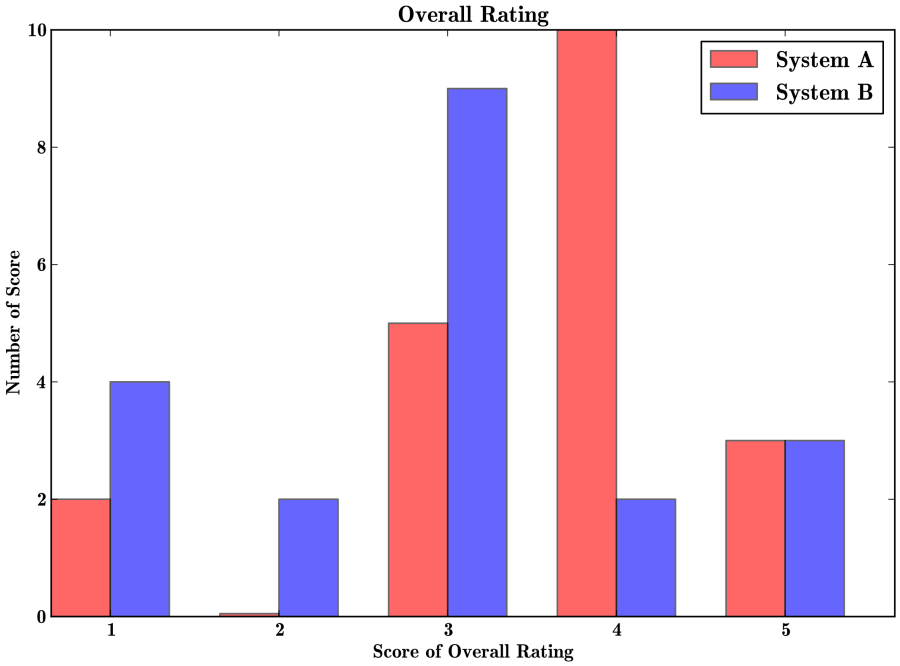



Figure 4.12: Distribution of Overall Score

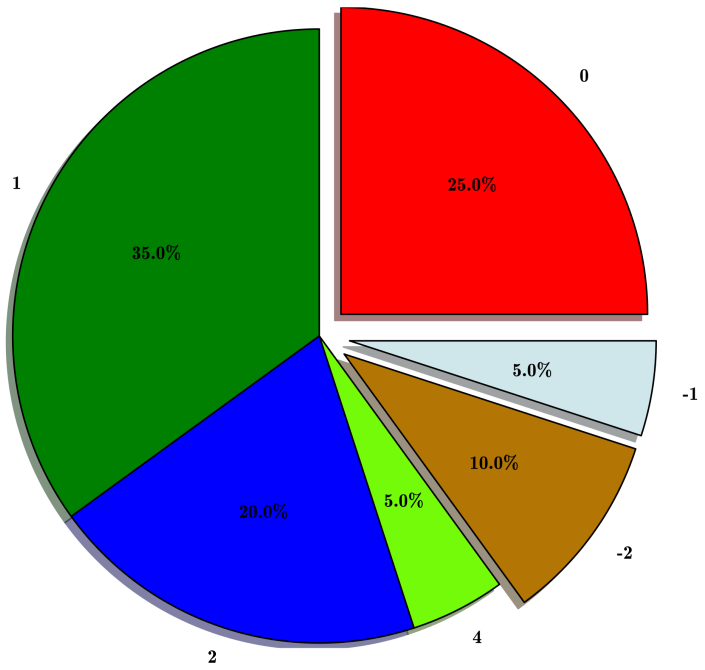



Figure 4.13: Distribution of Relative Score of Overall *(pie sector label: relative score)*

### Average Comparison

Figure 4.14 shows the average score and standard deviation of each property for both systems. We can observe that system A outperforms system B in the *accuracy*, *coverage*, *serendipity*, and *overall* properties, , and ties on *effort* property.

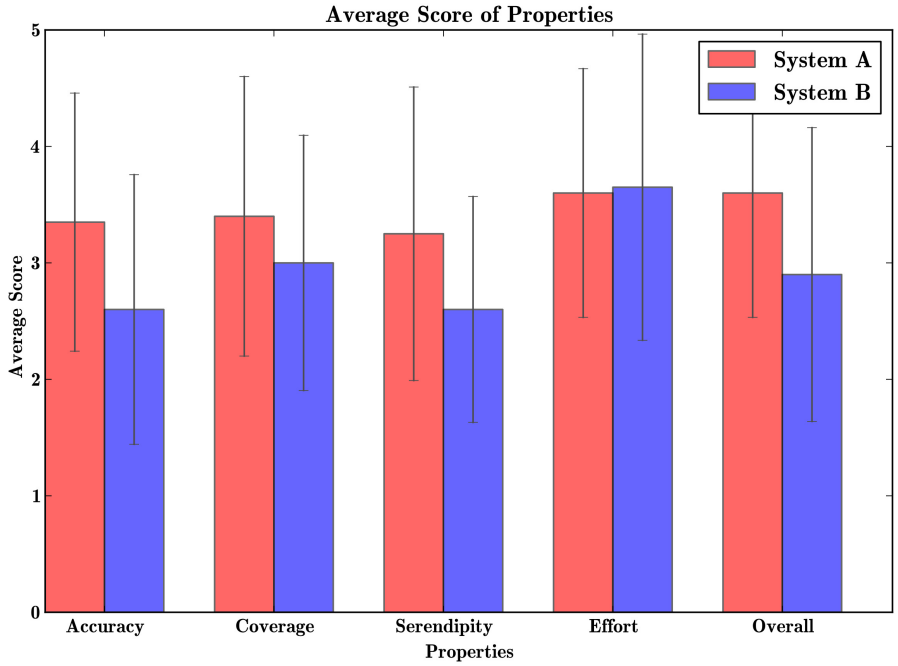



Figure 4.14: Average Ratings of Properties

Based on the above results and analysis, we validate our hypothesis that our personalized search engine (system A) outperforms the ordinary search engine (system B) in the *accuracy*, *coverage*, and *serendipity* properties. Also, the *effort* property for our search engine is on par with the ordinary search engine. The *overall* property for our search engine also outperforms the ordinary search engine.

## 4.3 Discussion

Personalized search can help making the search results more of interest to a user. As we can see from Figure 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, and 4.14, system A outperforms system B in *accuracy*, *coverage*, and *serendipity* properties. These properties are concerned with whether items of interest are ranked higher (*accuracy*), query-

irrelevant items ranked lower (*coverage*), or serendipitous items are ranked higher (*serendipity*). Thus, on these properties, the ranking is improved after personalization.

The performance of personalized search is subject to the original search results. We observed in the experiment that MAS did not always return good initial results. For example, in one particular search task created by a participant, "document cluster labeling" was the query string. There were around 500 results returned. However, no more than 10 papers had the keyword *labeling* in the whole dataset. In this case, the personalized search was unable to find more papers related to document labeling. Moreover, as the size of the class of *liked* papers was significantly smaller than the other two classes, the performance of the classifiers would be poor, which might mis-classify papers of interest that are already rare and annoy the user.

The performance of personalized search is subject to the actual user requirements. We observed in the experiment that our system could not interpret the actual intentions of users at times. For example, a participant searched "vegetation signal propagation" with the goal of finding papers emphasizing attenuation microwave and RET model. However, our system misinterpreted this as a preference for papers related to frequency, since microwave was a high frequency wave. We checked the log and found that it was because frequency and microwave were grouped in the same topic in the topic distribution, which meant the features were not fine grained enough to discriminate the two terms. Moreover, our system currently only supports a limited number of types of features, thus the user requirements that can be possibly addressed are also limited.

The performance of personalized search is also subject to the interaction between the user and the system. During the experiment, we noticed that inconsistent criteria, *e.g.*, the intended criteria was high citation yet the actual liked list contained several papers with no citations at all, would confuse the classifiers, which would generate unwanted ranking. If one class was too dominant, *e.g.*, the user selected only one *liked* paper, leaving the rest to be *unsure*, the performance of the classifiers would be poor since it could not identify what features the user was really interested in.

Better accuracy performance may come with the trade-off of effort. But if the reranked list is good, the users may be more tolerant with effort, and vice versa. We observe that the *effort* scores of the two systems are almost a tie, and only 25% of the questionnaires report more *effort* in system A. While it is arguable that this is only true in the experimental setting, we believe that some simple clicking will not require a noticeable amount of extra effort. The process is intuitive, but the reranked list may be surprisingly good, which may delight the user. Some comments say that it feels good to know that the system tries to fulfill his/her requirements, and that the system finds a paper of interest that would otherwise be hidden in the bottom pages (see Appendix E).

We also find that the reranking speed is so fast that it usually takes no more than 1 second to get the reranked list after the *Rerank* button is clicked. Compared with the time it takes to fetch the data from MAS, which is around 10 seconds, this time duration is tolerated by participants.

To conclude this section, depending on the initial dataset, a user's requirements, and his/her interactions with the system, our system can make the ranking more of the user's interest, with just some simple clicks and a small amount of time. The user can save a lot of page-flipping time when our system ranks initially lower ranked items of interest on the top pages.

## 4.4 Threats to Validity

The experiment shows the value of our system. However, there are threats to the validity of the experiment.

- The designer of the system is the experimenter of study. In this case, the experimenter may be biased when designing and conducting the experiment. For example, the experimenter created search tasks which might favor system A and not be representative of general tasks. In order to decrease this bias, we gave participants freedom to create their own search tasks. In the end, there were only 3 experimenter created tasks adopted by the participants. Also, in our experiment, we tried to keep everything the same between system A and

B. For example, we let each participant finish two tasks using the systems in alternate orders to reduce any potential bias caused by the order.

- The group of participants is limited. It is likely that if we invite more participants, the experimental results may be different. It is possible that people with different backgrounds will think differently about our system. We will evaluate our system on more people with more different backgrounds in the future.

- The user study has its inherent limitations. Different people have different rating scores for the same system performance. Thus we used the relative property rating score to reflect a participant's preference between system A and B. Moreover, a participant may prefer system A since it is a new system that people would feel interesting at first, which would cause bias. Thus we explicitly asked the user to be true to his/her actual preferences.

- The user interface may cause bias. Since we implemented our own user interface, the original MAS user interface is not supported. However, the user's rating towards the 2 systems may be influenced by the interface. For example, if both systems used the MAS user interface, the advantage of system A may be diluted by the convenience brought by the MAS user interface, which may influence the user's rating.

- The choice of data source may cause bias. As we have mentioned before, the performance of our personalized search engine is subject to the original search results. Although MAS often provides satisfying initial search results, there are still many research areas for which MAS does not provide enough papers.

- Manual comparison may cause bias. In our experiment, the participant needs to manually ignore labeled items, and manually compare the ranking of items of interest, query-irrelevant items, as well as serendipitous items. The manual process is error-prone and may cause bias. In our future experiment, we can make this process automated or semi-automated. First, we can change our

implementation in the experimental setting so that the labeled items can be eliminated from the result set for both systems. Thus, the participant does not have to manually ignore labeled items to guarantee the fair comparison. Second, we can implement a function for participants to specify the category of an item, *i.e.*, item of interest, query-irrelevant item, or serendipitous item, so that the system can compare the rankings between the 2 search engines for each category automatically. However, the participant still needs to manually specify the category of an item, which may not be accurate and cause bias.

- There is no standard or optimal solution when deciding how many pages should be viewed before reranking, or how many pages should be considered during the comparison. The number depends on the individual, the task, and the initial results. For example, different people may view the same result differently since they have different habits. This uncertainty is another source of bias. In our future experiment, we may try to set rigid numbers. This may not show the best performance of any of the systems; however, the potential bias caused by this form of uncertainty can be decreased.

# Chapter 5

# Related Work

There are a number of established and ongoing research in the personalized search area. And there are mainly four approaches to this research problem. The first approach is to use explicit user input to search, filter or sort items. The second approach is to build a user model to provide additional information and personalize the searching results. The third approach is to leverage user relevance feedback to improve search result. The fourth approach is to combine some or all of the above approaches to boost personalization performance.

## 5.1 Explicit Input

Simple, yet useful, explicit user input can help to accurately retrieve items of interest that match the conditions of the user input. There are mainly three types of user input information, *i.e.*, query string, filtering conditions, and ranking criteria. Depending on the area of search, there are different specific kinds of user input fields.

Take the publication search in the academic area as an example. Google Scholar [4], Microsoft Academic Search [5], and CiteSeer [1] provide advanced search functionalities such as searching for a query string in different publication fields (*e.g.*, title, abstract, author.), filtering according to publication years, venues, research domains, *etc.*, and ranking the result according to relevance, citations, or publication date. The retrieval result is the intersection of all items that meet the above conditions. ScienceDirect [6] further provides a functionality that some of

the conditions can be combined using one of the three logic operators, *i.e.*, *AND*, *OR*, and *NOT*. The retrieval result thus is the certain combination of intersections and unions of items that meet the above conditions. The advantages of these search engines are that the searching criteria is clear and specific, making the search results mostly relevant to these criteria. However, some of these advanced search functionalities are neither easy to learn nor to use. Sometimes, a user needs to see what items are out there before he/she gets a bit more clear about what his/her criteria may look like, and these criteria may not be fully expressed by the explicit input fields alone.

Faceted DBLP [3] is more intelligent in a sense that beyond traditional searching and getting results back to a user, it does a simple post-search analysis of the original search results by grouping items according to different criteria. For example, papers can be grouped together if they are published in the same conference, or written by the same author. In this way, a user can view the original results first, identify which groups he/she likes, and filter out papers outside of groups of interest. The grouping information provided by the post-search analysis makes the searching task easier since it comes up with filtering suggestions and eases the pain of coming up with and typing in these filtering criteria. However, the post-search analysis is still too simple and is limited to only criteria such as publication conference and year. Sometimes, a user may have more advanced requirements, *e.g.*, looking for theoretical papers that do not necessarily contain "theoretical" in the title or abstract fields. This kind of requirement cannot be addressed by the above approach. In our system, however, we can analyze the paper entries and apply user knowledge, which is more capable of addressing the user's requirements.

## 5.2 User Model

Building a user model is a popular approach to personalization. A user model can be built from the user's historical interaction with the system, *e.g.*, submitted query strings and corresponding viewed items, browser caches, even local desktop files. The model is usually represented as a document collection in common practice, and

reveals possible interests of the user.

Many different ways have been explored to leverage the user model. Pretschner and Gauch [30] try to monitor user web browsing behaviors to construct a user interest list, which can be used in reranking, filtering, or query expansion. Liu *et al.* [27] model and gather a user's search history, construct a user model based on the history, and construct a general profile based on the ODP (Open Directory Project) category hierarchy. Then they deduce appropriate new query categories from the user model and general profile, improving the web search by using the categories as a context. Ding and Patra [16] also try to generate a category in response to a user's query; however, they use a self-organizing map to cluster the user's search history into categories. Martinez *et al.* [17] uses machine learning methods to identify the user's intended category and incorporate it with user specified interest to form a user model, which is used to support their decision making and personalization engine.

These techniques are built on the assumption that the current user intention is always consistent with the user model. However, in real world scenarios, it is common to have ad-hoc queries that are significantly different from the user's history. Moreover, personalization only using the user model needs to deal with the *cold-start* problem, *i.e.*, when the user first uses the system, there may not be as much user information available to build a rich user model. Thus we think that using a user model alone cannot deliver useful personalization performance. Instead of maintaining a user profile, our system focuses on inferring user intention for each query through the interaction, making the system more flexible for ad-hoc needs. Nevertheless, the system is extensible to incorporate a user profile component, which can also be used for personalization [39].

Since we mostly use a user model to process existing search results (*e.g.*, reranking, filtering), or perform query expansion (*e.g.*, by adding context information to the old query), the concern of transferring personal information to the web is minimized.

## 5.3 Relevance Feedback

Relevance feedback is another popular approach to achieve personalized search.

It is recognized that a user is still capable of identifying items of relevance even if he/she does not know how to specify the query terms. [33]. This leads to leveraging user relevance feedback to improve search results. Relevance feedback can be regarded as a simple short term user model, since the relevance feedback indicates the user's current preferences.

There are mainly three types of feedback: explicit feedback, implicit feedback, and pseudo feedback [18]. In explicit feedback, a user explicitly specifies his/her preferences towards items. The feedback accurately reflects a user's opinion, yet the user needs to explicitly specify the opinion. While in implicit feedback, user opinion is inferred from his/her behaviors, such as mouse clicking, and time spent viewing a document. There are no obtrusive user inputs required, however, the user opinion might not be reflected as accurately. Surf Canyon Search uses the implicit feedback approach to personalize results of existing search engines, *e.g.*, Google and Bing, based on the user's browsing behaviors [9]. Pseudo feedback holds the assumption that the top ranked items in the original result is of great relevance. Thus using them as items of relevance can improve the original result. This approach does not need any human involvement; however, the performance may not be stable if the top ranked items are actually irrelevant.

Different methods have been designed to leverage relevance feedback. The most popular one is query expansion. The goal of query expansion is to revise the original query string to include terms that have higher weights in items of relevance. The updated query is sent to the server to request a new set of results, which is supposed to be more relevant since the query string contains more terms of interest. Much research has been done to find the best strategy to update the query string. The fundamental work is the Rocchio algorithm [34]. It uses a vector space model to represent both the query string and the items. The algorithm tries to find a new query string that maximizes the difference between the average vector of the relevant documents and the average vector of the irrelevant documents. More com-

prehensive query expansion methods can be found in Ruthven and Lalmas's survey [33].

There are also other methods to leverage relevance feedback. Allan [11] applies feedback incrementally and selectively to decrease the space and time overhead, while making the system adaptive to drifting user interests. Hijikata [21] studies the mouse operations of the user, analyzes the potential user interest, uses information from mouse operations to update the query, and gets better performance compared with a traditional *tf-idf* approach. Sheth [35] combines a genetic algorithm with relevance feedback. In this approach, different agents are created and used to recommend different items. Agents recommending items of relevance have higher health points and better chance of generating offspring.

In our approach, we also use relevance feedback. Our difference is that we use feedback as labels of items, and try to identify the shared attributes for each class, which represents the user preferences of the current search. Moreover, different from most approaches in query expansion that only uses textual information (*e.g., tf-idf*), our approach is able to leverage other available features, such as topic distribution and community number, which contains more domain knowledge and is more powerful in addressing user requirements.

## 5.4 Hybrid Approach

Personalization can be achieved with combined methods. Research has been done in the hybrid of different approaches.

Teevan *et al.* [38] combines a user model with relevance feedback to use the user interest information to rerank the original search result. They modify BM25 [37], a probabilistic weighting scheme that ranks documents based on their probability of relevance given a query, such that relevance feedback is incorporated. The modified version can compute the similarity of a query to a document, taking into account the relevance feedback influence. Moreover, the relevance feedback is calculated from the user model, which is implicitly built, thus the effort for the user is minimized. Elegant as it is, the personalization is still biased towards historical records, making

it less effective with new ad-hoc queries. Since the whole approach is built on a vector space model, the approach cannot adopt domain knowledge as what our system does.

Luxenburger *et al.* [28] try to match a task profile with user needs, such that the system can identify whether the user is making a new ad-hoc query or a historically similar query. They cluster the historical search tasks from user model, cluster search results from the current query, and try to find the two most similar clusters from the two different fields. If their divergence is greater than a certain threshold, then the query is identified as a new query, so the personalization should not be biased towards historical records. While this approach helps to alleviate the ad-hoc querying problem, it still needs to cope with the cold start problem, and lacks support for non-textual features.

## 5.5 Discussion

Different approaches emphasize different personalization aspects. For example, user model based approaches are more fit for longer term searching environments where the queries are more stable, while relevance feedback approach emphasizes more on current search goals.

While our approach is also more focused on current search goals, it provides a very flexible integration mechanism with other approaches, *e.g.*, user model based approaches. To be precise, it can be built on top of any search engine that provides lists of search results. Moreover, our approach is flexible in supporting more domain knowledge, which the user may require.

# Chapter 6

# Conclusion and Future Work

In this chapter we first summarize the contributions of this thesis, then we discuss future directions for the personalized search, followed by lessons learned during this research.

## 6.1 Conclusion

Personalizing search engines can better assist users in addressing their individual requirements. In this thesis, we proposed an interactive and iterative approach to improve the searching result of existing search engines by analyzing user feedback using machine learning techniques. We first construct a feature space from acquired data, allow users to select items of interest, then use machine learning techniques to train classifiers that reflect user preferences, and rerank a new list of items with improved relevance for further user interaction. The interaction happens iteratively to better converge to the user's requirements. We apply the approach in the academic search domain, explore a variety of techniques to build the paper's feature space and build a pool of classification algorithms that can dynamically select the best classifier for the current user selection. Moreover, we evaluate the above personalized academic search with real users, collect their ratings towards the original search and our personalized search in terms of several properties. The result shows users favor personalized search. Last but not the least, our system follows modular design principles, making it extensible with different algorithms and potentially applicable to various search domains.

## 6.2 Future Work

There are three directions for the future. Firstly, we will continue to improve the personalized publication academic search. Secondly, we plan to extend the application to other areas of academic search, such as searching for relevant experts (*i.e.*, researchers) or venues (*i.e.*, conference or journal). Thirdly, we will explore applying this approach to other domains such as web search or commodity search.

### 6.2.1 Improving Publication Academic Search

Our system has personalized the default ranking provided by the original search engine; however, there are still possible ways to improve the user's searching experience. Since our system is not a replacement for the original search engine, our approach can be seamlessly combined with other customization techniques. For example, some academic search engines, such as Faceted DBLP, provide explicit filters, *e.g.*, publication conference, to narrow down the results. While these filters are accurate to narrow down the results, they can not address some features such as topic distribution. We can combine the strengths of both explicit filtering and our implicit inference by building our approach on top of the filtering approach. In this case, the user will specify the filters before the search, then provide feedback to the backend system, and rerank after he/she gets the initial results. We can combine our approach with other relevance feedback techniques such as query expansion [33] in the same way.

There are also other techniques [26] with which our approach can be combined. We previously [39] proposed the use of a personal sphere that could record a user profile to support personalized search. We analyzed a data model for papers to add domain knowledge, and designed a ranking module to rank items by relevance. The personal sphere is useful when we need the user's background information, *e.g.*, to do the personalization. The analyzed data model precomputes the paper features, which can address more user requirements. The ranking module uses information from the personal sphere, and can help to rank items of the same class to make the ranking even more personalized. We can seamlessly incorporate these modules

into our existing system. We show the concept in Figure 6.1, where the add-on components are in shaded boxes.

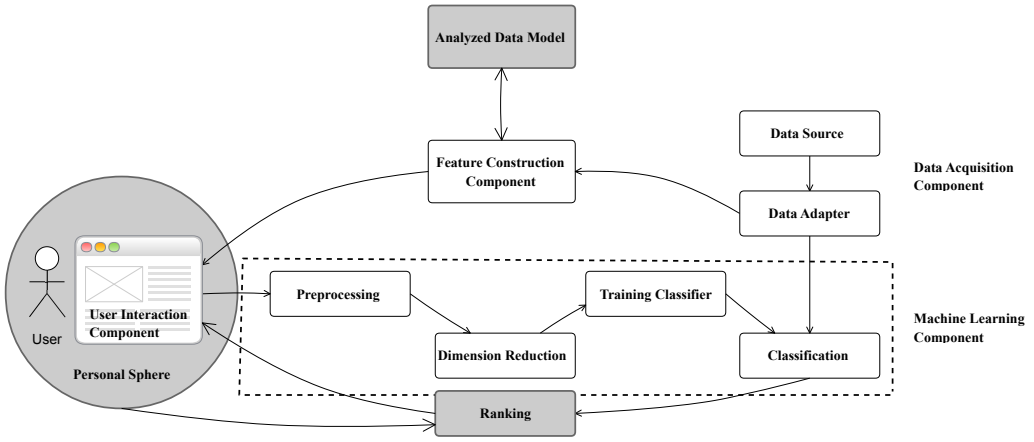



Figure 6.1: System Design with Addon

There are also some user interface enhancements suggested by the participants. For example, the user interface can be more intuitive by changing the radio button of like and dislike to figures of thumb up and thumb down. Also, we can store a user's historical classifiers to use in similar future search tasks, which helps to save the time and effort of clicking. Moreover, we can change the result set by including references and citations of papers of user interest. This can potentially improve the quality of the result set since if a paper is interesting, its references and citations will possibly be also interesting. We can do so by using the relationship-exploring functionality of the MAS API. At the algorithmic level, we can try the ensemble method [7] to implement the classifier, and compare with our current approach. We can try sampling techniques [14] to reduce the impact of dominant class(es). We also need to further evaluate our system as mentioned in Chapter 4.

### 6.2.2 Extending to Other Academic Search Areas

In the academic search area, searching for relevant experts (*i.e.*, researchers) or venues (*i.e.*, conference or journal) would naturally complement searching for relevant papers. The MAS API also supports author and venue search, which are similar to the publication search functionality, except that the returned entities are authors and venues. When applying our personalization approach to these searches, we

need to customize the Feature Construction component to represent possible user requirements when the user is doing author or venue search. For example, the user may be interested in the institution of the author. Then we need to include the institution ID as a feature. And the pool of classification algorithms should be able to deal with categorical features in this case.

### 6.2.3 Exploring Other Search Domains

Academic search is an important activity in academia. However, in our daily life, we are more involved with other kinds of searches, such as web search and commodity search. More often than not, we encounter a situation where we input our query words, get tens and hundreds of pages of results which are stuffed with some popular items surely not of our interest. There is no way to filter out this information to leave room for items of our interest, and searching with more specific query strings will possibly miss important items in other aspects. Thus, the user can do nothing but to flip through pages and scan through each page to not miss any valuable item in a sea of uninteresting items. If we apply our approach to these searches, the users can simply specify items not of interest, and the system can automatically rank those items at the bottom of the result list, leaving room for items of more interest.

Microsoft Bing Search[1] provides an API for developers to use the functionalities of Bing to search. However, as the items in the results of web search are much less structured, it is much harder to get common features than in academic search. We may start with simple features such as topic distribution then explore other features that can potentially help to address the user requirements.

## 6.3 Lessons Learned

After implementing the personalized search approach and experimenting it with real users, we find this approach helpful, easy-to-use, easy-to-implement, extensible, and can be built on top of existing search engines. It would be an interesting feature

[1] http://www.bing.com/

for MAS to incorporate. And since MAS has the full data warehouse of publications, it can perform more powerful data analysis and construct more features that can represent more user requirements. With the user accounts at Microsoft, MAS can mine user profiles both individually and globally upon the user's approval. The profile can further help with the personalization.

We suggest users to keep their requirements consistent when using our personalized search engine. We also suggest our users to try to keep the size of each class close by exploring new pages or new search queries to build a better classifier.

# Bibliography


[1] CiteSeer. http://citeseerx.ist.psu.edu/.

[2] Django Project. https://www.djangoproject.com/.

[3] Faced DBLP. http://dblp.l3s.de/.

[4] Google Scholar. http://scholar.google.com/.

[5] Microsoft Academic Search API User Manual. http://academic.research.microsoft.com/about/Microsoft%20Academic%20Search%20API%20User%20Manual.pdf.

[6] ScienceDirect. http://www.sciencedirect.com.

[7] Scikit-learn. http://scikit-learn.org/stable/.

[8] Source Code. https://haiming1@bitbucket.org/haiming1/workspace.git.

[9] Surf Canyon Search. http://www.surfcanyon.com/.

[10] Per Ahlgren, Bo Jarneving, and Ronald Rousseau. Requirements for a cocitation similarity measure, with special reference to Pearson's correlation coefficient. *Journal of the American Society for Information Science and Technology*, 54(6):550–560, 2003.

[11] James Allan. Incremental relevance feedback for information filtering. In *Proceedings of the 19th International SIGIR Conference on Research and Development in Information Retrieval*, pages 270–278. ACM, 1996.

[12] R.E. Bellman. *Dynamic Programming*. Dover Books on Computer Science Series. Dover Publications, 2003.

[13] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, March 2003.

[14] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *arXiv preprint arXiv:1106.1813*, 2011.

[15] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Generalized Louvain Method for community detection in large networks. In *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications*, pages 88–93. IEEE, 2011.

[16] Chen Ding and Jagdish C Patra. User modeling for personalized web search with self-organizing map. *Journal of the American Society for Information Science and Technology*, 58(4):494–507, 2007.

[17] E Frias-Martinez, G Magoulas, S Chen, and R Macredie. Automated user modeling for personalized digital libraries. *International Journal of Information Management*, 26(3):234–248, 2006.

[18] Gregory Gay, Sonia Haiduc, Andrian Marcus, and Tim Menzies. On the use of relevance feedback in IR-based concept location. In *Proceedings of the 25th International Conference on Software Maintenance*, pages 351–360, 2009.

[19] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: an overview with application to learning methods. *Neural computation*, 16(12):2639–2664, 2004.

[20] Douglas M Hawkins. The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004.

[21] Yoshinori Hijikata. Implicit user profiling for on demand relevance feedback. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, pages 198–205. ACM, 2004.

[22] David W Hosmer Jr and Stanley Lemeshow. *Applied Logistic Regression*. John Wiley & Sons, 2004.

[23] Walter L. Hrsch and Cristina Videira Lopes. Separation of concerns. Technical report, Northeastern University, 1995.

[24] Alan Julian Izenman. *Linear Discriminant Analysis*. Springer, 2008.

[25] Ian Jolliffe. *Principal Component Analysis*. Wiley Online Library, 2005.

[26] Xiaoxiao Li, Jiyang Chen, and Osmar Zaiane. *Text Document Topical Recursive Clustering and Automatic Labeling of a Hierarchy of Document Clusters*. Springer, 2013.

[27] Fang Liu, Clement Yu, and Weiyi Meng. Personalized web search for improving retrieval effectiveness. *IEEE transactions on Knowledge and Data Engineering*, 16(1):28–40, 2004.

[28] Julia Luxenburger, Shady Elbassuoni, and Gerhard Weikum. Matching task profiles and user needs in personalized web search. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 689–698. ACM, 2008.

[29] Thomas Dyhre Nielsen and Finn Verner Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2009.

[30] Alexander Pretschner and Susan Gauch. Ontology based personalized search. In *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, pages 391–398. IEEE, 1999.

[31] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[32] Juan Ramos. Using tf-idf to determine word relevance in document queries. In *Proceedings of the 1st Instructional Conference on Machine Learning*, 2003.

[33] Ian Ruthven and Mounia Lalmas. A survey on the use of relevance feedback for information access systems. *The Knowledge Engineering Review*, 18(02):95–145, 2003.

[34] Gerard Salton. The smart retrieval system—experiments in automatic document processing. 1971.

[35] Beerud Sheth and Pattie Maes. Evolving agents for personalized information filtering. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications*, pages 345–352. IEEE, 1993.

[36] Ya Tang. The design and study of pedagogical paper recommendation. 2008.

[37] Jaime Teevan, Christine Alvarado, Mark S Ackerman, and David R Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 415–422. ACM, 2004.

[38] Jaime Teevan, Susan T Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th Annual International SIGIR Conference on Research and Development in Information Retrieval*, pages 449–456. ACM, 2005.

[39] Haiming Wang and Kenny Wong. Recommendation-assisted personal web. In *Proceedings of the 1st International Workshop on Personalized Web Tasking at the 9th World Congress on Services*. IEEE, 2013.

[40] Haiming Wang and Kenny Wong. Personalized search: an interactive and iterative approach. In *Proceedings of the 2nd International Workshop on Personalized Web Tasking at the 10th World Congress on Services*. IEEE, 2014.

[41] Zhang Xuegong. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica*, 26(1):32–42, 2000.

# Appendix A

# Information Letter

**Study title** Searching for paper with different requirements, an interactive and iterative approach

**Background** Dear participants, we want to invite you to participate in this research because you have research experience in computing science. And you have the potential need to use our improved paper search engine. We would like you to try it to help us get some feedback and evaluate our system. Thanks!

**Purpose** The purpose of this research is to come up with a new way of interaction between researchers and academic search engine to satisfy different requirements.

**Study procedures** Participants(You) will fulfill 2 paper searching tasks. You can either choose your own task or pick an example from the sample list (see questionnaire). Each task is about searching for papers in a computing science research area. Before the task, we will ask you about your knowledge level in that area. During the task, you will use our provided search engines to search for papers. After the task, we will ask you about your preferences towards different search engines in terms of accuracy, coverage, serendipity, effort, and overall. We only record the experimental data in terms of questionnaire, user-system interaction data. The whole process takes 30 to 40 minutes. The experiment is conducted in-person. The research investigator will be there to provide you with non-biased help were there any questions.

**Benefits** Participants(You) can experience a different kind of search engine, knowing how you can use it to help with your paper searching activities. We can release our service to public to let more people use it. Participants(You) can expect refreshments during the experiment, which is the cost of the experiment.

**Risk** Please be prepared to do a 35 minutes experiment

**Voluntary participation** Participant can withdraw at any time during the experiment without any penalties. If you do so, we will automatically destroy any recorded data from you. Participant reserves the right to ask us to destroy his/her experiment-related data 2 weeks after the experiment.

**Confidentiality and anonymity** We do not ask for participants' personal information, we only collect experiment-related data. None of the recorded data will have any identifiable information directly or indirectly. The electronic data will be encrypted in a folder, and will be written in a writable DVD. The DVD and paper documents (interview notes and questionnaires) will be kept by professor Kenny Wong for five years. After five years, the DVD and paper documents will be destroyed.

**Further information** The plan for this study has been reviewed for its adherence to ethical guidelines by a Research Ethics Board at the University of Alberta. For questions regarding participant rights and ethical conduct of research, contact the Research Ethics Office at (780) 492-2615.

# Appendix B

# Consent Form

The Study of Searching for paper with different requirements, an interactive and iterative approach

I, _____, hereby agree to participate in the above research conducted by Haiming Wang.

I have read the information letter and I understand that the experimental result may be used for publications. I understand that the research investigators will only record the experiment-related data, which includes questionnaire, in-person interview notes, user-system interaction data. I understand that my personal identifiable information will not be recorded in any means directly or indirectly, my experimental data will be kept securely for five years, after which all of the data will be destroyed, and I have the right to protect my identity from being revealed.

I know that I am not obliged to participate in the research, and I can withdraw at any time without any penalty of any kind. If I withdraw, my experimental data will be destroyed automatically. I understand my benefits/risks. I reserve the right to ask the research investigators to destroy my experimental data.

Name of participant: _____

Signature of participant: _____

Name of researcher: _____

Signature of researcher: _____

Date: _____

# Appendix C

# Experimental Protocol

**Hypothesis** We assume that when researchers search for papers, they have either clear criteria, such as author, topic, that can be clearly addressed by current academic search engines using filters or keywords matching, or unclear, subjective, changeable criteria or combination of criterion that are hard to be represented. Thus we calculate a feature space representing these criterion and use a new way of interaction to infer the underlying user criteria. We hypothesize that our personalized search engine can make the ranking of the result list more of the user's interest.

**How the experiment is run** We call for participants, mostly graduate students with research background in computing science. Each participant will perform 2 paper searching tasks in different research areas. Participant can either create their own task or pick one from the questionnaire. The tasks will be conducted in person. The research investigator will stay with the participants during the experiment to set up the system, train the participant, give helps, and ask specific questions regarding to the experiment. The user interaction with the system will be recorded, the questionnaires will be collected, in-person interview notes will also be collected.

**Full sequence of phases for each participant** Firstly, each participant will need to read and sign all related documents, information letter and consent form. Secondly, each participant will read the questionnaire instructions, get trained with the system to get familiar. Thirdly, each participant will choose and perform 2 searching tasks. After finishing each task, each participant fills in the questionnaire, and

go to the next task. During this period, the research investigator will be there to provide non-biased help and ask experiment-related questions. Lastly, after the participant finishes both tasks, the research investigator will collect consent form, questionnaire, in-person interview notes and user-system interaction data.

**Timing of each phase of a run** Each task takes about 15 to 20 minutes. The participant is expected to finish both tasks in 30 to 40 minutes.

**User training or practice tasks** The research investigator will teach participants how to use the system, how to get most of the system, and what the experiment process is. They will also practice with the example task provided in the questionnaire.

**How many formal tasks** 2

**How you will analyze/use the data** For each task, we collect the user's rating of accuracy, coverage, serendipity, effort, and overall properties for our personalized system and the ordinary system. For each property, we calculate its distribution of scores for both systems. We also calculate the distribution of relative score of each property and analyze the correlation.

**How you will protect the data and anonymity of users** We do not ask for participants' personal information, we only collect experiment-related data. None of the recorded data will have any identifiable information directly or indirectly. The electronic data will be encrypted in a folder, and will be written in a writable DVD. The DVD and paper documents (interview notes and questionnaires) will be kept by professor Kenny Wong for five years. After five years, the DVD and paper documents will be destroyed. A participant reserves the right to ask us to delete all his/her experimental data during or after the experiment. Any future published paper will not contain any identifiable user data either.

# Appendix D

# Questionnaire

**Instructions**

1. Please read the information letter and sign the consent form.
2. Research conductor will introduce you how to use the system.
3. Please create two search tasks, or choose from the table below.
4. Please be aware of what the three options: like, dislike, and unsure, means.
5. Please try to be consistent with your criteria.
6. Feel free to ask the research conductor any questions.

**Example Searching Tasks List**

| Queries | Criteria |
|---|---|
| Hbase | Application side of Hbase |
| Natural language processing | Application of natural language processing |
| Image processing | New approaches to processing images |
| Social network analysis | New approaches in social network analysis |
| Community mining | Community mining techniques, applications, *etc.* |
| Recommender systems | Collaborative filtering approaches of recommender system |
| Machine learning | Machine learning publication with high citation and reference |

**Query:** _____

**Criteria:** _____

**Your familiarity with this topic (familiar/unfamiliar):** _____

Please rate A and B search engines according to the following properties (Score 1 to 5: 1 means worst, 3 means neutral, and 5 means best)

| **Parameters** | **A search engine** | **B search engine** |
|---|---|---|
| Accuracy (items of interest ranked higher) | | |
| Comments: (*e.g.* Why do you think A/B is good/bad in terms of accuracy?) | | |
| Coverage (query-irrelevant items ranked lower) | | |
| Comments: (e.g. Why do you think A/B is good/bad in terms of coverage?) | | |
| Serendipity (serendipitous items ranked higher) | | |
| Comments: (*e.g.* Why do you think A/B is good/bad in terms of serendipity?) | | |
| Effort (perceived effort during the search) | | |
| Comments: (*e.g.* Why do you think A/B is good/bad in terms of effort?) | | |
| Overall (general preference) | | |
| Comments: (e.g. Why do you think A/B is good/bad in terms of overall?) | | |

# Appendix E

# Comments From Participants

| Comments | Explanation |
|---|---|
| A gives me papers from other fields such as medication, biology. | preference in B in accuracy since A contains some irrelevant papers |
| A showed many useful results after rerank, the results in B is less useful, even though most of them are related to the query. | preference in A in accuracy since A's results are more tuned towards him/her |
| Most of papers shown in A are highly related to what I am interested in, the results in B are related to my query, but not as interesting as those in A. | preference in A in accuracy since A's results are more personalized |
| A showed a few results that are interesting and unexpected, B failed to eliminate old papers which are uninteresting. So there are few unexpected but useful results. | preference in A in serendipity |
| B is as simple as google. For A, I need to rate on some results, so not as easy as B, but it's still easy because it only costs a few clicks. | preference in B in effort yet mentions A is also easy |
| Even though A cost me some extra time to rate a few publications, it helped to eliminate unuseful results, so in general, it saved my time. | preference in A in overall |
| B gives more related paper. | preference in B in accuracy |
| A shows results that could be used after affiliation extraction. | preference in A in serendipity |
| A gives more serendipity. It is good to have a tool like this for research. | preference in A in serendipity |
| A has less duplicated inaccurate results and more close results of general application of large-scale data visualization. | preference in A in accuracy |
| A and B both find a useful but unexpected result but A ranks it higher. | preference in A in serendipity |

| | |
|---|---|
| Both are easy to use. It's no trouble to click a few more buttons. | same preference in effort |
| A can sometimes help us refine the result to some extent. In some cases, we may not think of the right search term to search in B. | preference in A in overall |
| The reranking helps find more relevant papers. | preference in A in overall |
| Without the keywords I was looking for. | low preference in both A and B |
| A filters out unrelated topics like 3D data or visual cortex. | preference in A in accuracy |
| A can help me find certain papers that otherwise will hardly appear if using B, since some papers do not use the common terms related to the interested topic. | preference in A in overall |
| After two pages of reranking, results become very good. | preference in A in accuracy |
| Some really interesting theoretical papers are pulled up by A, even if I didn't give much feedback. | preference in A in accuracy |
| A recommended another Chinese NLP paper based on my previous choices while I can't find that paper in the first 5 pages of B. | preference in A in accuracy |
| Feel comfortable that the machine is trying to fulfill my requirement. | same preference in effort |
| A is better because it learns to give better results. | preference in A in overall |