FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Enhancing a Job Recommender with Implicit User Feedback

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Matthias Hutterer
Matrikelnummer 0526201

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Ao. Univ. Prof. Dr. Jürgen Dorn

Wien, 21.06.2011

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Matthias Hutterer
Aggsbach 7, 4655 Vorchdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21.06.2011 _____

(Unterschrift Verfasser)

Abstract

Recommender systems assist individual users in finding the right items in large option space. Absolventen.at, an Austrian job board for graduates, uses such a system for recommending appropriate jobs to applicants. So far, this system has only considered the resume as input for the user profile, which is compared with the available jobs. However, only around half of the registered job seekers fill out the resume, for the other half no personalized recommendations can be generated. To improve this, the recommender system has been enhanced with implicit relevance feedback and the impacts of this approach have been examined in this thesis. Implicit feedback can be captured in an unobtrusive way and allows the system to infer user preferences. Four different user actions for implicit feedback have been identified on Absolventen.at, including reading of a job description, bookmarking, applying and searching for jobs. All of them provide different levels of evidence for interest, as an application is a more reliable indicator for interest than just reading a job description, which is taken into account with individual weighting parameters. In addition to that, gradual forgetting factors are used for adapting the profile over time. All of this information is included in the hybrid user profile, which is represented as hyperdimensional vector and calculated by a linear combination of the resume and the preferred jobs. To evaluate the new approach, the preferred jobs of 46 job seekers were compared with the recommendations. The results show that including implicit feedback helps to increase the user coverage, as well as the accuracy of the recommendations.

Kurzfassung

Recommender Systeme unterstützen Benutzer die richtigen Artikel aus einer großen Auswahlmöglichkeit zu finden. Absolventen.at, eine österreichische Jobplattform für Absolventen, verwendet ein solches System, um Bewerbern passende Jobs vorzuschlagen. Bis jetzt wurde nur der Lebenslauf als Bestandteil des Benutzerprofils, welches mit den verfügbaren Jobs verglichen wird, betrachtet. Jedoch füllen nur rund die Hälfte aller registrierten Benutzer einen solchen Lebenslauf aus, für die andere Hälfte können keine personalisierten Empfehlungen generiert werden. Um dies zu verbessern, wurde das System mit implizitem Relevanz Feedback erweitert und die Auswirkungen dieses Ansatzes wurden in dieser Diplomarbeit untersucht. Implizites Feedback kann in einer unauffälligen Weise aufgezeichnet werden und ermöglicht dem System Benutzerpräferenzen abzuleiten. Vier unterschiedliche Benutzeraktionen konnten auf Absolventen.at für implizites Feedback identifiziert werden, darunter Lesen einer Stellenbeschreibung, Hinzufügen von Lesezeichen, Schreiben von Bewerbungen und Suchen nach Jobs. Jede dieser Aktionen liefert unterschiedliche Evidenzen für Interesse. So ist eine Bewerbung ein zuverlässigerer Indikator für Interesse als nur das Lesen einer Stellenbeschreibung, was durch unterschiedliche Gewichtungsparameter berücksichtigt wird. Zusätzlich werden graduelle Vergessensfaktoren verwendet, um das Profil über die Zeit zu adaptieren. All diese Informationen werden im hybriden Benutzerprofil miteinbezogen, welches als hyperdimensionaler Vektor repräsentiert und mit Hilfe einer Linearkombination aus dem Lebenslauf und den bevorzugten Jobs berechnet wird. Für die Evaluierung des neuen Ansatzes wurden die bevorzugten Jobs von 46 Jobsuchenden mit dem Empfehlungen verglichen. Die Ergebnisse zeigen, dass implizites Feedback hilfreich ist, um sowohl die Benutzerreichweite als auch die Treffergenauigkeit der Empfehlungen zu erhöhen.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Since the late 1990's a clear shift from traditional to online-based recruitment methods has taken place. The Internet offers many opportunities to the recruitment process, from which both the companies as well as the job seekers benefit. The still increasing popularity comes, among other things, from the cost and time saving advantages [17], [27], [34].

The importance of online recruiting has also been underlined in a recent survey in Germany, including the top 1.000 companies. According to that, 87% of the job offers are published on the company's website, 61% on job boards and only 20% in the print media. Moreover, most job seekers apply through electronic channels and 71% of the hired persons are a result of online recruitment [57].

Although the Internet has become the main channel for recruitment and many new possibilities have been facilitated, some challenges remain unsolved. From the job seeker's point of view, still several open positions have to be read for identifying the most interesting ones. As thousands of job offers are spread over the Internet, and the capabilities of keyword-based search engines are rather limited for the complex job matching, which includes factors like region, education, skills, salary and so on, the job seeking process can still be a tough task.

Recommender systems have emerged in the E-Commerce domain and are one way to address this issue. Based on the needs of individuals, recommenders assist them in finding the right items [50]. In the online recruitment domain such systems can be used to analyze the job seeker's preferences and capabilities and to point them to the most appropriate jobs. According to numbers of CareerBuilder.com, an international recruiting platform, job seekers write three to four times more applications for recommended jobs [9].

Absolventen.at, an Austrian job board for graduates, uses such a job recommendation service. The website serves as umbrella brand for the niche portals Uni-/ FH-/ HTL-/ HAK-Absolventen.at and is based on the Content Management System Drupal. As other job boards, it allows job seekers, on the one hand, to fill out their resume and companies, on the other hand, to publish their job ads.

For finding the best matching job, the job recommender takes the user profile as input and compares it with the available job ads. The user profile currently consists of the resume, which is explicitly provided by the user. Typically such a resume contains information like the education and skills, representing the capabilities of the job seeker. Furthermore information from a taxonomy, which describes concepts like the skills and fields of study, enhances the matching process.

However, only the half of the registered users fill out their resume. Consequently the current implementation is unable to reach a wide range of users, as it requires the resume of the users to work. When looking at a typical job board, a lot can be learned from user actions, e.g. if a user applies for a job, the system can assume, that this job is highly interesting for the user and hence recommending jobs similar to this one sounds reasonable. Besides applying for a job, user actions like searching for a job, adding a job to the bookmarks or reading of a job description can be monitored and provide so called *implicit user feedback*, which allows the system to infer preferences. This feedback can be observed in an unobtrusive way, without any further user intervention, which is one major advantage compared to explicit feedback methods. Adding this information will lead to more comprehensive user profiles and will help the recommender system to cover a wider range of users. Furthermore, the quality of recommendations will probably increase as well, something which has yet to be evaluated.

This evaluation will be performed after architecting and implementing the new job recommender. In order to tell whether the new approach produces more accurate recommendations than the other, real user profiles from Absolventen.at will be analyzed and compared. For the comparison, it will be checked if the recommendations would also match with the user's preferred jobs. To have a set of preferred jobs per user, parts of the applications and bookmarks from the past will be taken, since it is assumed that these jobs are highly relevant for the user. As this data should also be used as input for the new job recommender, it has to be split into mutual exclusive training and validation data sets. The training set is passed as input to the recommender, while the validation set is used for counting the matches. The more recommendations match with the known preferred jobs, the better, something which can be measured and assessed with classical information retrieval evaluation metrics, like the precision and the recall.

To sum up, the aim of this thesis is to examine what kind of implicit user feedback can be captured from the job board website and how it affects the recommendations. Different levels of evidence for interest have to be considered, e.g. applying for a job

gives higher evidence for interest than just reading a job description. For evaluating the impacts of this approach, a prototype of a job recommender, which combines explicit user input as well as implicit feedback, will be developed and evaluated in this study.

The outline of this thesis is organized as follows: in the first part the theoretical foundations for building a recommender will be discussed. This includes methods for constructing user profiles, as well as various recommendation techniques. In addition to that, some concrete examples for job recommenders will be given. Afterwards, the paper guides through the complete process of architecting a job recommender and especially focuses on methods for integrating implicit user feedback. This part is followed by information on the implementation, and in the last section, an evaluation between the existing and the enhanced implementation will be performed.

CHAPTER 2

State of the Art

In this chapter the foundations for building a job recommender are going to be presented. For this purpose, a literature research in the field of recommender systems was conducted, covering the important topics of user profiling, various recommendation techniques and some concrete job recommender examples, which will be discussed in detail after a short definition of recommender systems.

Recommender systems are used on many E-Commerce websites, helping individuals to find the right items in a large option space. For that several different techniques exist and in order to provide personalized recommendations, knowledge about the user's interests is needed. These interests can be gathered from item ratings, a purchasing history or other interactions and are stored in the user profile [51].

2.1 User Profiles

User profiles contain information about individual users, enabling computer systems to adapt their behavior specifically to the user's needs. The first approaches of user modeling date back to 1979, when the need for personalization and distinct treatment of individuals has already been underlined [44].

As mentioned, recommender systems also need such profiles for personalizing the suggestions. In this case, a user profile may contain demographic information, like the age, country or education of a user or some interests and preferences, like which articles the user rated. Recommender systems exploit these profiles and, based on various techniques, try to find matching items or users. The user profiling step is the key task for recommender systems, as the quality and success of the systems depend on the correct representation of the user's interests [15], [32].

Figure 2.1 shows the three main phases of constructing and exploiting user profiles. At the beginning, in the *data collection* phase, information about the interests of the user is gathered. The data collection can either be done explicitly or implicitly. In an explicit data collection mode the user tells the system what items he likes through direct intervention, e.g. by rating a special item. In contrast, in implicit data collection, the system monitors the user's behavior and uses this information for inferring the preferences.

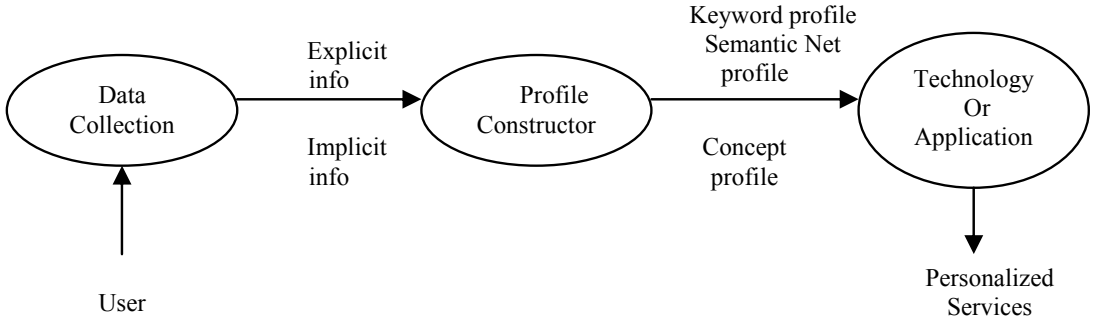


Figure 2.1: User profile-based personalization phases [15]

Afterwards, the *profile construction phase* transforms the collected information into an appropriate *profile representation*. The representation can be done in some form of weighted keywords, concepts or semantic networks and expresses the interests and preferences of a user. The construction of the profile depends on the representation and usually relies on machine learning or information retrieval techniques. The generated profiles may be *static*, meaning that the data is not going to be updated after the initial construction, or *dynamic*, where the profiles are adapted as the interests of a user may change over time. Dynamic profiles may furthermore distinguish between *short-term* and *long-term* interests. Long-term ones are collected over a longer time span and don't change that often. In opposite, short-term interests are related to a specific task a user performs and are likely to change regularly. For example, searching for a flight for the upcoming holidays represents the short-term interests of a user. After the holidays ended, this information is not relevant any more and the user continues reading the daily articles about IT, representing the long-term interests [15].

As soon as the final profile representation has been constructed, in the third and last phase a *technology or application* exploits the data to recommend appropriate items to the user [15].

Based on a detailed evaluation of recommender systems on the internet, Montaner et al. [32] present five design decisions for building and maintaining user profiles. These de-

sign decisions are similar to the phases shown in figure 2.1, but use a more fine-grained classification. The user profiling task starts with the definition of the *profile representation*, which needs to be tackled first, as every other step depends on it. In the data collection phase, first an *initial profile generation technique* is needed. Then *relevance feedback* is used for adding preferences, either from explicit or implicit sources. For constructing the profile, a *profile learning* algorithm uses all collected information and finally a *profile adaption technique* helps to maintain and adapt the profile over time.

Later on, these two models will build the foundations for architecting the user profiles for the job recommender. For that a more comprehensive analysis of the most important tasks, the profile representation, the data collection and the profile construction, will be presented in the following chapters.

2.1.1 Profile Representation

The profile representation is responsible for storing the user's current interests in an appropriate way. An expressive representation, which is able to capture the interests and preferences as detailed as needed, is a key factor for the success for a recommender system. The choice of the right schema is tightly coupled to the profile learning and recommendation technique being used. Several different representation schemas were identified by Montaner et al. [32] and Gauch et al. [15]:

- *History-based model*
  In a History-based user model, various user actions, like purchases, are stored. These actions give feedback, either positive or negative, and are later on used as input for the recommender.
- *Vector space model / Weighted keywords*
  The vector space model (also know as *weighted keywords* [15]) is one of the most common user profile representations. The profile is represented as a vector containing some features, which are usually keywords from a text and express some kind of interest. Each feature has a weight associated, which defines the degree of interest. The higher the value, the more the user is interested in.

  Several extensions to the vector based model exist:

  - *Concept profiles* contain concepts of a reference ontology or taxonomy.
  - *Weighted semantic networks* use weighted arcs between re-occurring concepts or keywords. The weights specify the level of interest and the connections allow to maintain the meaning of the words.

  - *Weighted associative networks* are similar to weighted semantic networks, but in contrast, they only have one generic relation type for grouping nodes into phrases.
  - *Weighted n-grams* store links between words with a fixed length of *n* characters, which enables the system to keep the context of words.

- *Classifier-based models*
  Classifier-based models rely on the generated structure of a classifier being used as profile learning technique. The resulting structure can be based on neural nets, decision trees, association rules and Bayesian networks.

- *User-item ratings matrix*
  Collaborative filtering recommenders compare users. One way of doing that is a comparison of user ratings. For this purpose a matrix containing the ratings of the users for each item is generated and used for the similarity calculation.

- *Demographic features*
  Another way of comparing users may rely on demographic data of a user. Such demographic features are stored as a list in the user profile. For building profiles based on demographic features, or an user-item ratings matrix, no profile learning techniques are necessary.

2.1.2 Data Collection

In this phase we want to gather as much information about the user's needs as possible. For this purpose various information sources exist, e.g. the user can explicitly define the interests after the registration has been completed. This data can be used for a so called *initial profile* [32], which helps to recommend appropriate items from the beginning on. Such *manual* systems require time and interaction with the users for filling out a form, usually a task they are not motivated to do. Furthermore, it is often difficult to precisely describe the own interests. Other, more automated systems may generate *stereotypes* based on demographic data or infer an initial profile based on the user's ratings on a pre-defined *training set*.

Besides the initial profile, most recommender systems rely on information from evaluations, where the users tells which items are relevant and which ones not. This process is usually referred as *relevance feedback*, originally introduced in the mid 1960s in the information retrieval area, used to refine searches [48]. The feedback can either be obtained *explicitly* through user interventions or observed by various user actions in an *implicit* way [32].

Explicit Relevance Feedback: In an explicit feedback mode, the system asks the user to evaluate items. Evaluations are usually done by some kind of rating system, like a numeric rating scale, a binary like / dislike button or textual comments. Textual comments require additional interpreting on whether the rating is positive or negative [32].

Such rating systems have the big advantage of simplicity: they can be easily integrated into the systems, are normally well understood by the users and provide precise evaluations of items. Additionally, several experiments showed that explicit feedback mechanisms resulted in good performance [48]. Nevertheless, these systems suffer from a serious problem: users are often not motivated to rate items, resulting in a sparsity of evaluations [23]. This is primarily caused by the need of user's intervention and time effort, without enough direct benefit for the user.

Implicit Relevance Feedback: To overcome the sparsity problem of explicit feedback methods, observations of user interactions can be used to infer whether a document is interesting or not, a method called implicit feedback [44]. Many different actions, like the user reads an article for a certain time, buys a product or bookmarks a webpage, can be monitored in an unobtrusive way, without any additional intervention and effort by the user. For example, purchases indicate interest in this item and therefore positive ratings can be inferred. Of course, the collected data might be noisy and such approaches are not as precise as explicit evaluations, e.g. long read times do not always indicate high ratings. Some of the drawbacks will be discussed later on, but first let's take a closer look on the observable behavior, which can be used for implicit feedback.

For this purpose a basic classification was published by Oard et al. [37] and extended by Kelly et al. [22]. The classification, which can be seen in table 2.1, is based on two dimensions: the behavior category and the minimum scope. The behavior category groups various similar actions, whereas the minimum scope defines the level at which the action can be monitored. A *segment* scope represents just a portion of an object, e.g. a paragraph of an article, the *object* column requires the whole item and the *class* column a collection of objects. Actions can also be observed at larger levels, but not on lower ones, e.g. a user can view a portion of the object, the whole object as well as a collection of objects. In the *examine* behavior category, we have actions like *viewing*, *listening*, *querying*, *selecting* and *browsing* through documents.

Viewing an object is a very common source for implicit feedback. Various studies showed that users viewing an article for a certain amount of time also tended to mark this article as interesting [52]. Morita et al. [33] found out that items with read times higher than 20 seconds were likely to be relevant, Konstan et al. [23] stated longer

Table 2.1: Classification of observable behavior [22]

Behavior Category \ Minimum Scope	**Segment**	**Object**	**Class**
Examine	View Listen Scroll Find Query	Select	Browse
Retain	Print	Bookmark Save Delete Purchase Email	Subscribe
Reference	Copy-and-paste Quote	Forward Reply Link Cite	
Annotate	Mark up	Rate Publish	Organize
Create	Type Edit	Author	

read times for highly rated items, but confirmed the correlation between read times and positive feedback.

Querying and *finding* actions are difficult to classify. First, they are used to locate interesting information for examination. But for performing these actions, the user needs to enter search query words. These words can be used as feedback as well and hence classifying these actions into the *create* category is also reasonable.

Actions in the *retain* category, like *printing*, *bookmarking*, *saving* etc., assume that the user will later on make use of the items. For example, users normally bookmark objects if they want to have a fast and direct access to the item for later viewing. The *deleting* action indicates that an item is less important than other retained objects and thus allows the inference of negative feedback. The *reference* category contains actions, which create some kind of links between objects. For example, *replying* can be used in a messaging system and establishes a link between the original and the new message. Such links, which can as well be added with *quotations* and *citations*, create a useful network of related objects and can be used for inferring relevant objects. *Annotations* add some extra information to an object, e.g. by *marking* a paragraph or *rating* an item.

As previously already mentioned, the rating action is usually seen as explicit relevance feedback, but this classification rather wants to give an overview of possible observable behavior than distinguishing between explicit and implicit feedback. The *publishing* action makes the object accessible to larger community and therefore adds some value to object, whereas the *organize* action is used to order a collection of objects, useful for faster access in the future. In the last category, we have actions that *create* content, for example someone *authors* a new article. Information from the written content can then be added to user profile for suggesting similar items.

This classification should be seen as an example for observable behavior and should help developers to identify and assess possible user actions. Further, different classifications are possible and the available actions and their naming are application-specific.

After the user actions have been monitored, the stored data has to be analyzed in order to infer correct preferences. According to Oard et al. [37] two different approaches for recommenders based on implicit feedback exist and are shown in figure 2.2.

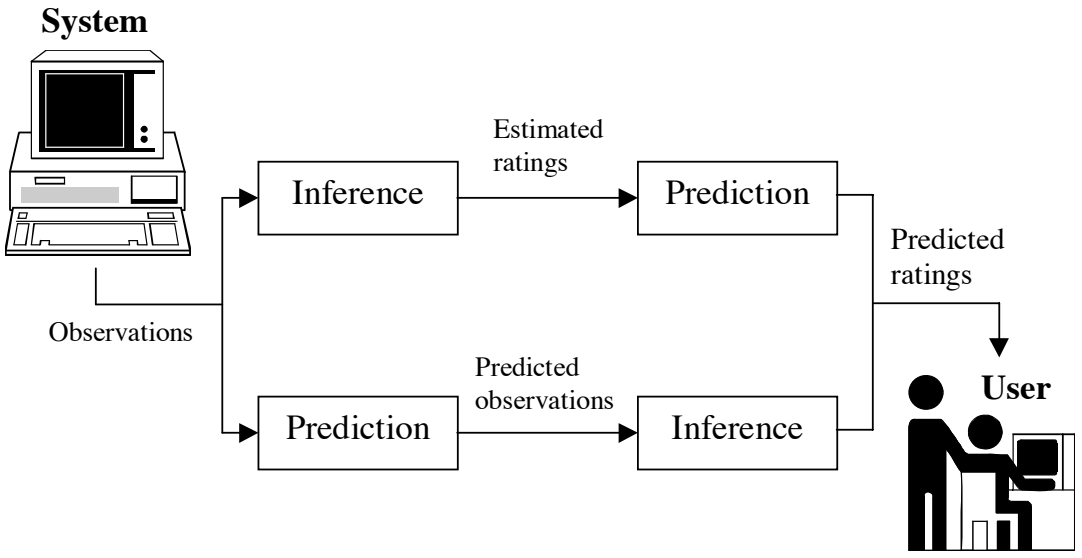


Figure 2.2: Recommendation alternatives based on observed behavior [37]

The first *Inference - Prediction* approach aims to generate ratings based on implicit feedback, for example the longer a user reads an article the higher ratings it receives, a strategy implemented by Konstan et al. [23]. After the ratings have been estimated, they are used as input for the prediction stage, as it would have been done with explicit

ratings. The prediction stage calculates appropriate items and recommends them to the users.

In opposite, the *Prediction - Inference* approach, tries to predict the behavior of users based on observed behavior from other users. The inferencer uses the predicted behavior in order to calculate the interestingness of new content. A content-based implementation of this approach exists, where the read time of new articles is estimated based the read times of similar articles. Afterwards a search query that would find the articles with long estimated read times is generated and applied [54].

Implicit feedback overcomes one of the most serious drawback of explicit feedback techniques: no additional effort of the user is needed, because the system just observes the user performing certain tasks on the website. As shown in table 2.1, many different behaviors can be considered for implicit feedback. As a result of that, much more feedback can be collected than by explicit methods. Furthermore, user monitoring can be easily implemented within a web-application, where all data is already available and no further client side installations are necessary [37].

Besides these advantages, personalization applications using implicit feedback have to take into account that the quality of received feedback might be lower [36]. Someone might just forget to close the browser window, resulting in long read times and positive feedback, although in reality, the user might not rate the item highly. Nevertheless, several studies in [33] and [23] showed that replacing explicit ratings by implicit feedback from read times resulted in similar accuracy, although both experiments weren't executed in real world environments [36]. In another experiment by [10], the accuracy of 80% with explicit feedback was reduced by 10% when relying on implicit feedback. Apart from that, different levels of evidence should be considered, e.g. purchasing gives more evidence for interest than just reading an article [22]. Furthermore, most observable actions provide positive feedback. Even though adding of negative feedback resulted in better performance in experiments by [20], it is usually much more difficult to obtain, as people tend to do things they like and ignore uninteresting stuff.

Hybrid Models: As the studies above indicate, there is no clear evidence whether implicit or explicit data collection provides more accurate results than the other. However, experiments with personalized search applications underlined that the more information a profile contained about the user, the better the application performed [56].

Hybrid user models make combinations of information received from explicit and implicit sources, leading to more comprehensive profiles. With this approach, the advantages of both data collection techniques can be combined, like lower user effort with observed behavior and higher accuracy with explicit evaluations [32]. The data from different sources can be combined in several ways, either both are added to the profile

or one feedback is used as validation for the other one, e.g. feedback from implicit sources validates explicit ratings [36].

2.1.3 Profile Construction

In the profile construction phase the collected information needs to be transformed into the profile representation, e.g. based on the relevance feedback, an interest vector needs to be calculated.

For this purpose *profile learning techniques* [32] are required, which often rely on *information retrieval* or *machine learning* techniques. As mentioned before, in some cases such techniques are not necessary: either the representation is based on a rating matrix, demographic features or on a simple history-based model, where all data is directly available in the database.

For building vector- or semantic network-based profiles, the most important terms of relevant documents are extracted and added to the profile. To identify the most important terms, often a pre-processing of the documents with unstructured text is necessary. *Information retrieval techniques* analyze the text, extract keywords and build a vector containing the keywords with associated weights, which denote the importance of a term. The weights are usually calculated with the TF-IDF (Term Frequency - Inverse Document Frequency), which is shown in equation 2.1. The term-frequency (TF) counts how many times a word appears within a document, whereas the inverse document frequency (equation 2.2) reduces the weight of terms, which occur in many documents. When searching for documents, usually a information retrieval application wants to achieve high recall, meaning that as many relevant documents as possible are actually retrieved, and high precision, where the proportion of relevant documents within the result is as high as possible. To increase the recall, broader terms with a high term frequency help, whereas on the other hand, more specific terms are better for the precision. For example, if someone searches a term that is included in every document, maximum recall is achieved, but it doesn't necessarily mean that all of them are really useful for the user. As a result of that, factors from both, the term frequency and the collection frequency should be taken into consideration for the final term weights. The TF-IDF schema follows this principle, as terms with the highest weight are those that occur often within the document and are rare over the collection of documents [47].

$$w_{d,t} = tf_{d,t} \cdot idf_t \tag{2.1}$$

$w_{d,t}$	Weight for term *t* in document *d*
$tf_{d,t}$	Term frequency factor for *t* in document *d*
idf_t	IDF for term *t*

$$idf_t = \log(\frac{N}{docFreq_t}) \tag{2.2}$$

idf_t	IDF for term *t*
N	Number of all documents
$docFreq_t$	Number of documents, where term *t* appears

As soon as the document vectors are built, extracted vectors from the preferred documents are accumulated into a single profile vector. In case certain documents should be weighted higher, a linear combination, following the principle of Rocciho's algorithm [45], can be used. This algorithm, which is shown in equation 2.3, again has its origin in the information retrieval area for reformulating searches with relevance feedback and has also been used in several recommenders [2], [39]. A search query $\vec{q}$ is represented as vector, which initially contains the user's search words and looks like a vector-based profile representation. The user receives the search results from the initial query $\vec{q_0}$ and might provide relevance feedback on the found documents. The information from the relevance feedback is then used to alter the initial query in order to match the user's needs more closely. For that, the query vector is enhanced with terms from relevant documents D_r, whereas the weight of terms from non-relevant documents is reduced. In this equation, the document vectors $\vec{d_j}$ are normalized by the Euclidean length $|d_j|$, resulting in the unit vector, where the weights range from 0 to 1. This normalization is optional and systems, where weights are not restricted, can use the original document vectors. α, β and γ are used as multipliers to weight the different information sources. Vector $\vec{q}$ is the result of this linear combination and used for the new search.

$$\vec{q} = \alpha\vec{q_0} + \beta \sum_{\vec{d_j} \in D_r} \frac{\vec{d_j}}{|d_j|} - \gamma \sum_{\vec{d_j} \in D_{nr}} \frac{\vec{d_j}}{|d_j|} \tag{2.3}$$

$\vec{q}$	Query vector
$\vec{q_0}$	Initial query
D_r	Relevant documents
D_{nr}	Non-relevant documents
α, β, γ	Weights

Concept-based profiles follow the same schema as vector-based ones, except that predefined concepts are added to the profile. This fact requires a classification of documents, which can either be done manually or automatically.

In contrast to vector-based representations, machine learning techniques, like clustering and classifiers, can be applied for analyzing the user. In case of clustering, users with similar characteristics are grouped into clusters, which can be used for collaborative

filtering. Classifiers, on the other hand, have the goal of calculating the user's degree of interest in an item and can be based on neural networks, decision trees, Bayesian networks etc.

The profile construction phase is usually an ongoing process, as user preferences change over time and hence updates of the profiles are necessary in order to produce accurate recommendations. For tackling this important problem, several *profile adaption technique* [32] are known. The techniques range from *manual* systems, requiring an explicit user interaction, to *automatic* systems, where new feedback is detected and added to the profile. Such automatic systems can furthermore be extended with approaches for forgetting old feedback. For example, a *window* based approach either defines within the relevance feedback has to occur or a maximum number of preferences, which helps to automatically drop out old feedback. More complex approaches can use a *gradual forgetting function*, which takes into account that recent feedback is more important than older one. This function weights the feedback according to the interaction date and one possible approach is shown in equation 2.4, where a forgetting factor for user actions is used. It is based on a half-life time span *hl*, which assumes that the preferences reduce by the half within this time [55].

$$\textit{Forgetting factor} = e^{-\frac{\ln(2)}{hl}d} \tag{2.4}$$

hl half-life time span in days
d days passed since action occurred

2.2 Recommendation Techniques

For finding the right items for users, different recommendation techniques and classifications exist. First of all, a distinction between *personalized* and *non-personalized* recommenders should be made [50]. Personalized recommenders are based on input from individual users, whereas non-personalized recommenders produce the same results for all users and may use statistical data, like the top selling items. For personalized recommenders, which are of interest in this thesis, several classifications in the literature ([1], [2], [7], [8], [42], [50]) were analyzed. All of them distinguish at least between the following techniques:

- *Collaborative filtering* recommenders make a user-to-user comparison in order to suggest preferred items from similar users.
- *Content-based* recommenders try to find items that are similar to ones the user likes.

- *Hybrid* approaches combine different techniques to achieve better performance.

In addition to that, other techniques were discussed in some papers, like:

- *Knowledge-based* [7], [8], [42] recommenders use additional knowledge in order to infer items, which best match to the user's needs.
- *Utility-based* [8] recommenders are similar to knowledge-based recommenders, as they try to match the user's needs to the items, but are based on a user-specific utility function.
- *Demographic* [8] recommenders are similar to collaborative filtering, but the user-to-user comparison specifically is based on demographic data.

All of these techniques use different approaches for recommending items and all have strong and weak sides, which makes some more suitable in specific domains than others. The main characteristics of collaborative filtering, content-based, knowledge-based and hybrid recommenders will be discussed in the following. Information on utility-based and demographic recommenders is included in the various subchapters.

2.2.1 Collaborative filtering Recommender

For suggesting items, collaborative filtering recommenders, also known as user-to-user correlation method [50], search for users with similar taste. Most often, such recommender systems are based on ratings, like one of the first recommender Tapestry [16], or the Grouplens [23] news recommender. These ratings are then used to match similar users, for example two users have the same rating for the same article. Afterwards the preferences of similar people are re-used for recommending items.

Algorithms for collaborative filtering are either *memory-based* or *model-based* [1]. In case of memory-based algorithms, the system tries to calculate ratings for unknown items and suggests the items with the highest predicted rating to the user. For example, for calculating the ratings, the average rating of similar users is taken. Hence, as a first step, the so called nearest neighboring user needs to be found, who is often determined with the Pearson correlation coefficient or the cosine method with n-dimensional rating vectors. In order to have the recommendations efficiently on demand, these inter-user similarities can be calculated in advance.

On the other side, for model-based approaches, all ratings are used as input for building a model for predicting ratings. Here a probabilistic model, like a Bayesian network, can be used to compute the likelihood a user will provide a certain rating for a certain item.

Both approaches do not regard any content of the items, which is one big advantage of collaborative filtering recommenders, as they are independent of the items and can consequently be re-used in other domains. Although the recommendations might stick to a certain portfolio, this technique is the only one that is able to recommend cross-genre items [8].

Nevertheless, some drawbacks of this technique are known [1], [8]:

- *New-user problem*
  For suggesting items, the users must first specify their preferences. Without this information, no recommendations can be made.
- *New-item problem*
  Items that are new in the system need ratings by users before being used in the recommendation process. This problem, as well as the new-user problem, is often referred as *cold start* or *ramp-up* problem.
- *Sparsity of ratings*
  People often tend to rate similar items, leading to a sparsity of ratings, where only a few items have many evaluations. This makes it difficult to produce recommendations in all circumstances. In case the critical mass isn't reached, demographic filtering can help to categorize users on a different basis.
- *Gray sheep problem*
  Especially users interested in rare items, are difficult to categorize.

2.2.2 Content-based Recommender

A content-based recommender suggest items that have similar content to ones the user prefers and hence is also referred as item-to-item correlation method [50]. This approach has its roots in the information retrieval area, as methods for searching for documents are involved. But compared to traditional information retrieval applications, recommenders require user profiles, which encode the user's preferences.

For example, if someone reads a lot of articles in the IT section, the recommender is going to search for similar articles in this category. For that, as explained in chapter 2.1.3, the system looks at the content attributes of a document and stores the attributes of items the user likes in the profile, in this case in the so called *content-based profile* [1]. Afterwards this profile is matched against the available documents.

A very common technique, which comes from the information retrieval area, stores the keywords of documents and profiles in n-dimensional vectors. Each word has a weight associated, telling how important it is. For defining the weights, the TF-IDF weighting

schema is very popular. After the vectors have been constructed, the cosine similarity, which is shown in equation 2.5, can be used to compare the user profile vector with the possible document vectors. The smaller the angle is, the closer and more similar the document is to the user profile. This method is based on the same formulas as the cosine method for collaborative filtering, but instead of vectors with users ratings, keyword-based vectors for user profiles and documents are used [1], [39].

$$cosine(\vec{up}, \vec{d}) = \frac{\vec{up} \cdot \vec{d}}{|\vec{up}| \cdot |\vec{d}|} \tag{2.5}$$

$\vec{up}$	User profile vector
$\vec{d}$	Document vector
$\|\vec{up}\|$	Euclidean length of vector $\vec{up}$
$\|\vec{d}\|$	Euclidean length of vector $\vec{d}$

In contrast to this memory-based approach, model-based methods, like a naive Bayesian classifier can be implemented, which predict the probability a document is either relevant or non-relevant for a user [1], [39].

Compared to collaborative filtering, no information of other users is involved in the recommendation process and due to this fact, content-based recommenders do not suffer from the new-item problem. This fact makes content-based recommenders more suitable in situations with a relatively low amount of users. Still, several problems with this recommendation technique are known [1], [8]:

- *New-user problem*
  Same as for collaborative filtering, user profiles are required as input.
- *Limited content analysis*
  The recommender highly depends on the available information from the documents. Therefore, the documents either need to contain some machine readable text or they need to be manually classified by users. Further, the popularity of two documents, as long as they have the same vectors, cannot be distinguished by such content-based systems.
- *Overspecialization*
  The recommender will only suggest items similar to the ones, the user has already known, leading to a portfolio effect.

2.2.3 Knowledge-based Recommender

Knowledge-based recommender systems use knowledge about the items, the users and on how to map the user's needs to the items features, the so called functional knowl-

edge. This technique is similar to utility-based recommenders, because both perform a direct user's needs to item features matching, but, compared to knowledge-based recommenders, end-users have to construct the mapping function between the interests and the items themselves [8].

For implementing a knowledge-based recommender, several different types are known [42]. This includes *Case-based reasoning*, *Constraint-based reasoning* and *Rule-based systems*.

Rule-based systems contain explicit rules, which are used to find possible items, whereas a Contraint-based system receives a set of constraints from the user, for which the constraint solver tries to find possible solutions, a task that is seen as Constraint Satisfaction Problem. FSAdvisor [14], a recommender for financial services, is one example for a Constraint-based system. If no solution can be found, repair actions for the conflicting constraints are suggested in order to satisfy the request.

Case-based reasoning systems, on the other hand, reuse the solution of previously solved problems. The given request, in this case the user's needs, is first compared to the description of solved problems from the case base. Then, the solution of the most similar case is used and adapted for the current problem [28].

For example, the Entree restaurant recommender [7] is based on Case-based reasoning. It has a more conversational and interactive user interface, which is often typical for knowledge-based systems. In this application, a user selects a restaurant at the beginning and asks for similar ones, including further tweaking possibilities. The recommender doesn't build long-term profiles, as collaborative filtering or content-based recommenders tend to do. For finding interesting restaurants, a similarity metric is needed in order to compare the user request with the available options. It forms the main part in such a recommender and often no uniform concept for the similarity can be applied. For example, prices have to be compared in a different way than the cuisine type of a restaurant. In this case the Entree system uses a semantic network of cuisine types and compares two restaurants by calculating the distances between the types.

Such semantic networks, like taxonomies or ontologies, can be useful in many areas. For example, the Quickstep application [31], a recommender for research papers, uses a hierarchical topic ontology about research areas. If a user has interest in a specific topic, each parent concept receives half of the interest weight and is added to the profile. Besides the specific topics, more general topics are added, leading to more comprehensive and rounded profiles.

Concerning the matching of concepts of a hierarchical network, two different use cases have to be considered: the *subsumption* and the *proximity matching* [4]. In case of the subsumption matching, an item containing the child concept perfectly matches a request with its parent concept. The proximity matching, on the other hand, refers to the sibling

concepts, where some similarities because of the common parent concept are given, but it should not be treated as perfect match. As done in the Entree system, as well as in several others ([3], [4]), a distance measure between concepts can be applied, where the distance between parent-child concepts is smaller than the distance between siblings.

Compared to the other recommendation techniques discussed so far, this one doesn't suffer from a cold start problem. On the other hand, the recommender highly depends on the available knowledge and additional effort for the knowledge acquisition and maintenance is required [8], [42].

2.2.4 Hybrid Recommender

To avoid some of the mentioned drawbacks of the several recommendation techniques, combinations of two or more techniques are possible. For example, a collaborative recommender can be combined with a content-based recommender to overcome the new-item start-up problem.

In [8], Burke discussed seven different ways for combining recommendation techniques. For example, a *weighted* hybrid recommender combines the results of two separately operating recommenders in one score, whereas a *switching* hybrid recommender selects the best recommender for a specific situation. A *mixed* hybrid, on the other hand, presents the results of two or more recommenders within one common result list. This way, if one recommender still doesn't return any results due to a start-up problem, recommendations might already be available from the other one.

2.3 Job Recommender Implementation Examples

After the foundations for recommender systems have been discussed, concrete implementation examples are going to be presented. As this thesis is about a job recommender, the examples are taken from the online recruitment domain, including the Proactive [25], the CASPER [4], as well as a bilateral person-job [29] recommender. The way user profiles are built and the concrete recommendation technique are of special interest and the findings will be a valuable input for architecting the job recommender. Further on, some information about the existing implementation on Absolventen.at, which forms the basis for the new job recommender, will be given.

2.3.1 Proactive

Proactive [25] is a job recommender, which suggests jobs in four different ways. First, the most recent jobs are presented to the job seekers. Then, the job seekers can search for jobs via an advanced interface or they can specify their preferences, like location, salary level and so on, for receiving appropriate jobs. These three job listings include a bookmark option for every job ad. The bookmarked jobs are stored in the user profile and then passed as input to the actual recommender, which searches for similar jobs in a content-based way. For comparing jobs, several ontologies for job categories, educational levels etc. are used for calculating the distances between individual concepts.

So far, only feedback from the bookmarks has been included in the user profile. In [26], these profiles were extended with preferences inferred from implicit negative feedback. Whenever a job seeker read a job description, without adding it to the bookmarks, it was counted as negative feedback. Results from the evaluation showed that adding of negative feedback was helpful in order to distinguish between relevant and non-relevant jobs.

2.3.2 CASPER

CASPER [4] (Case-based Profiling for Electronic Recruitment) is another example for a content personalization technique in the online recruitment domain. Its goal is to personalize the job search, a task that is performed in two steps: first there is a server side similarity-based retrieval of jobs and second, a client side case-based personalization is applied. For the first step, no traditional searches are executed, it rather takes a job from the case-base that best matches the query, and compares it with other jobs. For the comparison, different similarity metrics are used, for example the comparison of skills is supported by a domain ontology. After similar jobs have been retrieved, as a second step, the result is filtered according to the user profile. The profile consists of graded jobs and a classifier uses this data to group the search results into relevant and non-relevant jobs.

For constructing the user profile, implicit feedback from various user actions, including applying for a job, emailing, reading, and visiting a job, is used. Even though it can be difficult to calculate appropriate profiles out of this data - especially the read time data contained a lot of noise - it was shown that this data was a valuable input, as it could be monitored without any extra costs and the feedback from read times and job visits correlated with the user's interests [41].

This search personalization follows a content-based approach. Nevertheless, within the same environment a collaborative filtering recommender was developed and evaluated [40]. This was mainly motivated by the problem that users often failed in describing

their needs with a search query. For the collaborative filtering, two algorithms were tested. First, a memory-based method for retrieving the k-nearest neighbors was implemented, followed by a cluster-based approach. As the profile data was relatively sparse, the memory-based approach had too little overlap in between the profile vectors and as a result of that, the clustering showed better performance.

2.3.3 Bilateral Person-Job Recommender

In [29], a system for recommending jobs to people as well as for recommending people to jobs was presented. There, it was stated that both, the preferences of the recruiter and the preferences of the job seeker should be considered in the recommender. For this purpose, two separate recommenders, a CV-recommender, and a job-recommender were implemented. In the CV-recommender, the preferences of the recruiter were accumulated with the previously selected candidates, whereas the job seeker's profile consisted of information from relevant jobs. For the recommendation process itself, a probabilistic model was established that predicts whether a job or CV is relevant or not.

2.3.4 Absolventen.at

Absolventen.at consists of four niche job portals, which all run on the same code and database. Only the content, like jobs or companies, is specifically filtered for the target groups. The technology behind is based on the CMS Drupal version 5.x, which has been extended with many modules.

Over the last years, more than 16.000 job seekers have registered on the website and currently around 3.350 jobs are published[1]. After job seekers register, they can fill out a very comprehensive resume, including the education, work experience, skills and so on. Most form elements in the resume, like the skill catalog, are generated out of various domain taxonomies. Jobs, on the other hand, can either be published manually by companies, or, which applies for most of the available jobs, are automatically parsed and classified from the companies websites.

To assist users finding the right items, a job-people matching system for recommending jobs to applicants and applicants to jobs was implemented. The approach follows a clear content-based recommender and its architecture is shown in figure 2.3, which will be explain in the following.

For the matching, the resume of the job seekers, as well as all available jobs, get transformed into a vector-based representation. The vectors, typically hyper-dimensional,

[1] Numbers from 3rd March 2011

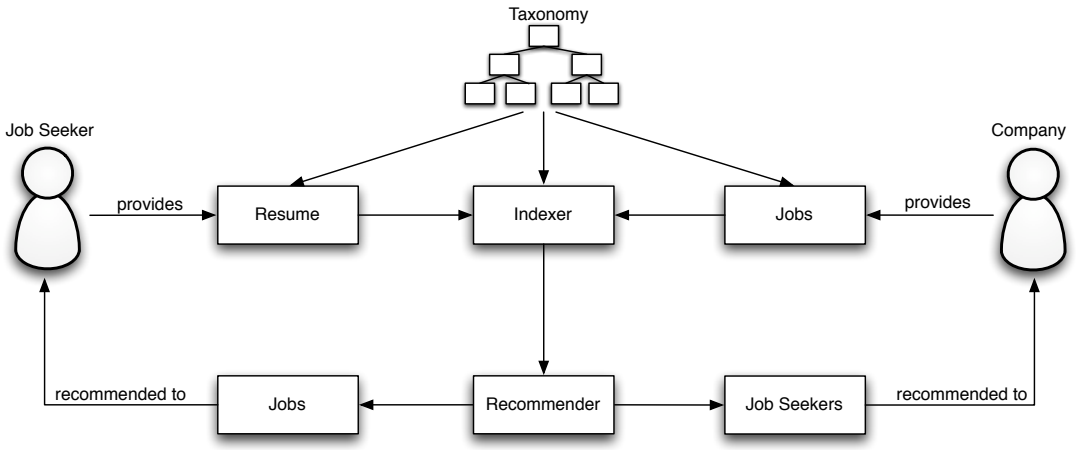


Figure 2.3: Architecture for the existing Absolventen.at recommender

contain some concepts with weights. The weights in the job seeker's user profile denote the skill level. For example, expert skills in Java gain higher weights than basic skills in C++.

Before the vector is finally stored by the *Indexer* component, it is enhanced with parent concepts of the taxonomy, e.g. if someone has skills in MySQL, the parent concept "Databases" will be added to the vector. This enables the matching algorithm to take sibling concepts into account, like for a job searching for PostgreSQL knowledge, an applicant with MySQL will match, as both concepts are siblings and have the same parent. Of course, this match should be less important than a perfect hit, which is somehow controlled by the parent weights.

For calculating the weights of parent concepts, the sum of child concept weights is considered, as the more specific skills, for example, someone has in an area, the more the parent concept should be weighted. Nevertheless, just summarizing all child concept weights might lead to fairly high weights and in order to control the influence of parent concepts the equation in 2.6 is applied, where the weight is calculated by a sublinear sum of child concept weights. Additionally, each weight is multiplied by the inverse document frequency (IDF), which decreases the weight of often occurring terms, like parent concepts, and emphasizes seldom ones, like very specialized skills.

$$w_p = 1 + \log_2 \sum_{c \in childnodes_p} w_c \tag{2.6}$$

w_p Weight for parent term in the vector
w_c Weight of a child term

Furthermore, the taxonomy is used in a similar way for matching the fields of study. An applicant always selects a specific field of study, like business informatics, whereas a job usually contains the general one, like computer science, which is a parent of business informatics. To achieve this subsumption matching functionality, the parent concept is indexed in the job seeker's resume vector.

Besides the fields of study, other data fields from the resume and jobs are taken into account, including the IT skills, the general skills, the languages and the preferred region. Some of these fields are more important for the recommender than others. In particular the field of study and the region are seen as crucial factors for the job seeker. Furthermore, some fields have higher term frequency weights by default than others, e.g. skill fields have higher weights than a normal region field. To control the influence of the different fields, a further field weighting schema, which is shown in equation 2.7, is applied to the vectors, whereas the according field weights are given in table 2.2.

$$w_{d,t} = tf_{d,t} \cdot idf_t \cdot fw_t \tag{2.7}$$

fw_t Weight of field that contains term t

Table 2.2: Field weights

Field	Weight
field_general_skills	1
field_it_skills	1
field_languages	1
field_field_of_study	20
field_region	20

As the resume and the job vector need to have the same structure for the matching, a transformation of the various data structures is necessary. Especially transforming the resume into a single vector is a complex task, since the data for one resume for one user is spread over multiple entities and database tables.

Once these vectors have been generated, they are sent to Apache Solr[2], an open source search framework. Solr is, on the one hand, responsible for storing the vectors in an efficient way, and on the other hand, for performing the content-based similarity calculations between jobs and resumes. For calculating the similarity, a *overlap score measure* [30] is used (see equation 2.8), which simply calculates the dot product of the vectors. An additional normalization, which is done by the cosine similarity, can be

[2]http://lucene.apache.org/solr

problematic in some cases, especially for recommending applicants to jobs. It is important that an applicant with expert skills is ranked before one with basic skills, but with a normalization by the Euclidean length, this information would be removed.

$$score(up, j) = \sum_{k=1}^{t} w_{up,k} \cdot w_{j,k} = \vec{up} \cdot \vec{j} \tag{2.8}$$

up	User profile
j	Job
$w_{d,t}$	Weight for term *t* in document *d* (*up*, *j*)

The results with the highest score are then sent back to Drupal, which displays the recommendations to the end users.

Previous to this implementation, database queries were used for finding appropriate jobs. These queries have been very limited in expressing complex statements and in calculating a similarity between two objects. The current vector-based approach has outperformed in many points, like the execution time, the similarity metric and the possibility to include various different data.

Still, during the last year, several experiences have been made and when taking a closer look at the implementation, some drawbacks can be identified:

- Preferences of both, the job seeker and the recruiter, have been ignored so far, although this information is an important input for recommender systems.
- Many indexing steps and representations are hardcoded in the Indexer component, what makes it difficult to adapt and to re-use this component.
- Both, the job recommender as well as the applicant recommender use the same configuration, even though they have slightly different requirements.
- Derived parent concepts of the taxonomy are added to the same field structure as their child concepts. This makes it difficult to separately address those parent concepts in the matching process.
- Lack of quality measures and hence no feedback on different configurations of the recommender component.

Those are some of the lessons learned with the existing implementation. As the current website will sooner or later switch to the newest Drupal version 7.x and will use the

open source distribution eRecruiter[3], which has mainly been developed for this website, the recommender will be completely re-implemented, allowing us to fix these issues.

[3] http://drupal.org/project/recruiter

CHAPTER 3

Conception

After the foundations of recommender systems and some examples have been discussed, we can now focus on implementing a full job recommender.

As pointed out in the last chapter, a basic implementation already exists, but for the integration of implicit user feedback and for solving some of the current drawbacks, a refinement of the overall architecture is needed.

This chapter covers the full process of architecting the job recommender. First of all, the requirements and objectives the job recommender should fulfill, are going to be analyzed. Afterwards an overall picture of the architecture will be drawn, followed by in-depth discussions of the two main components: the user profile and the recommendation technique. Especially the user profile, which needs to incorporate implicit relevance feedback, is of special interest.

3.1 Requirements

As a starting point for the conception phase, the requirements for the recommender have to be gathered. For defining the functional requirements, we have to take a look at the primary use case of the application: the job recommender for Absolventen.at. The applicant recommender, on the other hand, won't be further addressed in this thesis, but it can re-use many efforts from this work. The existing implementation, as well as the surrounding environment, have already been described in the last chapter. The use case involves three roles: the applicant, looking for jobs offers, the company, publishing jobs, and the administrator, who configures and monitors the recommender. Furthermore, an applicant performs various actions, like searching or bookmarking jobs. The

environment of a typical job board website is already available, so we can focus on the recommendation part.

For defining the main requirements following information sources have been considered: the use case description, experiences made with the current implementation, the problem description of the thesis as well as at the findings from the literature research. As this thesis is about the integration of implicit user feedback, we start with the requirements related to this specific topic:

- **Flexible system for observing user behavior**
  Users perform many different actions on the job board, like reading a job description, writing an application or searching for a job. These actions will form the main input for the relevance feedback and as a result of that the system has to keep persistent track of them in an unobtrusive way. The system needs to be able to act on various configurable events, like a user has bookmarked a job. Optimally, an administrator can configure these events and easily integrate them into the user profile via a user interface.

- **Integration of relevance feedback into the user profile**
  After the user actions have been recorded, preferences have to be inferred and merged into the existing user profile. As the current user profile consists of the resume and the preferences relate to jobs, a transformation and harmonization step of different data structures will be necessary.

- **Appropriate profile adaption technique**
  Older user actions are usually less relevant than later ones. This fact should be taken into consideration when inferring the preferences. It should be possible to either ignore outdated actions from the history or to appropriately weight actions according to their interaction date.

- **Weighting of user actions**
  Different actions provide different levels of evidence for interests. For example, applying for a job provides more evidence for interest in this job than just reading the job description. As a result of that, a weighting system for the different actions is required, where actions with higher level of evidence have higher influence on the recommendations. These weightings need to be configurable for the administrator.

Besides these specific points, also more general requirements for the complete system have to be taken into account. They are mainly a result of the lessons learned by the current recommender:

- **User interface for the recommender configuration**
  The current implementation suffers from one major drawback: many configurations are hardcoded in the program. Small adaptions of the recommendation process require changes in the code. These changes usually can only be done by programmers and exclude normal website administrators. Further, it is time-consuming to test different configurations, since every change requires a code update. As a consequence, a user interface for configuring the recommender is strongly needed.

- **Generic and re-usable implementation**
  As already mentioned, similar recommenders, either on the same website or on others, might follow the same recommendation method. Therefore, a more generic implementation, which does not rely on website or domain specific settings, like field names, should be achieved.

- **Evaluation component**
  The configuration of the recommender contains many different settings, like weights for fields and user actions, which influence the recommendation results. As the recommendation process is complex, without any performance measures, it is difficult to tell, whether changes in the configurations result in better accuracy or not. Further, part of this thesis is a comparison of the job recommender with and without implicit feedback. Hence, a system that measures the performance, like the precision and recall of different settings, is required.

- **Flexible integration of the taxonomy**
  Most data for the recommendation process is related to concepts of a taxonomy. This helps to infer more general concepts, which can be added to the matching process. At the moment, each data field that refers to the taxonomy is handled the same way: in a pre-processing step the general concepts are added to the field and therefore included in the matching query. As every concept is in the same field, no distinction between given and inferred concepts is possible any more. In some situations, different weightings might be interesting. Moreover, not every field should follow the same approach: some might only want to match over specific concepts, others want to include some parent concepts and so on. Accordingly a more flexible and configurable integration of the taxonomy is desired.

- **Consideration of different skills levels**
  While filling out the resume form, applicants are able to specify their skill levels, for example someone is expert in Java programming, but has only basic skills in PHP. This information needs to be represented in the recommendation algorithm, as expert skills should have more influence on the results than basic skills. The

current implementation is already capable of doing that. Hence, this functionality must be retained within the new implementation.

- **Separate configurations for various recommenders**
  Absolventen.at currently serves two recommenders: one for recommending jobs to applicants and one for recommending applicants to jobs. Although both follow the same recommendation principle, some configurations, like weightings or similarity metrics, should be specific to the recommender. Nevertheless, at the moment, some settings are shared. This fact should be fixed in the new version so that every recommender has its own configuration.

Before we go on with the architecture, we should look at the non-functional requirements. These requirements do not add any further behavior to the system, they rather ensure that the system performs within certain quality standards. Some of the main non-functional requirements are listed here:

- **High performance**
  The current implementation is able to calculate recommendations with a very low execution time. Adding of implicit feedback will lead to larger user profiles and consequently to larger matching queries. Still, performance is an important topic and should be taken into consideration. It needs to be decided, which information can already be calculated off-line and evaluated, whether the current matching query can be optimized.
- **High quality of recommendations**
  A high accuracy is desired, so that jobs fit the user's interests and capabilities. The accuracy is influenced by the recommendation method and by various configuration settings. The evaluation of different settings and thus the optimization of the accuracy is part of this thesis.

After the requirements have been clarified, we can start with the architecture of the job recommender. Most of the mentioned requirements will be addressed in the architecture, some of them, like providing a user interface for some configurations, are part of the implementation.

3.2 Architecture

In figure 3.1 the new architecture, which tries to solve the requirements for the job recommender, is given. It shows the main components and interactions on a very high abstraction level.

The architecture can be seen as a refinement of the existing one, which was explained in figure 2.3. When we compare it, new components for the *Hybrid User Profile*, the *Configuration* and the *Evaluation* have been introduced. The other components, like the *Indexer* and the *Recommender*, will mainly differ in their implementation. The diagram is based on the use case for recommending jobs to applicants, but the implementation of the components will be generic, so that it could also be re-used for other use cases, like recommending applicants to companies.

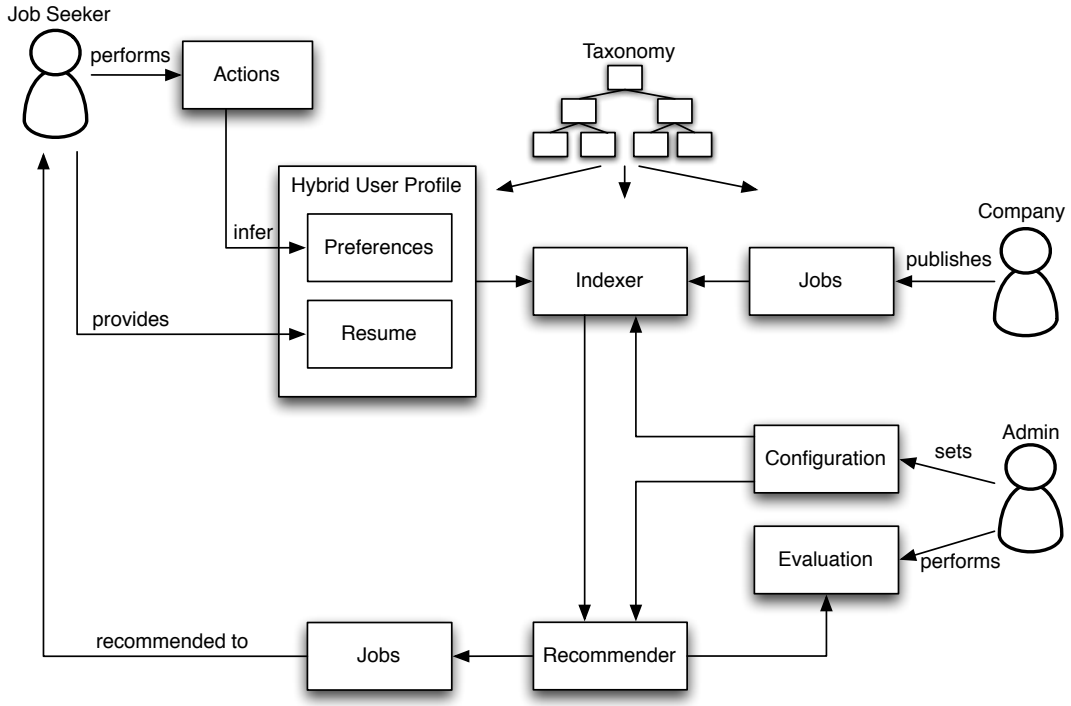


Figure 3.1: Architecture for the job recommender

The recommendation process starts with the job seeker, who either provides the resume or performs some actions on the website. This information is stored in the *Hybrid User Profile* and together with the jobs, the *Indexer* transforms and stores the data in an appropriate format. The data is then used by the *Recommender* to find the best matching jobs. The job seeker, who receives those, may perform some actions on them, like writing an application. These actions again influence the user profile and lead to an adaption of the recommendations, a process which is also referred as the *recommendation cycle* [35].

To give an overview of the separate components, the functionality of each of them is

described in the following:

- **Hybrid User Profile**
  The hybrid profile combines information from the resume as well as information from the relevance feedback into one representation. For calculating recommendations, the user profile needs to contain at least some preferences or some data from the resume.

- **Indexer**
  The Indexer is responsible for transforming the different data structures into a common and for the matching algorithm optimized representation. For that, various information for the user profile, as well as for the jobs, is fetched from the database and transformed into the appropriate representation. Furthermore, this component interacts with the taxonomy and adds derived concepts to the profiles and jobs.

- **Recommender**
  The Recommender uses the stored information and performs content-based queries for matching the user profile and the jobs. For tuning the process, the queries take several configuration parameters.

- **Configuration**
  An administrator is responsible for creating and managing recommender settings. This ranges from selecting the appropriate user actions for the relevance feedback, defining the mapping rules to various parameters for the recommender. All settings have to be accessible for the site administrator. An appropriate user interface will be presented in the implementation chapter.

- **Evaluation**
  For testing the performance of different parameter settings, an own Evaluation component is needed. This component receives the results from the recommender and compares them with a pre-defined set of relevant items, which allows the measuring of several quality indicators, like the precision and recall. More information about the evaluation process is discussed in chapter 5.

The next chapters provide in-depth discussions on the user profiling and recommendation techniques, including information on the Indexer and Configuration component as well.

3.3 User Profiles

User profiles build the main input for the recommender. The information from the user profile will later on be used for matching the jobs. So far, the user profile has only consisted of data from the resume. During this project, relevance feedback from implicit sources will be added, leading to more comprehensive profiles and requiring a re-work of the architecture.

According to Montaner et al. [32], five design decisions for building user profiles have to be made. These design decisions have intensively been discussed in chapter 2.1 and will be used to define appropriate profiles for the job recommender, which fulfill all necessary requirements.

3.3.1 Profile Representation

The definition of the profile representation is the first task to do. This specifies the final structure that will be passed to the recommender as input. Every other user profiling task depends on this representation, as several construction and adaption techniques need to be performed to receive the final user profile representation.

The existing implementation uses a vector-based representation of the user profile. The vector is separated into distinct fields, like a field for IT skills, a field for languages and so on. Each field may contain several values, associated with a weight, which represents the user's degree of interest.

Most values are known concepts of the taxonomy. Additionally, more general concepts are derived from the taxonomy and added to the vector. An example of this representation can be seen in table 3.1, where each row represents a dimension in the vector. Here the user has expert skills in Java and intermediate skills in C++. As a consequence, the weight of Java is higher than the weight for C++. Object-Oriented Programming (OOP) is a parent concept of Java and C++ and thus added to the vector. The weight results in the sum of both child concepts. Compared to the shown example, the actual representation uses identifier for the concepts, as names might change. Further, the final weight used by the recommender may differ from this example, as additional adaptions might be performed.

Until now, only data from the resume has been included in the profile vector. For adding preferences, we have to think about an ideal integration into the existing representation. This vector-based profile allows an efficient matching with jobs using the cosine similarity. Hence, the goal is to the keep this representation and to add the preferences from the relevance feedback in an appropriate way.

Table 3.1: Example for a vector-based profile representation

Field	Value	Weight
IT skills	Java	3
	C++	2
	OOP	5
Languages	German	4
	English	2
Field of study	Computer Science	1

To achieve that, concepts of preferred jobs are extracted and merged into the profile vector. For example, an applicant bookmarks a job for Java programmers. Then the concept Java, among others, is added to the profile. The complete profile construction process can be found in chapter 3.3.4.

But first of all, the user actions have to be persistently stored in a history-based user model. This user model is used as intermediate store and will be accessed by the construction technique to build the final vector. Directly adding of extracted concepts of a job, without storing the action is not enough. It has to be possible to rebuild the whole user profile at a later point of time. For example, when someone updates his resume, all performed actions from the past need to be included in the construction phase. Aside from that, for an adequate profile adaption technique, where old actions should be ignored and the importance of later ones increased, the exact interaction date is required.

To ease the integration of the user actions, a consistent access of the user history is needed. A common database table for storing all kinds of user actions will be difficult, as some actions require different information. For example, the bookmark action stores a reference to the job, the read action additionally the time, and the search action has no references, but some search words to store. As a result, a single table without too many optional attributes is not possible. An API that takes care of storing and providing an uniform access is preferred. Further details about this issue will be discussed in the chapter 4.2.1.1.

3.3.2 Initial Profile Generation

For providing recommendations from the beginning, an initial profile can be used. For creating such an initial profile, different methods exist, ranging from a manual specification after the registration to automatic generation of stereotypes or preferences.

On Absolventen.at, after users finish the registration, they are asked to fill out the resume, containing various demographic data and other relevant information. This can be

classified as manual approach and has not changed so far.

The only fact which changes is that the resume and hence the initial profile is not required any more by the recommender. If we take a look at the website statistics, we see that only the half of the registered users fill out the resume. The other half is excluded from the recommendation service. Now, an empty initial profile is possible, as long as users perform some other exploitable actions.

However, the resume helps to generate recommendations from the very beginning on and enables the recommender to suggest jobs according to the capabilities of the job seeker.

3.3.3 Relevance Feedback

Relevance feedback is typically used as input for recommenders. The user tells the system which items are interesting and which ones not. This can either be done in an explicit way, e.g. by a rating system, or the feedback can be gathered from implicit sources.

Currently the profile solely consists of the resume, representing the user's capabilities. In the new system, the user's preferences from various feedback will be added to the profile, resulting in a *hybrid user profile*, as we have multiple different data sources as input.

On Absolventen.at, no explicit feedback mechanisms, such as a rating system for jobs, exist. Still, many user actions can be observed and used as implicit feedback. When analyzing the functionality of the website, following possible user actions for implicit feedback can be identified:

- a user reads a job description
- a user bookmarks a job
- a user writes an applicant for a job
- a user searches for jobs

In chapter 2.1.2 a classification of observable behavior has been presented. This classification is now used for the observable user actions on Absolventen.at and can be seen in table 3.2. The four mentioned actions were put into an appropriate scope and a behavior category, even though specifying the category can be difficult in some cases. The *read* action (often also referred as *view* action) is observed at the *object* level, which represents a full job. This action could also be monitored on the *segment* scope, e.g. which

paragraph the user is actually reading, but capturing this information from the server-side is impossible and we don't expect much benefit from this information anyway. The *bookmark* action is in the category of *retention*, as we assume that the user wants to make further use of these job *objects*. *Applying* for a job is definitely not easy to classify in this table. When performing this action, the user writes an application, which references to the job. The writing of an application could be classified as *creation* behavior, but we don't plan to make further use of the created content. In addition to that, putting the action into the *reference* category is reasonable as well, since it establishes a link between the application and the job. Nevertheless, we expect that the user primarily wants to make further use of this job and therefore have classified it under the *retain* category. The last action, which can be observed and used as implicit feedback on this website, is *searching* for a job (often the *query* term is used instead). In this case the user enters some search words and as a result of that the action is classified as *creation* behavior in the *segment* scope. The search result is not relevant here, as we don't know if it meets the user's expectations.

Table 3.2: Classification of observable behavior on Absolventen.at

Behavior Category \ Scope	**Segment**	**Object**	**Class**
Examine		Read	
Retain		Bookmark Apply	
Reference			
Annotate			
Create	Search		

We assume that most actions indicate some kind of interest in this job. In case of reading, it is rather difficult to tell, whether a user finds the job interesting or not. In contrast, applying for a job indicates high interest. As can be seen, different levels of evidences must be considered. For that, appropriate weightings for each kind of user action should be defined.

The way user feedback is going to be integrated follows the *Inference - Prediction* approach (figure 2.2), as we use the observed behavior to infer ratings (= weights) and pass it as input for the prediction of relevant jobs. This topic about inferring appropriate weightings and other special characteristics of each action are going to be discussed in the following section.

Read Action: This action has already been discussed in several research papers, where a tendency of longer read times on documents with higher ratings than on ones with

lower ratings has been identified [10], [23], [33], [52].

The time a user spends on a webpage can be easily recorded in an unobtrusive way. On Absolventen.at, a JavaScript snippet was added, which logs the time to the database. For using this information, we need to know whether the given feedback is positive or not. For this purpose some kind of read time threshold should be defined, which tells how many seconds a user has to read a page before it is counted as positive feedback. Finding this threshold is a difficult task and it will be impossible to set it in a way to satisfy all cases. Some people need more time for reading a job description than others and it might happen that someone assesses a job as uninteresting while reading the very last paragraph.

To get a feeling for an appropriate threshold, we first look at results from several researches. Table 3.3 shows average read times for relevant and non-relevant documents. The numbers should only give an impression. For exact times, other statistical measures and information about the experiment itself, the literature references are given in the table. Except in the results from experiments by Kelly et al., there were clear differences between read times for relevant and non-relevant articles. Claypool et al. and Konstan et al. compared the read times with explicit ratings, where the rating scale ranged from 1 (least interest) to 5 (highest interest). Read times in Konstan were notable higher than in Claypool. Times for the seconds highest rating (4 on the rating scale) can still be categorized as relevant and ranged from 24,3 (Claypool) to 41 seconds (Konstan). Seo et al. reported that the biggest part of relevant documents had a read time between 10 and 20 seconds, followed by 20 to 30 seconds, whereas most neutral documents were read between 6 and 10 seconds and people spend around 6 to 20 seconds for non-relevant documents.

Table 3.3: Read times in seconds in the related work

	Claypool [10]	Konstan [23]	Kelly [21]	Seo [52]
Relevant / Highest Rating	26	49	27.6	10-30
Neutral / Avg. Rating	21,2	28	-	6-10
Non-Relevant / Lowest Rating	13,4	11	25.6	6-20

In experiments by Morita [33] a threshold of 20 seconds was used, which resulted in higher recall and precision than with explicit ratings. Furthermore, it was stated that the length and readability of an article had no influence on the read time.

Nevertheless, it has to be noted that most experiments weren't executed in real world environments and before specifying a threshold for our application, we have to look at the recorded read times on Absolventen.at.

For this purpose the data of 2 months (15th December 2010 - 15th February 2011) was analyzed. Within this time frame, 6.627 visits were logged. The time can only be recorded, if the user is logged in and has JavaScript activated[1]. If a user re-visits the same job, it is logged as separate entry and not summarized into one value. As soon as the webpage is fully loaded, the read time is counted until the user quits the page. The median of all read times is 18 seconds, whereas the mean is 342,6 seconds, a measure which is heavily biased by the outliers. It might happen that users forget to close the browser window, resulting in very long read times and noisy data. The highest value is 459.594,0 seconds (around 5 days!) and consequently the mean should not be considered in this way, as this measure is not robust against outliers. If we remove the outliers by cutting off read time higher than 10 minutes (affects 4% of all records), we get an adjusted and more realistic mean with 71,02 seconds.

As on Absolventen.at no ratings for comparing and evaluating the read times are available, some other information has to be used. For this, we assume that people who apply for a job or bookmark it would also categorize this job as relevant. Information about non-relevant jobs, on the other hand, is not available, because there are no sources for negative feedback.

The read times of jobs a user bookmarked or applied for can be seen in table 3.4. A visualization in form of boxplots is shown in figure 3.2. The boxplots exclude the extreme outliers and it has to be kept in mind, that the number of observations for bookmarked and jobs applied for is much lower. Especially the data sample for applications is very small. Concerning the read times, the median of bookmarked jobs is 34 seconds, whereas the median for jobs, for which the user applied, is even higher with 88 seconds. A tendency of longer read times for relevant jobs compared to average can be identified in this environment as well.

Table 3.4: Read times in seconds on Absolventen.at

	All job reads	**Bookmarked jobs**	**Jobs applied for**
Num. records	6.627	483	71
Minimum	0 sec.	0 sec.	0 sec.
1st Quantile	4,0	7,0	12,25
Median	18,0	34,0	88,0
Mean	342,6	501,1	532,87
Adjusted Mean[2]	71,02	126,4	203,69
3rd Quantile	55,0	121,0	356,25
Maximum	459.594,0	42.712,0	4.915,0

[1] On Absolventen.at 98% of all visitors have JavaScript activated

With the read time threshold it should be achieved that as many relevant jobs as possible are included, while the non-relevant ones are neglected. For defining the threshold, some examples for non-relevant jobs would be extremely useful. Anyway, when looking at the numbers from Absolventen.at and other experiments in the literature, a threshold between 15 and 30 seconds seems to be appropriate. In table 3.5 possible thresholds are listed with the percentage of included visits. For example, if we would have a threshold of only 5 seconds, 72,3% of all visits and 81,2% of the bookmarked and relevant ones would be included.

Table 3.5: Evaluation of possible read time thresholds

Threshold t in sec	**All**	**Bookmark**	**Apply**
$t \geq 5$	72,3%	81,2%	84,5%
$t \geq 10$	61,3%	69,4%	77,5%
$t \geq 15$	54,6%	66,0%	73,2%
$t \geq 20$	48,3%	62,1%	67,6%
$t \geq 25$	43,0%	55,9%	64,8%
$t \geq 30$	38,3%	52,0%	64,8%
$t \geq 35$	34,6%	49,7%	62,0%
$t \geq 40$	31,4%	46,4%	62,0%
$t \geq 45$	29,0%	42,7%	60,6%
$t \geq 50$	26,8%	39,3%	53,5%

A threshold of 5 seconds is probably too low, since too many non-relevant jobs would be counted. As default value for the start, a threshold of 20 seconds should be reasonable. There, we still have 48,3% of all visits, 62,1% of the bookmarked jobs and 67,6% of jobs, where the user applied for, included. This threshold should be adjustable later on and the table in 3.5 can be used as orientation.

If we use a threshold of 20 seconds, 51,7% of all recorded read times are useless, as we can not state whether the feedback is positive or negative. Still, it makes sense to use this data as neutral information. For example, we don't add any data of these jobs to the preferences, but would exclude these jobs from the recommendation result, as suggesting an item a user already knows doesn't make sense. This user action, where the read time is lower than the threshold, is referred as *click* action in the following.

In case this approach with a fixed threshold appears to be insufficient, more complex models are possible. For example, a method used in the CASPER project [41], the threshold could be adjusted to individuals by looking at their personal average read time. In addition to that, a more fine-grained weighting function for the feedback could

[2]Outliers with more than 600 seconds were cut off.

be applied, where higher read times gain higher scores. This way a strict threshold could be avoided.

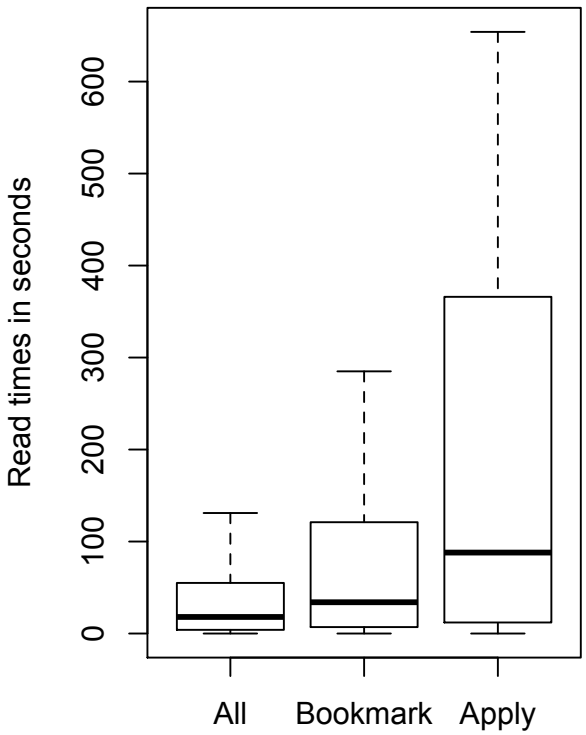


Figure 3.2: Read times on Absolventen.at

Bookmark Action: Users on Absolventen.at have the possibility of adding jobs to their bookmarks. In various job overviews and on the job description page itself, links for bookmarking jobs for registered users are available. Bookmarked jobs are then visible in a separate listing in the user account and can, of course, be unmarked and removed from their bookmarks again.

Seo et al. [52] reported a strong correlation between bookmarking and the relevancy of documents. In their experiment, 82% of the bookmarked documents were also categorized as relevant, 14% as neutral documents and only 4% of the bookmarks were non-relevant.

As can be seen from these numbers, bookmarking gives high evidence for interest. Additionally, a relative high number of jobs gets bookmarked and as a result of that, this action will play an important role for the job recommender. Within the time frame of

two months, 701 jobs were marked, whereof 191 were removed from the bookmarks again[3]. The information from the un-bookmarked jobs will be treated as neutral information, the same way as it is done for the click action.

Apply Action: This, for a job board specific action, allows users to write applications for a specific job offer. The company, which has published the job, receives an email with the application and gains access to the user's resume. Consequently, before using this application workflow, users need to complete their resume.

User who write an application clearly indicate interest in this job. On the other hand, only a fairly low number of job seekers (100 applications within two months[4]) use this functionality.

Search Action: Compared to the actions discussed so far, this one is slightly different. This feedback doesn't relate to a full job object, but rather contains a few search words.

Nevertheless, this information can form a valuable input for the user profile and it's part of the recommender to make use of it in an appropriate way. Furthermore, many job seekers use the search interface. Within two months, 1.937 search requests of logged in users have been recorded.

Now, various user actions have been discussed and before exploiting the information from the feedback, it should be kept in mind that not every action can be used in the same way. Some actions provide more evidence for interest than others. For this purpose some kind of weighting schema will be introduced, where higher weights mean higher evidence for interest. Afterwards, the profile construction phase has to regard those weightings.

Applications probably provide the highest evidence for interest, followed by bookmarks and read times higher than 20 seconds. This was taken into consideration when defining the weights in table 3.6. The weights should be seen as default values for the start and the recommender application should allow administrators later on to adjust these weights for optimization. The *apply* action received a weight of 5, *bookmarking* 3 and *reading* 1, meaning for example, that applying is 5 times more important than just reading a job description for more than 20 seconds. Concerning the *click* action, we don't know whether the feedback is positive or negative and hence considered it as

[3]This total bookmark number is higher than shown in table 3.4, but users can bookmark jobs without reading it or have JavaScript deactivated and hence no read time log exists.

[4]Again, more than in table 3.4, but users can start the application workflow before they login or have JavaScript deactivated and hence no read time log exists.

neutral information with the weight 0. The *search* action is not comparable to the others, as it does not relate to jobs, and got an initial weight of 1.

Table 3.6: Default weights for user actions

Action	Weight
Apply	5
Bookmark	3
Read	1
Search	1
Click	0
Un-Bookmark	0

3.3.4 Profile Learning and Construction Technique

After information about the user has been collected, the final user profile can be constructed. For this we have to consider the initial profile, in our case the resume, and the relevance feedback from implicit sources.

In table 3.1 the final representation can be seen. To recap, the representation is based on vectors, which are additionally divided into separate fields. For generating the profiles, concepts out of the resume and the relevant documents have to be extracted and added to the profile vector, a task which is mainly performed by the Indexer component.

For building a common user profile, which considers the resume and the relevance feedback with different evidence levels, we first take a look at Rocchio's algorithm for relevance feedback, which is explained in chapter 2.1.3. This algorithm is usually used for reformulating an initial search query with positive and negative relevance feedback, including weighting parameters for controlling the influence of the relevance feedback.

If we adapt this algorithm to our situation, we can use the resume vector $\vec{r}$ as initial query. In case the user doesn't fill out the resume, the initial query vector is empty and the user profile will only consist of the relevance feedback. In addition, as we have different feedback categories, like bookmarking, reading etc., which have different levels of evidence for interest, separate weighting parameters β_c are needed. Feedback with higher quality gets higher weights and thus has higher influence on the final profile vector. In the equation 3.1, this is represented as a sum over all available feedback categories. Negative feedback is of course still possible with β_c smaller than 0.

$$\vec{up} = \alpha\vec{r} + \sum_{c \in RF} (\beta_c \sum_{\vec{d_j} \in D_c} \vec{d_j}) \tag{3.1}$$

$\vec{up}$	User profile vector
$\vec{r}$	Resume vector
RF	Relevance feedback categories (apply, bookmark, ...)
α, β_c	Weights

Furthermore, in our case document vectors should not be normalized by their length. All job vectors contain concepts of the taxonomy with a weight of 1. At the moment, no distinction between the importance of certain concepts within a document is made, although it would be possible to consider, for example, that the required skill Java is more important than the optional skill Office. Nevertheless, if the vector would get normalized, the weights of vectors with more concepts would get smaller than ones of a vector with less concepts, although they should have equal influence.

The more feedback is available, the more the final user profile $\vec{up}$ moves away from the resume and as a result of that, the resume will loose importance. Of course, how far and how fast the profile vector moves away can strongly be regulated by the weights α and β_c. For β_c, the weights defined in table 3.6 can be used, where the apply action, for example, has a weight of β_{apply} = 5. If a job, for which the user has applied, contains the term Java with the weight 1, it will be added to the profile vector with the weight of 5.

In contrast to the job ads, different term weights are possible in the resume. If a user is expert in Java, the weight is 3, whereas it is 2 for intermediate skills and 1 for basic skills. Now, if we define an α = 2, and have expert skills in Java, the final weight is 6. When we compare this to the relevance feedback, a user would need to bookmark ($\beta_{bookmark}$ = 3) a job with Java twice, or read (β_{read} = 1) it six times to have the same influence on the final profile vector. Calibrating all these parameters and keeping a balance between the resume and the relevance feedback is difficult, but evaluations later on will help to adjust them.

Another possible approach would be to normalize the relevance feedback by the number of user actions. This would allow to keep a fixed proportion between the resume and the relevance feedback, but on the other side, more feedback and thus more evidence for interest wouldn't lead to higher weights in the user profile and as a result of that, this doesn't seem to be suitable here.

To illustrate the profile construction algorithm more closely, the equation with its default weights is shown in 3.2. It sums up the resume vector and the job vectors from various feedback. As can be seen, the weights for clicked or un-bookmarked jobs is 0 and

hence they have no influence on the final user profile vector. Still, this information can be used in the recommender, so that these clicked jobs are excluded from the result. Additionally, feedback from the user's searches is added to the profile. This feedback is slightly different to the others, as it's not related to other objects and only consists of a few words, which are matched against the fulltext of jobs and stored in a separate field.

$$\vec{up} = 2\vec{r} + 5 \sum_{\vec{d_j} \in D_a} \vec{d_j} + 3 \sum_{\vec{d_j} \in D_b} \vec{d_j} + 0 \sum_{\vec{d_j} \in D_{ub}} \vec{d_j} + 1 \sum_{\vec{d_j} \in D_r} \vec{d_j} + 0 \sum_{\vec{d_j} \in D_c} \vec{d_j} + 1 \sum_{\vec{q_j} \in Q} \vec{q_j} \quad (3.2)$$

D_a	Jobs applied for
D_b	Bookmarked jobs
D_{ub}	Un-bookmarked jobs
D_r	Read jobs
D_c	Clicked jobs
Q	Search queries

To give a concrete example for this algorithm, some vectors for a resume and jobs are defined in table 3.7. In order to simplify the example, these vectors only contain concepts from the IT skills. As can be seen from the resume vector, the user has expert skills in Java, intermediate skills in PHP and basic skills in C++ and Prolog. The jobs A to C, on the other hand, are classified with some IT skills with the weight 1.

Table 3.7: Example vectors for the resume and jobs

	Resume	**Job A**	**Job B**	**Job C**
Java	3	1		
C++	1		1	
C#		1		
PHP	2			1
Python				1
Prolog	1			

After the user has provided the resume, he performs various actions, which give feedback on the jobs (see table 3.8). For example, Job A is bookmarked, Job B is read for more than 20 seconds and Job C is just read for a few seconds.

If we use the data from the resume and from the relevance feedback as input for equation 3.2, we receive the final user profile. This result is summarized in table 3.9, showing the weighted resume vector, the weighted feedback vector and the final user profile vector.

Table 3.8: Example for relevance feedback

Action	Job
Apply	-
Bookmark	Job A
Read	Job B
Click	Job C

Table 3.9: Resulting user profile

	Weighted Resume	Weighted Feedback	Final User Profile
Java	6	3	9
C++	2	1	3
C#	0	3	3
PHP	4	0	4
Python	0	0	0
Prolog	2	0	2

3.3.5 Profile Adaption Technique

The preferences of a job seeker may change over time, e.g. someone might search for different jobs after finishing an education. So, recent feedback is often more relevant than older one. To take this into account, the user profile needs to be adapted over time.

In case of the resume, the user has to adapt it manually, e.g. by adding a new education, job experience or by updating the skills. But, on the other hand, an automatic adaption of the preferences from the relevance feedback is possible.

Several automatic methods have been discussed in chapter 2.1.3 . One possible solution is a fixed time window, within the feedback needs to take place. Too old feedback gets kicked out and ignored in the user profile. This simple approach already enables a change of preferences over the time, but it is a very coarse-grained strategy - either the feedback is in the user profile or not. If a user comes back after a period of time that is longer than the fixed time window, nothing can be said about the user's preferences.

Another similar approach uses a fixed window size, e.g. include the last 10 user actions and ignore the rest. This way, if a user comes back after a longer time span, some feedback will be still within the window size. On the other hand, if the user performs many actions within a small period, some relevant actions might get dropped due to the fixed window size. Additionally, both strategies don't consider the changes within a

window, e.g. feedback from a week ago should be less important than one from today.

A gradual forgetting function fixes the mentioned issues above. It weights each feedback according to its interaction date in a very fine-grained way. A possible function for that has been discussed in chapter 2.1.3 (equation 2.4), where a forgetting factor is used to weight a user action. In the following, this forgetting factor is referred as λ. Its values are between 1 and approximately 0 and the function is based on a half-life time span, which assumes that the preferences reduce by the half within this time.

For example, if we take a half-life time span of 14 days, we receive a λ of 0,5 after 14 days, 0,25 after 28 days and so on. Only feedback that takes place on exactly the same day gets the maximum weight of $\lambda = 1$. A graph with this half-life time span is given in figure 3.3. As can be seen, the exponential curve drops very fast at the beginning, but it takes a long time until λ gets close to 0. This allows to include feedback over a long period of time and to accordingly weight later feedback.

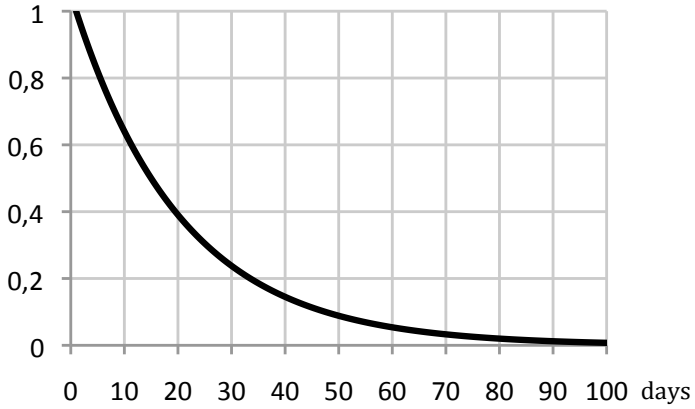


Figure 3.3: Forgetting factor λ with a half-life time span of 14 days

The forgetting factor can be easily integrated into the existing profile construction algorithm, which is shown in equation 3.3. The vector $\vec{d_j}$ is additionally multiplied by λ_j. In case the feedback is older than one day, this leads to a reduction of the vector weights and hence the influence of this feedback in the overall user profile is reduced day by day.

$$\vec{up} = \alpha\vec{r} + \sum_{c \in RF} (\beta_c \sum_{\vec{d_j} \in D_c} \lambda_j \vec{d_j}) \tag{3.3}$$

After we defined the forgetting function and the integration into the profile construction algorithm, the half-life time span parameters for the user actions still need to be deter-

mined. It will be possible to define different values for every user action category. The definition of appropriate default values for these parameters is again difficult and will be based on some assumptions. First, we assume that the job seeking process stretches over a longer period of time and that the preferences rather stay stable. Second, we know that some actions, like reading a job description, occur more often than others, like applying for job. For actions which re-occur often within a small period of time lower half-life time spans allow a more fine-grained weighting of the individual actions, e.g. a job read today should really be more important than one from two weeks ago. Even though the curve is steep at the beginning, with very long half-life values (e.g. 100 days), the differences in the weights λ between jobs from today and one from 2 weeks ago are small. Contrary, if the time span is very short (e.g. 7 days), the actions loose the influence very fast and it would be difficult to generate preferences when the user re-visits the site after a longer period of time.

In table 3.10 an overview of the default values for the half-life time span is given. Here, 14 days will be the half-life for the read action, which means that a read action older than 93 day has no influence on the user profile any more ($\lambda < 0{,}01$). Concerning the other actions, which occur less often, higher values will be used. The bookmark and search actions have a *hl* of 28 days, which is the double than the one for the read action, whereas the apply action has again the double with a *hl* of 56 days. This implies that applications have a long influence on the user profile and it will take around a year until this action is neglectable ($\lambda < 0{,}01$).

Table 3.10: Half-life time spans for the user actions

Action	Half-life in days
Apply	56
Bookmark	28
Read	14
Search	28

3.3.6 Summary of the Design Decisions

In the previous chapters the main design decisions have been discussed in detail. To give a quick overview, the decisions are summarized in the following:

- **Profile representation**
  The representation is based on vectors, which are structured by fields and mostly contain concepts of the taxonomy. The weight for each concept defines the degree of interest in this topic.

- **Initial profile**
  The resume, which has to be manually provided by the user, builds the initial profile. However, the resume is optional for the final user profile.
- **Relevance feedback**
  Relevance feedback will be collected in an implicit way from various user actions. This includes applying for a job, (un-) bookmarking, reading (longer than 20 seconds), clicking and searching. The actions will be weighted according to their level of evidence for interest.
- **Profile learning and construction technique**
  A linear combination of the resume and the relevance feedback will lead to the final user profile. The approach follows the principle of Rocchio's algorithm and allows the weighting of user actions.
- **Profile adaption technique**
  A gradual forgetting function will be used to decay the user's preferences as the time goes on.

3.4 Recommendation Technique

In the existing implementation, a content-based recommender has been used for matching the resumes and jobs. This technique makes sense in this case, as it compares the capabilities of an applicant with the required skills in a job ad. The more skills match, the higher the score. If the system would be based on a collaborative filtering recommender, it wouldn't be possible to ensure that a job fits the user's capabilities, because only the fact that a similar user likes this job, doesn't say much about current user.

In the new implementation, the preferences of a job seeker will be added to the user profile and it is planned to stick to the content-based recommender, because still, directly comparing the skills a user prefers to the available ones sounds reasonable. Of course, other ways, like a hybrid combination, which compares the resume in a content-based way and the preferences of a user with a collaborative filtering recommender, would be possible, but this is not the primarily topic of this thesis.

Although later on additional knowledge of the taxonomies will be added to the user profiles and job ads, this recommender still follows more a content-based than a typical knowledge-based approach. Compared to knowledge-based recommenders, we have long term user profiles, no conversational user interface and no typical knowledge-based matching method, like case-based reasoning.

In the last chapter, a central part of the content-based recommender, the content-based profile, has already been discussed in detail. This profile has to be compared to the

available jobs. As we use a vector-based user profile representation, the jobs ads need to be transformed to the same structure. This transformation task and the storing of the data structures will be part of the Indexer component, which will be further addressed in the implementation chapter.

But before moving on to the implementation, some adaptions concerning the content-based recommender, which are necessary due to new requirements, have to be discussed first. This includes the method information from the taxonomy is added to the vectors and which similarity metric for comparing user profiles and job ads is used.

3.4.1 Taxonomy

Absolventen.at has several taxonomies about topics like IT skills, fields of study and so on. Concepts in these taxonomies are structured in a hierarchical way, so that specific concepts are grouped by more generic concepts.

Information from the taxonomies is another useful input for the recommender and allows a matching over more general concepts. As explained in chapter 2.2.3 , for adding data from the taxonomy, two different use cases have to be considered: the *subsumption* and the *proximity matching* [4].

In case of subsumption matching, a user profile containing a child concept exactly matches a job that is annotated with its parent concept. In figure 3.4, a concrete example is given. It shows a part of the fields of study taxonomy, which is grouped by general categories, like computer science, and its corresponding specific fields of study, like software engineering or media informatics. When companies publish jobs, they always select the general category, whereas applicants specify the concrete education and field of study in their resume. If an applicant has studied business informatics, then he should perfectly match a job that requires someone from the computer science field.

To implement the subsumption matching functionality, only parent concepts are going to be stored in the vectors. For the jobs, this happens by default, as the companies already provide these values. On the other hand, for user profiles, parent concepts need to be derived by the Indexer and stored in the profile vector, the same way as it has been done so far.

The proximity matching covers a slightly different use case. For example, both, the job as well as the resume contain the most specific IT skill concepts. An exact match is only given, if both vectors contain the same concept. But even if the concepts are different, they might have the same parent concept. So, at least this parent concept matches, but it should have less influence on the final score than an exact match. In figure 3.5, part of the IT skills taxonomy is given. In this example, an applicant has Java skills, whereas a job requires C++ skills. Although the skills are not the same, they are sibling concepts

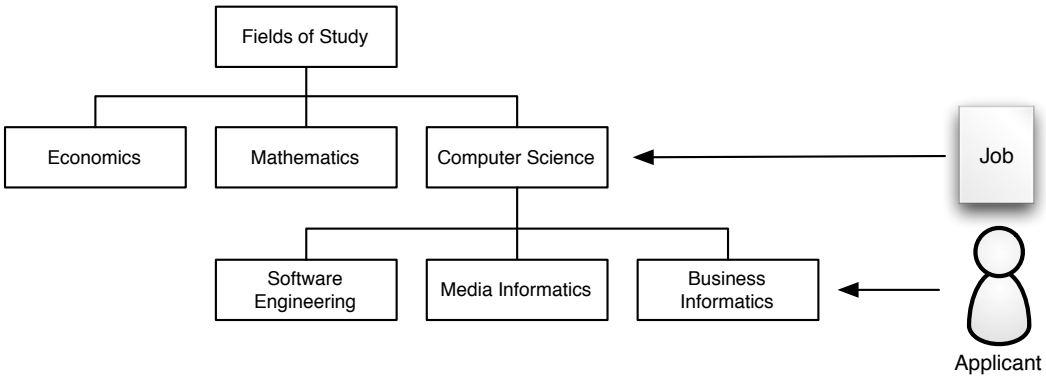


Figure 3.4: Subsumption matching

and both require programming skills. A further distinction of programming skills, like object-oriented, procedural etc, of course would make sense here.

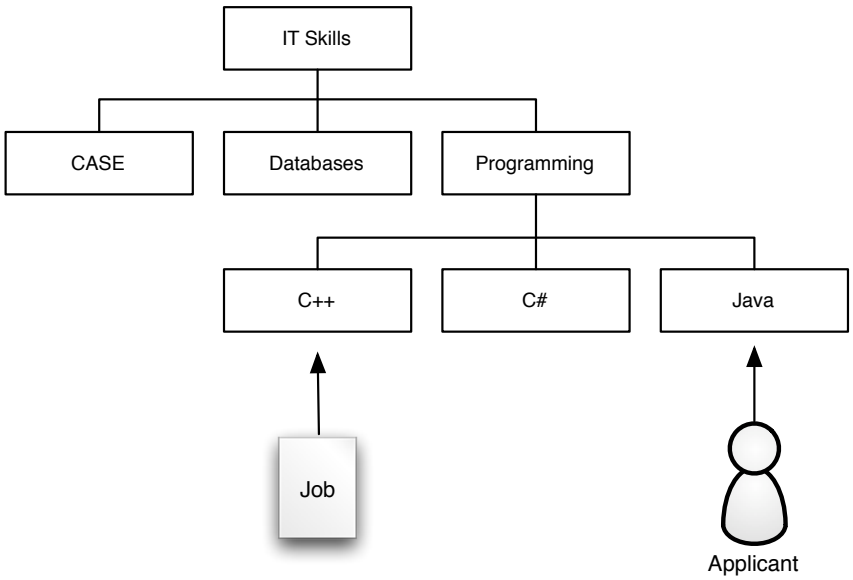


Figure 3.5: Proximity matching

In the existing implementation all parent concepts have been added into the same data structure as the child concepts and hence it has been difficult to express that a match over a common parent is less important than an exact match. A sublinear weighting schema

and the IDF have been used to decrease the influence of parent concept matches, but this is rather inflexible. To improve that, a separate field in the vectors will be introduced, which only stores the parent concepts. As a result, lower field weights can be applied for parent fields than for child fields, leading to a lower parent match score. In case the taxonomy has more hierarchy levels than the provided example, a separate field for each level is the most flexible approach. Furthermore, even though the influence of parent fields can now be separately adjusted with weights, a sublinear weighting is still appropriate in order to keep the weights low. Otherwise, extreme weights for one parent concept might lead to recommendations that only stick a certain topic. An overview of all field weights, including the new ones for the parent concepts, is shown in table 3.11. By default, parent fields will only have the half weight than their child fields.

Table 3.11: Field weights

Field	Default weight
field_general_skills	1
field_general_skills_parents	0,5
field_it_skills	1
field_it_skills_parents	0,5
field_languages	1
field_field_of_study	20
field_region	20

As explained in chapter 2.2.3, on other systems (e.g. [3], [4], [7]) a distance measure between concepts is applied, where the distance between parent-child concepts, like computer science - business informatics, is smaller than the distance between siblings, like Java and C++. With this measure both use cases would be supported as well. Anyhow, it is difficult to integrate such a measure into the existing content-based comparison and storing the parents in a separate field should have the same effect.

3.4.2 Similarity Metric

After the vectors with its terms and weights have been generated, a content-based comparison between the user profile and the available jobs needs to be performed. For finding the most similar jobs or the most similar applicants, a simple *overlap score measure* has been applied so far, which calculates the dot product between the vectors. The higher the score, the more relevant a document is.

The decision for the overlap score similarity metric has mainly been motivated by one requirement: for recommending applicants to jobs, applicants need to be ordered ac-

cording to their skill level. Someone with many expert skills should have a higher score than someone with basic skills. This still holds true for the applicant recommender, but for the job recommender other metrics might be more appropriate.

In the information retrieval area, problems with this similarity metric are known. Documents with more text, and as a result of that larger vectors, have a higher chance for a high score than shorter documents, even though shorter documents might be as relevant [47].

In our case, a similar problem exists, mainly introduced by the way parent concepts are added to the vectors for proximity matching. A score between a user profile and a job can be high in case the parent concepts match, even if no child concepts are common. This problem will later on be illustrated by a concrete example, but first, let's see how this problem can be tackled.

A possible way, used in the information retrieval area, is a normalization of vectors by their Euclidean length, resulting in the unit vectors. If this normalization is applied to the dot product of vectors, the cosine between those two vectors is calculated (see chapter 2.2.2, equation 2.5). The score will be a number between 0 and 1, the higher it is, the closer and thus more similar two vectors are.

Several other ways for normalizations exist, and in some cases it can be problematic to remove the length information. For example, if we compare two resumes, one, which has only basic skills, will have the same unit vector as one, which has the same skills with expert level. Still, for recommending jobs, a normalization can be useful and jobs have, except for parent concepts, equal weights for the concepts anyway.

Assume the scenario given in table 3.12, which shows vectors for an applicant and for some possible jobs (weights are without the IDF, but include the field weights and the sublinear weighting schema for parent concepts). The applicant has expert skills in Java and intermediate skills in C++, resulting in a weight of 1,65 $(0,5 \cdot (1 + \log_2 5))$ for the parent concept Programming. Job 1 requires exactly the same skills as the applicant has. The other ones are slightly different and require more skills, leading to a higher weight for the parent concept. In this situation, we would expect Job 1 is recommended to the applicant at first place.

In table 3.13 some example IDF values for the similarity calculations are given. The data is based on the available resumes, for the final recommender, the data from the jobs will be included as well, which slightly changes the IDF values. As can be seen, a rather seldom skill C# has almost the double IDF than the parent term Programming.

If we apply both similarity metrics, the results are quite different, as shown in table 3.14. With the overlap score measure, we receive a ranking with: Job 4 > Job 2 > Job 1 > Job 3. Job 4 requires the most skills, resulting in a high weight for the parent concept and as a result of that also in a high score. Of course, the differences are small and it would

Table 3.12: Example vectors for the resume and jobs

	Applicant	**Job 1**	**Job 2**	**Job 3**	**Job 4**
Java	3	1	1	1	1
C++	2	1	1	0	1
C#	0	0	1	1	1
PHP	0	0	0	1	1
Programming	1,65	1	1,25	1,25	1,5

Table 3.13: Example values for the IDF

	DocFreq	**IDF**
	N = 5614	
Java	874	0,81
C++	1108	0,70
C#	627	0,95
PHP	726	0,89
Programming	1856	0,48

be possible to improve this with smaller weights for the parent concepts, but contrary, the results from the cosine similarity match the expectations by suggesting a ranking as follows: Job 1 > Job 2 > Job 4 > Job 3.

Table 3.14: Scores from example vectors

	Job 1	**Job 2**	**Job 3**	**Job 4**
Overlap	3,33	3,34	2,43	3,52
Cosine	0,97	0,76	0,51	0,66

Consequently, the job recommender will use the cosine similarity metric, while the applicant recommender might still stick to the original overlap score similarity metric. For this purpose, the implementation should regard that the similarity part should be pluggable for different metrics. Furthermore, it has to be mentioned that the given examples for a resume and jobs are fairly simple. It might happen that real world examples behave different. An evaluation of different metrics would also be an interesting topic, but can't be further addressed in this thesis, as the focus lies on other issues.

When looking at the requirements, it has been stated that the execution performance is important for the recommender. As long as the vectors can be retrieved in an efficient

way, the similarity calculation itself is pretty simple and fast. Even though a user profile has to be compared to a large amount of jobs, the existing implementation has shown good performance and the adaptions in the similarity metric won't have any negative influence. Regressions in the performance are more likely be caused by more comprehensive user profiles.

CHAPTER 4

Implementation

In this chapter details about the implementation of the job recommender will be presented. The requirements and its architecture have been pointed out in the previous chapter and now the available tools will be used to realize the plans.

Relying on the CMS Drupal 7 as a job board platform was a prerequisite of the project, but the choice of a technology for implementing the recommender was left open, although good experiences with Apache Solr have been made so far. For this purpose, some alternatives were analyzed, which are going to be discussed in the next section, followed by the description of the implementation.

4.1 Recommender Technologies

For implementing a job recommender, four different and freely available technologies were taken into consideration, including Apache Mahout[1], easyrec[2], Drupal's Recommender API[3] and Apache Solr[4]. The main characteristics and their strong and weak sides are given in the following.

4.1.1 Apache Mahout

Mahout is an open source Java library, which supports many Machine Learning algorithms, including a wide range of recommender techniques, like collaborative filtering

[1] http://mahout.apache.org
[2] http://www.easyrec.org
[3] http://drupal.org/project/recommender
[4] http://lucene.apache.org/solr

and content-based recommenders. A main goal of this library is to provide a scalable solution, which can be achieved by distributing it via a Hadoop cluster [38].

Advantages:

- Supports many algorithms out of the box.
- Scalable implementation.
- Integrated evaluation component.

Disadvantages:

- No integration for Drupal yet exists, even though it is planned to be added during a Google Summer of Code Project 2011 [59].
- Difficult to realize a hybrid user profile, as it mainly focuses on user preferences.

4.1.2 easyrec

easyrec is another example for a recommender library, which is written in Java and available as open source project. Recently (March 2011) an integration for Drupal's Ubercat[5] shop system has been published. Furthermore, as mentioned in a blog post [13], an integration of Mahout into easyrec might be implemented in future.

Advantages:

- Easy to use via a REST API.
- Integration into Drupal 6 & 7 exists, although it hasn't been available when starting with the implementation.

Disadvantages:

- Rather designed for simple recommender use cases, like web shops.
- No content-based algorithms are available at the moment.
- As Mahout, based on user preferences and hence an integration of information out of a resume seems to be difficult.

[5] http://drupal.org/project/easyrec_for_ubercart

4.1.3 Drupal's Recommender API

Drupal's Recommender API is a contributed PHP module, which can be easily installed on any Drupal website. Its algorithms are implemented in PHP and it mainly focuses on the web shop use case.

Advantages:

- Drupal module.
- It's planned to use Apache Mahout in future (Google Summer of Code Project 2011) [59].

Disadvantages:

- Drupal 6 only.
- Implemented in PHP, which isn't fast nor scalable for such resource-intensive tasks.
- Similar user model as Mahout and easyrec.

4.1.4 Apache SoIr

Apache Solr is an open source search framework, which is also written in Java. Internally it is based on Apache Lucene and provides a standalone server for it, which communicates with other applications via REST-like HTTP requests. It is mainly used for fulltext search and offers many additional features, like faceted searches and so. The use of vectors for the document representation makes it also useable as recommendation framework, which has been done on Absolventen.at so far.

Advantages:

- Drupal 7 integration already exists.
- Fast and scalable implementation.
- Possibility to extend Solr with own plugins for the current use case.
- Lots of experiences with Solr are already available.

Disadvantages:

- Focus on fulltext searches.
- Fixed vector-based implementation and due to that, other algorithms are not possible.
- Requires own plugin for recommendations.

The presented recommendation frameworks, especially Mahout and easyrec, are very interesting and promising solutions. Nevertheless, it should be mentioned that they rather fit for typical recommender use cases, like web shops, and are not appropriate for the job recommender, where different kinds of information, like a resume, skills levels, etc., need to be taken into account. That's why it was decided to stick to Apache Solr, even though the complete recommender will be re-implemented anyways.

4.2 Overview

As shown in figure 4.1, Drupal will be responsible for providing the functionality of a job board, including the recording of the various user actions for the implicit relevance feedback, as well as transforming jobs, resumes and user feedback into appropriate data structures, that are transmitted to Solr, which implements the recommender. Both components communicate via a REST-like HTTP connection. As a result of that, integrating the recommender on other platforms than Drupal is possible.

Furthermore, Drupal will serve as administration interface for creating, managing and evaluating the recommender configurations. The configurations are usually sent together with the recommender request to Solr, which is also highly configurable via the passed parameters.

In order to implement the job recommender, which fulfills all necessary requirements, several components for both, Drupal as well as Solr, were programed, which will be discussed in the next chapters.

4.2.1 Drupal

For realizing the recommender, two separate parts have been necessary for Drupal. First, a generic system for logging the user actions and second the recommender itself, which provides configuration possibilities and interacts with Solr. As it has been stated in the requirements, a generic and re-usable implementation of the recommender is desired.

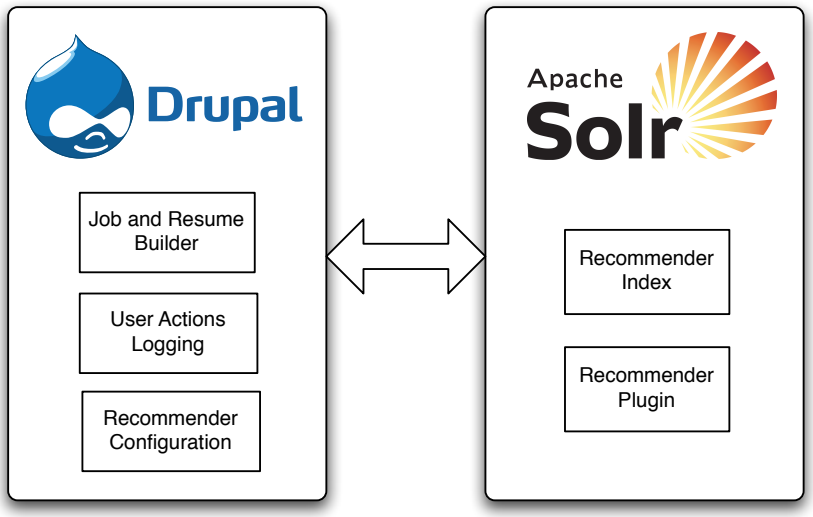


Figure 4.1: Overview of used technologies

To achieve that, it was decided to build upon Drupal 7 and its most common third party module where possible.

4.2.1.1 User Actions Logging System

On the job board, we have actions like reading a job, bookmarking, searching etc. Some of them provide references to a job, others search words. Furthermore, the functionality for such actions, like bookmarking a job, is provided by various different modules.

In order to avoid implementing a logging system for each kind of possible action, it was decided to base the system on two pillars: the Message and the Rules module.

The Message module itself is based on Drupal's core entity system and uses extended functionality of the third party module Entity API. The messages can be utilized to log any kind of information, not necessarily text messages, for a certain user. In addition to that, different types in form of entity bundles can exist, like one for searches, one for read times and so on. The message entity itself is fieldable, which means that the basic data structure of a message bundle can be easily extended with fields from Drupal's Field API. So we decided to add a node[6] reference field on a relevance feedback message

[6]The term "node" is used in Drupal for the main content entity, like a job.

type, which stores the referenced job, e.g. which job has been read. Any other fields are possible, like one for storing the read times, one for the search words, etc.

The Field API takes care of creating the according database tables and of storing and retrieving the data. Furthermore, many important modules are based on the entity and field concept, which simplifies the use of the recorded data. For example, the Views modules enables showing the complete message log to administrators.

For each user action, a separate message type will be created. To ease the start, a default message type for relevance feedback was defined and exported via the Features modules, which allows others to import the default configuration.

The only thing that's yet missing, is the creation of message logs upon some user actions. For this purpose, the Rules module is used, which enables website administrators to create actions, like writing a log for a certain user, which are triggered on certain events, like the user bookmarks a job. These rules also may have some conditions attached, like the acting user needs to be an applicant. With this system, it is easy to create the complete message log, the only requirement is that modules provide appropriate Rules events.

For logging the read times, an own module for measuring the times and invoking a Rules event was implemented. In order to see how long the user views a webpage, a JavaScript code is included in the webpage, which fires an AJAX callback when the user quits the page. The callback receives the information about the acting user, the webpage and the read time, and invokes the Rules event, upon which the message log will be written to the database.

These messages will then be used by the recommender system for the user preferences. The way it integrates with the rest of the system is shown in the next chapter.

4.2.1.2 Recommender Module

An overview of the main involved components, and how they interact, is given in figure 4.2. The recommender system, which was implemented for this project, is referred as "Reco" in the following.

When creating a recommender, the administrator starts with the basic Reco Config, which contains information about the involved entities, like the job node and the resume profile, which data structures in form of special matching fields will be used and which messages for relevance feedback are available. One such configuration is created for handling the complete job and applicant use case. However, for actually recommending jobs to applicants or for recommending applicants to jobs, separate Reco Search Configs, which are based on the same Reco Config, are necessary.

But first, the data from jobs, resumes and the relevance feedback needs to be indexed in Solr, which is done by the Search API. The so called matching fields, which are actually Entity Metadata properties and not persistently stored, need to be indexed for Solr's recommender. Such a matching field contains multiple value - weight pairs, representing a vector. Rules components, which can be configured by the administrator, are used to map the different data structures to the matching fields before the data is sent to Solr. These components extract information from different data entities, transform it, if required, and write the data into the properties. For example, the specific fields of study from several educations of a resume are fetched, the parent concepts derived, the weights calculated and finally written into the property. All of these processing tasks can be configured by the administrator via Rules actions. The message logs for the relevance feedback do not need any further transformation and are kept in an own Search API index and will be further addressed in Solr.

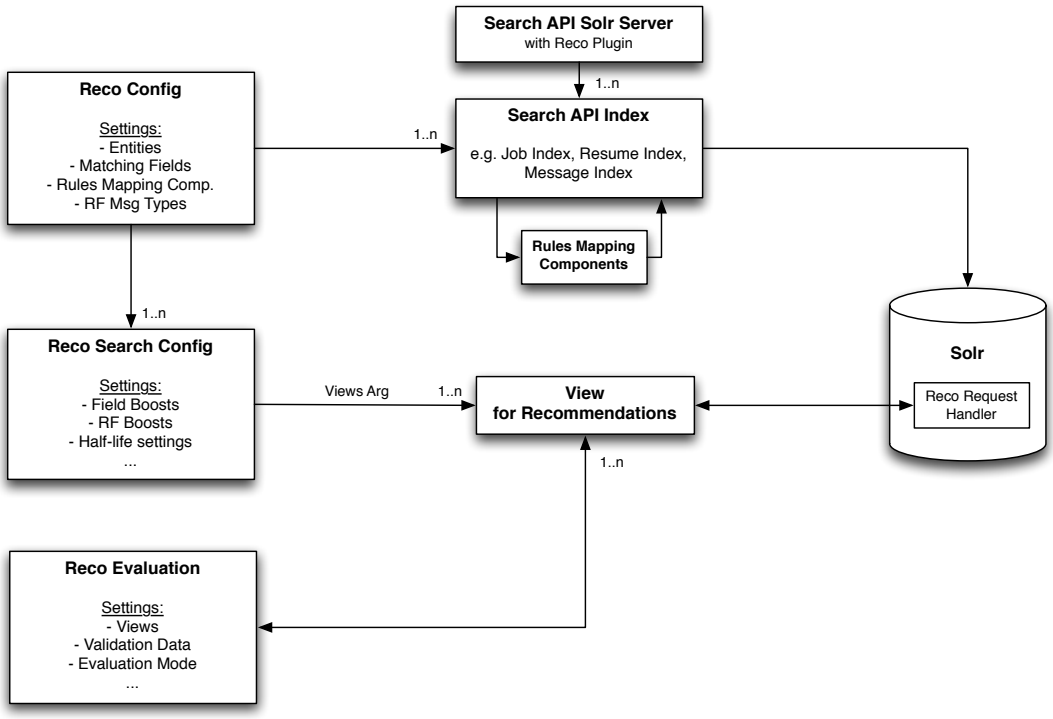
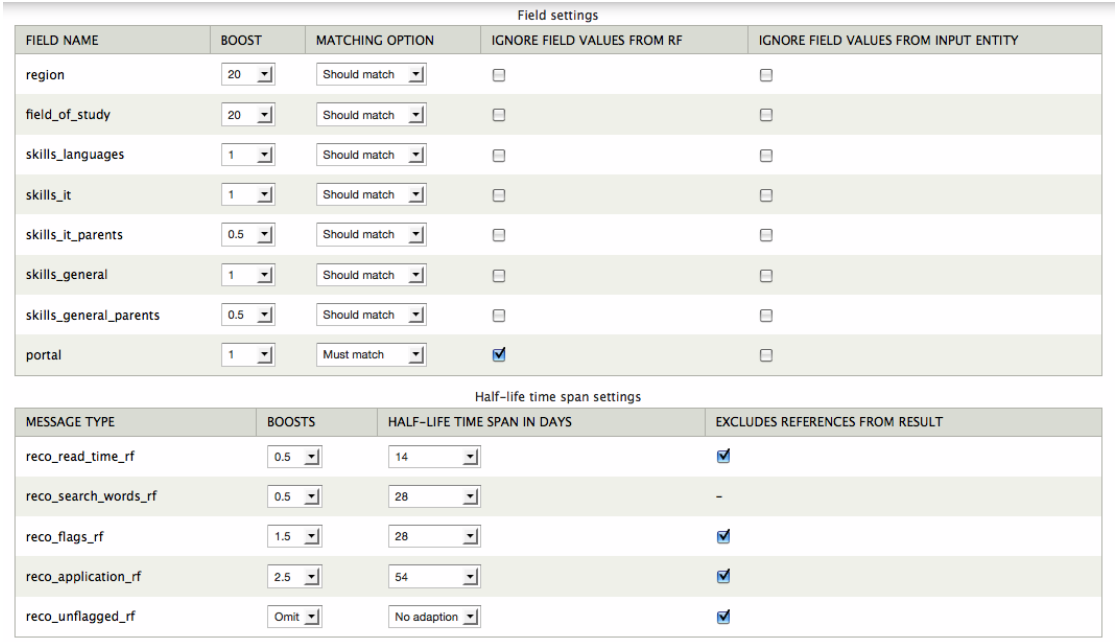


Figure 4.2: Drupal architecture for the recommender

As soon as the data is indexed by Solr, recommendations can be retrieved. For that, a Reco Search Config needs to be created, which specifies the recommendation type, like jobs to applicants or the other way round, the involved matching fields with their

weights, the available relevance feedback, again including weights and half-life time span settings, and some other advanced options, like the read time threshold or the similarity metric. A snippet of the administration UI is shown in figure 4.3.



Field settings

FIELD NAME	BOOST	MATCHING OPTION	IGNORE FIELD VALUES FROM RF	IGNORE FIELD VALUES FROM INPUT ENTITY
region	20	Should match	☐	☐
field_of_study	20	Should match	☐	☐
skills_languages	1	Should match	☐	☐
skills_it	1	Should match	☐	☐
skills_it_parents	0.5	Should match	☐	☐
skills_general	1	Should match	☐	☐
skills_general_parents	0.5	Should match	☐	☐
portal	1	Must match	☑	☐

Half-life time span settings

MESSAGE TYPE	BOOSTS	HALF-LIFE TIME SPAN IN DAYS	EXCLUDES REFERENCES FROM RESULT
reco_read_time_rf	0.5	14	☑
reco_search_words_rf	0.5	28	-
reco_flags_rf	1.5	28	☑
reco_application_rf	2.5	54	☑
reco_unflagged_rf	Omit	No adaption	☑

Figure 4.3: Screenshot of the admin interface for managing the recommender

For each of the Reco Search Configs, a Views argument is automatically generated. This argument can be used together with Views for retrieving and displaying the recommendations. Views allows to create flexible listings of the recommended entities and further filters, like including only the currently published jobs, can be added.

As can be seen, many different settings are possible for the recommender. To see how changes affect the recommendations, an Evaluation component was implemented. This component enables administrators to compare the results of different recommender views with a predefined set of relevant items, which are usually taken from the relevance feedback. Which relevance feedback categories are used, how the data is split into a training and validation set and if cross-validations showed be performed, can all be configured in the Evaluation component. The evaluation itself is executed as a batch operation and returns a summary, as well as detailed results for each user sample, including metrics like the precision, recall, F-Measure, execution times and so on.

All of these configurations are based on the Entity API and as a result of that, they can be easily exported and imported on other sites, as well as being versioned in a Features module.

4.2.2 Solr

Apache Solr is responsible for computing the actual recommendations. For that, two components have been required: first an optimized index for storing the data structures and second a plugin for retrieving the recommendations.

4.2.2.1 Recommender Index

For the job recommender, three different entity types, including the jobs, the resumes and the relevance feedback messages, need to be stored by Solr. According to Drupal's Search API naming conventions, three different indexes are used for this purpose, but actually all information is kept in the same Solr index and the entities are just distinguished by an additional field.

The most important information for the recommender is located in the matching fields, which have some kind of a vector format. For correctly storing the value-weight pairs, some further configurations in the schema.xml file were done, like setting the *termVectors* variable to true in order to save the weights. Solr then keeps a separate file just for the term vectors, which can be loaded via the Solr API in a plugin or retrieved via a Solr request in an external application.

4.2.2.2 Recommender Plugin

Solr already provides some request handlers, but they rather focus on traditional searches. Only the MoreLikeThis handler, which is typically used for retrieving similar content, e.g. jobs similar to the current one, has some commonalities, but is again not powerful and flexible enough for our use case. Nevertheless, Solr can be extended with own plugins, which just need to be registered in the solrconfig.xml file.

Such a plugin was written for this project. It receives a request containing all necessary information and configurations from Drupal, or any other application. The plugin then extracts the parameters and translates them to the internal configuration options. The main task of the plugin is the calculation of the query vector, the way it is described in chapter 3.3.4 for constructing the hybrid profile (equation 3.3). This query vector will then be applied for executing searches and the recommendation result is returned to Drupal.

In figure 4.4 the main classes of the plugin and the interactions are drawn. It's not meant to contain all classes nor connections, it rather wants to give an overview of the most important implementation parts. In order to provide a more detailed insight, the process for handling a typical recommendation request is described in the following.

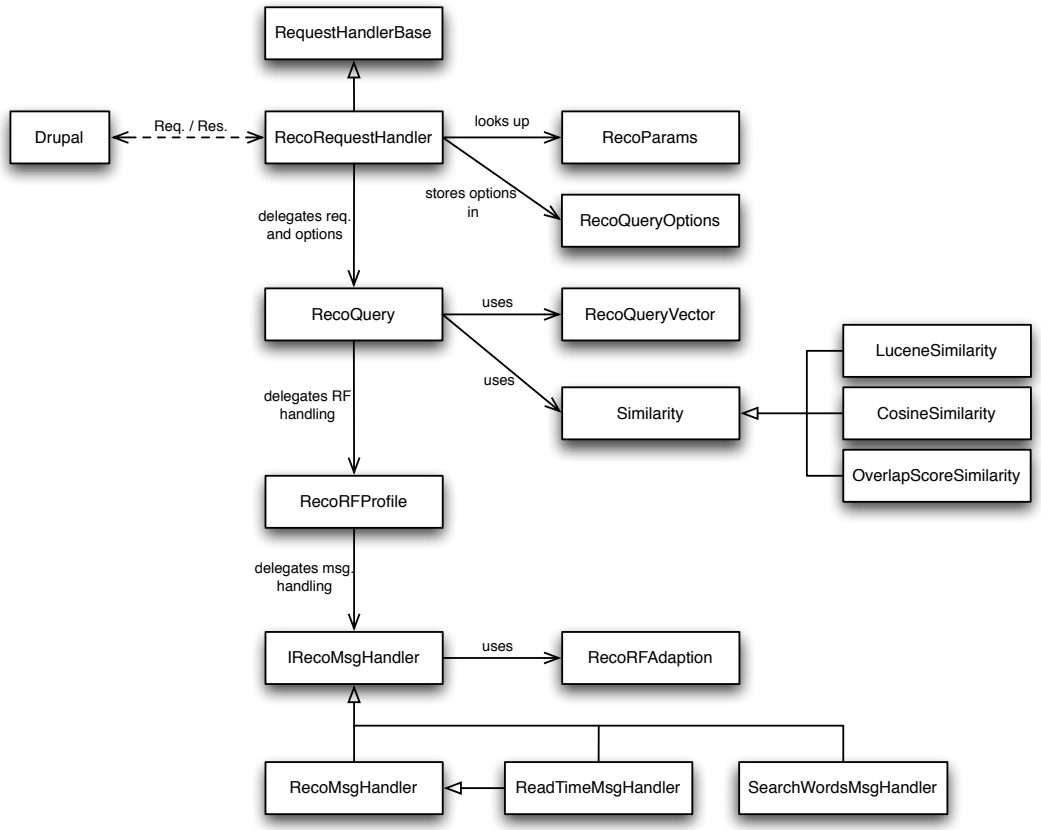


Figure 4.4: Solr architecture for the recommender plugin

1. The RecoRequestHandler, which extends the default Solr RequestHandlerBase class, receives a query string via a HTTP request. Then it looks up all possible configuration names, which are defined in RecoParams, for extracting the recommender options, like for which user recommendations should be made, the field weights and so on. All of these options are put into a RecoQueryOptions object, which is further on used by the others.

2. The recommender request and its options are then passed to a RecoQuery object, which is responsible for coordinating the main tasks. First, it initializes a query vector, which optionally contains data from the base document, in our case the resume. Then, according to the settings, this vector is extended with relevance feedback.

3. For the relevance feedback, a RecoRFProfile object looks up all available feed-

back from the message log belonging to the current user.

4. Each message type may have its own handler, which needs to implement the IRecoMsgHandler interface and needs to be available to Solr. A default implemenation with the RecoMsgHandler class already exists, which handles simple feedback like the bookmarks or applications. Other feedback types, which have additional data, like the read times or the search words, have their own message handler. The ReadTimeMsgHandler, for example, additionally checks, whether the read time is higher than the given threshold.

5. The information from the relevance feedback, in most cases an extracted term vector from referenced job documents, is then merged into the query vector. But before merging it, the weights of the term vector are adapted by the RecoRFAdaption class.

6. As soon as the query vector is completed, the RecoQuery object executes the query, which maps the values from the vector to optional boolean clauses with appropriate weights. The similarity metric for the query execution is configurable, but the provided class must extend Lucene's similarity class. The results from the query, including a score for each possible document, are then sent back to Drupal.

This plugin offers many configuration possibilities and as a result of that, it is not only limited to be applied for the job recommender. Basically it can be used for any kind of content-based recommenders, which should follow the described algorithm. Furthermore, it doesn't depend on Drupal, any other application may make use of it, as long as the request parameters are correctly set.

CHAPTER 5

Evaluation

After the implementation of the recommender system, we want to assess whether the introduced changes lead to a more successful system or not. For this purpose an evaluation system that is able to compare two or more different recommender approaches is needed. This system will allow us to discuss the main question of this thesis, whether implicit feedback helps to improve the job recommender, as well as it will allow the website administrators to fine-tune the parameters of the recommender.

But what makes a recommender "better" than the other? Some kind of measure is required that indicates the performance of one recommender and which can be compared to others. Usually recommender systems want to maximize the user satisfaction, which might be counted by the number of visits, purchases of products, applications for jobs and so on. For a job recommender, not only the user satisfaction, but also how well a job fits the user's capabilities might be relevant considerations. Always recommending jobs that require more education than the user has might be frustrating.

In the field of information retrieval, and more specifically in the recommender systems area, several approaches for evaluating a system are known. An overview of different methods will be given in the next chapter, followed by possible evaluation scenarios for our job recommender.

Afterwards the main evaluation with the job recommender will be performed, which compares a hybrid user profile with one that only consists of the resume. The results will then be discussed and presented in an appropriate way.

5.1 Theoretical Foundations

For evaluating a recommender system, a distinction between two different approaches should be made first: [24]

- **Off-line evaluations** use the recorded data from the past in order to perform experiments with different recommenders.
- **On-line evaluations**, on the opposite, observe people using the recommender system and analyze the observations for making statements about the performance of a recommender.

For online evaluations the availability of users who are going to participate in the experiments is crucial [24]. If the system is already running on an active website with a large community of users, evaluations with a deployment of different recommenders are possible, an approach that is also known as *A/B testing* [30]. In such a case, users are randomly divided into two groups. One part, e.g. 1-10% of the overall users, is directed to the new system, whereas the other users are still using the old one.

A framework for online evaluations of recommender system, which follows the principle of *A/B testing*, was proposed by Hayes et al. [18]. The architecture of this framework can be seen in figure 5.1. The results of two different recommenders are directed through the presentation layer, which either combines the results of both in one list, shows two separate lists, or switches between the recommenders. Then the behavior of the user is recorded by the feedback detector. Again, explicit feedback, like ratings, as well as implicit feedback, like reading times, can be gathered and used for comparing the two recommenders.

Yang et al. [58] also noted that for correct experiments, a control group with random recommendations was needed. This can be problematic, in particular, on live websites, as by showing random and probably inappropriate recommendations a low user satisfaction is risked.

In case a real live system with users isn't available, controlled user studies can be performed as well, e.g. which was done in studies by Claypool et al. [10] and Lee et al. [26]. However, those studies are often time consuming and expensive.

Offline evaluations, on the other hand, highly depend on the availability of collected test data sets. As this approach doesn't necessarily require a working live site, nor real users, it is often easier to perform [24].

Such offline data sets usually contain statements about relevant items for a user. In the information retrieval area, the same approach has been widely used, e.g. the TREC

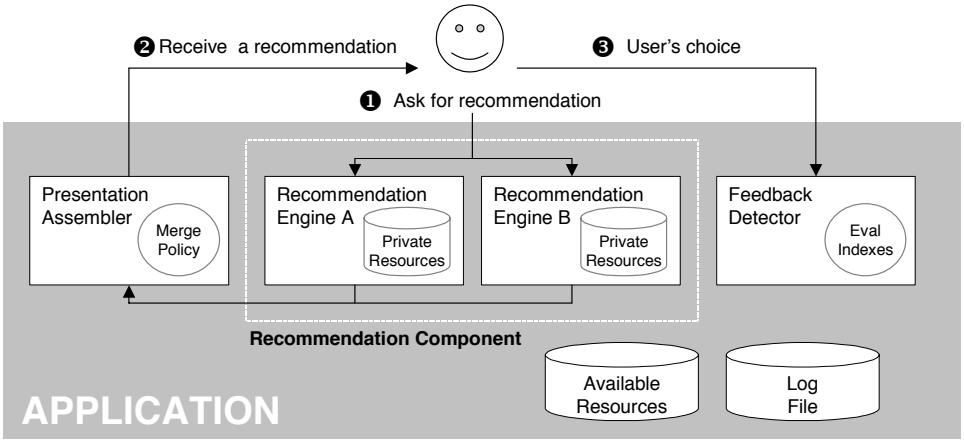


Figure 5.1: Online evaluation framework for recommender systems [18]

system contains over 1,8 millions of documents with relevance statements for 450 information needs (search queries). These statements, whether an item is relevant or non-relevant for a specific search query, are also referred as the *gold standard* or *ground truth* [30]. They are either manually created by experts, who judge if a document fits a certain information need, or automatically generated from user observations, which is mostly done for recommenders.

For example, the user's ratings, purchases or applications can be used as relevant items, since we know the user definitely likes them. The more recommendations match with the relevant ones, the better. In case this information should also be used as input for the recommender, it has to be split into mutual exclusive sets for training and validation data, an approach commonly used in machine learning. The training set is used for the recommender, while the validation set tests whether the recommender is able to predict the relevant items [18].

This way of performing evaluations, for example, was done in studies by Burke et al. [8] and Rafter et al. [41]. Especially the evaluation of Rafter et al. is of special interest for this thesis, as it was performed with a job recommender as well. There, the jobs for which the user applied, were used to validate the recommendations.

Additionally, this method allows *k-fold cross validations*, where the evaluation for each user sample is cross checked with swapped training and validation data sets. k denotes the amount of sets, e.g. in a 10-fold cross validation, the data is split into 10 sets. For each k, a separate check will be performed, where the current fold is used as validation set and the rest $k-1$ sets as training data. The Leave-One-Out cross validation (LOOCV) and the repeated cross validation are special cases. In LOOCV, each observation repre-

sents a separate fold and only one observation is used for the validation, while the rest is passed as training data to the system. For repeated cross validations, the same cross evaluation is performed multiple times with reshuffled training and validation data splits, e.g. in a 5x2-fold cross validation, the observations are randomly split in two equal sized training and validation sets and the evaluation is repeated 5 times with different splits, leading to 10 comparisons (5 repetitions with 2 cross checks each) [43].

Besides the decision for the evaluation approach, whether offline, online etc., appropriate metrics for measuring the performance must be identified before performing the evaluation. The most common ones will be discussed in the following chapter.

5.1.1 Metrics

When measuring the performance of a recommender system, the commonly used metrics can be divided into two groups [19]:

- **Coverage**, where the amount of items that can be recommended to the users is counted. For example, in an collaborative filtering environment, an item with few ratings might not be recommended to users.
- **Accuracy** metrics, on the other side, compare the recommendations with the actual known relevant items or ratings. For this purpose statistical or decision-support metrics can be applied. With statistical metrics, for example, the Mean Absolute Error (MAE) between the predicted and the actual ratings is measured. In contrast, decision-support metrics compare the recommended items with the relevant ones, e.g. by counting the overlap.

While the coverage might be an important measure for collaborative-filtering, it might not be that relevant in a content-based recommender system, where usually any item containing some exploitable information can be recommended. In addition to that, what has been missing in our point of view in [19], is the user coverage, where the percentage of users for whom personalized recommendations can be made is measured. Later on, this measure will be used when comparing the job recommender approaches.

Concerning the accuracy measures, in particular the decision-support metrics will play an important role for the job recommender evaluations. Many metrics for this purpose are well known from the information retrieval area [30] and will be discussed in the following.

The *precision* (equation 5.1) measures the portion of items that are relevant within the received result.

$$Precision\ p = \frac{\#(relevant\ items\ retrieved)}{\#(retrieved\ items)} \tag{5.1}$$

In contrast, the *recall* (equation 5.2) measures the portion of relevant items that have been retrieved. Both metrics should be used in common, as with increasing the amount of retrieved items, the recall increases, whereas the precision usually drops with larger result sizes.

$$\textit{Recall } r = \frac{\textit{\#(relevant items retrieved)}}{\textit{\#(relevant items)}} \tag{5.2}$$

A metric that considers both values is the *F-Measure* (equation 5.3), which calculates the mean of the recall and the precision. β can be used to weight the influence of one of both, where $\beta > 1$ increases the importance of the precision and $\beta < 1$, on the opposite, raises the influence of the recall. For a balanced F-Measure, $\beta = 1$ is used. Whether the precision or the recall is more important, depends on the goals of an application. In some cases, it is desired that the first shown results are really relevant (thus high precision). On the other hand, for some applications it might be necessary that every relevant item is actually retrieved (high recall), even though the user potentially has to read through a long result list.

$$\textit{F-Measure } F = \frac{(\beta^2 + 1) \cdot p \cdot r}{\beta^2 \cdot p + r} \tag{5.3}$$

Another possible metric is the *accuracy* itself (equation 5.4), which returns the fraction of relevant items retrieved (true positives) and the non-relevant items not retrieved (true negatives) versus all items. This measure can be maximized by categorizing all items as non-relevant and returning no results and, as in many cases 99,9% of all items are non-relevant, this metric is often inappropriate. The precision and recall, which focus on the relevant items, are usually a better choice.

$$\textit{Accuracy } a = \frac{\textit{\#(relevant items retrieved)} + \textit{\#(non-relevant items not retrieved)}}{\textit{\#(items)}} \tag{5.4}$$

So far the mentioned metrics do not consider the ranking of the retrieved items, e.g. if the result size is 10, it has no influence on the precision nor on the recall whether the relevant item is on the first or on the last place. Nevertheless, many applications, like most search engines or recommenders, generate ranked results and a user usually expects the first items to be the most relevant ones. As a result, metrics that take the ranking of relevant items into account are needed. One way to indicate the performance over ranked results are precision - recall curves, where the precision and recall values are plotted with different result sizes.

Another possible metric for ranked retrieval is the *Average Precision* (equation 5.5), which averages the precision values for each retrieved relevant item. For example, if we

retrieve 5 items with 2 relevant ones, one on the second position, the other one on the fifth position, the *AveP* is $\frac{1}{2} \cdot (\frac{1}{2} + \frac{1}{5}) = \frac{7}{20}$. In case a relevant item isn't retrieved at all, its precision is counted as 0.

$$\textit{Average Precision AveP} = \frac{1}{|R|} \sum_{i=1}^{|R|} P(i) \tag{5.5}$$

$|R|$ Number of relevant items

As evaluations are usually performed over multiple samples, this *AveP* is again averaged for the *Mean Average Precision* (MAE), which is shown in equation 5.6. The measure represents the area under the precision-recall curve and is commonly used for comparing the performance of information retrieval systems.

$$\textit{Mean Average Precision MAP} = \frac{1}{|S|} \sum_{s=1}^{|S|} AveP(s) \tag{5.6}$$

$|S|$ Number of samples (e.g. users)

The list of possible metrics could be continued with the *Precision at k*, the *R-Precision*, the *ROC curve* (receiver operating curve), the *NDCG* (Normalized discounted cumulative gain) and so on, but this chapter should rather give an overview of the most important ones, which can be used for comparing the performance of recommender systems.

In [5], the evaluation measures themselves were analyzed in experiments and it was stated that the Average Precision measure had a better stability than other IR measures, like the *Precision at k* and the *R-Precision*.

Furthermore, in [6], the stability of the mentioned measures with incomplete relevance judgments, meaning that not all of the possible documents have judgements for a certain information need (in our case the user), was investigated. For documents with missing judgements, it is simply assumed that they are non-relevant. This can especially be problematic in cases, where the relevance judgments are derived from the user's preferences, which often contain just a few items. The measures, like the AveP, highly depend on the amount of relevance judgments and monotonically decrease with smaller sets of relevant documents. In addition, no distinction between documents judged as non-relevant and those with missing information, is made. The *bpref* measure addresses this issue by considering the distinction. As a result of that, the experiments underlined a better stability with a decreasing amount of relevant documents. Of course, this measure only helps, if explicit judgments for non-relevant documents are available at all, which might not always be the case, as we see later on when evaluating the job recommender.

These metrics are mainly useful for comparing the recommender result with a set of known relevant items. In online evaluations, other metrics, like the clickthrough rate, the purchases, or more general the positive feedback on recommendations, are informative performance indicators. In [1], additionally some economic-oriented measures, like the ROI (Return of Investment) and the customer lifetime (LTV) were mentioned.

Before going on to possible evaluation scenarios for the job recommender, some theoretical foundations about statistical significance testing will be discussed.

5.1.2 Statistical Significance Testing

Statistical significance tests help to determine whether differences in measurements are real or just a result of random noise and chance. Such tests are necessary when judging that one recommender performs better than the other in a specific situation.

For conducting a significance tests, following parts are required [53]:

- A *test statistic* that is valid and powerful for the compared measure, like the t-test, Wilcoxon test or the sign test.
- A *null hypothesis* that will be accepted or rejected by the test. Normally the null hypothesis states that there is no difference between two systems. One-sided as well as two-sided formulations are possible.
- The *p-value* is the result of the test, which provides confidence for the null hypothesis. If the p-value is lower than the predefined significance level α (typically 0,05), the null hypothesis can be rejected and the differences between the two systems are statistically significant.

Several papers [11], [49], [53] discussed the use of significance testing in combination with the mentioned accuracy metrics, especially tests for the MAP were intensively evaluated. All those papers stated that for comparing two MAP values, the paired t-test was the most reliable one.

Still, it should be noted that for using the paired t-test, the prerequisite of a normal distribution of the pair differences should be approximately achieved [46].

Alternatively, the Wilcoxon test, which is also commonly used in the information retrieval domain, as well as the sign test do not expect a normal distribution. In evaluations by [11], it was reported that the Wilcoxon test had a greater power than the t-test, while the error rate was also higher. The sign test, on the other hand, couldn't convince in both aspects. Moreover, [53] rather warned using the Wilcoxon and sign test, as their ability for assessing significance was worse than others.

In case of cross validations, different tests have to be used, as the correlation between the repetitions violates the prerequisites of the mentioned tests. Dietterich [12] proposed a 5x2cv test statistics, but this test can only be applied for one data sample and as we typically have multiple samples for a recommender evaluations, this test statistic doesn't help here.

The sample size plays an important role for the evaluations. With a lower number of samples, even obvious differences might not be detected by statistical tests, whereas large sample sizes might lead to assumptions that very small and often meaningless differences are statistically significant [46].

In [5], the sample size for information retrieval experiments was examined. For the Average Precision measure, and for most others, 50 samples were stated to be enough for an acceptable error rate.

5.2 Scenarios

Based on the foundations discussed so far, four different scenarios for evaluating the job recommender will be presented. For each of them, a short description of the methodology, including the possible metrics, and the pro's and con's will be given.

5.2.1 Online Evaluation with A/B Testing

In an online evaluation, the job seekers are randomly split into two groups: one group receives recommendations from the original system, the other group from the new one, e.g. from the job recommender enhanced with implicit user feedback. The behavior of the users will then be observed and afterwards compared, e.g. on basis of the application rate for recommended jobs.

This way, the real impacts of a recommender can be well examined, but on the other hand, a live website using the job recommender is necessary. In our case, the new job recommender isn't deployed on the Absolventen.at website yet, as the current website still runs on an older Drupal version than the job recommender, which makes this evaluation scenario impossible for the moment.

Metrics: Positive feedback rate (clicks, applications, bookmarks, ...), User coverage

Advantages:

- Evaluates the real impacts of different recommendations on live websites, e.g. which recommendations the users read, bookmark etc.

- Many possible measures, including economic-oriented ones as well.
- All users on the website are included in the experiment.

Disadvantages:

- A live website with a user base is needed.
- Only two different recommenders can be tested at once.
- Difficult to obtain a control group with random recommendations, as this would risk a low user satisfaction.

5.2.2 Controlled User Studies

For controlled user studies, first some people who are willing to take part in the experiment are needed. Then these people are asked to perform certain tasks on a job board, like filling out a resume, searching for jobs etc. Afterwards they can be surveyed on whether the different recommendations fit their expectations, preferences or capabilities.

Metrics: Ratings from participants on recommendations

Advantages:

- Very precise experiments are possible.
- Participants can be surveyed: e.g. do the job recommendations fit their preferences, capabilities etc.

Disadvantages:

- Expensive and time-consuming.
- Doubtable, whether users in controlled experiments behave like on real websites. As a result of that, the monitored user behavior for the implicit feedback should be used with caution.

5.2.3 Offline Evaluation with Inferred Relevant Jobs

In this scenario, data from a live site is taken for the evaluation. This data will be used to compare the generated recommendations with the user's preferences from the past, the more items match, the better. As usually the recommender takes the preferences of the user as input, the user profile containing these preferences has to be split into a training and validation data set, as described in chapter 5.1.

In our case, we assume that jobs for which the user has applied or has added to the bookmarks, can be seen as highly relevant and due to this, we compare the recommendations with this data.

This way the evaluation can be repeated with multiple recommenders and it could also be used for fine-tuning the parameters, although this should not be done on the same data set for optimizing the results of one recommender. In addition, it has to be noted that only users with a fairly large profile size, as it will be divided into two parts, can be considered in the evaluation and it can be argued that this selection of samples is neither random nor really representative for the complete user base.

Metrics: Accuracy (MAP, Precision, Recall, ...), User coverage

Advantages:

- Evaluation with real data samples.
- No manual work is involved.
- Can be repeated with multiple recommender approaches.
- Cross-validations are possible.

Disadvantages:

- User profiles are split into a training and validation data set, although all data would be an important input for the recommender.
- Only active users with a reasonable user profile size can be included in the evaluation. This selection might not necessarily be representative for the complete user base.
- Recommendations can only be compared with the job seeker's preferences.
- Most jobs are assumed to be non-relevant, since no positive feedback for them exists.

5.2.4 Offline Evaluation with Predefined Relevant Jobs

One disadvantage of the previous scenario is that typically only a low amount of known preferred jobs per user exists. Due to this fact, most jobs without feedback are categorized as non-relevant, even though this might not necessarily be true. Furthermore, it can only be tested, if the recommendations fit the preferences, no validations on whether the job fits the user's capabilities are possible.

One way to overcome these issues can be a manual creation of the gold standard with a larger set of relevancy judgements. Additionally, these judgements do not necessarily have be binary (relevant, non-relevant), as they can be divided into a more fine-grained scale, as well as categorized into different groups, like one for matching the preferences and one for matching the capabilities.

Even if these advantages sound convincing, it should be regarded that creating such a gold standard involves much work and in many cases it can be difficult to make appropriate judgements for relevant jobs for a user.

Metrics: Accuracy (MAP, Precision, Recall, ...), User coverage

Advantages:

- A random and representative sample of users can be taken.
- Non-binary and multi-criteria relevance judgements are possible, e.g. highly relevant, relevant, less relevant, non-relevant, or fits preferences, fits capabilities etc.
- Can be repeated with multiple recommender approaches.
- Cross-validations are possible.

Disadvantages:

- Creating the gold standard requires much manual work.
- Difficult to judge whether a job fits to a certain user.
- All preferences of a user might match with the gold standard and as the job recommender, for example, doesn't suggest jobs the user already knows, the gold standard together with the user preferences would have to be split into a training and a validation data set as well.

All of the mentioned scenarios have their strong and weak sides. In addition to that, some can measure the achievement of slightly different goals than others, e.g. optimizing the recommendations to fit the user's preferences might be different than optimizing

it for the user's capabilities. In our situation multiple criteria for the goals of the job recommender can be identified. First of all, it is important that the recommender can address as many users as possible with personalized recommendations. Furthermore, maximizing the application rate and thus optimizing the recommender against the user's preferences is important for the job board and for the user satisfaction. Nevertheless, the capabilities of a user should be kept in mind.

For this thesis, where the effects of implicit feedback should be examined, all scenarios seem to be possible. As within the scope of this work only one evaluation scenario is feasible, it was decided to stick to scenario presented in chapter 5.2.3, as it offers some practical advantages in our situation. First, no live site that already uses the new job recommender is needed. Instead the data from a live site can be imported on a different testing environment running the new system. Second, it's not as time consuming as others and can be easily repeated with different recommender settings.

Nevertheless, for long-term evaluations, the other scenarios are interesting as well, and it would be possible to perform them in parallel, e.g. use an offline evaluation for fine-tuning some parameters and examine the overall impacts of two approaches in a big online evaluation.

5.3 Test Data

For comparing the recommendations with the job seeker's preferences, it was possible to make use of real data samples from Absolventen.at. When starting with this thesis, a simple version of the user actions logging system for Absolventen.at was one of the first parts we implemented. This was necessary in order to have some usable data for the evaluation at the end. This logging system has been deployed on the 15th of December 2010.

In the meantime, the architecture for the new job recommender was designed, followed by the implementation. As already mentioned, the new job recommender is based on a newer Drupal version and as a result of that, can't be installed on the current Absolventen.at website. However, for the evaluation, real data from Absolventen.at was imported, including the job seekers' resumes, the user actions and the jobs.

The import was performed with the data until the 29th of April 2011. Over almost four and a half months, around 22.036 user actions were recorded and 4.075 registered job seekers were active[1] on the website. Most of the recorded user actions were in form of 16.348 read times logs, followed by 4.124 searches, 965 bookmarked jobs, 321 unbookmarked jobs and 278 applications (see table 5.1).

[1] Includes users who have either created an account or logged into the system within this time.

Table 5.1: User actions

Action Type	Count	Percentage
Total	22.036	100%
Read Times	16.348	74,19%
Job Searches	4.124	18,71%
Bookmarked Jobs	965	4,38%
Unbookmarked Jobs	321	1,46%
Applications	278	1,26%

The import of jobs into the recommender test system turned out to be more complicated than expected. Even though it was originally planned to keep all published jobs on Absolventen.at within the evaluation timeframe, due to technical reasons, many of them have been removed from the database after they have been unpublished. As a result of that, several job ads had to be extracted and restored from backups in order to have enough data for the evaluation. Otherwise, too many user actions would have pointed to non-existing jobs, which would have dramatically reduced the user sample size.

With the existing and the restored ones, more than 5.000 jobs were imported. This only included the ones which were at least once read by any logged-in job seeker. On Absolventen.at, usually between 3.200 and 3.900 of jobs are available to the users. After a fixed time, or if the job went offline from the company's website, the job is unpublished and in certain cases deleted from the system.

To obtain realistic measures for the evaluation, the selection of jobs was reduced to a more reasonable size of 3.322 items. For the deletion of jobs, the oldest ones with neither bookmark nor application feedback were taken. In addition to that, it has to be noted that each job is published on one or more of the possible portals, including HTL-, HAK-, FH- and UNI-Absolventen.at. As job seekers only receive job recommendations belonging to their original portal, the typical size of possible jobs for recommendations is in average 1.641,25 items per portal.

For calculating the accuracy of recommendations, the job seeker's profile had to be split into two equal sized training and validation sets. It is important that both sets contain mutual exclusive job references, as otherwise the evaluation wouldn't work. For example, if we would take a bookmarked job as validation data, and would include the read time feedback for the same job in the recommender, the job suggestions would never match with the validation data, as the job recommender never returns items the user already knows.

The user profile size with highly relevant items, which are those from applications and bookmarks, plays an important role for the evaluation. The bigger the validation set,

the more we know about relevant jobs for a user, which will allow to make more stable statements about the performance of a recommender. For example, if we assume the extreme case, where the validation data consists only of one job, it can be difficult to find this job within the first recommendations. And even if, the comparison of recommenders with only one job as validation data might be heavily influenced by chance. Furthermore, the preferred jobs are also an important input for the recommender and by cutting the half of the preferences it is likely that the performance of the recommender with a hybrid user profile is reduced.

To have a reasonable sample as well as a user profile size, a minimum threshold of six highly relevant jobs out of applications or bookmarks was defined. The selection of this threshold is always a trade-off between the sample and profile size, as increasing the threshold will lead to more relevant jobs and more stable measurements, but on the other hand, reduces the amount of available samples, which is also very important for the stability of the evaluation. Additionally, for the evaluation, only users who have provided a resume were taken into account, as otherwise, if we would include user samples that only contain relevance feedback, the comparison wouldn't be fair. This way, we had 46 possible user profiles for the evaluation, which should be enough for examining the issues of this thesis.

Besides the amount of relevance feedback, which is available for a job seeker, the completion rate of the resume is also important for interpreting the results. As mentioned before, only those who have provided a resume, which contains some exploitable information, were taken into account. If the resume contains only little information, it is likely that the overall profile vector rapidly shifts towards the relevance feedback. In figure 5.2 an overview of the completion rate of the resumes of the test samples is given. A resume consists of many different parts, but five of these parts are exploitable for the recommender. This includes the education, languages skills, general skills, IT skills and the preferred region. As can be seen in the chart, the major fraction of the test sample, 32 users, filled out all relevant parts, 9 users provided 80%, 3 users 60% and each 1 with 40% and 20%. It could be argued that the user samples with relatively little information (less than 50%) should be dropped from the test sample, but as this only affects 4,3% of the overall user samples, no noticeable differences in the results are expected with this way or the other.

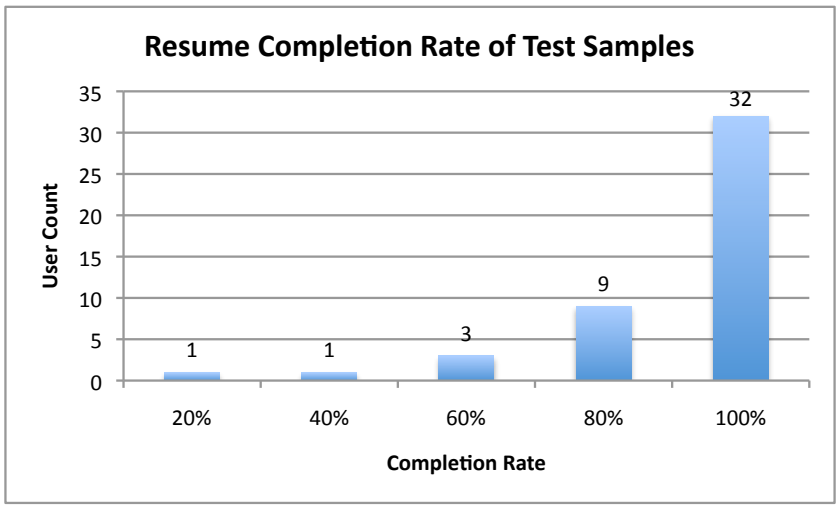


Figure 5.2: Resume completion rate of test sample

5.4 Results

In this chapter, the results from the evaluations will be presented. First, the user coverage will be discussed, followed by accuracy measures.

5.4.1 User Coverage

The user coverage measures the percentage of users, for whom personalized job recommendations can be generated. So far, the existing implementation has required the resume of the job seeker, which results in a low user coverage.

To see the impacts of the new approach in the means of the user coverage, four different user profile types were defined:

- *Hybrid* profiles are those that contain an exploitable resume as well as some relevance feedback.
- *Resume-only* profiles include job seekers who have provided a resume but no relevance feedback.
- *Relevance-feedback-only* (RF-only) profiles include the job seekers who have only provided relevance feedback, but no resume.
- *No information* includes users who haven't left any personal data.

The 4.075 active users were analyzed and grouped into one of the four categories. The detailed results can be seen in table 5.2, as well as in the chart in 5.3.

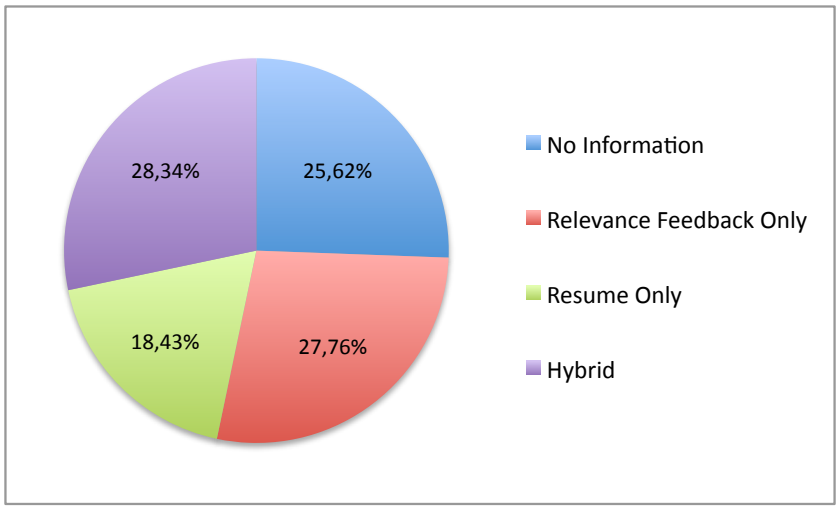


Figure 5.3: User groups

Table 5.2: User groups

Group	Users	Percentage
Total	4.075	100%
No information	1.044	25,62%
Relevance Feedback Only	1125	27,76%
Resume Only	751	18,43%
Hybrid	1155	28,34%

Around 25% of all registered users do not provide any exploitable information. Many of them have just created an account and probably never log in, which might also be a result of spam. 27% of the users provide relevance feedback by using the job board, but do not fill out the resume. On the opposite, fewer people, 18% of all registered users, provide a resume but no feedback, at least non when they are logged in. The biggest part, 28% of the users intensively use the website and provide both information. As can be seen, 56,19% (hybrid + RF-only) provide implicit feedback, meaning that 22.036 actions are spread over 2280 users, resulting in 9,66 actions per users who provide at least one action.

With these numbers it is easy to determine the user coverage. As already mentioned, the existing implementation requires a resume, whereas the new hybrid profile based approach can produce recommendations for users with either a resume, or relevance feedback or both. For the rest of the users with no profile data, no personalized recommendation can be calculated.

In table 5.3 the user coverage for the different recommenders is given. Additionally to a recommender based on a hybrid and the resume-only profile, a relevance-feedback-only profile is given for comparison.

Table 5.3: User coverage

Profile type	**User coverage**
Hybrid	74,53%
Resume Only	46,77%
Relevance Feedback Only	56,10%

With a system that supports hybrid profiles, a coverage of 74,5% can be reached (includes the user categories: hybrid, resume-only and relevance-feedback-only). Previously, for only 46,7% of the job seekers personalized recommendations could have been generated, even less than a solely relevance feedback based profile with 56,1% user coverage.

These numbers clearly indicate that the hybrid profile approach provides advantages for the user coverage. Whether it also results in a better accuracy, will be examined in the next chapter.

5.4.2 Accuracy

The main evaluation scenario for examining the accuracy has already been discussed in the chapter 5.2.3. According to that, the feedback from the applications and the bookmarks will be split into two equal sized folds for a training and validation data set. No cross validations will be performed, even though the evaluation component supports it. A 46-sized user sample is thought to be enough for examining the main questions and with cross validations, the additionally created comparisons strongly correlate with the original ones, which makes it more difficult to argue whether differences are significant or not.

For the evaluation, all parameters were set as initially described in chapter 3. No optimizations for these parameters with the test samples were performed. Still, this can be done after this evaluation in order to optimize the recommender for the usage on

Absolventen.at. Concerning the profile adaption, it is questionable whether this can be tested with this scenario in a meaningful way, as it is rather a feature for the daily use. Nevertheless, it was included in the evaluation and the forgetting factor is computed for the user's last feedback date. As a result of that, the feedback used as training data was selected to be the latest one, whereas the older one was usually used for the validation data.

For comparing different recommender approaches, the Mean Average Precision (MAP) is used as main metric. Additionally, for showing the performance over increasing result sizes, precision - recall curves will be used, but the MAP expresses the area under this curve anyway.

Three different approaches were tested in the main evaluation, including a hybrid user profile, a resume-only profile and a RF-only profile. The results in form of precision - recall curves are shown in figure 5.4. The plotted points start with a recommendation result size of 1, followed by 5, 10, 25, 50, 100, 250, 500 and 1.000. It should be noted that the seeker usually receives only the first 5 results, so most attention should be paid to the first few recommendations.

As can be seen from the curves, the hybrid profile and the RF-only profile approaches are almost identical, whereas the resume-only profile approach clearly has a lower overlap with the preferred jobs. On the first sight, it could be assumed that the hybrid and the RF-only profiles return nearly the same results, but the scores for the recommendations are usually different. Nevertheless, it seems that a profile containing at least three relevancy judgements out of applications and bookmarks almost ignores the values from the resume. Of course, it is possible to reduce the influence of the relevance feedback with other weights, but it should also be kept in mind that the evaluation was performed with fairly large relevance feedback profiles. For the average user with less relevance feedback, the results might be more balanced. On the other hand, just dropping the resume, which would simplify the implementation, isn't recommended either, as it would cost user coverage and it is thought to be still an important input for the overall recommendations.

The precision - recall curve for the resume-only profile is extremely low at the beginning and due to this fact, it might seem that this approach is inaccurate at all. However, it has to be pointed out that we compared the recommendation results with a fairly low number of preferred jobs, which led to extremely low precision values for all approaches. Correctly identifying 3 preferred jobs out of around 1.641 possible ones is not always easy. Furthermore, it is debatable how well the preferred jobs really fit the job seeker's capabilities, but this can't be answered by this evaluation scenario.

In table 5.4 the exact MAPs and the results of the significance tests in form of the p-values are given. The hybrid profile approach receives a MAP of 0,0575, which is slightly higher than 0,0570 for the RF-only profile and notable higher than 0,0278 for

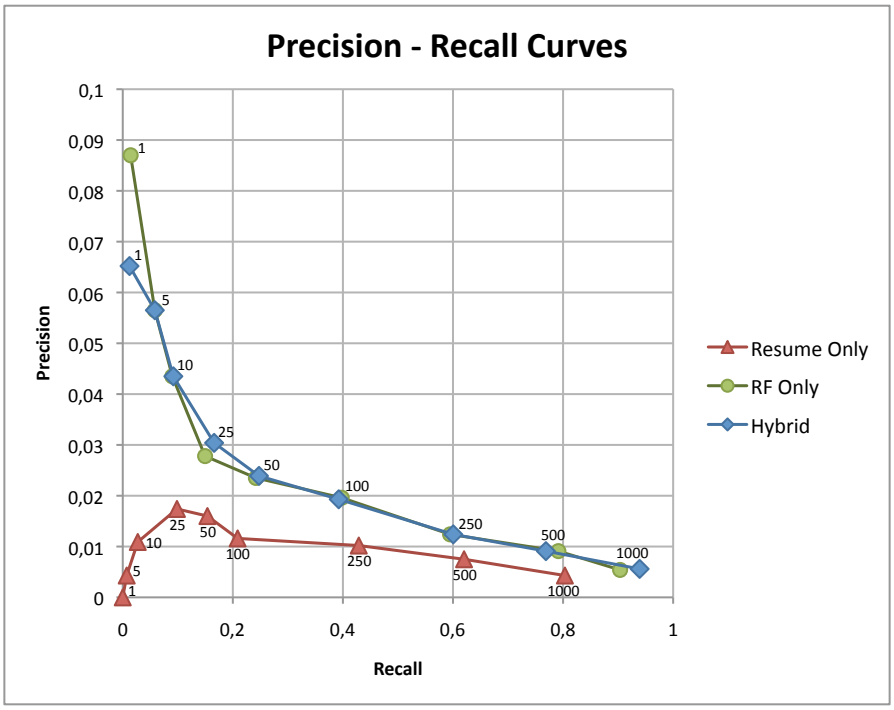


Figure 5.4: Precision - Recall curves

the resume-only profile.

To see if the differences are statistically significant, Wilcoxon tests with a two-sided null hypothesis, which states that both values are identically distributed, were performed. Previously, some criticism concerning the Wilcoxon test has been mentioned, although the opinions about that haven't been confirmed. Still the t-test, which should be used in favor, couldn't be applied here, as the pair differences of the Average Precisions weren't normally distributed at all (Shapiro test). Additionally, the results from the Wilcoxon test are clear and match the observations made on the precision - recall curves. The differences in the Average Precision between a hybrid and a resume-only profile are statistically significant with a p-value of 0,0044 (significance level $\alpha = 0,05$), which is almost the same as for the difference between a RF- and a resume-only profile with a p-value of 0,0059. On the other hand, for the comparison of a hybrid and RF-only profile, the null hypothesis can't be rejected.

During the conception phase, many other interesting questions have arisen, e.g. which read time threshold should be used, or how much evidence for interest the different

[2]p-value that both Average Precisions are equal (twosided Wilcoxon test)

Table 5.4: MAPs and statistical significance values

	MAP	equals to *Resume Only*[2]	equals to *RF Only*
Hybrid	0,0575	0,0044	0,7219
RF Only	0,0570	0,0059	-
Resume Only	0,0278	-	-

relevance feedback categories provide. In the next subchapters, parts of these questions will be examined with additional evaluations.

5.4.2.1 Relevance Feedback Comparison

For testing the influence and importance of separate feedback categories, the recommender was configured to include only one feedback category at once and to exclude the rest as well as the resume. Again, all other parameters were left the same as discussed in the conception chapter.

Four different categories were compared with each other, as well as to the RF-only profile approach, which combines all categories. It has to be noted that the read times feedback included all feedback below the threshold (click action) as neutral information, the same way as it was done for un-bookmarked jobs in the bookmark category.

The results are given in figure 5.5 and in the table 5.5. Interestingly, the bookmark category has with a MAP of 0,0534 almost the same accuracy as all relevance feedback together (0,0570), of course no difference in the means of statistical significance. Especially the precision - recall curve shows better values for result sizes greater than 25 (the same result sizes are used in all precision - recall charts). Only at the beginning the combined relevance feedback approach has a better hit ratio. Additionally, the read times performed quite well and do not show any statistically significant difference to the combined and bookmarks-only results. The application- and search-based profiles led to worse accuracy values, which are also significantly different to the others.

These results should be regarded together with the distribution of the relevance feedback (see table 5.1). It is not recommended to solely included the bookmarks in the recommender, as this would only take around 5% of the total user actions into consideration, which would drastically reduce the user coverage. This fact also helps to understand the rather low performance of an application-only profile, as not many application feedback actions were recorded at all. If we had the same amount of applications as bookmarks, we would expect a rather comparable result.

[3] p-value that both Average Precisions are equal (twosided Wilcoxon test)

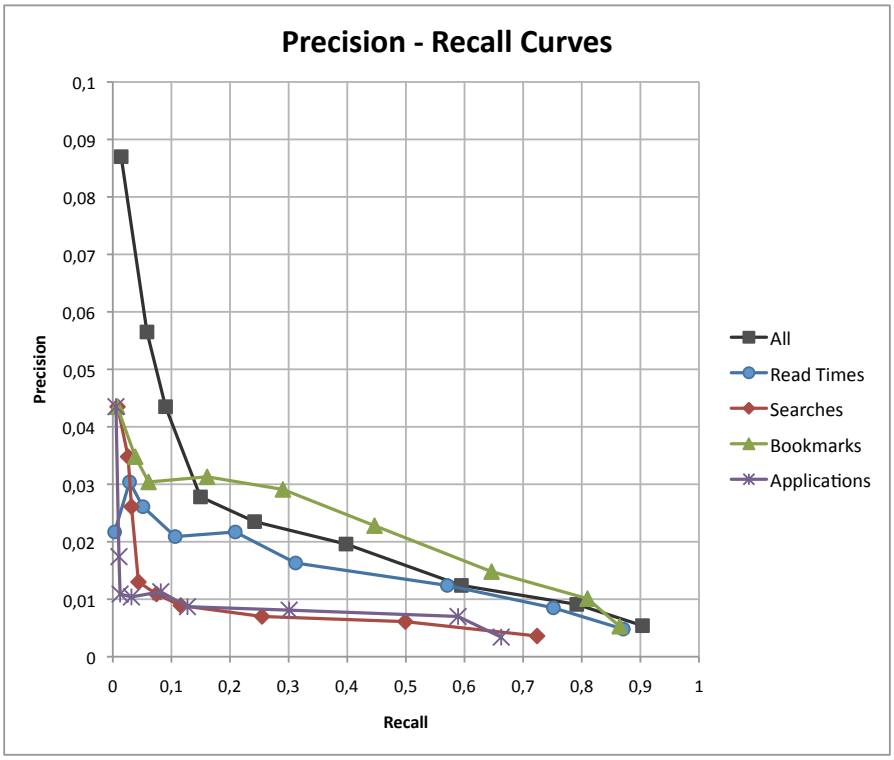


Figure 5.5: Precision - Recall curves for different relevance feedback categories

Table 5.5: MAPs and statistical significance values for relevance feedback

	MAP	= *Read Times*[3]	= *Searches*	= *Bookmarks*	= *App.*
All RF	0,0570	0,0843	0,0024	0,7867	0,0002
Read Times	0.0374	-	0,0417	0,1112	0,0069
Searches	0.0263	-	-	0,0013	0,9658
Bookmarks	0.0534	-	-	-	0,0000
Applications	0.0208	-	-	-	-

The user submitted searches, a feedback that represents the second biggest part with 18,7%, do not help to identify the relevant jobs as much as we expected. This also indicates that the information extracted from preferred jobs is better for constructing the user profiles. Later on, it will be shown that removing this feedback does not only improve the execution performance, but also slightly raises the MAP of a hybrid user profile for our test sample. A result that also shows that combinations of relevance feedback categories do not necessarily outperform profiles with less feedback.

Still, even though not all combinations are perfect and some might lead to an overfitting of user profiles, it is important for the overall recommender system that many different information sources can be captured and integrated, as this allows to address a wide range of job seekers.

5.4.2.2 Read Time Thresholds Comparison

In chapter 3.3.3, a system for read time thresholds has been proposed. It has been stated that low read times do not necessarily indicate relevancy. In addition, longer read times for relevant jobs, compared to the average, were observed. As a consequence, a threshold of 20 seconds was introduced and applied for the evaluations. All read times below this threshold are treated as neutral information, meaning that they are excluded from the recommendation result and no extracted concepts of such jobs are added to the user profile.

As defining the threshold is difficult and based on many assumptions, evaluations with different threshold are of interest. Thus the accuracy performances of the recommender containing only the read times feedback with different thresholds (0, 10, 20, 40, 60 and 80 seconds) was tested. Before discussing the results, which were a bit surprising for us, let's first see how the read times of the user from the test sample are distributed, which also helps to examine the evaluation results.

From the 46-sized test sample, 788 job reads with a median of 29 seconds were counted. This median is notable higher than the overall median with 18 seconds (see chapter 3.3.3). A histogram showing the distribution is given in figure 5.6. As can be seen, most recorded read times are between 0 and 10 seconds. Read times with more than 60 seconds are already fairly rare, but at the end a large amount of outliers with long read times exists. So setting the threshold to 20 seconds or a bit higher still seems to be a reasonable approach.

The evaluation results for all thresholds are shown in table 5.6. Because of very close and insignificant differences, no precision - recall curves are given. Surprisingly the evaluation results show no reaction upon different thresholds. Even a system with no threshold, which would be easier to implement, shows comparable performance.

Several interpretations for these results are possible. First, for assessing such a detail, the sample size might neither be large enough nor representative for the overall user base (see differences in the median for the read times). Second, as users already see the job title, the employer and the place of employment in the job overview, we assume that they mostly click on jobs with interesting information. If someone stops reading a job description, we don't know why. Maybe some parts fit, whereas a specific one doesn't, which would still allow to build quite accurate profiles over all read times. In addition

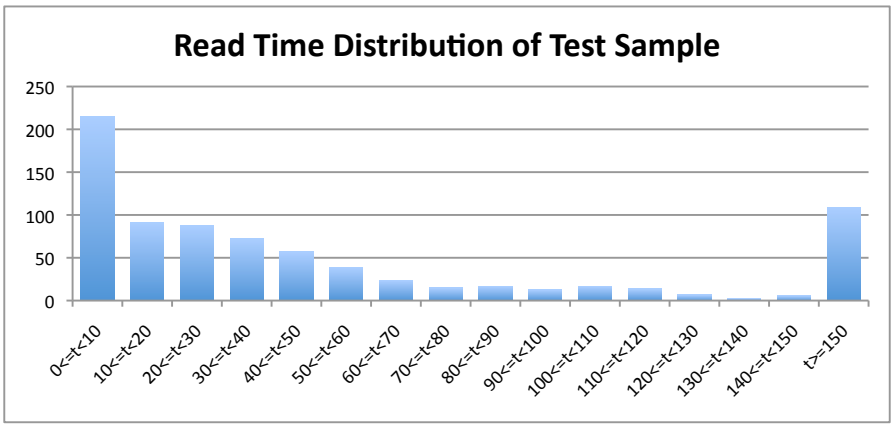


Figure 5.6: Read time distribution of the test sample

Table 5.6: MAPs for different read time thresholds *t* (in sec.)

	MAP
t = 0	0,0377
t = 10	0,0396
t = 20	0,0374
t = 40	0,0401
t = 60	0,0376
t = 80	0,0380

to that, as long as the amount of correct relevancy assumptions is large enough, the final user profile shifts into the right direction this way or the other.

In order to investigate this issue in more detail, long term evaluations would be necessary.

5.4.3 Performance

So far, only the user coverage and the accuracy have been examined. Still, for the overall evaluation, the execution time of the recommender is important, as noted in the requirements.

According to experiences made on Absolventen.at, a recommender based only on the resume works quite fast, but it is likely that with more information, which is included in the recommender, the execution performance will drop. Whether worse execution

times are still acceptable, is left to the website operator. In addition to that, Solr, which computes the recommendations, can be perfectly scaled and distributed over multiple server instances, although requiring such computational resources is rather unrealistic for job boards like Absolventen.at. Furthermore, scaling primarily helps to handle more requests at once, but not necessarily to decrease the execution time of a single request.

The recommender was tested on a development server, which was used in parallel for other applications as well. Therefore, it is expected that the performance is a bit better on a productive environment, where more resources are available. Determining and comparing the execution times of Solr is quite tricky, as it has several caching layers, like a document cache, a field value cache, a filter cache, a query cache and so on. Comparing the recommenders with differently filled caches wouldn't be fair at all. Due to this fact, three runs for each recommender with a result size of 5 items were conducted, one after Solr was started and the caches were empty, followed by two repetition, where the caches were filled one after the other. An additional repetition of the execution wouldn't have performed any better than the 3rd run.

For the daily use of the recommender, we expect that some caches, like the document cache, can be used. This is also confirmed by long-term performance analysis, but other caches, like the query cache, only help if the recommender is called more often for the same profile vector.

In chart 5.7 the results of the evaluation are given. Besides the three recommender approaches, like the hybrid profile, the RF-only profile and the resume-only profile, a forth one is included, a hybrid based profile without the search feedback. This search feedback is used to execute fulltext searches for the recommendations, a task that was identified as serious performance blocker.

The hybrid approach behaves almost the same as the RF-only approach, whereas the resume-only one is much faster (around 10 times for the first two repetitions). Consequently most computation time is required by handling the relevance feedback, but it should also be kept in mind that the used profile sizes are the largest ones the job board has available. On the last repetition, all times are very fast, ranging from 2 to 8 milliseconds.

As already mentioned, the search feedback requires much time and thus the hybrid approach without the searches is close to the resume-only approach and shows good performance. Interestingly, this combination without the searches has a MAP of 0,0626 that is higher than the one for the complete hybrid profile with a MAP of 0,0575 (no significant difference with a p-value of 0,1215).

To sum up, the execution performance should be acceptable in most cases. If not, it can be chosen between more computational power or removing resource-intensive feedback categories, like the searches.

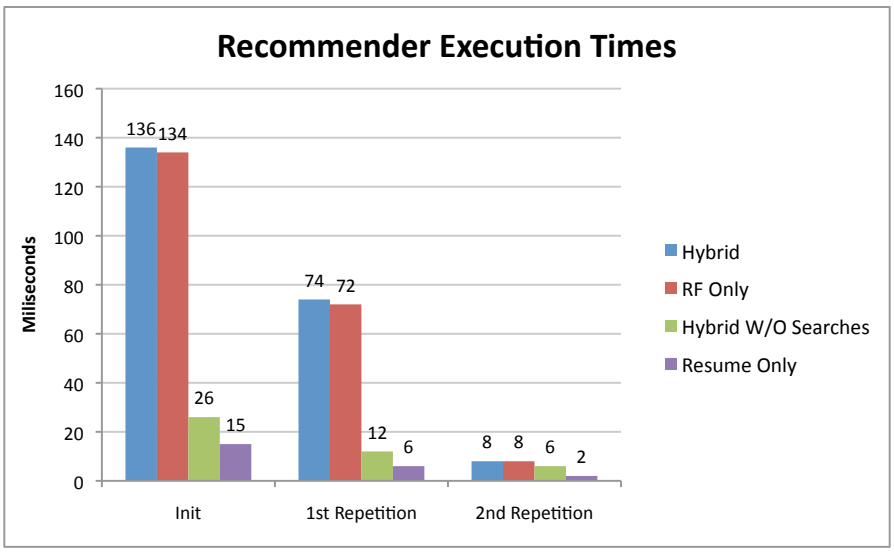


Figure 5.7: Execution times

CHAPTER 6

Conclusion and Outlook

6.1 Conclusion

The aim of this thesis was to examine the impacts of implicit relevance feedback for a job recommender system. In particular, methods for monitoring and integrating the feedback were of interest.

For this purpose an exhaustive literature research in the field of recommender systems was conducted, followed by an analysis of existing implementations, by the conception and implementation of the recommender system and finally by the evaluation.

This project mainly focused on the Austrian job board Absolventen.at, for which the recommender system was designed. Furthermore, this website also served as test environment in order to evaluate the impacts of the newly developed system. As a starting point, a recommender has already been available, which is based on comparing the resume with the available jobs. A major drawback of this approach is the requirement of a resume, which only the half of the registered job seekers provide. For the others, no personalized recommendations have been possible so far. This was the primary motivation for enhancing the recommender with preferences from the job seeker, which should not only help to reach a wider range of users, but also to increase the quality of the recommendations. As no explicit relevance feedback mechanisms for providing the preferences on Absolventen.at exist, implicit feedback is used as main information source. User actions, like applying for a job, indicate interest in this job and, therefore, can be used to infer user preferences. Besides applying for a job, user actions like reading a job description, bookmarking a job and searching were identified as possible feedback sources.

In order to make predictions for jobs to applicants, a content-based recommender was

implemented. For finding the best fitting job, a comparison of the user profile and the available jobs is performed, whereas both are represented as vectors and the similarity is calculated with the cosine in between. Previously, the user profile of an applicant solely consisted of the resume. During this project, it was extended with relevance feedback from the various user actions, leading to a hybrid user profile, as multiple different data sources are combined. To generate such a hybrid profile, it was decided to use the Rocchio's algorithm for relevance feedback and to adapt it to this use case. The resume is used as initial query vector, which is then extended by a linear combination of preferred jobs from the relevance feedback. In addition to that, each user action can have its own weights in order to control the influence on the final vector, as some actions, like writing an application, provide more evidence for interest than others, like just reading a job. Especially the read time feedback has to be treated in a special way, as just viewing a job for a few seconds doesn't necessarily express interest in this job. Hence, a read time threshold of 20 seconds, which defines the minimum time for indicating interest, was introduced.

Furthermore, it is likely that the preferences of a job seeker change as time passes. To take this into account, a profile adaption technique was implemented, which degrades the influence of older feedback and hence emphasizes the latest one. This was achieved by a gradual forgetting factor based on half life time spans.

For the implementation, a combination of the CMS Drupal and the search framework Apache Solr was used. Drupal is responsible for providing the job board functionality, including the monitoring of user actions and an administration interface for the recommender configurations, whereas Solr takes care of the actual recommender logic.

For evaluation, real user samples and jobs from Absolventen.at were taken. In order to see the impacts of implicit feedback, three different approaches, including a hybrid user profile, a resume-only profile and a relevance-feedback-only profile, were compared with the preferred jobs of applicants. The preferred jobs themselves are based on the applications and bookmarks, which are seen as reliable indicator for interest. As this information is also relevant for the recommender, it was split into two equal sized training and validation data sets. Therefore, only users with a resume and a large amount of preferred jobs could be used for the evaluation, leading to 46 test samples.

The performance of the approaches was assessed by two metrics: the user coverage, counting the portion of users for whom personalized recommendations can be generated, and the accuracy, which counts the overlap between the recommendations and the preferred jobs. Concerning the user coverage, a clear improvement can be seen, as it rises from 47% for a resume-only profile, to 56% for a relevance-feedback-only profile and to 75% for the hybrid profile.

The accuracy was analyzed with precision - recall curves and the Mean Average Precision (MAP). The Average Precisions are significantly better for the hybrid profile

compared to a resume-only profile, whereas the differences between the hybrid and relevance-feedback-only profile are not distinguishable, which also indicates that large profiles, which were used for this evaluation, already shift towards a relevance-feedback-only profile.

6.2 Outlook

Even though positive impacts of implicit feedback for a job recommender have been confirmed, many questions that go beyond this thesis remain unsolved and would need further examination.

As mentioned in this study, the recommender can be configured with a bunch of parameters, like field boosts, half life times, relevance feedback boosts and so on. Calibrating all these parameters was based on many assumptions, some of them have already been underlined or rejected by the evaluation. Nevertheless, further examinations would be necessary for a more detailed analysis of some settings. Of course, the current system already helps a lot, as the evaluation can be repeated with different settings, but a longer-term evaluation, which also examines the effects on a live website, would be interesting for this purpose.

Apart from that, recommender systems are still a very active research area. Many possible algorithms exist, which might improve the job recommender, and many new approaches will probably appear in future. Hence developing such a recommender system should be seen as ongoing project, as many improvements, either new algorithms, new configuration possibilities or an integration on other platforms, are still possible.

Bibliography

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.

[2] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40:66–72, March 1997.

[3] Christian Bizer, Ralf Heese, Malgorzata Mochol, Radoslaw Oldakowski, Robert Tolksdorf, and Rainer Eckstein. The impact of semantic web technologies on job recruitment processes. In *Wirtschaftsinformatik 2005*, pages 1367–1381. Physica-Verlag HD, 2005.

[4] Keith Bradley, Rachael Rafter, and Barry Smyth. Case-based user profiling for content personalisation. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, pages 62–72. Springer-Verlag, 2000.

[5] Chris Buckley and Ellen M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, pages 33–40, New York, NY, USA, 2000. ACM.

[6] Chris Buckley and Ellen M. Voorhees. Retrieval evaluation with incomplete information. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 25–32, New York, NY, USA, 2004. ACM.

[7] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Science*, 2000.

[8] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[9] CareerBuilder.com. The job recommendation engine increases job seeker application rates. http://www.careerbuilder.com/jobposter/enterprise/page.aspx?pagever=ENT_TechJRE . Last access: March 14th 2011.

[10] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces*, IUI '01, pages 33–40, New York, USA, 2001. ACM.

[11] Gordon V. Cormack and Thomas R. Lynam. Validity and power of t-test for comparing map and gmap. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 753–754, New York, NY, USA, 2007. ACM.

[12] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[13] Easyrec Team. Easyrec on twitter and upcoming features. http://sourceforge.net/apps/wordpress/easyrec/2010/10/05/easyrec-on-twitter-and-upcoming-features . Last access: May 15th 2011.

[14] Alexander Felfernig and A. Kiener. Knowledge-based interactive selling of financial services with fsadvisor. In *17th Innovative Applications of Artificial Intelligence Conference (IAAI05)*, pages 1475—-1482. AAAI Press, 2005.

[15] Susan Gauch, Mirco Speretta, Aravind Chandramouli, and Alessandro Micarelli. User profiles for personalized information access. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The adaptive web*, pages 54–89. Springer-Verlag, Berlin, Heidelberg, 2007.

[16] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992.

[17] Yael S. Hadass. The effect of internet recruiting on the matching of workers and employers, 2004.

[18] C. Hayes, P. Massa, P. Avesani, and P. Cunningham. An online evaluation framework for recommender systems. In *In Workshop on Personalization and Recommendation in E-Commerce (Malaga)*, 2002.

[19] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 230–237, New York, NY, USA, 1999. ACM.

[20] Robert Holte and John Yan. Inferring what a user is not interested in. In Gordon McCalla, editor, *Advances in Artifical Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, pages 159–171. Springer Berlin / Heidelberg, 1996.

[21] Diane Kelly and Nicholas J. Belkin. Reading time, scrolling and interaction: exploring implicit sources of user preferences for relevance feedback. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 408–409, New York, NY, USA, 2001. ACM.

[22] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37:18–28, September 2003.

[23] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40:77–87, March 1997.

[24] Joseph A. Konstan and John Rield. Research resources for recommender systems. In *CHI' 99 Workshop Interacting with Recommender Systems*, 1999.

[25] Danielle H. Lee and Peter Brusilovsky. Fighting information overflow with personalized comprehensive information access: A proactive job recommender. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems*, pages 21–, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Danielle H. Lee and Peter Brusilovsky. Reinforcing recommendation using implicit negative feedback. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH*, UMAP '09, pages 422–427, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] In Lee. An architecture for a next-generation holistic e-recruiting system. *Commun. ACM*, 50:81–85, July 2007.

[28] Fabiana Lorenzi and Francesco Ricci. Case-based recommender systems: A unifying view. In Bamshad Mobasher and Sarabjot Anand, editors, *Intelligent Techniques for Web Personalization*, volume 3169 of *Lecture Notes in Computer Science*, pages 89–113. Springer Berlin / Heidelberg, 2005.

[29] Jochen Malinowski, Tobias Keim, Oliver Wendt, and Tim Weitzel. Matching people and jobs: A bilateral recommendation approach. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 06*, page 137c, Washington, DC, USA, 2006. IEEE Computer Society.

[30] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[31] Stuart E. Middleton, David C. De Roure, and Nigel R. Shadbolt. Capturing knowledge of user preferences: ontologies in recommender systems. In *Proceedings of the 1st international conference on Knowledge capture*, K-CAP '01, pages 100–107, New York, NY, USA, 2001. ACM.

[32] Miquel Montaner, Beatriz López, and Josep Lluís De La Rosa. A taxonomy of recommender agents on the internet. *Artif. Intell. Rev.*, 19:285–330, June 2003.

[33] Masahiro Morita and Yoichi Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 272–281, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[34] Roger Munger. Technical communicators beware: the next generation of high-tech recruiting methods. In *IEEE Transactions on Professional Communication*, Vol. 45, pages 276–290, 2002.

[35] Quang Nhat Nguyen and Francesco Ricci. User preferences initialization and integration in critique-based mobile recommender systems. In *Proceedings of 5th International Workshop on Artificial Intelligence in Mobile Systems (AIMS'04)*, 2004.

[36] David M. Nichols. Implicit rating and filtering. In *In Proceedings of the Fifth DELOS Workshop on Filtering and Collaborative Filtering*, pages 31–36, 1998.

[37] D. Oard and J. Kim. Modeling information content using observable behavior, 2001.

[38] Sean Owen and Robin Anil. *Mahout in action (MEAP)*. Manning, 2010.

[39] Michael J. Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The adaptive web*, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.

[40] Rachael Rafter, Keith Bradley, and Barry Smyth. Automated collaborative filtering applications for online recruitment services. In *Proceedings of the International Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, pages 363–368. Springer-Verlag, 2000.

[41] Rachael Rafter and Barry Smyth. Passive profiling from server logs in an online recruitment environment, 2001.

[42] Maryam Ramezani, Lawrence Bergman, Rich Thompson, Robin Burke, and Bamshad Mobasher. Selecting and applying recommendation technology. In *Proceedings of International Workshop on Recommendation and Collaboration, in Conjunction with 2008 International ACM Conference on Intelligent User Interfaces (IUI 2008)*, 2008.

[43] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-Validation. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009.

[44] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979.

[45] J. Rocchio. *Relevance feedback in information retrieval*. Prentice-Hall, 1971.

[46] Lothar Sachs. *Angewandte Statistik*. Springer, November 2003.

[47] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval, 1987.

[48] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.

[49] Mark Sanderson and Justin Zobel. Information retrieval system evaluation: effort, sensitivity, and reliability. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 162–169, New York, NY, USA, 2005. ACM.

[50] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[51] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5:115–153, 2001.

[52] Young-Woo Seo and Byoung-Tak Zhang. Learning user's preferences by analyzing web-browsing behaviors. In *Proceedings of the fourth international conference on Autonomous agents*, AGENTS '00, pages 381–387, New York, NY, USA, 2000. ACM.

[53] Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, CIKM '07, pages 623–632, New York, NY, USA, 2007. ACM.

[54] Frank Curtis Stevens. *Knowledge-based assistance for accessing large, poorly structured information spaces*. PhD thesis, University of Colorado at Boulder, Boulder, CO, USA, 1993.

[55] Kazunari Sugiyama, Kenji Hatano, and Masatoshi Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th international conference on World Wide Web*, WWW '04, pages 675–684, New York, NY, USA, 2004. ACM.

[56] Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 449–456, New York, NY, USA, 2005. ACM.

[57] Tim Weitzel, Andreas Eckhardt, Alexander von Stetten, and Sven Laumer. Recruiting Trends 2011 (Management-Zusammenfassung), 2011.

[58] Yinghui Yang and Balaji Padmanabhan. Evaluation of Online Personalization Systems: A Survey of Evaluation Schemes and A Knowledge-Based Approach. *Journal of Electronic Commerce Research*, 6(2):112–122, 2005.

[59] Zhou Daniel. Recommender module performance enhancement & drupal for data-intensive computing. http://groups.drupal.org/node/145339 . Last access: May 15th 2011.