Iván Kubinyi

Designing an Automatic Recommendation System for a Web Based Social e-Learning Application

Helsinki Metropolia University of Applied Sciences

Bachelor

Media Engineering

Thesis

28.2.2013

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Iván Kubinyi Designing an Automatic Recommendation System for a Web Based Social e-Learning Application 47 pages 28 February 2013
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Application development
Instructor(s)	Antti Keurulainen, CEO of Bitville Oy Hannu Markkanen, Principal Lecturer

The purpose of this study was to create a prototype of an automatic recommendation system to demonstrate how recommendation systems work, what the possible pitfalls are and how they can be avoided. The results were evaluated from Bitville's web based social learning application's point of view.

First the concept of social e-learning is introduced. Then the technology and the taxonomy of the recommendation system are presented. Some of the mathematical tools that are the base of recommendation engines are introduced and it is shown how they can be used in applications.

The outcome of the study was a working prototype that was built on the same programming framework as Bitville's social e-learning application, thus it can be incorporated.

With the prototype it was proven that it is possible to build a recommendation engine that takes into account the ratings of the users for different content items and in this way bringing the social element into the recommendation process.

Keywords	automatic recommendation systems, collaborative filtering, singular value decomposition, e-learning

Contents

1 Introduction

The purpose of this thesis is to design an automatic recommendation system for a web based social e-learning application. In the first part of the thesis, a short overview about community based e-learning from the viewpoint of companies is given. A training consultancy company, Bitville's social e-learning application is introduced and the points where and how an automatic recommender system could help improving the application is revealed. Then the theory of computer generated recommendations is presented and a short comparison of the different techniques that are used on various commercial and non-commercial sites is made. Also, the theoretical background of recommendation systems and what are the main considerations are briefly introduced. Furthermore, a plan how the recommendation system could be improved, once the need arises, is proposed.

After the evaluation of the existing systems, the results are analysed from Bitville's web based social learning application point of view. A solution, based on the introduced methods that suits best the application's need at the current state of development is finally proposed.

2 Social e-learning applications

2.1 Community based learning within companies

In most of the companies, one of the big challenges is to train employees continuously, to make them more competent in their fields. Usually, a great deal of knowledge has accumulated within the company but in many cases there is no elaborated way to share that knowledge. There are rarely workshops, personal trainings or coaching, besides the higher management. Most of the employees get some induction for their work but as time passes trainings may get less frequent. [1,1]

Usually, there are experts within the organizations, employees who have worked for the company for many years and have improved their professional skills in their field. The first step for organizations is to get their experts share that knowledge, to make it available for all those who could benefit from it because the experts of the company are the best candidates to provide learning material for training.

It is very important to create time and possibility to share the knowledge, because many (and hopefully most) of the employees have a natural tendency to excel in their professional field and they are ready to learn. The learning sessions can be class room trainings, peer support, personal coaching, but the main aim is to convey the knowledge and raise the competence of the employees. There should be a way that experts can share their knowledge, either by creating documentation: wikis, PowerPoint presentations, screencasts or similar. In addition, everyone, within the company, who has some level of competence, should be given the chance to contribute. [1,1]

Some of the company experts might raise the following questions: Why would it be good for them? Why should they get involved, spend their time with creating learning material if they do not get anything in exchange? The big task here is to get the commitment of the authors and the experts of the company. They have to be kept reminded that their work in providing material for e-learning is valuable. However, personal commitment also raises the question of authorship.

Who should own the created material? Sometimes the authors do not wish to transfer their copyrights. The original author might feel that any alteration or reuse of his or her work is one way of stealing, or that work does not belong to him or her anymore. They

might not feel like sharing any learning material because of that. On the other hand, other content creators might even see strengths in the collaborative content creation. They might feel that more eyes see more and more viewpoints might help also to improve the quality of the study material. Some other employee might appreciate their own content and think it worth continuing their work, even if they leave the company. [1,2]

Once the e-learning content is in place, the next challenge is to activate the users, to get people to learn, to share ideas, and in the meanwhile raise their competence level. A good e-learning application attracts the users; it creates a need for knowledge and fulfils it. Besides supporting the creation and sharing of learning material, the application should also reward the work that users did in the system. When they finish some longer study material, the students should get some kind of reward that would provide positive feedback for them. Prizes could be given out not just for finishing some course, but also uploading content, commenting material, or in any other way being active in the system. Rewards could be as simple as diplomas or entering the hall of fame. If the rewarding system works well, it can help the students to refine their knowledge and indirectly raise the value of the company.

Besides rewarding the participation, another pulling factor is how easy and fun it is to use the system. People like to use an application that does not require training or reading of user manuals, in other words they do not have to learn how to interact with the system. Today's most popular applications such as YouTube, Facebook or Google mail do not need instruction manuals. However, this does not apply in many cases of e-learning applications. "One of the fundamental bottlenecks considering traditional eLearnings is the poor usability." [1,3] It might be due to the complexity of the material or the outdated pedagogical concept that many e-learning systems are neither intuitive nor easy to use. Therefore, those who want to be successful in the e-learning field have to tackle this issue.

Community based learning is not just about the content. Applications do not only provide the learning material but also raise interest and awareness in students. If the user of the system likes something, he or she has to be given the possibility to share it with others. If one found a video or an article particularly useful, there should be a chance to share that interest with the community. Users should be able to express their feelings while using the system, exchange opinions and discuss topics. This does not

only help to improve the feeling of community but could also enhance the learning process.

Once the system is in use and employees start to visit it and study courses regularly, as well as comment on items, create content and discuss with their peers, another possibility opens up. With a good e-learning application in place, it becomes easier to find expertise and find experts for certain fields. Users of the system can recognize new experts, draw attention to those who are experts, in other words employees who have a good deal of knowledge about some theme. The community can help to find these but there can be also automatic solutions that search and find users who seem to excel in their field. These users can be then drawn into the creation of new study material. At the end more expertise and more competence can lead to innovation.

Bitville's intention is to create an application that can provide all these features. It wants to create a tool which main aim is to raise the competence level of the employees and to be an effective platform for sharing and creating e-learning material.

2.2 Bitville's social e-learning application, Soclet

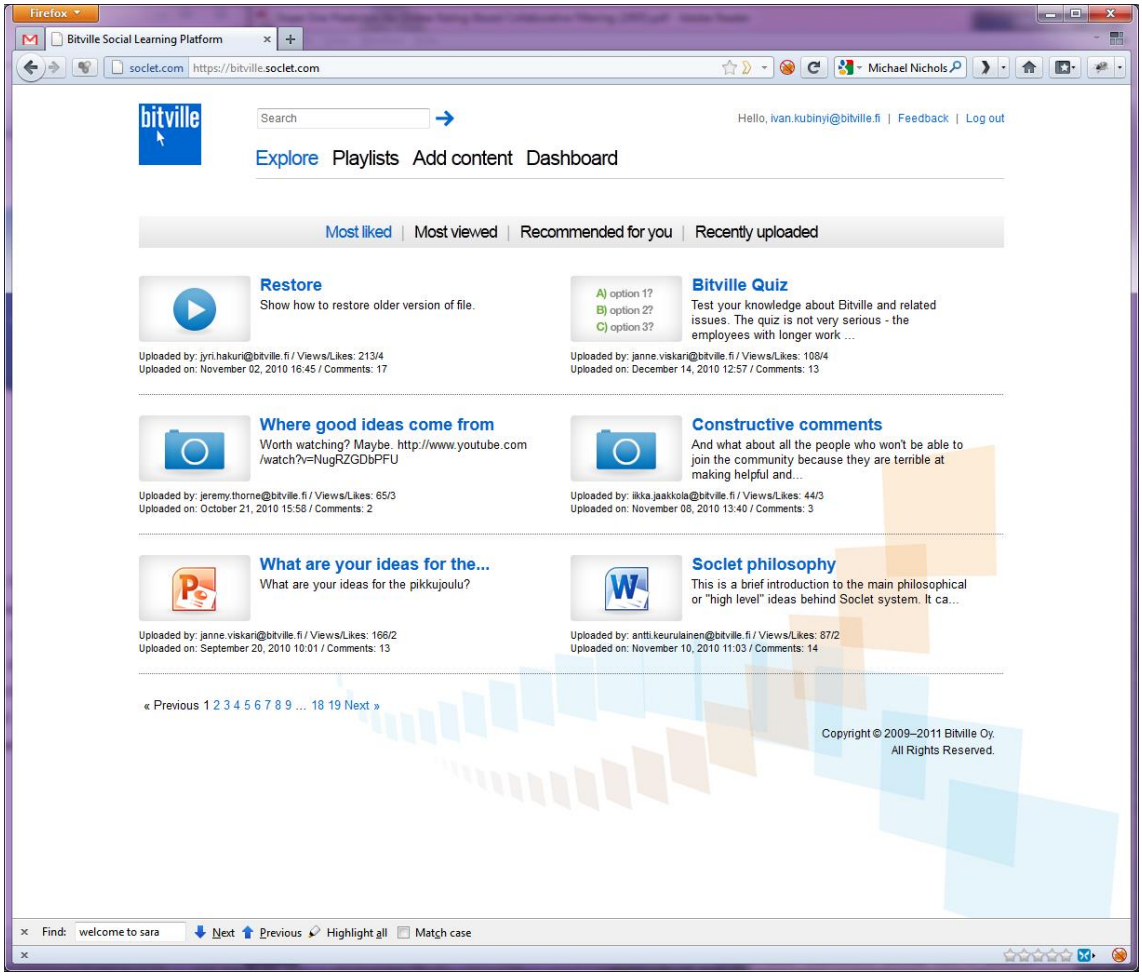
Soclet is community based e-learning application for companies. Its main aim is to create a virtual space where employees can study and raise their competence level on their field of work. The content can originate from the employees of the company. They can upload different types of content such as PowerPoint presentations, screencasts or text documents. It can be used also to share e-learning material from outside vendors for example short videos and Adobe Flash animations.

As a result the content can be watched online. There is an inbuilt player for Flash animations (SWF), Flash videos (FLV) and screencasts. This way users can leave comments for every item, initiate discussion, and exchange ideas. They can also give out "likes", thus raising the chance that other users would bump into the "liked" item.

The content elements can be organized into playlists, which can be created by anyone. They can be used as part of a course, preparation for some kind of exams, or simply to group items that can be connected into a logical chain. They have a description so the user can find what they want to see. A playlist can also be "liked", thus making it possible to reach a wider audience.

Playlists can be also remixed which means that if some other user wants to add or remove items from the playlist, they can do that.

Picture 1 illustrates the landing page that greets the user. It is the Explore page which shows the items that got the most “likes”. Under the other tabs, the user can see the most viewed items, items that are recommended for the user and the items that have been recently uploaded.

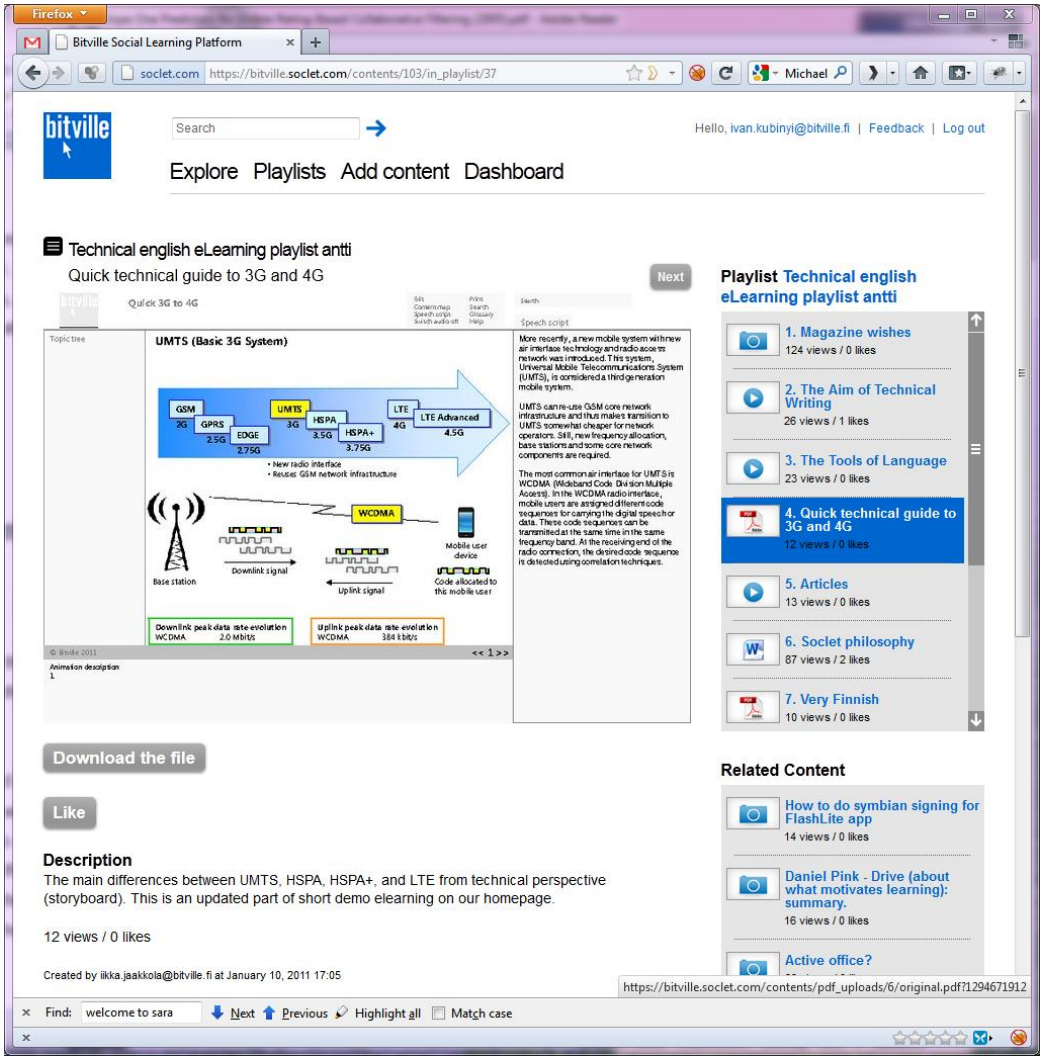


Picture 1: Landing page of Soclet

Picture 1 shows the first line of navigation: after the Explore, you can find the Playlist, Add content and Dashboard menu items. Under the Playlists menu you can browse or create new playlists and under the Add content menu item you can upload content. You have to select the type of the content, give it a title, write a short description and

provide a couple of keywords for it. Under the Dashboard menu, you can find your profile: what content you have added and what playlists you have created.

Picture 2 illustrates the content view. This is the main view of a content item. Naturally, the biggest part of the page is occupied by the actual content item. Under it there are two buttons: one is to download the content item and the other one is to “like”. Under them follows the description of the items and some statistics of the item: how many time it was viewed or liked. There is also a placeholder for comments.



Picture 2: The content page

Next to the main content, on the left side, you can see different lists in connection with the current item. If you are watching an item from a playlist, you can see the full playlist on the sidebar. Under that comes a list that is named Related Content. The relation is

based on the keywords that the content items have. The third list is called Related Playlists. The connection here is also based on the keywords that the playlist have.

The current of implementation of the “Recommended for you” feature is rather simple, it has room to improve the quality of the recommendations. A decision to design a recommendation engine which can produce more appropriate results and can be further developed was made. The aim is that the user would get immediately useful recommendations what to check out when he or she logs into the system. In the following chapters the theory behind the recommendation systems will be introduced.

3 Introduction to recommendation systems

"Show me your friends and I'll tell you who you are."

Ancient Greek proverb

Recommendations have been long with us. Humans make big part of their decisions based on others' opinions: what they think about certain things, if they think it is useful, or should they not bother. When we are young, we probably ask our parents' opinion whether we should go to a school camp or should we start to take music lessons. Later on we tend to ask our friends or colleagues if we should buy a certain T-shirt or phone, what they think about those products.

Concurrently with the blooming of the internet era, the number of the services and consumer products started to grow. At the same time they started to diverge and get more specialised. The wider the variety of the available products, the harder it is to decide what to choose. For example, if someone wants to buy a good book for reading during his or her holiday. One can naturally ask his or her friends or relatives to recommend something but that can take a long time. This has the advantage that the friends and the relatives know the person and probably have an idea about his or her taste, so they can probably give a good recommendation. Or one can just simply go to the local bookshop and ask the shopkeepers to recommend some good book. The shopkeeper likely does not know the person but using his intuition he can still give a rather good recommendation based on the mood, the way of talking, or even on the clothes of the customer. Still, in the local bookshop the variety might not be wide enough.

Commercial applications such as Amazon realized this fact and started to incorporate automatic recommender systems into their website. Usually, these systems are collecting huge amount of the data about the user: about their behaviour on the site, what the user has rated or bought, if the user has wrote any comment on a certain item, etc. Based on the collected data, the system can offer items that the user might like and hopefully buy.

The main challenge in giving recommendations is to give personalized ones. Personal recommendations can be based on many different factors. Commonly, they require some kind of initial user profile, so that the system could start to work reasonably well. It requires an initial profile of the user to be created containing the information about

the preferences of the user. This initial profile can be set up and then exploited in many ways. Table 1 demonstrates the categorization of the different recommendation methods.

This thesis focuses on the collaborative filtering and content based methods which were found to be the most useful ones for Soclet. Knowledge based method is also introduced shortly because it could be an interesting way of development for the future.

Table 1: Recommendation categories

Main category	**Sub category**
Collaborative filtering	Memory based (*user based technique*)
	Model based (*item based technique*)
Content based recommendation	Tag based
	Description based
Knowledge based recommendation	

Collaborative filtering is based on the explicit preference of the users. Usually the users express their likings on a scale: either some kind of numeric scale, or some binary ones (like, dislike). When the users gave out enough ratings, when his or her rating base is big enough, the recommender system can start to work. The recommender system can find close neighbours who have liked the same items approximately as much as the user in question. Then it can recommend items from the list of the close neighbour that the user has not yet seen.

One great advantage of this approach is that it is content-agnostic; the pure collaborative filtering recommendation system does not have to know anything about the content. Recommendations are solely based on filtering the ratings of the user base. It does not require that there would be any information uploaded to the system: what is the content about, what its genre is, who is the author or other features. This makes it less error prone and the data less noisy.

Two approaches can be identified in collaborative filtering: user based or item based. The user based tries to identify which users are more similar and recommend items from the users' list. Item based systems have the focus on the item. It tries to find content which has been similarly rated by other users. [2,13-21] Typically, the user

based approach is classified as memory based technique, while the item based approach as model based technique [2,26].

As the collaborative filtering recommendation is content-agnostic, it does not know anything about the content itself, and this can lead to surprising result. For example, the purely collaborative system might start to recommend items that the user feels have nothing in common because “with a pure collaborative filtering approach, a very intuitive way of selecting recommendable products based on their characteristics and the specific preferences of user is not possible” [2,51].

To overcome the obstacle, information retrieval methods were taken into use, to exploit the content and use the data for recommendation. *Content-based recommenders* analyse the textual content and the information is used to build up the initial profile of the user. Once the user indicates his or her preference regarding a few items, the recommender can start to work. In general, content based recommenders need much less user ratings then collaborative filtering. [2,77]

Analysing content is a rather resource intensive task but it can be done offline. Depending on the exact method, the analysing can happen in many ways. For example, the recommender filters through the content to find keywords, or to find the most used terms. When the user marks an item “liked”, the keywords of that item become a part of his or her profile. Then the recommender can recommend items that contain the same keywords.

The third main category is the *knowledge based recommendation* systems. These are usually rather sophisticated applications: their main task is to find items that the user might be interested from a vast number of possibilities. Knowledge based recommenders are based on interactivity: the user has to express his or her preferences and the recommender presents him or her with possibly interesting or useful objects. Knowledge based recommender systems are used when the intention is to find one particular item or product and the decision is affected by many factors.

One example of knowledge based recommender could be an application that recommends bicycles. The recommender has to offer bicycles that fulfil the needs of the user, match his or her preference and his or her body. The user’s requirements can be for example, that “the price of the bike should be not more than 500€”, or “the bike

should be suitable for downhill cycling”. The recommender system then translates the requirements of the user to product features. It offers than a selection of bicycles that match the requirements. If there are no suitable products then it tries to adjust the requirements so, that there would be some suitable items: like suggesting a higher price range.

This kind of system requires a lot of structured data. In case of the bicycles, the recommender system has to know the price, the size of the wheels, the type of the tyres, the colour of the bike and so on.

3.1 Collaborative filtering

Collaborative filtering uses the assumption that users who have similar tastes like similar items [3,135]. It also presumes that those, whose preference matched in the past, tend to be similar in the future [4,624].

> As online information continues to grow at an exponential rate our ability to access this information effectively does not, and users are often frustrated by how difficult it is to locate the right information quickly and easily. So-called personalization technology is a potential solution to this information overload problem: by automatically learning about the needs and preferences of users, personalized information access solutions have the potential to offer users a more proactive and intelligent form of information access that is sensitive to their long-term preferences and current needs. [5,3]

Giving good recommendation requires a good deal of knowledge about the users. If we still stick to the book buying example: the shopkeeper might remember to his or her customer, what he or she bought earlier. Maybe, the shopkeeper has even a full record of history what the customer has bought so far. Once the seller has to recommend something for that particular user, the seller could just look up the history of the customer and offer a new book based on the previous choices of the customer. Friends would probably use a different method, they would just recall their friend’s character and figure out what book he or she would probably like not knowing anything about his or her reading history.

Similar approaches could be identified in the collaborative recommendations methods. The memory based method can be compared to a very studious, accurate shopkeeper, who keeps the purchase history of his customers. Once a customer returns to his shop,

he uses his customer’s purchase history to recommend new book. He looks up in the list what the customer has bought so far and then compares it to the purchase history of other customers. If our shopkeeper wants to get even more info about his customers he might even ask them about how much they liked some books. Once he has enough information about his customers, he can set up a user-ratings matrix, and compare different customers based on their purchase history and on their ratings. So, when the next time the customer enters the shop, the shopkeeper could say: "Others who liked books that you had bought before, liked also this book, so probably you would like it too."

The friends' method is most similar to the model based method. One's friend usually has some picture in their mind, what he or she likes. Friends form their opinion on all sort of things, not just what he or she has read but also other information that are not closely related to buying books: friends might know his or her musical taste, if he or she likes to travel and they might even share some common memories. In a way the friend creates the “model” of the buyer and he or she tries to find items that would be in line with that model. In the following chapters, the memory based and model based approach of collaborative filtering will be discussed.

3.1.1 Memory based

The main idea of the collaborative filtering is to track the users’ opinions, preference and based on collected data, predict how the customer would feel about an unknown item. [2,13] Depending on the actual solutions, the stored data can be only ratings, or other things like, number of views, comments. The rating, in this context, is just a simple number that incorporates various human attitudes: it shows how much someone likes something, what is his or her feelings towards the rated item, many things that we consider while deciding what rate to give to an item. For the sake of clarity and simplicity, a pure user and item ratings matrix will be used in the examples.

The matrix is a 2 dimensional matrix, where columns represent the items with ratings and the rows represent the users. When the user gives a rating for an item, it appears in the user’s row, in the column of item as demonstrated in Table 2. The question marks denote the unknown ratings. Usually, when an application with a collaborative recommender is set up, there are just very few user ratings available. The user, item-

rating matrix is empty or at least really sparse. This makes the prediction of ratings for the unknown items very unreliable. It is hard to find similar users (users with similar ratings) or similar items (items with similar ratings) that are significantly similar. This problem is called the “cold start” problem. See chapter 3.3.

Table 2: Users and their ratings matrix

Users \ Items	**Episode 1**	**Episode 2**	**Episode 3**	**Episode 4**	**Episode 5**
John	5	?	4	2	3
Eric	?	1	3	2	4
Mary	4	?	1	5	?
Paul	5	4	4	5	3
Peter	4	5	3	?	5

In its simplest form, the user-ratings matrix serves as a base for the calculations. There are various mathematical tools to be used to predict ratings for a user who has not yet rated that particular item.

Why is it important to predict the user’s ratings for an item? Knowing the ratings of the user, we can build up a preference list, what the user would presumably like or dislike. When we have an idea about items that the user has not rated yet, we can create a list of recommendations with items that the user might like, based on the prediction of the ratings.

To predict ratings mathematical algorithms have been developed. Historically, the first recommenders were memory based. The algorithms processed the whole user-ratings matrix and based on the results gave predictions to ratings for items that user had not rated yet.

Because the data has to be available in the memory, the memory based methods are rather resource intensive and they scale rather badly. As the user base grows, so does the user-ratings matrix, in direct correlation. Bigger matrices require more memory, thus the computation time grows. In commercial applications, like Amazon, where recommendations have to be given in a fraction of a second, the strictly memory based approach is not feasible. Hundreds of thousand users with millions of ratings put an enormous strain on computing machinery.

However, these memory based algorithms are still in use: either as part of a more complex solution, or if the application does not have massive amount of users and items. Since the data structure is less complex, the memory based algorithms are less complicated, therefore, they are easier to maintain. Memory based algorithms can produce results that are on a par with other, more complicated algorithms but they require a lot of explicit or implicit ratings from the users and the required computational capacity grows with size of the users-items matrix. [3,136]

3.1.2 Model based

The scalability issues urged developers to find new ways to be able to recommend items. Simple memory based applications soon ran out of breath when they had to process couple millions of users and their ratings.

The solutions came from data mining and machine learning. Instead of using the calculation capacity to handle the whole user-ratings matrix, model based methods employs only some pre-processed data. Model based methods utilize the whole or part of the database to create a model that can be used to predict the rating of a user for an item. The initial data, a training set is used to create a model using data mining and machine learning methods. These identify the different users' preferences and other attributes that can be used to compare the users and their ratings to each other. [4,625]

Theoretically, the process of recommendations consists of three main steps:

1. Create an initial profile for the user: a training set of data is used to model the user and place him or her in an imaginary space, where his or her position can be calculated based on his or her preferences and that how it relates to the other users. If their preferences are similar, they are closer, if their preferences differ a lot, then they are further away from each other.
2. Select the nearest neighbours who are most similar to the user: when the user is placed in the virtual space, it can be defined which other users are the closest to him or her.
3. Recommend items based on the nearest neighbour: the recommendations can be given out using the items that the neighbours have already rated but the user has not. Naturally, the highest rated items should appear on the list. [6,243]

Creating a model and initializing the learning data usually takes a lot of time, so it has to be done offline. Calculations can be run outside the rush hours of the application or on an isolated server. After the profile has been created and the initial data has been processed, it can be handed over to the online application that can start to give out recommendations. This time, not the whole database is used to create a list of recommendations but only part of the available data, that are most relevant, is initialized by the intelligent algorithms. Therefore the processing time is decreased to an acceptable level.

Although, the model based approach requires more time investment in the beginning, it usually pays off in terms of scalability, and the speed of the recommender system. The models used are usually more complex, more elaborated, that makes the fine tuning possible, but at the same time, it is harder to maintain such application. The big advantage over the simple memory based method is that the model based method scales better; the required computational capacity does not grow linearly with the number of users and items. The computation intensive, complex models can be prepared offline, so then later the model can be used to give recommendation for the user at run time. [2,26]

3.2 Content based filtering

Another way to recommend items to users is based on content analysis. “Content-based recommendation systems analyse item descriptions to identify items that are of particular interest to the user.” [7,325] When the user sees or watches a certain item, the system can recommend other items based on the relative similarity to the actual content. The basis of the comparison is the actual item that the user sees.

Most of the items in Bitville’s application are linear active content: i.e. videos, presentations. These kinds of content cannot be easily analysed. Instead of analysing the actual physical data, some textual attributes needs to be attached to the item, and the recommendations can be made based on those.

3.2.1 Tag based

Tags are keywords that characterize the items. It gives some information to the user what the item is about, what is the main topic, what other topics occur in the item. Items can have multiple tags. They usually represent the content in a rather inordinate, non-hierarchical manner. Usually there are no restrictions what kind of tags can be used to describe the content: some may describe the content, some of them the genres, some of them the location (e.g. in case of a photo shot), etc.

Yet, well-chosen tags can describe the content and they can be used in the recommendation process. Tags can be used to build an item-tags matrix, where every tag represents a column, and every item represents a row. If the item has that tag, then it is marked in the appropriate cell with 1, otherwise the cell contains 0. With the help of the tags, vectors can be built for each item and the similarity can be counted between them using cosine similarity. The similarity value can be counted for every item pair and they can be saved to the database. When a new item is added to the system, the similarity would be counted against each of the other items and then the values are saved to the database.

Then the recommendation would work the following way. When the user sees a content item, the recommendation system would suggest items that are most similar to the current item. All is needed to query the database for rows where one of the items is the actual one and the similarity measurement is bigger than an arbitrary threshold.

3.2.2 Description based

A different approach is to use the description metadata that is available for the items. In many applications, items have a description. Usually, description is a shorter, coherent text that explains what the item is about, what is its content. Though, it is more useful for the users, analysing longer texts requires more complicated algorithms and more computational capacity.

Analysing coherent text is one of the big tasks of the data mining. The text is filtered through to find the terms that appear the most. The number of how many times a term can be found in a text is called term frequency (TF). Naturally, if the text is longer, then

the TF of the words will be bigger, which makes the comparing of different documents biased. The comparison should not depend on the lengths of the documents. To eliminate the effect, the number of other terms in the document is counted which is called inverse document frequency (IDF). The product of TF and IDF gives then the number that can be used as the base of the comparison in relation of a certain term. The TF-IDF products of the different terms are counted and these products become the items of a document vector. The documents vectors then can be used to compare different documents. [2,54-56]

Naturally, there are a lot of extra words that do not provide additional information for the comparison. These can be articles and different prepositions, like “a”, “the” or “to”. They are the so called stop words that can be removed from the documents during the analysis because they can be found in most of the texts in approximately the same prevalence. The analysis can be further refined if only the stem of the keywords are used. [2,56]

Advanced matrix factorization methods can be used then, not only for collaborative filtering but in the further analysis as well. The singular value decomposition (SVD) (see chapter 3.5.3) can help to find hidden, latent features items using only the ratings as input. For example, in case of movies, it can point out factors that can be easily associated with the genre of the movie, but there can be other factors that are not explicable [4,634-636].

The idea to find hidden factors has been successfully applied in the information retrieval domain. The main purpose of the information retrieval is to find relevant documents that the user is searching for. Documents can be analysed using TF-IDF and then depending what terms the user is searching for, the system can return those which have the biggest overlap in terms and in frequency. Unfortunately, in this way the similarity is based only on the exact word. The system cannot find something that is semantically similar, it cannot handle synonyms. If the user is using a search term like computer, the system returns only such documents that contain that exact word and it does not offer any that talks about laptops. SVD can help here, because it can identify the relationship between synonyms, so it can return also documents that contain only one of the terms. The technique is called *latent semantic indexing* or *analysis* (LSA).

The LSA filters through a document to find the most frequent words, phrases and it also analyse their relationships. It can find synonyms, homonyms. In a similar fashion, LSA can also be used to measure similarities among full texts, paragraphs or just sentences, words. The documents and the search queries are both encoded as term vectors. SVD is used to collapse the big matrix of the document vectors into smaller vectors that still preserve the main features of the original term vectors. Then the result document vectors are compared to the search query vector. [2,26-27]

Descriptions could be utilized to calculate the term vectors of the items. Then these term vectors can be used to check against the users' profile: if they liked a particular item earlier or not. Using LSA can also bring some novelty in the content based recommendation because it can bring up "latent" connection between items. [2,27] For example if the user is checking items that are about cars, it can also offer items that are about motorcycles or other motorized vehicles.

3.3 The cold start problem

Usually, recommender systems rely heavily on user provided data to provide recommendations. The data might come from surveys, asking the user about his or her preferences (explicit data) or by examining the behaviour of the user, what he or she is doing within the system (implicit data). Asking the user about his or her preferences might not be very successful because users, in general, do not like to fill in surveys or even being asked questions too much. Tracking the actions of the user usually provides more data but to give recommendation it requires that the user already have had some interaction with the system. This leads us to the "cold start" problem.

"The 'cold-start' problem describes situations in which a recommender is unable to make meaningful recommendations due to an initial lack of ratings." [8,311]. The main reasons of the cold start problem can be:

- *New item:* when new item is added to the system and it has no ratings yet.
- *New user:* users without ratings or any initial data causes that the system cannot give reliable prediction if the user would like a given item or not.
- *New community:* this is the most challenging one. In this case a whole new community is introduced to the system with new items and new users. [8,311-312]

When a new user is added to the system, very little, possibly nothing, is known about the user's preferences. On many commercial sites, therefore during the registration process the user is asked for demographic information. This can be then used to place the user into a demographic group that can be used to give out some initial recommendations. Another technique that sites may use is to ask the user to rate a set of predefined items, and in this way provide information about his or her preferences. A new user might be also asked about his or her taste: what kind of content he or she is interested in.

The cold start problem does not occur only when the user is new but also when a new item is added to the system. As none of the user has seen the new item, there are no "likes", comments and statistics available about it. The system cannot know how the users feel about the new content; it cannot define the characteristic of the content based on the opinions of the users.

3.3.1 Possible solutions

The cold start problem applies also to Bitville's application, for Soclet. Before the automatic recommendation system could start to work and propose valid recommendations, the possibilities have to be examined. To eliminate, or at least minimize the cold start problem, the following solutions could be considered:

1. Creating playlist beforehand for the new users. This can be a manually set up list that editors or the administrators of the system found useful. Still another way to provide the user with other, dynamically created lists, simple aggregates: e.g. most popular item, most watched etc.
2. If someone invites another person who has not yet registered in the system (future feature) then it can be presumed that the inviter's preference is somewhat similar to the one who he or she invited. Then the system can offer items that the inviter has rated the highest.

After the user has made the first interaction with the system, like searched for an item, or watched an item, the system can provide the user with similar items based on the similarity of the content, or some other features of the item. The important features can

be tags or the description, but the similarity filter also can take into account genres, author or other attributes.

As the user has more interaction with the system there could be a shift from content based methods to collaborative filtering. The system can collect more explicit and implicit ratings of the user, it can find more easily users who are similar to each other. Thus, the application could provide some novelty in the recommendations, instead of the apparent content similarity.

3.4 Hybrid method

Both the collaborative filtering and the content based filtering have their shortcomings. Collaborative filtering requires relatively big amount of input data from the users' side before it could provide sensible recommendation. The input data can be the given ratings to some of the items, the written comments to some items or simply the time spent on particular content page. Users have to interact with the system for a while before an appropriate profile can be set up for the user.

There are problems also with content based recommendations: the user has to express their preference at least for a couple of items that the system could build up an initial profile him or her. Before this profile is set up, the system cannot provide personalized recommendations. This is the so called *cold start* problem that has been described earlier (see chapter 3.3).

Content based filtering requires less initial data from the user-item relation but it needs a lot of metadata and/or content analysis. If the recommendations are based solely on the similarity of the items, it can lead to overspecialization [9,737] and the list of recommendations would lack novelty: only very similar items would appear on it. To provide more quality recommendations and to overcome the weaknesses of the different methods, the different techniques can be combined.

Combining the different methods is called hybridization. There are many ways to create a hybrid recommender system: for example a collaborative filtering recommender and content based recommender can be run within the same system, each of them produce their own lit of recommendation. The final recommendation list would then be created from the combination of the different recommenders. Another approach is to put the

different recommender engines in a chain, so that the output of one would be the input of the second, and so on and then the final recommendation list would be produced at the end of the chain.

Table 3: Hybridization designs

Main categories	Sub categories
Monolithic design	Feature combination
	Feature augmentation
Parallelized design	Mixed
	Weighted
	Switching
Pipelined	Cascade
	Meta-level

In the followings the three main hybridization designs (see Table 3) that can be identified in recent systems will be introduced. [2,124-142]

3.4.1 Monolithic design

Unlike in the following designs, where two or more recommender components are combined, in the monolithic design, there is one which incorporates multiple different techniques. Different methods and algorithms are used to process the data and give recommendation list. Usually, the data is pre-processed to prepare it for the recommendations component.

Figure 1 presents the schema of the monolithic design. The different algorithms and sources are combined to produce a final recommendation list. One way can be to combine the information that comes from the collaborative filtering with content features. For example, we know about our user what items he or she likes or dislikes. Then we can also check the feature set of the items: in case of films, if it were horror movies, comedies or art movies. Then the system concludes that if the user liked The Shining form Stanley Kubrick then he or she probably likes horror movies.

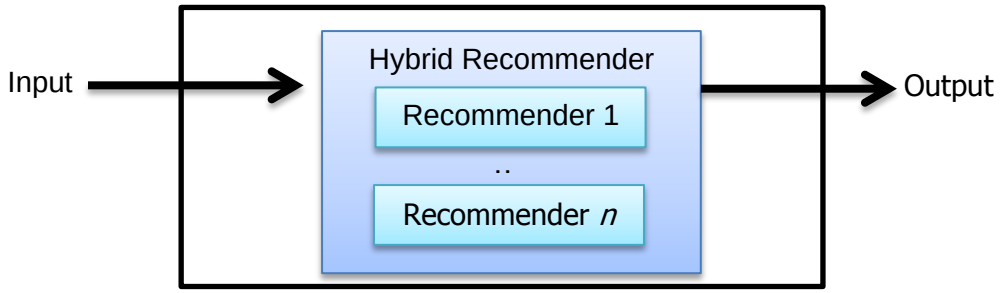


Figure 1: Monolithic design. Modified from Jannach, D., et al [2,128]

In this way, also those users who are regarded rather similar based on the collaborative filtering results can be differentiated. The similar users have many items in common that both of them like, but because of the content-agnostic nature of the collaborative filtering we cannot know what kind of items they liked. After further examination of the preferred content we might find out that actually some of the similar users like totally different genres, so it would be a mistake to recommend them items from their not so liked groups. This approach is called feature combination hybrids. [2,130-132]

In the book, Recommender systems: an introduction, a more sophisticated method is mentioned, which “does not simply combine and pre-process several types of input, but rather applies more complex transformation steps” [2,132]. It is called feature augmentation. Let us use an example to present the idea. The technique applies several different variables to provide more punctual prediction of rating for a certain item for User A. It uses two users (User 1 and User 2) to predict the rating for User A. The calculation is based on the Pearson correlation of the different users (shows how similar two users based on their ratings to User A), the number of mutually rated items (in relation with User A), the number of the ratings the users gave and the known ratings for the item from User 1 and User 2.

An initial predicted rating is counted for the User A using a content based method. Then this initial rating is used to predict a more probable rating for the user. The method takes into account that how many ratings the user has, how many items User A and the other user rated mutually, giving a higher weight for those who have more ratings and more common items. After applying the factors the result is a more accurate prediction for User A’s rating for the item.

3.4.2 Parallelized design

The parallelized design uses two or more recommender components concurrently. Each of them makes builds their own recommendation list, which is then modified in the hybridization step. The schema of the parallelized design can be seen in Figure 2.

One approach to parallelized design is called mixed hybridization. This is a rather straightforward method: the different recommender components all make their list of recommendations and then those lists are merged and presented to the user. For example in case of large web store there might be different recommenders for the different product domains: one for movies, one for books, one for sport equipment, etc. Each of these components can make their own recommendation and only the final list is presented to the user, where he or she can see movies, books and sport equipment. [2,134-135]

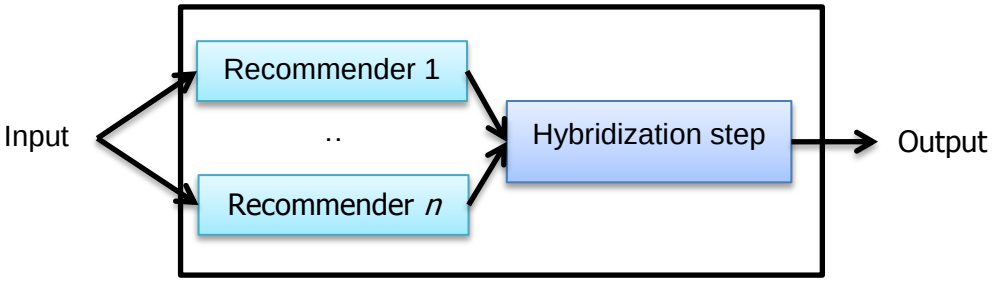


Figure 2: Parallelized design. Copied from Jannach, D., et al [2,129]

The second approach within the parallelized hybridization design is called weighted hybrids. In this case the each of the calculated ranking score of the different recommender components are taken into account with different weights. Typically, the sum of the weight across the components should be 1.

Let us assume that we have two recommendations, two ranking score from two recommender components for one single item: r_1=0,6 and r_2=0,8. The weight for recommenders are ω_1=0,3 and ω_2=0,7. Then we can calculate the final rating, like so: $r_1\ \omega_1 + r_2\ \omega_2 = 0{,}18 + 0{,}56 = 0{,}74$.

The weighting method can be used to adjust the accuracy of the recommender system, to give recommendation that the user feels more accurate. For example if we have a

system with a collaborative filter recommender component and with a content-based recommender, the weights can be adjusted so that the community data based rankings or the content similarity would be taken more into account.

The weights can be also adjusted dynamically based on the known ratings of the system. Some of the known rating values can be recalculated with the recommender system and check what is the difference between the real and the calculated values. Then the system can be adjusted so, that the predicted rating would be more close to the real value. In extreme situations, the weights can be 1. Then we speak about switching design. [2,134-137]

In switching design the results of the recommender components are taken into account based on the confidence of their results. For example, if the new user has just a few ratings available, the collaborative filtering will not give good results for recommendations. Thus, recommendation from the content-based recommender or some other knowledge based recommender can provide more accurate recommendations. [2,137-138]

3.4.3 Pipelined design

The third big family of the hybridization designs is the pipelined. Here the different recommender components are put after each other in a chain, so that the output of one of the recommender system is the input of the following one. The main difference among pipelined hybrids is the output of the recommender components: if they produce a recommendation list that the consecutive components refine, or the output is a pre-processed data model which then, in the final stage, becomes a list of recommended items. Figure 3 demonstrate the schema of the pipelined hybridization design.

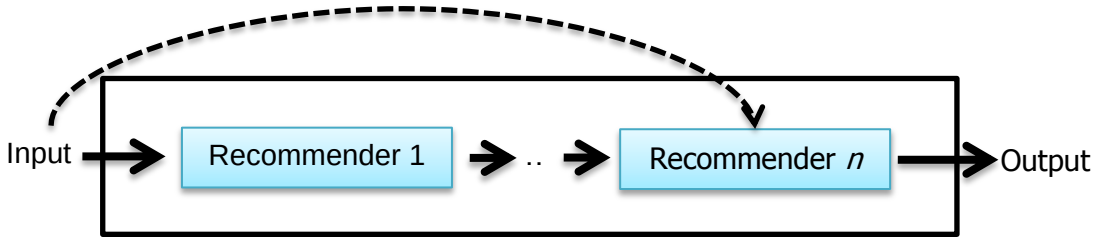


Figure 3: Pipelined design. Copied from Jannach, D., et al [2,129]

In cascade hybrids each consecutive recommender component refines the recommendations of the previous one. The first recommender component creates a list of recommended items which is then further refined by the following components. Each of the components, except the first one, works with a list of recommended items. It recalculates the ranking of the items according to its logic and takes away those that are found not to be valid any more, then it passes on the results to the next recommender component. Therefore, the list in the sequential processing can only get shorter. This might be undesirable in some systems, because many times the recommendation list is just not long enough. In this case, the applications usually use some fall-back technique to keep number of recommended items big enough: for example, switching back to some weighting method, when the number of the items does not reach a certain threshold.

When the output of the recommender component is a not a list of recommended items, but a data model which then utilized by a consecutive recommender, we talk about meta-level hybrids. Dietmar Jannach et al. describes an application which works on the online news domain. The content-based recommender component is used to create an initial user profile, which is then further processed with a collaborative filtering engine. Items are recommended not based on the user profile but from the preference list of those peers who are most similar to the user. [2,138-141]

3.5 The mathematics of automatic recommendations

This subsection discusses some of the most used mathematic techniques and logics. Some of the names cover just pure mathematical, statistical tools; while others describe a set of tools or methods.

These tools were chosen because all of them appear at certain stages of the recommendation process. Some of them are used mainly in collaborative filtering and some of them employed characteristically in content based filtering.

3.5.1 Cosine similarity

Cosine similarity is one of the most basic and most used mathematical tool in automatic recommendation system. It is used to give a numerical value what is the

similarity between two items. Usually, it is done so, that the items are represented by vectors in some virtual space, and their similarity is counted from the angle of the vectors. It gives a numeric representation that how much two vectors point to the same direction.

$$\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\| \, \|\boldsymbol{b}\| \, \cos\theta$$

The similarity ranges from -1 to 1, where 1 means the perfect similarity and 0 the least similarity and -1 that the vector the exact opposite.

$$similarity = \cos\theta = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \, \|\boldsymbol{b}\|}$$

It is used in many applications because its implementation is rather straightforward and the result is easily interpretable. In information retrieval the similarity cannot be -1 because every item of a vector is 0 or bigger.

3.5.2 Pearson correlation

Pearson correlation is mainly used in statistics to define how big the correlation between two or more variables. The values of the Pearson correlation is between -1, denotes strong negative correlation, and +1 that denotes strong positive correlation. If the correlation between any given two variables is positive, then we can argue that they are similar to each other. In its simplest form, it can be used in collaborative filtering.

To introduce the concept, I would refer back to the Table 2: Users and their ratings matrix. In this case the task of the recommender system is simply to find out how would Peter rate Episode 4, based on other users rating. To find this out the system should calculate which user's rating is the most similar to Peter's and presume that he would give the same rating. Based on the rating then the system can decide whether it should recommend that item.

Paul's ratings for the different episodes are in order: 5,4,4,5,3 and Peter's: 4,5,3,?,5. The equation to be used to count the similarity is the following:

$$sim(a,b) = \frac{\sum_{e \in E} (r_{a,e} - \bar{r}_a)(r_{b,e} - \bar{r}_b)}{\sqrt{\sum_{e \in E} (r_{a,e} - \bar{r}_a)^2} \sqrt{\sum_{e \in E} (r_{b,e} - \bar{r}_b)^2}}$$

Where “e” represents one of the episodes from all (“E”). $r_{a,e}$ means the rating of user “a” for the given episode, and $\bar{r}_a$ means the average rating of user “a”. To count the Pearson correlation, only those items are taken into accounts that have been rated by both users. Thus we get that, Paul’s average rating is $\bar{r}_a = \frac{5+4+4+3}{4} = 4$, and Peter’s is $\bar{r}_b = \frac{4+5+3+5}{4} = 4{,}25$. Thus we get:

$$\frac{(5 - \bar{r}_a)(4 - \bar{r}_b) + (4 - \bar{r}_a)(5 - \bar{r}_b) + \cdots (3 - \bar{r}_a)(5 - \bar{r}_b)}{\sqrt{(5 - \bar{r}_a)^2 + (4 - \bar{r}_a)^2 + \cdots} \sqrt{(4 - \bar{r}_b)^2 + (5 - \bar{r}_b)^2 + \cdots}} = (-0{,}45)$$

Thus, it can be said that the 2 user is not very similar to each other. After counting the similarity measure in relation with other users, we can have a list with users who are most similar to Peter and presume with a big probability that the Peter would give the same rating to an unrated item, which the similar user gave. [2,14-15]

3.5.3 Singular value decomposition

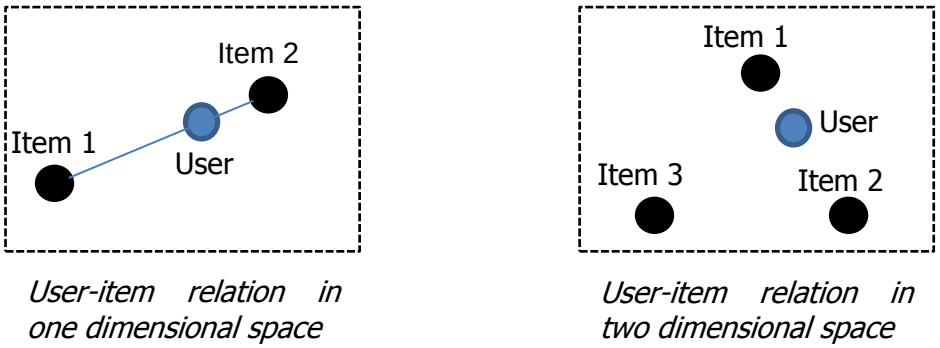
Singular value decomposition (SVD) is a matrix factorization method. It has not been developed primarily for recommendation engines but it can be used as an effective part of it. Its biggest advantage is that it can reduce the multidimensionality of a given dataset and helps to choose the most relevant values that can be used to predict the unknown ratings for users.

The singular value decomposition theorem states that “a given matrix M can be decomposed into a product of three matrices as follows, where U and V are called left and right singular vectors and the values of the diagonal of ∑ are called singular values.” (Golub and Kahan, 1965, ref. in [2,28])

$$\mathrm{M} = U \Sigma V^T$$

To show how SVD can be applied, I use again a user-rating matrix. When we have two items, we can present the preference of the user on a straight line: if the user prefers one of the items over the other, then we can show that by placing the point of the user

closer to the preferred item. With three items we have to use two dimensional space to show the same kind of preference. With four items we would need a three dimensional representation and so on. If there are a lot of items and a lot of users, the representation of the items and users might be really cumbersome.



User-item relation in one dimensional space

User-item relation in two dimensional space

Figure 4: User-item relation in one and two dimensional space

Figure 4 presents how the relationship can be presented in low dimensional spaces. The one dimensional representation shows that the user prefers Item 2 over Item 1. The distance from the items can be counted for example from the ratings that the user gave for the items. In the two dimensional space, the relationships are a bit more complex, but the results can be still easily presented. In this case, the user prefers Item 1 over Item 2 but dislikes Item 3.

Table 4: Users and ratings for SVD

Users\ Items	**Episode 1**	**Episode 2**	**Episode 3**	**Episode 4**
John	5	4	4	2
Eric	2	1	3	2
Mary	4	3	1	5
Paul	5	4	4	5

The aim of the decomposition is to break down the original matrix (see Table 4) into its component and find out which are the most relevant elements in ratings and user relation. SVD makes it possible to identify and separate the most relevant values. Values with what we can still approximate the original matrix within certain margins of error.

The outcome of the decomposition are three matrices that contains the most relevant elements in the beginning of the matrix (left side) and less relevant elements on the other end. Thus even if we start to reduce the number of columns in the resulted matrices, we can still get a good approximation of the original matrix. Therefore, if we take only the first two columns of the matrices, we can still preserve some of the characteristic of the original matrix, but the result can be presented on a two dimensional space. [2,28-29]

3.5.4 OpenSlopeOne

Daniel Lemire and Anna Maclachlan published a set of algorithms that they named Slope One. Many implementations of the algorithm can be found on the internet. This chapter presents the idea published in Lemiere's and Maclachlan's paper [10]. The basic idea, to give predictions for a user for a previously unrated item based on the already rated items and the ratings of other users.

Daniel Lemire and Sean McGarth have designed a solution based on PHP and MySQL as well. They claim that even though the algorithm is simple to implement, it works almost as well as other more elaborate algorithms. [11]

They had a set of aims they wanted to reach with their algorithms. It should be so maintainable and easy to implement that a common engineer would be able work with the code. The application should react to changes right away: when a user rates an item, its effect should be immediately visible in the relationship with the other items. Usually, the latter one is rather hard to achieve, especially if there is the third constraint namely that the system should provide recommendations in a reasonably short time. And finally, a good recommendations system should be able to give recommendations even to users with few rating, for example for new users.

The core of the idea is to store the average differences of ratings of users for different items, that the authors called "popularity differential". The basic idea is rather simple. Let us say, there are two users, John and Mary. John rated the movie, Matrix to 2,5 points and Existenz to 3,5. Mary rated the Matrix to 3 and we would like to know how she would rate Existent. Table 5 shows this:

Table 5: Rating matrix

	John	**Mary**
Matrix	2,5	3
Existenz	3,5	?

Let us presume that John's and Mary's tastes are rather similar and then we can give an estimation how Mary would rate Existenz. John's ratings difference is 3,5 – 2,5 = 1, so if Mary rather gave Matrix 3, we can predict that she would give Existenz 4. Of course, with this small amount of user and items, our prediction is rather unreliable but as the available data is growing, the difference in the predictions evens out. The different slope one algorithms take the average of the differentials.

Their proposals for different set of algorithms also reveal some major concerns users rate certain items. As their solution based solely on the ratings of the users, it is very important to get an idea how the users think, what are the basic patterns that they use in their ratings, etc.

In the most simple solution for prediction is the "per user average". The average ratings of the user are counted and we predict that the user would rate the unrated items with the average. A more sophisticated variation takes into account the deviation from the averages, thus giving an idea how much the different ratings depend on each other. In other words, how reliable the connection between the ratings of the items or the ratings that the user gave out.

The Slope One solutions "take into account both information from other users who rated the same item … and from the other items rated by the same user" [10,3]. The scheme uses ratings only from users who have common items with the user for whom the prediction is counted, and only those ratings of items that the actual user has rated themselves. Their "implementation of slope one doesn't depend on how the user rated individual items, but only on the user's average rating and crucially on which items the user has rated" [10].

A further developed solution takes into account the size of these arrays. The scheme gives more weight to the difference among arrays if the arrays are bigger. It represents practically that if there are users who have very similar taste, meaning that their ratings

for different items are very similar, then their arrays of ratings will weigh more in the calculation. From the weighing factor, the scheme is called *weighted Slope One*.

The authors also came up with a third solution where they take into account how a user generally rates items. Defining the characteristic of the person who gives ratings can affect the accuracy of the prediction. They counted that in their training sets “more than 70% of all ratings … are above the middle of the scale” [10,4]. The bi-polar scheme takes into account also how users generally rates, to define what a certain ratings means in the case of a user. For example, if a really optimistic person gives out only 10s and 9s (on a 10 points scale) for items that he or she really loves, while 7 and 6 already shows that he or she dislikes the item, because almost never give lower ratings than those. While our pessimistic user easily gives out 3 and 4 items that he or she dislikes and never wants to give more than 6 or 7 to something that is close to his or her heart. So, the average of the ratings is counted: if something falls below the average, then it can be considered as disliked. Thus the attitude of the users also can be taken into account when predicting a rating for a previously unrated item. They claim that a possible improvement could be “splitting ratings into dislike and like subsets can be an effective technique for improving accuracy” [10,5].

3.6 Amazon

Amazon is one of the best known and most used e-commerce site that uses automatic recommendation system to offer its users items that they might be interested in. The main purpose naturally is to increase the sales, to provide some novelty in the variety of the products and also to push items that might be new or not very known yet [12,77].

At such large companies, like Amazon, the biggest challenge is to provide real-time recommendations on huge datasets (possibly tens of millions of users with millions of different items) [12,79]. New customers usually have very little data that can be used for recommendations: purchased products or rated items, while regular customers probably have much more. Because of the relative sparseness of the input data, every interaction has to be tracked because it might provide valuable information concerning the preferences of the user. Amazon refers their recommendation system as *item-to-item collaborative filtering*.

Amazon's recommendation system consists of both offline and online elements. To provide good recommendations the system computes the most resource intensive similar items table offline. The algorithm browses through all items in the product catalogue and check if a given item, A, was bought by a customer, C. Then it checks if customer C bought also item B, and if so, records it. Thus, it can be counted that how many persons who bought item A bought also item B, and based on those numbers, the similarity between items can be counted. Cosine vector similarity is used to measure the similarity, where the items are represented by vectors and the dimensions of the vectors corresponds to the number of the customers who bought the items. [12,78-79]

The items are grouped into clusters based on their similarity. The online component of the recommendation system looks up that into which cluster the purchased, or the currently checked product belongs to and gives recommendations from those clusters.

The engineers of Amazon claim that their solution scales much better than other existing algorithms because it creates the resource intensive items similarity table offline. The online component, which looks up for similar items based on the customers previous purchases and ratings, scales independently from the similarity table. The speed of the online component depends only on the number of the items the customer purchased or rated. The number of the items and ratings do affect the speed of the application but they scale well because of the clustering. The new items are connected to clusters, thus they can be also recommended. Using item-to-item collaborative filtering, the application can give personalized recommendations in under a second. [12,79]

3.7 Apache Mahout

There are not so many commercial recommender applications available that could be easily tailored to an already existing application. Automatic recommenders require a lot of data that are gathered during the usage of the application or the user has to explicitly express his or her preferences. The recommender algorithms also have to "know" the content, have to have loads of information about it. If the infrastructure is not standardized, ready-made solutions usually does not suit the applications. There are commercial solutions available that are able to give automatic recommendations but in

most of the cases they only work with web stores, where the items have prices, purchase histories and the sole purpose is to sell more.

As the tailoring of an already existing automatic recommender is a cumbersome task, usually it is developed together with the rest of the application. There are open source tools that are available for the developers who can then create their own recommender engines using those. In this subsection one of the biggest open source software library built for data mining and machine learning will be introduced.

The biggest and probably the most developed from the open source recommender systems is Apache Mahout. It is developed by the Apache Software Foundation that develops the most successful web server of the internet.

The Mahout project is a rather new initiative. They are not providing recommendation systems as such, instead a set of tools that can be used to build one. Implementations of the most known and researched algorithms are included in the project. One can download the framework and then the tools can be used to build a full blown recommendation system. The implementations include:

- Collaborative Filtering
- User and Item based recommenders
- Singular value decomposition

One can build really sophisticated recommender system using these tools, however the actual logic to be designed still stays the responsibilities of the system architect. They have to decide what kind of information they want to use to provide recommendations: recommendations based on users provided ratings, on their page views, etc.

Apache Mahout is built on the Apache Hadoop software library that “is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model” [13]. The project includes also three subprojects, as they are listed on hadoop.apache.org [13]:

- Hadoop Common: The common utilities that support the other Hadoop subprojects.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.

- Hadoop MapReduce: A software framework for distributed processing of large data sets on compute clusters.

The Hadoop, thus also the Mahout projects are based on the Java programming languages. It can be run on Windows or on Linux operating system, but on production it has been tested only on Linux [14].

4 Implementation of a prototype

4.1 Why an automatic recommender is needed?

In the case of Bitville's social e-learning application the automatic recommender would be used to enhance the learning process, to provide the learners with similar content that they deal with and to aid them to find the right content for themselves. At the same time, it could also be useful for keeping up the interest of users by presenting ever new content.

In the current setup there is not much that can be known about the users of the application, so creating an initial profile might be cumbersome. Practically, new users only have to provide email addresses and passwords. It helps to engage them faster but provides very little data for creating an initial profile for the automatic recommender.

However, there is implicit data available once the user starts to use the application: we can track what items the user watched, how long he or she watched it. Users can express their preferences pressing the "Like" button when they see something that they liked. All this information could be exploited to enhance the quality of the recommendations. Even more precise information can be collected for example by analysing what the users commented on certain items.

As first step in proving the validity of the concept, I created a prototype that uses a simple memory based method using collaborative filtering. I chose this approach because it brings in the personal element in the recommendation, it utilizes the "likes" (or ratings in case of the prototype) feature of Soclet, still the complexity of the solution enables multiple direction of development. It is rather intuitive but at the same time it provides novel results as well.

The calculation is based on a linear algebra tool, singular value decomposition (see Chapter 3.5.3: Singular value decomposition). I chose this technique because the solution is quite intuitive, and the different steps of the calculations can be shown on graphs. Even if the original working method of the prototype would be deemed to be inefficient, the tools, the mathematics and the programming solutions can be used to redesign or further develop them into a more elaborated recommender engine.

4.2 Collaborative filtering with singular value decomposition

Usually collaborative filtering can be used effectively only when there are enough ratings in the system. Because in the Soclet application there was not enough data that could be used for the prototype, I used a publicly available dataset: the MovieLens dataset from the GroupLens Research group (http://www.grouplens.org/node/73). Its initial data is dense enough that the recommender could start to work and that presenting graphs would show the results.

I used the MovieLens 100k dataset that consists of 100 000 ratings on 1700 movies from 1000 users. The data was collected in 7 months in 1998. Apart from the ratings he dataset contains some information about the movies: e.g. title, genres, date of release; and some information about the users.

For the prototype I used only part of the data to illustrate the approach how the recommendation engine could work: 21 movies and 20 users. The rating scale is a 5 level scale: from 1 to 5. In the prototype, I also used 0. This means that an item is not rated, probably has not been seen by the user, so it can be recommended for the user.

The prototype was built using Ruby on Rails which is an open source web framework. The framework applies the model, view and controller software architecture pattern. Typing one command into the console the framework is capable of creating the model, the controller for the model and the view for it. The framework is easily expandable by using plugins that are called “gems”. To have a unified look I installed the Bootstrap CSS using its *gem*.

To make all the matrix calculations, I used a precompiled Java executable that was created from a Java package called JAMA. It takes a matrix as an input and outputs 3 matrices, the singular value decomposition of the input matrix. The communication between the Ruby on Rails application and the Java executable happens with JSON. Figure 5 shows the high level structure of the application.

Two models were created in the application: the movie and the rating models. Only these were needed because the prototype recommends only for one single user and the other users are just represented by numbers in the user-item matrix. Because I used datasets that were text files, parsers had to be programmed to import the data

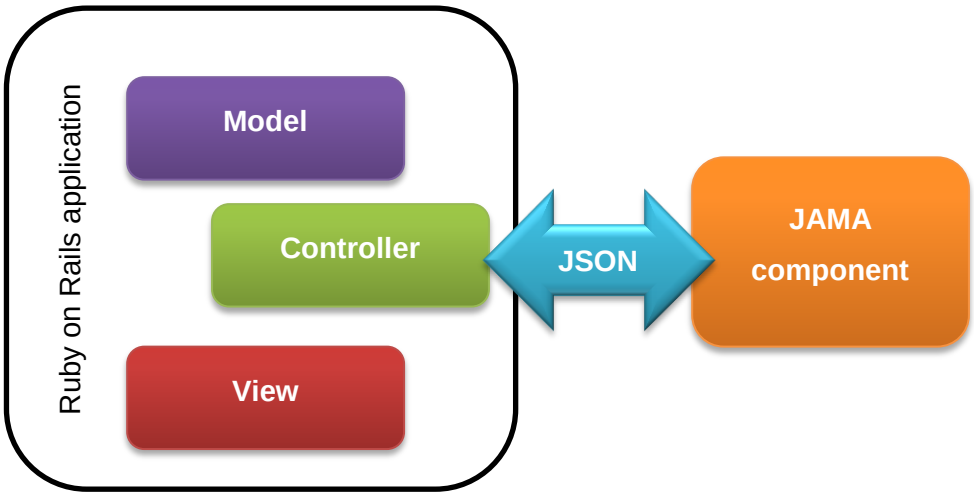


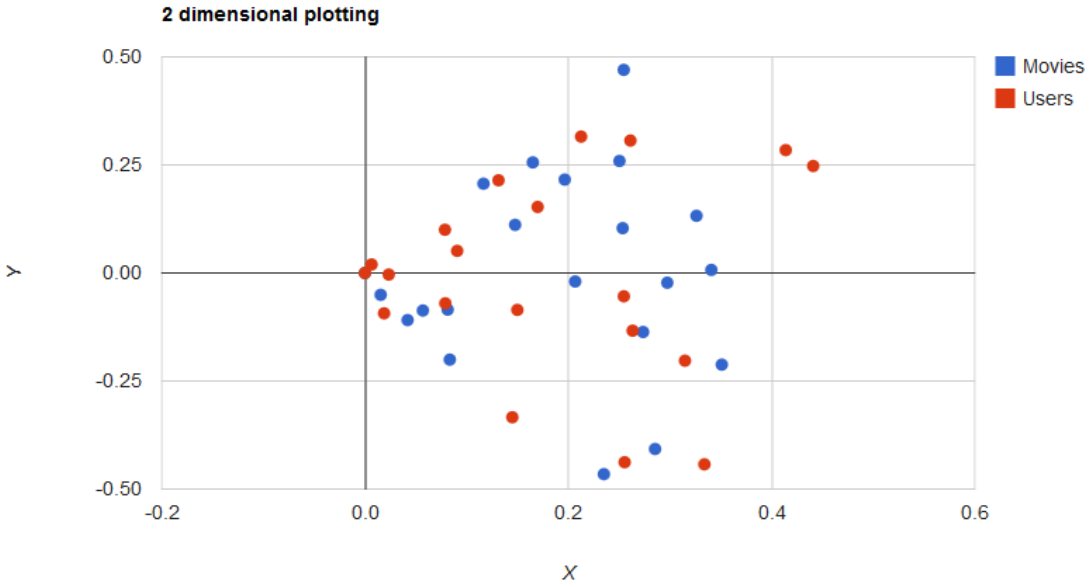
Figure 5: Structure of the application

into the database. The data in the dataset defined that what properties the models have: for example a movie has title, release date, genres and link to the IMDB record. Using the parsers I imported into the database the movies and their ratings.

In order to create recommendation I used Ilya Grigorik's idea [15]. Grigorik used singular value decomposition to place the users and the rated movies into a virtual space. In this space the similarity can be defined with simple geometric tools. By using singular value decomposition I can also demonstrate how the calculation is done because the results of the different steps can be plotted on graphs.

First a user-item matrix was created where columns represents the users and the rows the movies. The ratings are read out form the database and the values are put into the appropriate position. A mathematical technique, SVD (see Chapter 3.5.3) is applied to process the item-rating matrix. The precompiled Java executable calculates the singular value decomposition of the input matrix which results in three matrices U (21 x 21), $\sum$ (21 x 20) and V^t (20 x 20). The U and V^t matrices represents the movies and the users. In the prototype this calculation is made only once and the resulting matrices are saved on disk in JSON format to be used in the following steps. In other words this calculation is not made at runtime.

Picture 3 shows how the calculated matrices can be seen plotted on a graph. For plotting the results, I used the Google Charts API. Only the first two columns of the U and V^t matrices are used: the first column represents the x, the second columns the y



Picture 3: Plotting the results

coordinates. Thus, on a two-dimensional graph the relation among the movies and users can be shown.

The calculations after this point are made runtime. First the U, ∑ and V^t matrices are read in the memory from the JSON files that were earlier created. For the calculations, only the first two columns of the matrices are used, in order to use the same results that are visible on the graphs. I indicated this with the number 2 in the index of the matrices. Then, the actual ratings of the user are read out from the database. From the ratings a column vector (User) is created and then the following calculation is done:

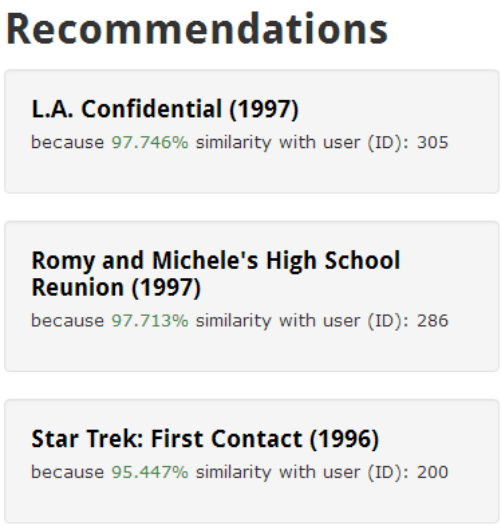
$$User_{2D} = User \times U_2 \times \sum_2^{-1}$$

$User_{2D}$ is a row vector with two elements, these are the coordinates of the user for whom the recommendation is calculated.

After the position of the user is calculated based on his or her ratings, the next step is to find the most similar neighbours of the user. There are many possible techniques to find those but I chose a rather simple one for presentational purpose: cosine similarity. The cosine similarity shows how much two vectors (representing the users) point to the same direction.

Once a list of the most similar users is created, the next step is to find what items to recommend. My algorithm browses through the similar users starting with the most similar one. It takes the items that are rated highest by the most similar user and then it iterates through his or her rated movies as long as it finds an item which has not been rated (presumably not seen) by the actual user and its rating is at least 3.

Once a movie is found or the conditions are not met, the algorithm jumps to the second most similar user and repeats the process again. The algorithm runs as long as it can find three recommendable items or runs out of recommendable items.



Picture 4: Recommendation list

In the final step (see Picture 4), the prototype presents the user with a list of recommended items based on the ratings of the user and of all the other users.

4.3 Outlook

In this chapter I lay out a plan how the automatic recommender system could be further developed using some of the solution that I incorporated in the prototype. The aim of a successful recommendation engine is to present the user with more personal and more relevant items which would enhance the user experience. The recommendation list should contain materials that are based more on their personal history of interest and at the same time it should suggest items that are in same way similar to the actually watched item. To achieve this, I propose the following steps:

1. Content based recommendations using tags
2. Collaborative filtering using rating, “likes”
3. Collaborative filtering using rating AND implicit measures (comments, clicks, spent time, etc.)
4. Content based recommendations using descriptions, LSA
5. Hybrid methods

Figure 6 shows the possible route of development. Each step raises the complexity of the system but produces more appropriate results. Step 1 is a really intuitive method to recommend items based on what the user is currently watching. It uses tags that are attached to the content items. This way of recommending item is already part of Soclet but it misses the personal element of the recommendation: it always recommends items that have similar tags.

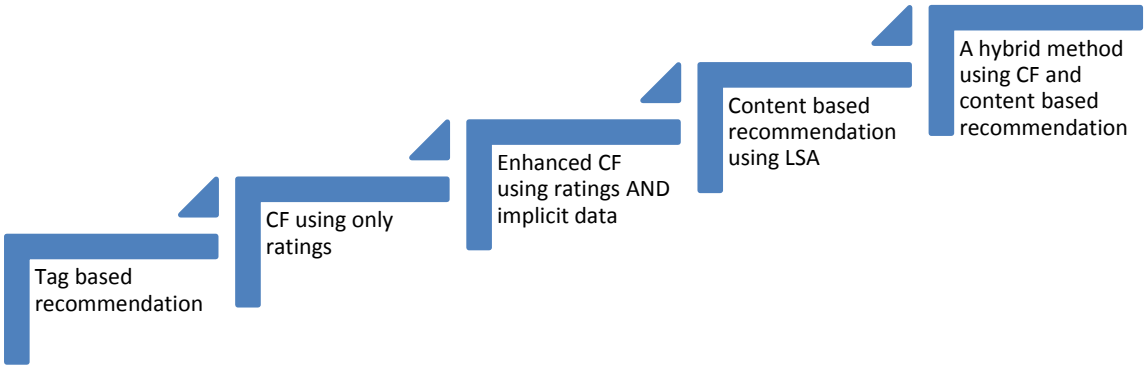


Figure 6: Steps of development

Step 2 exploits the personal element of the recommendation. It takes the ratings as the base of the calculations, profiles the users based on their ratings. It brings in the personal element but content-wise the recommendations might seem irrelevant. Step 3 enhances the process by using more implicit data and not just relying on explicit user ratings. This should enhance the recommendation process and makes the recommendation relying less on ratings.

The main idea is that instead of relying only on the explicit binary rating of the users (“likes” and “dislikes”, “thumb up” and “thumb down”) other implicit factors would be taken into account to calculate a “personal ranking”. For example: how many times the user has watched an item, how much time he or she spent on the page of an item, did he or she leave a comment. Figure 7 illustrates these factors. Each of these factors

Figure 7: Components of personal ranking

could get some arbitrary multiplier and then the number could be summed up. The sum would show that how much some content item is appreciated by a certain user. This way also those users would rate the content “unintentionally” who otherwise do not like to give out ratings.

Step 4 is where the content is analysed in order to build a more thorough user profile which would describe better the actual interest of the user. Ratings would be used also in this step but they would be used to find out what kind of items the user might like. For example, in case of a movie rental place the profile would contain information about the genres preferred by the user. This gives a better user model than that can be exploited for the better recommendations. The user exits in the system not just as an array of ratings but he or she would have a full profile: what items the person liked and why.

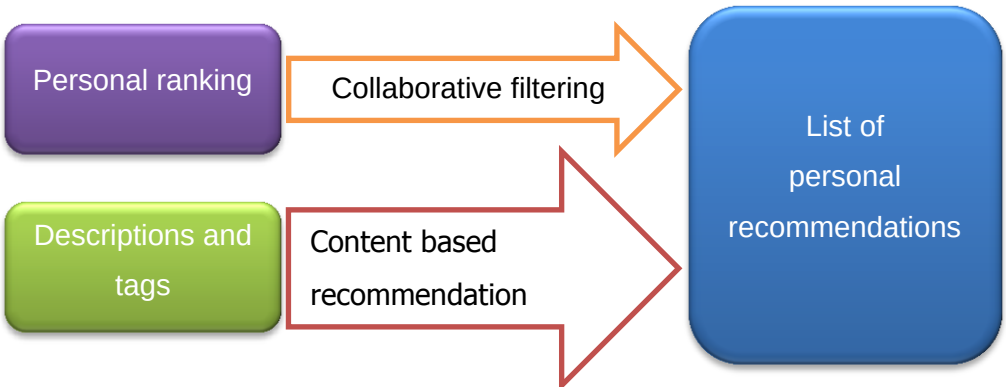


Figure 8: Hybrid design

Step 5 could be the final stage of the development. It would use collaborative filtering and content based recommendation in a hybrid setup (see Figure 8). When a new user arrives to the system all the recommendation would be based solemnly on tags. As the user would start to give out ratings and interact with the system the collaborative filtering and content based recommendation engine would start working. They would both produce their list of recommendations for the user then the algorithm would pick up only those recommendations which have higher confidence, which the user is expected to rate higher. When nothing can be produced for some reason the system would fall back to tag based recommendation. It should then tackle problems like the cold start problem, but after a while it could produce more personal and more relevant recommendation for the user.

5 Discussion

It is problematic to measure the accuracy and the relevancy of the recommendation given by an automatic recommender system. The efficiency of the algorithms can be measured with a test and a control dataset: the automatic recommender is run with the test dataset and the predicted ratings can be compared with the control dataset [2,166]. However, this is a topic that I am not able to cover in this thesis in more details.

I will focus more on the findings that I have learned during the creation of the prototype and the questions that arose.

- User-item matrix:
  There has to be enough ratings data in the system. Both the collaborative filtering and the content based filtering require that there would be sufficient amount of ratings. It is also important that the user-item matrix would not be very spare, so that there would not be users who have not rated any items and there would not be items that have not been rated.
- The rating system:
  When I started to work on the prototype I was thinking if the rating scale should be more detailed or is it enough to have likes. Shiva Rajaraman, the product manager of YouTube in a blog post was questioning the usefulness of 1-5 star ratings of YouTube videos. He questions if a simple thumb up or down would be enough to show if we like or dislike an item. [16] Also Lemire and Maclachlan argue in their paper [10,5] that dividing the ratings into “likes” and “dislikes” can improve the accuracy of the calculations. Currently in Soclet the user can only “like” an item. So all those items that a user did not like are in the same group with the unwatched items. In Soclet’s case and for successful recommendation engine it might prove useful that the user could show if she or he does not like an item.
- Finding the nearest neighbour:
  In the prototype I used cosine similarity to find users who are similar to the actual user in a virtual space. Cosine similarity is an easy and computationally rather inexpensive way to identify similar users but there are other ways to find the neighbours of a certain user.
- Algorithm how to build the recommendation list:
  I presented only one way how to create a list of recommended items once the

most similar users are known. With other algorithms better results could probably be produced.

With the creation of the prototype I wanted to show how an automatic recommendation system could be implemented as a part of the Soclet application. It requires some changes from the current state of the application and it also requires that the Soclet users would actively use the content: leave comments, spend time watching the different items and most of all, rate the items positively or negatively. By users expressing their preference the automatic recommender is capable of recommending relevant items that can improve the effectiveness of the learning process.

6 Conclusions

Building a prototype for a web based social e-learning application I examined the various phases of how a recommendation engine works and what are the possible pitfalls. At the same time I went through the theory and the taxonomy of different automatic recommender systems. Finally, I would like to sum up my findings and analyse what they mean in the case of Soclet.

The main goal was to enhance the learning process of the learners so that they should not have to search for the content which interests them but instead it would be offered. “The focus of digital personalization has shifted from what I am interested in now to what I might be interested in next.” [17,82] At the same time an automatic recommendation system helps to find the most popular content, helps to find prominent quality without external administering. It offers content to the actual user that is highly rated by other users and whose interests are similar to actual user’s.

A challenging task in case of Soclet is the versatility of the types of the content: they can be PowerPoint presentations, textual documents, videos or animations, but generally they provide little amount of metadata. There is a possibility in Soclet to attach description and tags to content items but it has to be done by administrators or the content creators.

The other important part is that the users as well should contribute to the operation of the automatic recommender: by rating the content items, expressing their like or dislike, leaving comments or just spending time on a page with content. If the users do not express their preferences towards the items, the automatic recommender cannot create a profile for the user and cannot recommend items reliably.

For a successful recommendation engine both the administrators, content creators and the user have to actively work with the contents. The former ones have to maintain and keep up to date the metadata of the content items, while the latter ones have to actively rate, comment and use the same items.

References

1. **Keurulainen, Antti.** Soclet - tool for corporate competence development. Helsinki : s.n., 2011.

2. **Jannach, Dietmar, et al., et al.** *Recommender systems: an introduction.* New York : Cambridge University Press, 2011. ISBN 978-0-521-49336-9.

3. **Berkovsky, Schlomo, Eytani, Yaniv and Manevitz, Larry.** Efficient Collaborative Filtering in Content-Addressable Spaces. [ed.] Matthew Y. Ma and Gulden Uchyigit. *Personalization Techniques and Recommender Systems.* River Edge : World Scientific, 2008, Vol. 70, 6, p. 135.

4. **Takács, Gábor, et al.** Scalable Collaborative Filtering Approaches for Large Recommender Systems. [ed.] Paolo Frasconi, et al. *Journal of Machine Learning Research.* March 2009, 10, p. 624.

5. **Smyth, Barry.** Personalization-Privacy Tradeoffs in Adaptive Information Access. [ed.] Matthew Y. Ma and Gulden Uchyigit. *Personalization Techniques and Recommender Systems.* River Edge : Scientific World, 2008, Vol. 70, 1, p. 3.

6. *Explaining collaborative filtering recommendations.* **Herlocker, Jonathan L., Konstan, Joseph A. and Riedl, John.** New York, NY, USA : Association for Computing Machinery, 2000. ACM conference on Computer supported cooperative work. ISBN:1-58113-222-0.

7. **Pazzani, Michael J. and Billsus, Daniel.** Content-Based Recommendation Systems. [ed.] P. Brusilovsky, A. Kobsa and W. Nejdl. *The Adaptive Web.* Berlin : Springer Berlin / Heidelberg, 2007, Vol. 4321, p. 325.

8. **Schafer, J. Ben, et al.** Collaborative Filtering Recommender Systems. [ed.] Peter Brusilovsky, Alfred Kobsa and Wolfgang Nejdl. *The Adaptive Web.* Berlin : Springer Berlin / Heidleberg, 2007, Vol. 4321, p. 311.

9. **Adomavicius, Gediminas and Tuzhilin, Alexander.** 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.* June 2005, Vol. 17, 6, p. 736.

10. *Slope one predictors for online rating-based collaborative filtering.* **Lemiere, Daniel and Maclachlan, Anna.** Newport Beach, California, USA : s.n., 2005. In SIAM Data Mining (SDM'05).

11. *Implementing a Rating-Based Item-to-Item Recommender System in PHP/SQL.* **Lemire, Daniel and McGrath, Sean.** D-01, January 2005.

12. **Linden, Greg, Smith, Brent and York, Jeremy.** Amazon.com recommendations. *IEEE Internet Computing.* January, February 2003.

13. **Apache Software Foundation.** Apache Mahaout: Scalable machine learning and data mining. [Online] Apache Software Foundation, 13 August 2011. [Cited: 2 November 2011.] http://hadoop.apache.org/#What+Is+Apache+Hadoop%3F.

14. —. Quick Start. *Hadoop core.* [Online] Apache Software Foundation, 19 February 2010. [Cited: 2 November 2011.] http://hadoop.apache.org/common/docs/r0.20.2/quickstart.html.

15. **Grigorik, Ilya.** Ilya Grigorik - igvita.com. [Online] 15 January 2007. [Cited: 11 April 2012.] http://www.igvita.com/2007/01/15/svd-recommendation-system-in-ruby/.

16. **Rajaraman, Shiva.** Five Stars Dominate Ratings. *The Official YouTube Blog.* [Online] 22 September 2009. [Cited: 8 August 2011.] http://youtube-global.blogspot.com/2009/09/five-stars-dominate-ratings.html.

17. **Schrage, Michael.** Recommendation Nation. *Technical Review.* 2008, May/June.