ISSN 2277-3061

An Extended Recommendation System using Data Mining Implemented for Smart Phones

Robert Győrödi, Cornelia Győrödi, Mihai Derșidan
Professor phd. eng, Corresponding author, Department of Computer Science and Information Technology, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania, 410087 Oradea, Universității str., 1
rgyorodi@uoradea.ro
Professor phd. eng, Department of Computer Science and Information Technology, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania, 410087 Oradea, Universității str., 1
cgyorodi@uoradea.ro
MSc student, Department of Computer Science and Information Technology, Faculty of Electrical Engineering and Information Technology, University of Oradea, Romania, 410087 Oradea, Universității str., 1
m_dersidan@yahoo.com

ABSTRACT

In this paper we introduce a recommendation system that attempts to solve some of the issues of classical recommendation systems: the need for huge amounts of data to be able to extract meaningful patterns, or for directly visible connections between the users. We propose a rating method that is based on a hierarchical tag system used to describe the subjects of recommendation, and two data mining algorithms that run on the data gathered through this rating system.

Indexing terms/Keywords

GPS, LBS, Android OS, data mining, recommendation system.

Academic Discipline And Sub-Disciplines

Computer Science, Data Mining, Mobile Technologies

Council for Innovative Research
Peer Review Research Publishing System
Journal: International Journal of Computers & Technology
Vol 11, No.3
editor@cirworld.com
www.cirworld.com, member.cirworld.com

1. INTRODUCTION

The process of searching and finding entities that are in harmony with an individual's personality and needs has always been a critical problem in the human existence, present on all levels: physical, emotional, intellectual, and spiritual. This behavior of continuous search has led to the creation of religions, ideologies, sciences and arts, the individuals reflected their personalities in their discoveries and modeled them to represent them, by this becoming a part of their creations. Passing from the level of searching for the purpose of existence, a problem that is present during our entire lives, we get to a smaller level: the daily problem of searching for and finding things that we currently need: the place where we parked our car, details about a new movie, a new place to visit. If for the first two problems, the technology has given us good solutions through applications like 'Car Finder' or 'Google', for the last problem a satisfactory solution hasn't been created yet, because the queries haven't been created to be subjective enough to the user that makes them.

Recommendation systems ([10], [26], [22], [24], [7]) are a solution, and systems that are more and more user-sensitive are being developed; we can find simple recommendation systems everywhere in one form or another, from video websites (user ratings on YouTube, Metacafe), to social networks (likes on Facebook), and online tourist guides (Hotels' ratings on Trip Advisor, Frommer's, etc.) ([10], [26], [17]). The problem with most of these systems is that they are objective to the user, and they display the exact same values for everyone, regardless of their tastes and interests. This often isn't meaningful information in the user's context, because his needs might be different from those of the average user, or from the category of people that appreciate a certain object. A simple example might be built from a user's taste in music. Let's assume that John likes classical music, especially Beethoven. He lands on YouTube for the first time, and starts to browse for music videos. He will find hundreds of very popular videos, with high ratings, but when he plays them, he will be surprised that he hears teenage pop artists, whose music he dislikes. The issue in this particular case is that the recommendation system doesn't take into account John's preferences in music, which are totally different from those of the pop artists' fans. However, if the system would observe John's behavior for enough time, it could gather information about which videos he's watching, for how long he is watching, and what ratings he is giving them. Using this information it could observe John's interest in classical music, and when he browses for new music, it could adjust the ratings to be more appropriate for him. YouTube implements this, but at a very superficial level, through the 'Recommended for you' section, which displays videos with similar tags with videos that have been viewed in the past.

Most rating and recommendation systems are blind to the user's needs, and because of this they are often misguiding. Some systems that take the user's needs into account have been designed, but usually they only work with huge databases ([8], [18], [4], [21], [11], [5]), and estimate the user's preferences indirectly, by comparing them with those of similar users. An example is the system present on some e-shops, which, simply described, works based on the assumption that 'people who buy x also buy y'. Other systems use the k-nearest neighborhood approach ([9], [6]), but these usually only work on social networks (where connections between users is directly visible through direct or indirect friendship connections) ([10], [26], [7], [1], [14], [3]).

Our system attempts to solve some of the downfalls present in other recommendation systems (user-blindness, need for large amounts of data, or direct connections between the users), by completely redesigning the rating system on both the rater side (the user), and rated side (the subjects to be recommended). We applied the system on a particular domain, mainly real-life locations that a user might be interested in visiting (the locations can be anything from entertainment sources like bars, restaurants, hotels, museums, operas, to service offerers like supermarkets, car shops, stadiums, factories, and so on). We tried to automate the system as much as possible, and considered that visited places should be logged to a user's account automatically, and the ratings should be made as easily as possible. Because of these needs, we implemented the system on a mobile platform, namely smartphones running on the Android OS, because of the things it offers: fine-coarse localization sensors (GPS), mobility, and constant presence of the smartphone at the user.

The rating system was designed with the purpose of representing the locations' characteristics on more than just one level. While classical rating systems use just one value to rate the entire experience, our rating system tries to represent every characteristic or at least every important characteristic of the location separately. This allows us to obtain meaningful information with small amounts of data, and to easily extract both the user's preferences, and the location's styles very fast, without requiring hundreds of thousands of ratings.

2. THE TAG SYSTEM

The core of the system's functionality lies in determining the user's preferences, in obtaining relevant data from these preferences, and in finding the right suggestions for the current user by comparing his preferences with other users'.

To meet this requirement, a set of textual descriptors is attributed to each location (tags) that will in the end be keywords for the respective location. Initially the locations will have a set of tags added by the owners while registering the location, but later the users will be able to suggest tags for each location they visited (more on this later). After a visit to a certain location, a user will be allowed to rate it, by giving marks to the tags that interested him during his visit. A user's preferences will later be extracted from all the ratings he has given throughout the history of him using the system, and when he will make a query on the system for receiving recommendations that fit him, these preferences will be used as input in the search algorithms.

The tag system has been designed starting from the requirements presented below. These requirements must be respected to allow the tag system to represent as much information as possible about both the locations and the users, without becoming redundant. Here are the requirements:

1. The tags must be quantified to a level that will allow them to represent correctly the aspect for which they are intended. For example, a tag with the name food is too general (because it covers a domain that is too large, and doesn’t give any information, because everybody likes, or at least requires, food), and doesn’t represent correctly the aspect for which it was intended. Another bad example could be Fanta (the juice); in this case the tag creates too much fragmentation and we will not be able to extract the user’s global preferences from tags that are too specialized. Some good examples could be:italian-pasta, fizzy-drinks, etc.
2. The tags must be organized in a hierarchical structure, or a tree, with the tags on the higher levels (closer to the root) being more general, and the tags on the lower levels (closer to the leaves) being more specialized. This requirement is important for the search algorithms and for an efficient organization and data representation.

Taking these requirements into account, the tag system has been built like this: the tags will be structured hierarchically (tree structure) with a height of maximum K (K will be predefined in the system; for this iteration of the system, K has been chosen to be equal to 3) + 1 (the root). The generality of the tag will decrease as we descend into the tree, and the leaf will be the actual description tag (its parents will only be present for structuring, not for representing the locations’ attributes). The user will see the actual tag with all its parents, in the following representation:

category:subcategory:tag

The parent-child relation is represented by the semicolon. Some examples using this representation could be:

food:sweets:ice-cream

food:cooked:sea-food

music:rock:classic

music:classical:romantic

This structure solves certain aspects from the requirements presented above, but some other problems appear due to the fact that users can add their own tags. The first problem is duplication: two tags that are supposed to represent the same information can be present in the system with different names. The best solution for this issue is to merge the two tags and keep the name from one of them. Another problem can be the over or under-specialization of the leaf tags. This is a sensible issue and can only be tackled when enough information is available in the system. However, it can be solved by creating a globalized system where the users control the tags that are created and associated with locations, with specific rules and permissions associated to users depending on their level of contribution. In general, it would be similar to a wiki, which is a website that allows the creation and editing of any number of interlinked web pages via a web browser using a simplified markup language or a WYSIWYG text editor, typically powered by wiki software and often used collaboratively by multiple users ([15], [19]).

2.1. Operations on the tagsystem

At the moment we defined two operations on the tag system: addition of new tags, additions of new tags to a location, and merger of two tags that are identical, but have different names.

2.1.1. Adding tags to the system

The addition of tags will be very important throughout the entire usage time of the application, because a good tag system is vital to the correct functionality of the application. To add new tags, the users will have a special section in the application, used only for suggesting the addition of new tags, and for voting for the addition of new tags.

The addition will thus be held in two parts: the sug-gestion, and the voting. The pseudocode that describes this process is presented below:

```
1. let numSuggestionsMinimumAdd and numVotesMinimumAdd be parameters with
   numVotesMinimumAdd>= 10 * numSuggestionsMinimumAdd
2. let Userx suggest the addition of a new tag Tx in the system
3. lookup similar exiting tags and present them to Userx
4. if Userx is satisfied with an existing tag stop the process.
5. start the suggestion process (last indefinitely):
6. while (suggestions(add Tx) <numSuggestionsMinimumAdd) wait.
7. start the voting process (lasts 1 week).
8. if (votes(add) > votes(don't add) ANDvotes(total) >numVotesMinimumAdd)
           Add Tx as a new tag in the application
   else
           Don't add Tx as a new tag.
```

The process of adding a tag to a location is similar to the process of adding a new tag to the tag system: a user must suggest the addition of the tag first, then this suggestion must receive enough approvements, after which the voting is started. The pseudocode for adding a tag to a location is presented below:

```
1. let numSuggestionsMinimumAdd and numLocationVotesMinimumAdd be parameters with
   numLocationVotesMinimumAdd>= 4 * numSuggestionsMinimumAdd
2. let Userx suggest the addition of a new tag Tx to the location
3. lookup similar exiting tags and present them to Userx
4. if Userx is satisfied with an existing tag stop the process.
5. start the suggestion process (last indefinitely):
6. while (suggestions(addTx) <numSuggestionsMinimumAdd) wait.
7. start the voting process (lasts 1 week).
8. if (votes(add) > votes(don't add) ANDvotes(total) >numLocationVotesMinimumAdd)
           Add Tx as a new tag in the application
   else
           Don't add Tx as a new tag.
```

2.1.2. Merging tags

Because the users will be allowed to add new tags, the tag duplication problem could appear: two tags that should be identical are present in the system with different names. The duplication situations could arise from spelling, usage of synonyms while creating the tags, and so on (even if a similarity lookup service is performed in the adding phase). Here are some cases that describe these situations:

- *food:sweets:ice-cream* and *food:sweets:icecream*. In this case two tags that obviously represent the same characteristic are stored differently because of the hyphen.
- *drinks:non-alcoholic:fizzy-drinks* and *drinks:non-alcoholic:bubbly-drinks*. Here the duplication appears because of the usage of synonyms (*fizzy* and *bubbly*) while creating the tags.

The scope of this operation will be the entire application's tag system, and not the tags of a specific location. After two tags are merged, just one of them is kept, and this change will be visible for every location that originally had one of the two tags in its tag set.

The merge operation is done using a simple vote system. Every user can suggest a merger, and when enough suggestions have been made for a pair of tags (this value will be variable, depending on the number of active users, and of the tags' popularity), a poll will be initiated for merging the two tags. Every user will be able to vote for the merger. If the number of votes is at least twice the number of merging suggestions, and at least 50%+1 voters agree on the merger, it will be implemented. The merging will be supervised by an administrator of the tag system, and will be done in the following manner:

1. if both tags are grammatically incorrect, e.g.: *food:sweets:ise-cream* and *food:sweets:ice-creame*, the administrator will create a new tag that is correct.
2. if just one tag is grammatically incorrect, the correct tag will be kept.
3. if merging two grammatically correct or acceptable tags (e.g. both *color* and *colour* are acceptable, the former in USA, the later in UK), the most popular tag will be kept. Popularity is computed as the sum of the durations of all the visits effectuated on locations with the tag present in their tag set, visits in which this tag has been rated. The tag with the highest popularity will be kept.

The pseudocode for the merging operation is the following:

```
1. start: let Ta, Tb be the two tags that represent the same information
2. while (suggestions(mergeTawithTb) <numSuggestionsMinimum) wait for a new merge Ta with Tb suggestion.
3. run a vote for merging Ta and Tblasting 1 week.
4. if (votes(merge) > votes(dont-merge) ANDvotes(TOTAL) > 2*numSuggestions)
   begin
           merge Ta with Tb. Let Tc be the new tag name,
                   whereTc can be Ta or Tb or neither.
           replace Ta and Tb with Tc in the system.
```

```
end
```

This algorithm might be tweaked or changed once we have enough active users in the system, to allow mergers to take place more correctly. At this point, the only number that can be changed to control the functionality of the merger is the **numSuggestionsMinimum** value, the minimum number of suggestions that has to be reached for a merger vote to be started. We used a value of 30 for testing the application.

2.1.3. Absolute tags

To better classify and differentiate the locations, a special class of tags has been introduced, whose only scope will be to describe the locations (i.e. they will not be used for rating purposes, like the normal tags). These are called absolute tags exactly for this reason: they describe certain absolute characteristics of the location (such as geographic localization, type, spoken languages and so on). Although the geometry of the space is stored separately from the tags, it can also be considered a type of absolute tag. Some examples for these type of tags are:

1. absolute:location-city:Oradea
    a. Here, the category (absolute) specifies the tag type. It is going to be a descriptive tag. The subcategory (location-city) specifies the characteristic that will be described (i.e. the location's city; here, 'location' means geographic positioning, and should not be confused with the actual locations that are the objects of our recommendations system). The tag, "Oradea", is the value of the current absolute parameter. Other similar examples could be:
        i. absolute:location-country:France
        ii. absolute:location-city:Paris
2. absolute:type:hotel
    a. Same as for the previous example: absolute specifies the tag type, type specifies the described characteristic, and the hotel is the value for this tag. Other examples in the same category could be:
        i. absolute:type:restaurant
        ii. absolute:type:museum
        iii. absolute:type:stadium
3. absolute:spoken-language:Romanian
    a. Self-explanatory. For tourists this information will be interesting if they want to make sure that the place they visit has employees that speak their language.
4. absolute:hotel-rating:5-stars

The purpose of the absolute tags is the usage in user searches as filters, for further differentiation of the results. For example, the users can search for a specific type of location (e.g. museums), for hotels that have employees who speak a certain language etc.

2.2. Rating the locations

While the tags are used to describe the locations, the user's preferences will be extracted from the ratings they give to specific tags for the locations they visit. After each visit, a user will have the option to give ratings to the tags for the location he visited. This means that each single visit will have independent ratings. A visit will also be from now on called an instance. The ratings will have 10 possible values, 5 positive and 5 negative i.e. the integers between -5 and 5, with 0 missing from this interval. Beside the ratings, the time spent in the location will also be taken into account when determining the importance of the current instance ([10], [26], [20]). In plain words, the more time a user spends during a visit, the more important will be the ratings he gives for it. For example, a visit that lasts for 10 hours is most likely going to be more important than 10 visits that last for 5 minutes. This separates the ratings from different instances. Another usage of the time spent in a location will be when computing the global preferences of a user. The more total time he spends in a location, the higher the ratings given for this location will score in the overall preference factor.

3. THE RECOMMENDATION ALGORITHM

The main purpose of the application is to obtain recommendations for new locations that can fit the user, based on his recorded preferences. The tag system and location rating is the component that contains the user's preferences, the algorithm being the part that uses this data to extract certain preference rules, and find new recommendations based on these rules.

The implementation of such an algorithm can be difficult, mainly because we cannot know precisely which will be the aspect of the preferences database when a very large number of users will have rated many locations. Certain rating patterns or preference personalities could emerge on such a database, but before that data is available, we still need to implement algorithms that obtain relevant results. In this paper we present a recommendation algorithm that works in similar ways, extending on the preferences of the user and on the types of locations he visits to find recommendations. We designed these algorithms ourselves, based on the format of the data gathered through the rating system, and on the needs of our system.

Before presenting the algorithms, here are some general requirements that these or some future algorithms should meet:

1. The algorithms should be able to obtain new recommendations for different types of locations, using preferences that do not describe those types. For example, a user should be able to obtain recommendations for museums and operas even if the application holds records of him only visiting hotels.
2. They should obtain recommendations that extend the users' preference base, not staying just inside the domain of his preferences. The purpose of this requirement is to encourage the user to try new experiences that he has not tried before and that he might enjoy.
3. The algorithms should work and give useful recommendations in any place in the world, even if the user is not connected at all to that place. This is where the universality of the tag system will play a very important role, allowing local preferences to be relevant on a global level.

Another important term will be useful for the algorithms: the locations' personality, or style. This is aonedimensional attribute, and it is obtained from all the ratings given for a location's tags. It is obtained simply by computing a coefficient for each tag, the coefficient being directly (but not linearly) proportional with the ratings, the time visitors spend in this location, and the numbers of visitors that visit it. All these coefficients form a vector that is the location's personality or style. While the tags give descriptive information on the locations' personality, these coefficients quantify the overall level of appreciation and quality. To use an analogy, if the tags are subjects, the coefficients are adjectives.

Our algorithms will work on an input set of selected locations. The selection is done as follows: the user selects a geographic position on the globe, and a search radius. For example, he can select the center of his hometown, and a radius of 10 kilometers. After this, he selects the search filters, if required. He can select the location type, spoken language, and so on. The locations that match these parameters (geographic location and search filters) will be candidates for recommendations. We call this set C (from candidates).

Our algorithms will also use the set of all the locations the user has visited and rated inside the application. We will call this set U (from users' locations). The application can only work with data it has recorded, and doesn't have any knowledge of the places the user visited prior to its usage.

To help better understand our algorithms, they will first be presented shortly, and then in greater detail. Explained briefly, they will compute the user's general preferences, and based on them will find new places, that match those preferences. They can work globally, obtaining recommendations in any place in the world. However, these algorithms do not extend on the user's preferences, and can't find places that are different from what the user already likes. They are pattern-finding and searching algorithms, not creation algorithms (here by creation we understand the ability of finding patterns that are not similar to the original ones, but are still meaningful in the user's class of patterns).

3.1. Extracting patterns from the user's preferences

Our first algorithm will use all the ratings the user has given for all the tags throughout the history of using the application, without considering which locations have received the ratings. This is the input data, together with the localization filters (the place on the globe where the search will be effectuated), and further location filters (type, spoken languages, etc.).

From the user's ratings, this algorithm will first compute a coefficient for each tag that has at least one rating in this set, using the following formula:

$$CT_{u_k} = \frac{\sum_{i=1}^{n} t_i * v_i}{\sum_i^n t_i} \qquad (1)$$

where:

- T_u is the set of tags for which the user has at least one rating
- k is the tag's index from the above set for which the coefficient is computed
- CT_{u_k} is the computed coefficient for the T_{u_k} tag
- $n$is the total number of ratings given for the T_{u_k} tag
- $t_i$is the time spent in the location when the i-th rating was given
- $v_i$is the i-th rating given for the T_{u_k} tag.

This coefficient is just a weighted average, where the weights are the times spent inside the locations corresponding to each rating. The weights are linear to allow a correct evaluation of the preference for the aspect represented for the current tag.

After the coefficients for all the tags have been computed, they are sorted descending, and used in the search process. For every location inside the input set of possible candidates, C, the following steps will be executed:

1. We intersect the set of all the location's tags with the set of all the tags the user has rated. These are the tags that appear both for the user, and for the current candidate location. The size of this new set is going to be called N.

2. For each tag of the current location, compute the similarity degree (T_f) between this tag's coefficient in the location's personality (T_{cp}), and this tag's coefficient in the user's preferences (T_{cu}). The purpose of this step is to see how much the place meets the user's requirements in the different aspects he's interested in. This factor is computed using the following formula, created by us for the current algorithm:

$$T_f = 1 - (T_{cu} - T_{cp})/T_{cu} \quad (2)$$

Here are some examples to clarify this formula. Let's consider that the current tag is defined as follows: *music:rock:classic-rock*. We will take different cases into consideration:

Case 1. The user likes classic rock, and the current place also plays this style of music.

$T_{cu} = 4.5$. The user likes this music style to a high degree.

$T_{cp} = 4$. The place is appreciated for this type of music, but not strongly.

$T_f = 1 - 0.5/4.5 = 0.(8)$. The location meets the user's requirements to 88%.

Case 2. The user likes classic rock moderately.

$T_{cu} = 2$

$T_{cp} = 4$

$T_f = 1 - (-2)/2 = 2$. The location over-meets the user's requirements by 100%.

Case 3. The user likes classic rock highly, but the location doesn't offer enough.

$T_{cu} = 4.5$

$T_{cp} = 2$

$T_f = 1 - 2.5/4.5 = 0.(4)$.

By computing T_f, we will obtain the degree of how much of what the user likes in the current tag he is going to find in the current location. We will add these degrees for all the N tags that are common to both the user and the location, and will obtain the following formula:

$$S_{location} = \sum_{i=1}^{N} T_f \quad (3)$$

The location sum ($S_{location}$) will represent how much the current location has to offer of what the user likes. We will divide this value with the number of all the user's tags, which will be all the tags that the user has rated throughout the history of using the application (in total, M tags).

The final recommendation level for the current location is computed as:

$$Rec_{location} = S_{location}/M \quad (4)$$

3.2. Extracting patterns from the user's preferred locations' personalities

This algorithm works similarly with the previous one, but instead of using as a reference all the tags the user has rated, it uses the personalities of the locations the user visits most of the time. This algorithm requires as an input filter at least the location type. For example, if the user is interested in looking for museums in New York, the search will start from the following start tags:

absolute:type:museum

absolute:location-city:New-York

This algorithm requires the user to have visited the same type of location at least once. It then searches through the database for all these locations (for the current example, museums), and orders them in descending order based on the total time the user has spent in them; the total time spent inside a location will be called$Time_i$, with i being the index of the current location. We will then create a set of all the tags present in these locations, called $S_{reference-tags}$ which will be the union of all the tags for each specific location. If, in total, we have N locations, then:

$$S_{reference-tags} = \cup_{i=1}^{N} S_{location-i-tags} \quad (5)$$

Let the total number of tags in this set be called$M$. For each of these tags, we will compute a coefficient that will be used when searching for similar locations in the same way as the T_{cu} coefficient was used in the previous algorithm for finding new locations. This tag is computed using the following formula:

$$T_{cup-j} = \frac{\sum_{i=1}^{n} t_i * v_i}{\sum_{i=1}^{n} t_i} \quad (6)$$

where:

- $j$is the current tag.
- $n$is the number of all the ratings that have been given for this tag by all the users that visited this location.

- $t_i$is the time spent by the user who gave the i-th rating during the respective visit.
- $v_i$is the value of the i-th rating.

Now that we have found the archetype of the places the user prefers, we will re-run the previous algorithm, only that instead of using the T_{cu} coefficients, we will use the T_{cup} coefficients. I.e., the following steps are performed.

After the coefficients for all the tags have been computed, they are sorted descending, and used in the search process. For every location inside the input set of possible candidates, *C*, the following steps will be executed:

1. We intersect the set of all the location's tags with the set $S_{reference-tags}$. These are the tags that appear both for the user, and for the archetype of the preferred locations. The size of this new set is going to be called N.
2. For each tag of the current location, compute the similarity degree (T_f) between this tag's coefficient in the location's personality (T_{cp}), and this tag's coefficient in the user's preferred locations personalities (T_{cup}). This factor is computed using the following formula:

$$T_f = 1 - (T_{cup} - T_{cp})/T_{cup} \tag{7}$$

By computing T_f, we will obtain the degree of how much of what the user's preferred locations offer he is going to find in the current location. We will add these degrees for all the N tags that are common to both the user and the $S_{reference-tags}$ set, and will obtain the following formula:

$$S_{location} = \sum_{i=1}^{N} T_{f_i} \tag{8}$$

The location sum ($S_{location}$) will represent how much the current location has to offer of what the user's most preferred locations offer. We will divide this value with the size of the $S_{reference-tags}$ set, which is$M$.

The final recommendation level for the current location is computed as:

$$Rec_{location} = S_{location}/M \tag{9}$$

4. THE ANDROID APPLICATION

In this section we will shortly present the Android application that was created for our recommendation system; some screenshots with the interface are included.

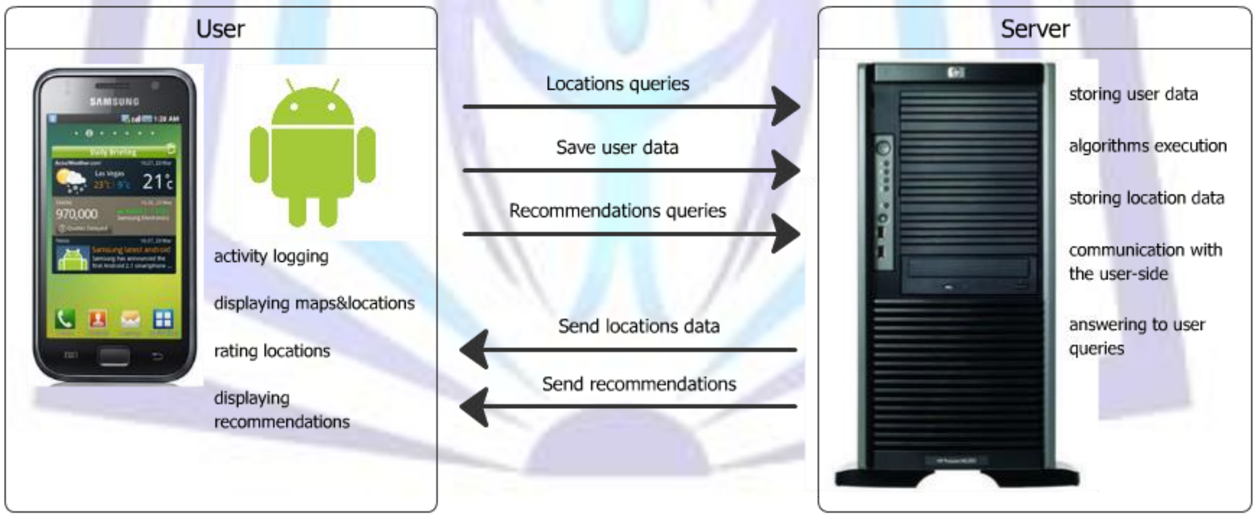


Figure 1. The Android application and communication with the server.

The application has two main parts: the user side, and the server side, which are interconnected and communicating constantly. The user side is responsible with gathering user data, and sending it to the server. The server collects all the information that it receives from all the users, and uses it to run the recommenda-tion algorithms and send results to the users. The separation of tasks and communication between the two sides is represented in **Figure 1**.

4.1. The server/web service

This is the most important part of the application, because it is used for collecting and storing all the data that is logged on the user side, and running the recommendation algorithms; the server-side also stores information about the locations: the names and details of each location, together with the tag list for each location. The tag system, and current operations being ran on it: mergers, suggestions for new tags, votes, are also stored on the server.

The server communicates with the user side whenever a request is made from the Android application. Requests from the user-side to the server are made in the following cases:

1. when the user needs to receive information about the map of all the locations in a certain area; since all the locations are stored on the server, when the user arrives to a new area, for which he does not have the map, a request is made to the server to receive a list of all the locations that are in a certain area surrounding the user.
2. whenever the user performs an operation in the Android application that needs to be persisted on the server: a new rating is given for a visit, a new visit to a location has started or ended, or the user makes an action regarding the tag system: suggestion for a new tag, merger between two tags, etc.
3. to make a recommendation request: when the user wants to receive recommendations for a certain location, a request is made to the server. The server runs the two algorithms, and sends the results given by the algorithms to the user, in the form of a list that contains all the locations that match the user's preferences, together with a recommendation value, ranging from 0 to 5, where 5 is given to the most recommended, and 0 to the least recommended place.

4.2. The Android application

4.2.1. Functionality

The Android application consists of two components: a service that is always running, and the user interface, which is active only when the user starts the program and wants to do something in it: to give ratings for a recent visit, to get recommendations, or to view his current location.

The service is responsible with logging the user's visits, and sending this information to the server. To do this, it uses the phone's geolocation sensors (a GPS sensor is required for enough precision) to track the user's movement. The user is considered to be stationary if his movement speed is below 10 km/h; the service checks periodically if the user is stationary, and if this condition is met, the server checks to see if the user is inside one of the locations of its current map. When the user first gets inside a new location, the service sends this information to the server, and remembers the current time, which will be used to calculate the duration of the visit. An heuristic algorithm is used to check if the user remains inside this location, and when he leaves, all the required information for this visit is logged to the server, and in the local database.

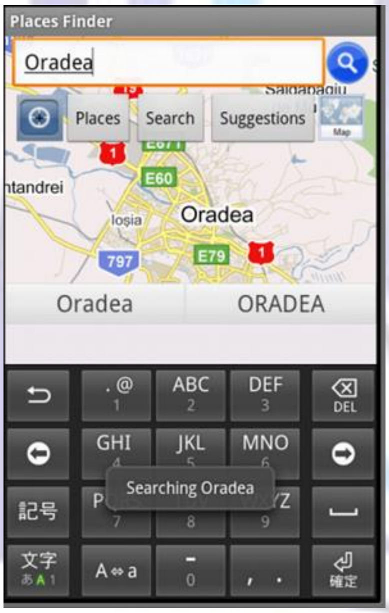
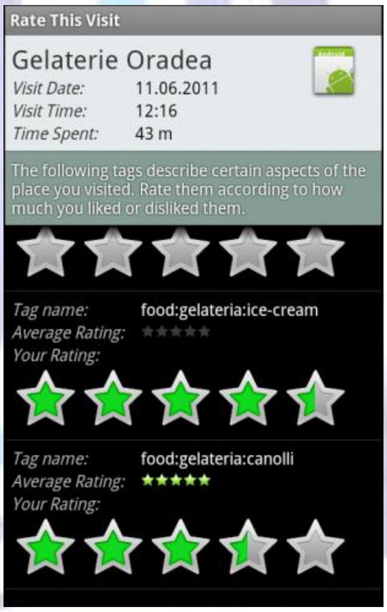


Figure 2. The start screen.


Rate This Visit

Gelaterie Oradea

Visit Date: 11.06.2011
Visit Time: 12:16
Time Spent: 43 m

The following tags describe certain aspects of the place you visited. Rate them according to how much you liked or disliked them.

Tag name: food:gelateria:ice-cream
Average Rating:
Your Rating:

Tag name: food:gelateria:canolli
Average Rating:
Your Rating:

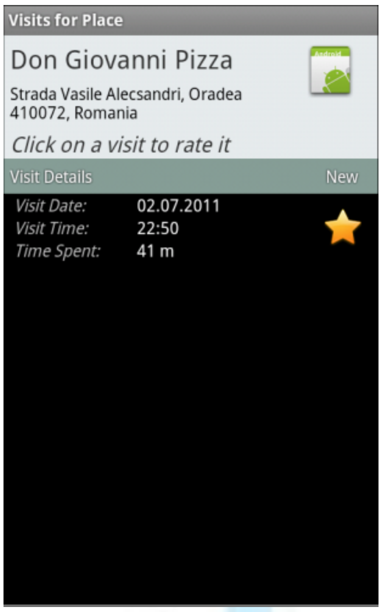
Figure 3. Rate a visit.

The service also makes periodic requests to the server for obtaining the map of the locations surrounding the current position of the user. For this, a simple algorithm is used that works on the condition of always having a map with all the locations in a radius of *X* available. To achieve this, when a request for a new map is made to the server, all the locations within a 2**X* radius is asked for; because of this, the user can move inside a circle with radius *X* before a new map request has to be made. The requests for new maps are made only when the user is stationary, because if *X* is too small, and the user is traveling for a long period (e.g. by car, train, etc.), a lot of unnecessary information will be asked.

4.2.2. Interface

The application's start screen (**Figure 2**) displays a map based on the Google Maps API, and several other buttons:

- a bull's-eye used for always centering the map to the user's current position.
- 'Places' button, used for viewing the user's loca-tions visit history.
- 'Search' button, used for searching for a specific address.
- 'Suggestions' button, used for obtaining recommendations.
- A Map/Satellite toggle button used for changing the map's view mode.



Visits for Place

Don Giovanni Pizza

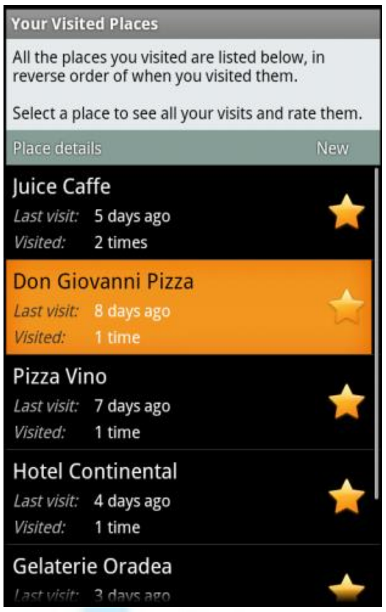
Strada Vasile Alecsandri, Oradea
410072, Romania

Click on a visit to rate it

Visit Details New

Visit Date: 02.07.2011
Visit Time: 22:50
Time Spent: 41 m

Figure 4. Visits for a place.



Your Visited Places

All the places you visited are listed below, in reverse order of when you visited them.

Select a place to see all your visits and rate them.

Place details New

Juice Caffe
Last visit: 5 days ago
Visited: 2 times

Don Giovanni Pizza
Last visit: 8 days ago
Visited: 1 time

Pizza Vino
Last visit: 7 days ago
Visited: 1 time

Hotel Continental
Last visit: 4 days ago
Visited: 1 time

Gelaterie Oradea
Last visit: 3 days ago

Figure 5. Places visited.

The rating process is presented in the **Figure 3**, **Figure 4** and **Figure 5**: after the user enters the 'Places' menu, he sees a list with all the locations he's ever visited, with the locations that have unrated visits marked with a star. The user then selects a location, and sees a list with all the visits, the unrated visits being marked as new, just like in the previous list.

After selecting a visit that he wants to rate, he gets to the rating page, where all the tags for the current loca-tionare displayed, with the option of rating them, and with the average rating offered by all the users being displayed.

The recommendation page simply displays all the recommended places with a recommendation degree assigned to them, with the most recommended place being shown first.

5. COMPARISON WITH EXISTING TECHNOLOGIES

In this section we are trying to explain what the main differences from classic rating systems are, and how these differences affect the performance, and the output of our system. We are going to compare our system with that of a popular online site that has a very similar purpose, namely Trip Advisor - a travel website that presents travel locations and attractions, along with user ratings and opinions.

Comparing the output of the two systems, these are the main differences:

5.1. Rating presentation

One of the advantages offered by our system is that all ratings are subjective - each user gets his own personalized rating, based on his profile in our system; for example, if we take the 1-5 rating scale, one user might see a '5' rating for a location, while another might see a '1', depending on what their preferences are. What this means is that we show no absolute ratings to the users, but instead all the ratings are subjective and specific to each person. If we compare this with how the ratings are shown on trip advisor, we see that in their case, the ratings are always the same for all users.

Because of this difference, we can say that the rating presented by our system for each location is actually a 'matching index', which indicates how much a person and his preferences matches a location - we are trying to show to the user how much he would enjoy a specific place, and not what is the average opinion.

5.2. Preference-weighted ratings

First, we will consider how and if the system used by Trip Advisor uses any weights for the ratings that the users give to specific locations. Based on their outputs (the final ratings), and on the fact that they are static for different users, there are two possibilities:

1. the final ratings are un-weighted averages of all the ratings given by the users; i.e., we have a perfect democracy, and each user's opinion has the same impact.
2. the final ratings are weighted averages of all the ratings given by the users, with the weights being determined by some specific rules (how many ratings a user has given, how objective he is, etc.)

In both cases, there is a constant problem: the votes are either all equal, or their weights are always constant, regardless of what the user that is viewing the rating likes or dislikes. For example, let's take into account a music club which plays a very specific genre of music, which is appreciated only by some people. Also, let's consider that this club is located in a very popular area for tourists, and has an average rating due to this fact - being located in a popular area, a lot of people visit it, and many end up disliking it because of the type of music it plays. This means that everyone that checks this location’s rating sees an average rating, even if some would love it, while others would hate it.

What our system does to solve this issue, is that it uses different weights for the ratings depending on the user for which the rating is being calculated. We still have a weighted average, but the weighs depend not only on the users that have given the ratings, but also on the user that is viewing the rating. If we apply this to the above example, this is how the weights change: if someone who likes the music type played at the club is checking its rating, the previous visitors who like the same music type will have high weights for their ratings, while the visitors who dislike the music type will have lower weighs. The reverse is also true, and if someone who dislikes the music type is interested in the club’s rating, people who disliked the music will have higher weights for their ratings.

5.3. Time-weighted ratings

Another factor that we considered very important in offering an accurate rating was the time during which the users were exposed to the experience they were rating. From our knowledge, this factor isn’t included in any other rating system at the present time, and while it is not critical, it is definitely important, because a longer duration of an experience gives people the possibility of forming a better and more correct opinion of the respective experience. For example, a person that stays in a hotel for one night might base his rating entirely on his first impression, or on a strong impression formed during that night, be it negative or positive. His rating might be totally different had he stayed more days, a week or more at the hotel.

This factor can’t be used for all rating systems however. For example, we can’t use the time weights for ratings given for movies, because most of the people will watch a movie for its whole length, so they’ll have equal influences.

However, for the case in which our system is being used, the experiences from which the ratings are determined have durations that vary greatly. For example, a person that dislikes a bar might stay for only a couple of minutes in it and give it a very bad rating, while someone who likes it can be a regular customer, and spend hours during each visit. This shows that the bar fits this customer’s profile and preferences, and this is why we will give him a higher weight when we consider his rating.

The fact that we included the time-factor when calculating the ratings’ weights was a strong reason for making us build the application as a mobile one - this allows us to log the time automatically, on the user’s phone. If we hadn’t built the system with a mobile side, we couldn’t have included the time factor, because asking the users to specify how much time their visits lasted would have been too unreliable.

5.4. Rating fragmentation

Another big difference in our system when compared with other rating technologies is the tag system - this allows the users to determine the different characteris-tics that define a location, and to break up a rating into more ratings, each for a different aspect of that location. For example, a restaurant will be rated not only for the food, but also for the music, the staff, the interior style, etc.

This difference turns the one-dimensional ratings of most systems (i.e. just one rating to represent the entire experience) into multi-dimensional ratings that more accurately get to represent what a location has to offer, and what are its strong and weak points.

This also gives the users the possibility of choosing the places that give them the things they are interested in, without caring if other aspects that they’re not interested in have lower ratings.

Summing up all these differences, the main goal is to show, for each user, and for each location, a rating that is as close as possible to what the actual rating the user would give to that respective location, after he had visited it. This is the point from which we started when we designed the system, and, being its central point, it is another difference from classic rating systems, that just offer a static global average.

6. CONCLUSION

In this paper we introduced a new rating method based on a tag system which is used to describe the most important characteristics of the objects that have to be rated, in our case, real-life locations. Using this tag system, we automatically classify the locations and find meaningful information about them, without even needing any ratings in the system. This allows us to find details about the user’s preferences and to offer them good recommendations with very small amounts of data (tens to hundreds of records), which is a significant improvement over classical recommendation systems that require databases with hundreds of thousands or millions of records to be able to find useful preferential patterns.

The tag system is the core of our recommendation system, and on this core we designed an algorithm that can use the data to obtain useful recommendations. This algorithm takes into account the user’s behavior by considering his preferences and finding locations that match them. As in all recommendation systems, better results are obtained when more algorithms are used on the present data, and their results combined ([5], [1]), and we intend to design and implement new algorithms for our tagging and rating system, that take into account preference connections (we can construct a graph on the locations, with connections being created between locations that have weights proportional to the number of people, place-to-place, user-to-user and place-to-user preference matching - our algorithms use user-to-place matching), behavior patterns that take the time spent in locations into account, and similarities between branches in the tag system. The great advantage given by our tag system is that it already offers us all the information we need about the users' behavior and the places' style, information which we couldn't find with just one rating per location, or would require huge databases to be inferred.

REFERENCES

[1] Adomavicius, G., Tuzhilin, A. (2005), 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 (6): 734–749, doi:10.1109/TKDE.2005.99.

[2] Athanasios P., Christos Z. (2007), Searchius: A Collaborative Search Engine, ENC '07: *Proceedings of the Eighth Mexican International Conference on Current Trends in Computer Science*: 88–98, doi:10.1109/ENC.2007.34.

[3] Barry S., Evelyn B., Oisin B., Keith B., Peter B., Maurice C., Jill F. (2005), A Live-User Evaluation of Collaborative Web Search, IJCAI.

[4] Barry, S., Evelyn B., Peter B., Maurice C., Jill F. (2003), Collaborative Web Search, IJCAI: 1417–1419.

[5] Bell R., Koren Y., Volinsky C. (2007), The BellKor solution to the Netflix Prize, 2007.

[6] Cover T.M., Hart P.E. (1967), Nearest neighbor pattern classification, *IEEE Transactions on Information Theory* 13 (1), 21–27. doi:10.1109/TIT.1967.1053964.

[7] Dietmar, J., Markus Z., Alexander F., Gerhard F., (2010), Recommender Systems: An Introduction, *Cambridge University Press*, 2010, ISBN-13: 978-0521493369.

[8] Golovchinsky, G., Pickens, J. (2007), Collaborative Exploratory Search, *Proceedings of HCIR 2007 workshop*.

[9] Hall P, Park BU, Samworth R. J.(2008), Choice of neighbor order in nearest-neighbor classification, *Annals of Statistics* 36 (5): 2135–2152. doi:10.1214/07-AOS537.

[10] Herlocker, J. L., Konstan, J. A., Terveen, L. G., Riedl, J. T., (2004), Evaluating collaborative filtering recommender system, *ACM Trans. Inf. Syst.* 22 (1): 5–53, doi:10.1145/963770.963772.

[11] Longo L., Barrett S., Dondio P. (2009), Toward Social Search - From Explicit to Implicit Collaboration to Predict Users' Interests, WEBIST 2009 - *Proceedings of the Fifth International Conference on Web Information Systems and Technologies*, Lisbon, Portugal, March 23-26, 2009 1: 693-696, ISBN 978-989-8111-81-4.

[12] Longo L., Barrett S., Dondio P. (2010), Enhancing Social Search: A Computational Collective Intelligence Model of Behavioural Traits, Trust and Time, Transaction Computational Collective Intelligence II 2: 46-69, doi:10.1007/978-3-642-17155-0-3.

[13] Longo L., Barrett S., Dondio P. (2009), Information Foraging Theory as a Form of Collective Intelligence for Social Search, Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, *First International Conference, ICCCI 2009*, Wroclaw, Poland, October 5-7, 2009. Proceedings 1: 63-74, ISBN 978-3-642-04440-3.

[14] Maurice C. and Barry S. (2008), Nejdl, Wolfgang; Kay, Judy; Pu, Pearl et al., eds., Social Aspects of a Collaborative, Community-Based Search Network, *Adaptive Hypermedia and Adaptive Web-Based Systems*, Volume 5149/2008: 103–112, doi:10.1007/978-3-540-70987-9, ISBN 978-3-540-70984-8.

[15] Majchrzak A., Wagner C., Yates D. (2006), Corporate wiki users, Corporate wiki users: results of a survey, Symposium on Wikis, pp. 99, doi:10.1145/1149453.1149472, ISBN 1595934138, retrieved 2011-04-25.

[16] Meredith R. M., Eric H. (2007), SearchTogether: An Interface for Collaborative Web Search, UIST.

[17] Montaner, M., Lopez, B., de la Rosa, J. L.(2003), A Taxonomy of Recommender Agents on the Internet, *Artificial Intelligence Review*, Vol. 19, Issue 4, June 2003, pg. 285-330.

[18] Morris, M., Teevan, J. (2008), Understanding Groups' Properties as a Means of Improving Collaborative Search Systems.

[19] Müller C., Birn L (2006), Wikis for Collaborative Software Documentation, *Proceedings of I-KNOW '06*.

[20] Parsons, J.; Ralph, P.; Gallagher, K.(2004), Using viewing time to infer user preference in recommender systems, *AAAI Workshop in Semantic Web Personalization*, San Jose, Cali-fornia, July 2004.

[21] Pickens, J., Golovchinsky, G., Shah, C., Qvarfordt, P., Back, M. (2008), Collaborative Exploratory Search, pp. 315–322, doi:10.1145/1390334.1390389.

[22] Rennie, J., Srebro, N. (2005), Fast Maximum Margin Matrix Factorization for Collaborative Prediction. In Luc De Raedt, Stefan Wrobel (PDF). *Proceedings of the 22nd Annual International Conference on Machine Learning*. ACM Press.

[23] Rohini U, Vamshi A. (2002), A Collaborative Filtering based Re-ranking Strategy for Search in Digital Libraries, ICADL2005: *The 8th International Conference on Asian Digital Libraries*.

[24] Sarwar, B., Karypis, G., Konstan, J., Riedl, J. (2000), Application of Dimensionality Reduction in Recommender System A Case Study.

[25] Seikyung J., Juntae K., Herlocker, J.L. (2004), Applying Collaborative Filtering for Efficient Document Search, Inf. Retr.: 640–643.

[26] Takács G., Pilászy I., Németh B., Tikk D. (2009), Scalable Collaborative Filtering Approaches for Large Recommender Systems, Journal of Machine Learning Research 10: 623–656.

[27] Thorben B., Erik B., Klemens B., (2008), Discovering the Scope of Privacy Needs in Collaborative Search, Web Intelligence (WI).

Author' biography

Robert Győrödi received his MSc in Computer Science (1995) and PhD in Computer Science (2001) from "Politehnica" University of Timișoara, Romania. Now he is full professor of computer science at the Department of Computer Science and Information Technology, University of Oradea, Romania.

His current research interests include a combination of image processing, artificial intelligence, data mining and database management systems subjects.

He also has over 15 years of IT consulting experience.

He has (co-)authored more than 6 books and 70 papers.

Cornelia Győrödi received her PhD in Computer Science (2004) from "Politehnica" University of Timișoara, Romania. Now she is full professor of computer science at the Department of Computer Science and Information Technology, University of Oradea, Romania.

Her current research interests include database management systems, knowledge discovery in data-bases and data mining techniques.

She has (co-)authored more than 6 books and 70 papers.

Mihai Derșidan is a MSc student in the Department of Computer Science and Information Technology at University of Oradea, Romania with an interest in mobile computing and data mining.