Juggle Mobile: Recommending Music to Individuals and Groups

José Devezas
jld@fe.up.pt

Sérgio Nunes
ssn@fe.up.pt

Abstract

We present Juggle Mobile, a music recommender system capable of suggesting new artists to an individual or a group of nearby people. We describe the full process of creating such a system, including data gathering, and recommender system training and prediction. In this work, we also tackled recommendation to groups, proposing a balanced rating aggregation methodology. We validated our recommendation algorithm and further evaluated the system's usability and recommendations quality based on human feedback. Two iterations of Juggle Mobile were evaluated: the first one corresponding to our base system, and the second one incorporating user feedback from the first iteration. The recommender system suffered no change from the first to the second iteration. Results of the survey show an average quality of recommendations, as expected from the baseline algorithm we used. On the other hand, we were able to surpass the average value of 68 in the System Usability Scale (SUS), visibly improving from the first to the second iteration. We believe this work might be useful to the community in the sense that it compiles the whole process from conception to deployment of a complete recommender system.

1 Introduction

We present Juggle Mobile, a responsively designed system that can help users find music recommendations by building a profile based on the ratings of a randomly ordered collection of artists — using a five-star scale, presented as *Love*, *Like*, *Neutral*, *Dislike*, and *Hate* — as well as by optionally taking advantage of Facebook and Last.fm account data and combining it with the locally generated profile. As opposed to music discovery services that are also music players, we offer a system that provides a silent experience where the user can explore information about recommended artists and then listen to them by using popular services such as Spotify, Grooveshark or Last.fm. While we provide individual recommendations, we also provide a location based recommendation to groups feature, where the user can select from a list of people around, based on a sample of what they enjoy and avoid listening to, in order to combine his/her profile with the tastes of other users. We see two main use cases for the recommendation to groups module: *(i)* provide a recommendation to a group of friends gathered at the same location, or *(ii)* introduce a social bias in the user's profile to help deviate from an established taste, based on people from a similar geographical context. There are several techniques for recommendation to groups, where either user profiles are merged into a group profile or individual recommendations are combined in order to obtain a mutual list of recommendations. We introduce the awareness that these techniques might produce unbalanced results and that we should provide a result that is independent of the profile size of each group element, so that users with a large number of ratings do not dominate the results.

This paper serves to introduce this idea and to describe the architecture and user interface of our recommendation system. In Section 2, we provide a short survey on recommender systems. In Section 3, we describe the modules of our system and the architecture of the most relevant parts. In Section 4, we describe how the system interacts with the user, explaining some of the main challenges of design. In Section 5 we evaluate our system regarding usability and quality of the recommendations, based on cross-validation and on human

input. Finally, in Section 6, we conclude by highlighting some of the considerations to have when building recommender systems and by proposing some lines of future work.

2 Reference Work

In the following sections we will present an overview on recommender systems, focusing on collaborative filtering, particularly based on latent factor models, and music recommendation to groups.

2.1 Collaborative Filtering

As part of the winning team of the Netflix Prize competition[1], Koren et al. [9] demonstrated that matrix factorization models are superior to nearest neighbor techniques, for collaborative filtering. Neighborhood methods tackle the problem of recommendation by either finding other users that rated items similarly to the user, or by finding items that were similarly rated by all users when compared to the user's highly rated items. This is usually done by comparing user or item vectors based on metrics such as the Pearson's correlation coefficient, the cosine similarity or the Jaccard index.

On the other hand, latent factor models, which are usually based on matrix factorization, help uncover the hidden features that describe the items (e.g. in movies: comedy vs drama, amount of action, orientation to children), as well as how much the users like items based on each of the uncovered features. We can obtain these latent features by using Singular Value Decomposition (SVD), which, given a user-item matrix M, results in the user-features U matrix and the item-features V matrix, along with a matrix Σ that contains the singular values in the diagonal, which measure the importance of each feature [3]:

$$M = U\Sigma V^* \tag{1}$$

However, issues still arise from this technique, as the user-item matrix is sparse, i.e. most of the users haven't rated most of the items. To deal with sparsity, earlier systems used imputation, for instance by filling missing values with the mean of the row or column, thus making the matrix dense. This presents an issue, as it's more costly memory-wise to compute the SVD for a dense matrix than it is for a sparse matrix, but also because inaccurate imputation of values might distort the data. An alternative is to confine to available ratings, while avoiding overfitting through a regularization procedure. Two methods are described to solve the problem of minimizing the difference between known and predicted ratings: stochastic gradient descent and alternating least squares. Koren et al. also describe temporal dynamics, modeling this problem based on item biases, user biases and user preferences, over time. While temporal features are certainly relevant to improve recommender systems, we are even more interested in the techniques used to integrate this information into the predicted ratings matrix, since one of our goals is to account for social, cultural or community bias in our approach to music recommendation.

Sarwar et al. [15] proposed and validated an incremental technique for matrix factorization based on singular value decomposition, in the context of recommender systems. This technique, described as folding-in, consists of adding a new vector of user ratings to an existing factorization, which assumes that the item collection will be static after the initial factorization. To add a new user u, we first calculate its projection onto the latent factor space, using the formula $\hat{u} = uV\Sigma^{-1}$, and append the projected vector $\hat{u}$ to the user-features matrix U. While this same technique had been previously explored by Berry et al. [1] in the context of information retrieval, Sarwar et al. not only proposed a new application to the area of recommender systems, but also provided an illustration to help clarify the folding-in process.

2.2 Recommendation to Groups

In 2007, Jameson and Smyth [8] have explored the challenges posed by the task of recommending for groups in the context of the adaptive web. They identified four steps in this recommendation process: (*i*) acquire information about the user's preferences, (*ii*) generate recommendations, (*iii*) explain recommendations to the users, and (*iv*) help the users settle on a final decision. This approach takes advantage of the collective intelligence of users, adapting results based on a collaborative process. We argue that, given the recent boom in

[1] http://netflixprize.com

ubiquitous computing, there are several open opportunities for contribution, specially regarding the contextual automatization of the recommendation adaptive process. In their survey, Jameson and Smyth cite some of the existing group recommender systems, such as PolyLens, by O'Connor et al. [12], or Flytrap, an intelligent group music recommender system by Crossen et al. [4]. Both systems were developed in 2002, a time when smartphone usage was far from achieving its peak. In fact, that same year, BlackBerry launched what would be considered the first real smartphone. In 2004, Masthoff [11] studied user behavior in the selection of a sequence of television items for a group of viewers. She noticed that users instinctively took advantage of the *Average* strategy, the *Average without Misery* strategy, and the *Least Misery* strategy, thus having the whole group in mind when making their choice. In 2005, having noticed the rapid developments in mobile communications technologies, Zhiwen et al. [18], proposed taking advantage of these advancements to build an in-vehicle adaptive multimedia recommender for groups of users. Their approach was based on connecting the mobile devices of the users and the automobile's multimedia system through a Wi-Fi LAN. They collected individual user preferences and fed this information to the multimedia system, which received content from available providers through a GPRS network and played the best content based on the tastes of the current passengers. Smyth et al. [17] presented, in 2005, their work on adaptive web search, where, using a similar approach, they identified the implicit preferences for communities of searchers, in order to improve the results of future searches from users with the same characteristics and social context.

In 2013, Deventer et al. [5] have experimented with group recommendation in a television context. They merged the individual user profiles by using the *Least Misery* rating aggregation technique to model the group preferences, which is an effective strategy for small groups. The system's recommendations were based on a user-genre matrix, instead of a user-item matrix, which is quite smaller than the latter. They also experimented with user identification schemes, both using facial recognition and QR codes generated through a mobile application. They left some open challenges, namely discovering the correlation between the real group preferences and the *Least Misery* aggregation, providing reasons for the recommendations, and shielding the user's privacy given the shared environment with co-watching friends and family.

Piliponyte et al. [13] proposed a new *Balancing* approach to music recommendation to groups, as an alternative to the traditional rating aggregation strategies: *Average*, *Average without Misery*, or *Least Misery* (see Ricci et al. [14, Chapter 21, Table 21.3] for a complete list of rating aggregation techniques). They compared three approaches:

1. *Average* (the baseline), which consists of computing the predicted rating for each item and calculating the average over all the members of the group.

2. *Balancing without Decay*, which is done in two steps: first the "Average" aggregation is done and a section of the highest rated items is selected as the candidate set; then the individual satisfaction of each user in the group is calculated iteratively, for the candidate set, when adding a new track to the playlist, given the sequence of previously built recommendations, and, for each track, the sum of all the differences between user satisfactions is calculated, recommending the next track by selecting the lowest of these values. This means that each track is selected sequentially based on the satisfaction agreement of the group and the previously recommended tracks.

3. *Balancing with Decay*, which is similar to the previous, but the satisfaction score is calculated differently, assuming that user satisfaction is higher for tracks played more recently, but also that the tracks with the highest predicted rating for each user have a higher weight the largest their predicted rating for the user.

Based on the system they implemented to test their hypothesis, they conclude that the *Balancing* techniques can achieve better performance than the *Average*, as well as return results that are comparable to humanly generated playlists, as created by a randomly chosen group member.

Herr et al. [7] present a selection of social psychological concepts for group recommender systems, including group identification, group norms and social roles. The authors describe several items

within each topic, mapping their impact in group recommender systems. Within the group identification topic, they list: interpersonal attraction, self-categorization, and interdependence. Within the group norms, they list: communication rules and attitude formation. And within the social roles, they list: cognitive centrality, individual characteristics, and expertise. This type of study is valuable to understand group dynamics and to better tune group recommender systems to account for real-world group behavior.

3 System Architecture

Juggle Mobile's backend was built in Python, mainly supported on the flask, numpy and h5py libraries. The recommender system's module has been released as open source[2], while the web services module, which represents a more trivial implementation, hasn't been released publicly. The frontend was developed purely using HTML5, CSS3 and JavaScript, strongly supported on SAPO Ink[3] for a responsive design adapted to the desktop, tablets or smartphones.

Our system's main task is to help users discover new artists that might interest them. While the user needs to build a profile to be able to make the most of recommendation algorithms, we aimed at making this task the least painful as possible. We enabled the user to import tastes from Facebook and Last.fm, but also provided random artist biographies as a means of exploration and of rating known artists, in order to improve the user's profile.

3.1 Artist Data Gathering

To be able to do what we just described, we must first obtain a dataset of artist biographies along with play counts for these artists so that we can build our recommender engine. We started by building a Last.fm crawler to gather user profiles containing a series of records with at least three fields: user, artist and play count. Given we have released this as open source software[4], we urge you

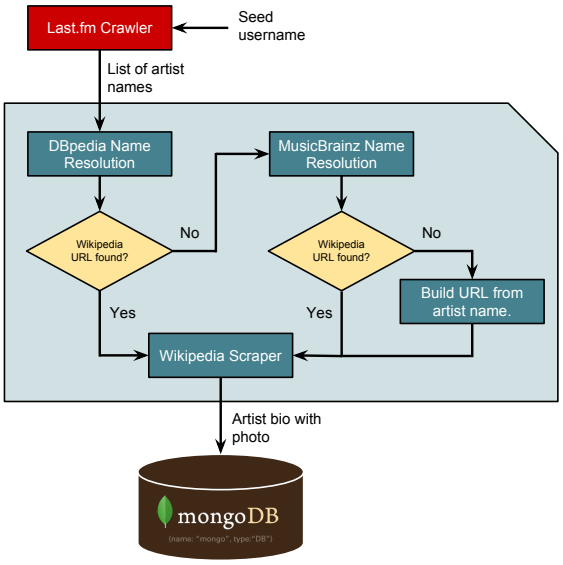


Figure 1: Subsystem for gathering listened to artists and obtaining a MongoDB collection of artist biographies, with a photo when available.

to download it, try it and improve it, as it can crawl user profiles including not only artist play counts, but also the social graph of these users. The crawling algorithm does a breadth-first search on the Last.fm social graph. It starts from a seed user, gathering data about this user and all of its neighbors, and repeating the process by randomly selecting a neighbor as the new seed user. A seed user is never crawled twice unless the crawler is restarted — a restart process was done a few times in order to build a larger user profile, as the crawled playback data was based on the recently listened tracks.

While the Last.fm crawler is a fundamental piece to help us build our recommender system, it also allows us to obtain a list of relevant artists whose biographies we must gather and store in our database. Figure 1 describes our Wikipedia scraping tool[5] and the process we used to gather artist information. As you can see, we started by getting a list of unique artist names that were crawled from Last.fm, which we feed to a name resolver that converts an artist name into its Wikipedia article title. We use a main resolver based on DBpedia

[2] https://github.com/jldevezas/phd/tree/master/python-jldlab

[3] http://ink.sapo.pt

[4] https://github.com/jldevezas/phd/tree/master/lastfm-crawler

[5] https://github.com/jldevezas/phd/tree/master/wikipedia-scraper

that tries to find resources belonging to any subclass of `http://schema.org/MusicGroup` with a `foaf:name` matching the artist name. As a fallback, we try to search MusicBrainz for the artist name, run a string similarity heuristic on the top result to ensure it's within a given threshold and use a CSS selector to obtain any available Wikipedia link. If both name resolvers fail, we directly download the Wikipedia page corresponding to the artist name after replacing spaces with underscore characters. We use some simple constrains to skip ambiguous and other irrelevant pages, in which case we skip the artist completely. Using this simple method, we are able to obtain over 45% of the artist biographies — the remaining 55% include artists that do not have a Wikipedia entry and ignored redirect pages.

We then randomly present these biographies to the user, so that he/she can rate known artists using a *Love*, *Like*, *Neutral*, *Dislike* and *Hate* scale. We will describe this in more depth in Section 4 together with the interface details.

3.2 Recommender System

Juggle Mobile's recommender system consists of two main modules: the training method, and the rating prediction and recommendation methods. The whole system is disk supported, using the HDF5 format [6] to store the matrices and vectors that represent the latent factors model. HDF5, as the name implies, is a Hierarchical Data Format designed to store and efficiently retrieve large amounts of numerical data. Figure 2 illustrates the SVD-based training (2a) and recommendation (2b) methodologies showing what data is stored in disk as HDF5.

As we can see in Figure 2a, the first step of the process is to create a CSV in the format `<user>,<item>,<rating>` which, in our case translates to `<user>,<artist>,<play count>`. The first step of the training is to read the CSV file, creating a user-artist matrix and normalizing the play count values for each user vector using a 0 to 1 scale, thus making users comparable independently of the total number of tracks they have listened to. Having this matrix stored in disk using the HDF5 format, the next step was to factorize the matrix using SVD — for this we used the pymf package[6], which directly supports HDF5 through h5py. This resulted in three stored matrices: a user-features matrix U, an item-features matrix V and matrix Σ which represents the weight of each latent factor/feature in the model. While we store each matrix individually, were we to require only two matrices, U_b and V_b, out of the factorization, similarly to the Non-negative Matrix Factorization (NMF), we would calculate $U_b = U\Sigma^{-1}$ and $V_b = \Sigma^{-1}V^*$, that is, include the square root of the features weight matrix, so that when the two matrices are multiplied each contributes with "half" of the features weight matrix to the product, thus creating an equivalence between $U\Sigma V^* \Leftrightarrow U_b V_b$. This might be useful to avoid an extra calculations during the ratings prediction phase, however we opted to keep all data separate for future experiments.

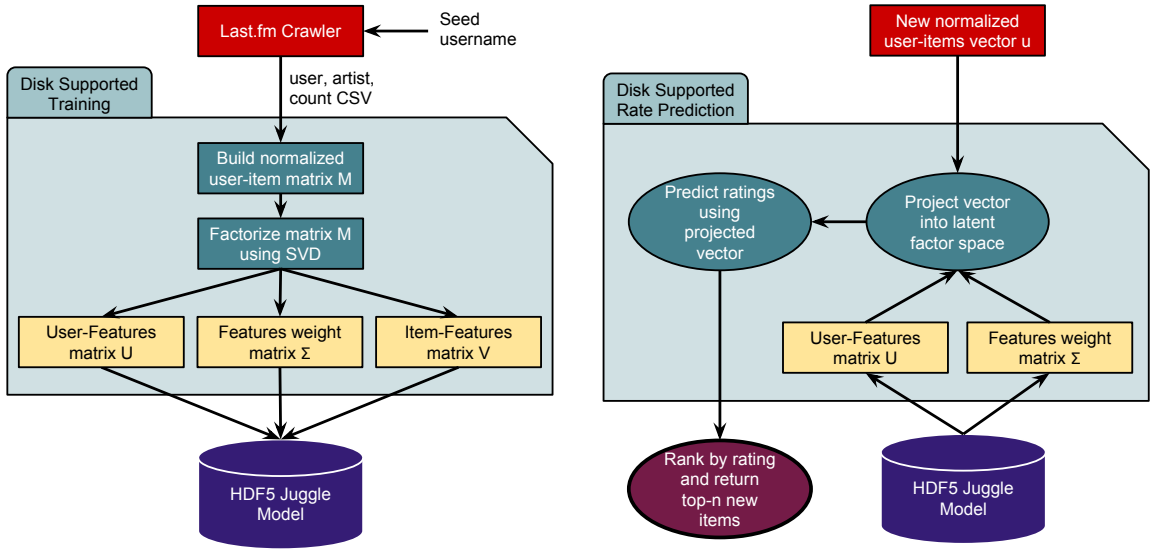
In Figure 2b, we illustrate the recommendation process, starting from a normalized user-artists vector u representing the tastes of a new user. We take this vector and calculate its projection into the latent factor space using the formula $\hat{u} = uV\Sigma^{-1}$. We take this projection and do rating prediction on the projected user-artists vector using the formula $r = \hat{u}\Sigma V^*$. We then rank the artists by the predicted rating, remove any artists the user already listened to and return the top-n artists as our final recommendations.

While this methodology has been used frequently [9], we believe the summarized explanation we provide can help the novice recommender systems scientist grasp how latent factor models based on matrix factorization work, without having to repeat the process of gathering all the scattered information again, which encompasses knowledge from information retrieval, as well as more recent content by authors such as the winners of the Netflix Prize [9].

3.2.1 Training Data

We trained our recommender system using data for 955 users from the same connected social graph. Each user contained the play counts for $3,988$ artists. Thus, our user-items matrix M had a $955 \times 3,988$ dimension. We filled missing values by the imputation of a zero, which accurately represents our use case where, for each user and artist,

[6] https://code.google.com/p/pymf/



(a) Disk supported model training using numpy and h5py.

(b) Disk supported rating prediction and top-n recommendations.

Figure 2: A simple recommender system based on a latent factor model trained with Singular Value Decomposition (SVD) matrix factorization.

it is natural to use a play count of zero for missing (never played) artists. This is not always the case as, for instance, using a zero to replace missing ratings in a five-star scale would not accurately represent the data; the zero would be interpreted as a rather low rating instead of a missing value.

Each user had information for at least 1 artist, having on average 6.521 non-zero ratings and a maximum of 118 ratings. Each item had at least 1 rating from a user, having on average 1.562 user ratings and a maximum of 37 user ratings (the most popular artist). Unnormalized ratings, corresponding to play counts, varied between 1 and 26, having an average value of 2.178.

3.2.2 Recommendation to Groups

Besides the individual recommendations module that we just described, Juggle Mobile also provides an option to obtain recommendations for a group of people at the user's location who have been online using the "Discover With People" feature in the past few hours (we're currently using a three-hour span).

We chose to combine the group members profiles in order to build a group profile that we used to compute our recommendations. Our first experiments were done with the average aggregation technique, which consists of calculating the average vector for all the group members. Having profiles with a large number of rated items, as well as profiles with only a few rated artists, we rapidly noticed that the recommendation results were highly biased towards the user with the highest number of ratings. Recommending to groups makes little sense if the results do not represent all group members equally, thus we decided to use a weighted average instead.

Balancing Rating Aggregation In order to create a balanced group profile, which gives everybody a chance of getting enjoyable recommendations, we created a weights vector w with lower values for users with a large number of rated items and high values for users with a small number of rated items, in an attempt to generate a fairer group profile. We call this rating aggregation technique *Fair Average*. Fair Average is calculated by first generating vector w and then calculating the weighted average, by w_u, of the user ratings u_i. Vector w is computed by counting the non-zero ratings of each

member profile and normalizing these values by dividing by the maximum number of non-zero ratings — corresponding to the user with the largest profile. This guarantees that the values are within a $[0, 1]$ scale. We then take these values and subtract them from 1, which gives us a larger weight for smaller profiles and vice versa. However, to avoid giving a 100% weight to the smallest profiles and a 0% weight to the largest profile — which would be equally unfair, as this user would be ignored in the group recommendations —, we scale these values to a $[\alpha, \beta]$ interval, where $\alpha > 0$ and $\beta < 1$ (in our system we used $[\alpha = 0.3, \beta = 0.7]$). Finally, we multiply each individual weight by the ratings in its corresponding member profile, obtaining a group profile vector. Recommendations to the group are then computed using the same process we described for individual users, except they are based on the group vector.

Next, we provide an example of the *Fair Average* rating aggregation technique. Take a group of three people, u_1, u_2 and u_3 and four different items, i_1, i_2, i_3 and i_4, as depicted in Table 1a. For each user u, we show the rating (in this case the play count) of each item i. In Table 1b, we show the weights w_u of each user's profile in the final group profile. Let's assume the following scaling function:

$$f(v, \alpha, \beta) = \frac{(\beta - \alpha)(v_i - v_{min})}{(v_{max} - v_{min})} + \alpha \qquad (2)$$

that takes a vector v and scales each of its elements to fit the interval $[\alpha, \beta]$. We calculate the unscaled weight vector as follows: $v = [1 - \frac{1}{3}, 1 - \frac{2}{3}, 1 - \frac{3}{3}]$. Each entry v_i, for each user, is given by:

$$1 - \frac{UserProfileSize}{MaxUserProfileSize} \qquad (3)$$

where $UserProfileSize$ is given by the number of non-zero values and $MaxUserProfileSize$ is the size of the largest user profile. We then obtain w by scaling v: $w = f(v, \alpha = 0.3, \beta = 0.7)$. Resulting values of w are shown in Table 1c.

The average vector in Table 1a represents the group profile according to the traditional average rating aggregation technique, while the average vector in Table 1c represents the group profile according to the fair average rating aggregation technique. When comparing the group profile vectors given by the *Fair Average* and given by the *Average*, we can see two things. The first is that the fair average approximates user vectors by diminishing the range. The second is that, while it maintains high values for highly rated items (e.g. i_1 is the highest rated group item in both methods), it also gives less relevance to items that effectively do not represent the group strongly (e.g. i_3 was only rated by u_3, so its value was lowered). Our intuition for this weighting scheme in the context of music recommendation was that the users have a tendency to be biased towards a small set of musical genres. For a particular group containing a user with an extensive music library, a strong bias towards the favorite genres would be introduced, and other group members with possibly different tastes would not be covered by the recommendations (manual tests with both rating aggregation methodologies supported this intuition).

Table 1: Comparison between original and weighted user-item vectors.

(a) Original normalized user vectors.

	i_1	i_2	i_3	i_3
u_1	1.00	0.00	0.00	0.00
u_2	0.00	0.50	0.00	1.00
u_3	0.50	0.00	1.00	0.15
Avg.	0.50	0.17	0.33	0.38

(b) Weight vector (one weight per user vector).

	w_{u_1}	w_{u_2}	w_{u_3}
w	0.70	0.50	0.30

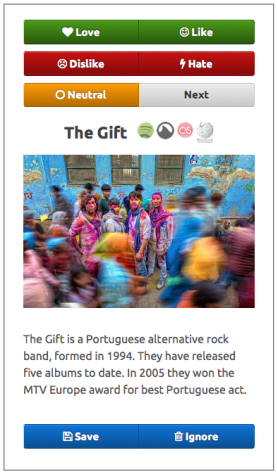
(c) Weighted user vectors.

	i_1	i_2	i_3	i_3
u_1	0.70	0.00	0.00	0.00
u_2	0.00	0.75	0.00	0.50
u_3	0.15	0.00	0.30	0.05
Avg.	0.43	0.25	0.01	0.18

4 User Interface

In this section, we will describe the user interface and the interaction flow of our responsive web application — designed with SAPO Ink[7] and compat-

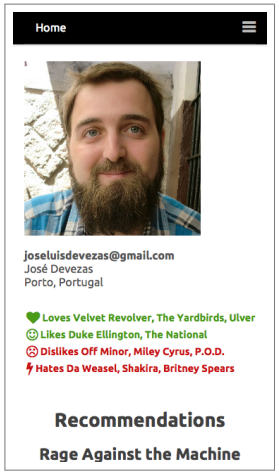
[7] http://ink.sapo.pt



(a) A 5-option rating of "The Gift". User can skip using "Next", "Save" to view later, and "Ignore" to never show the artist again.

(b) Most loved, liked, neutral, disliked, hated, saved and ignored artists.

Figure 3: Artist rating and statistics.



(a) Profile showing some of the loved, liked, disliked and hated artists, and individual recommendations.

(b) Paginated list of saved artists.

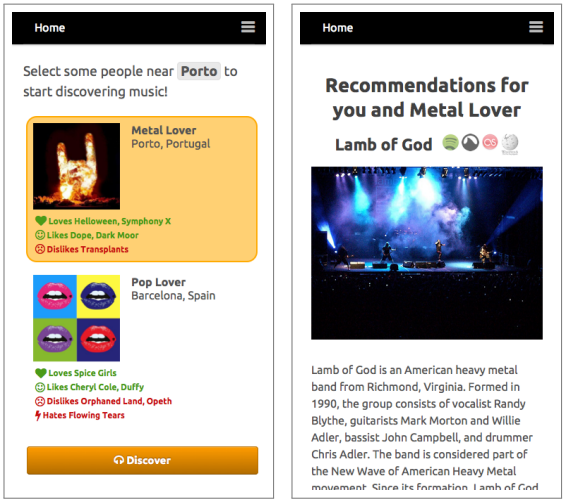
Figure 4: User's individual account data.

ible with desktop, tablet and smartphone browsers.

After registering an account, the user can optionally connect to Facebook and/or Last.fm to periodically import his/her liked or top listened artists to the Juggle Mobile database. This will enable the user to instantly obtain some recommendations. However, Juggle Mobile provides a "Rate Artists" feature (Figure 3a), which randomly presents an artist's biography giving the user the option to *Love*, *Like*, *Dislike*, *Hate* or set an artist as *Neutral*. Positive actions are colored in green, while negative actions are colored in red, and neutral is colored in yellow. The user also has the option to skip using *Next*, whenever he/she doesn't know the artist or is too indecisive. Artists can always be shown again, unless the user marks them as *Ignore* in which case they will be hidden from "Rate Artists" and from recommendation results. The "Rate Artists" feature can also be used as an exploration feature, enabling the user to randomly discover new artists and *Save* them in his/her account for later consultation. Each listed artist includes a quick search link for popular services such as Spotify, Grooveshark, Last.fm and Wikipedia, enabling the user to listen to or find more information about an artist of interest. Based on all the qualitative ratings available in the system, Juggle Mobile also provides a simply statistics module showing the top-3 artists in each category. This is illustrated in Figure 3b, where we show the top most loved artist, Dave Grohl.

In Figure 4a, we can see that each user has a profile showing individual information, as well as a small sample of positively and negatively rated artists, which will be publicly shown to other users to reflect individual profiles. Within the user's home, we also include the best artist recommendations, given the currently available ratings for the user. While this should be cached, it is currently computed on-the-fly as the user visits the profile. The user can also access his/her "Saved" artists (Figure 4b) which are paginated and shown as groups of three per page, in columns for desktop resolutions and vertically for mobile devices.

Finally, in Figure 5, we show the "Discover With People" feature. As seen in Figure 5a, after acquiring the HTML5 Geolocation from the device and doing a reverse geocoding lookup to obtain the location name, Juggle Mobile shows the nearby active users in the last hours. Then, based on the positive and negative ratings of each nearby person, the user can select several profiles to combine



(a) List nearby people and select some of them based on their tastes.

(b) Show combined recommendations for user and selected people.

Figure 5: Artist recommendations for location-based group.

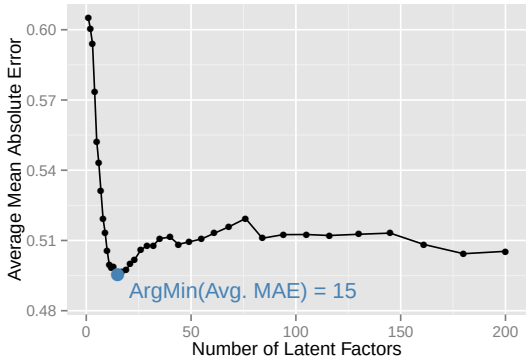


Figure 6: Average value for the Mean Absolute Error of the test sets from 10-fold cross-validation, for different numbers of latent factors. Figure shows that using 15 latent factors result in the best recommendations.

with his/her own. Recommendations will be computed as previously described in Section 3.2.2 and results will be shown for the group as depicted in Figure 5b.

5 Evaluation

In this section, we validate our system and compute the number of latent factors that result in the best recommendations. We also assess the usability of the system, as well as the human perception of the quality of the results, correlating user profile size with the scores given to each of the recommendation questions.

5.1 Model Validation

We used 10-fold cross-validation to determine the quality of our recommender as well as the ideal number of latent factors that minimize the mean absolute error. We used the same dataset from the training phase for validation. We did this by dividing it into 10 random subsets of users. We then used nine of the subsets to train the recommender and the remaining subset to test the recommender (after removing a fraction of know ratings). We compared the predicted ratings with the original ratings using the mean absolute error. We repeated this process to use each of the 10 subsets as test sets. Figure 6 shows the mean absolute error, averaged over the 10-folds, for increasing numbers of latent factors (the left and right singular vectors in U and V matrices). Given that the error tends to stabilize as the number of factors increase, we did a denser sampling for lower values, relaxing it as the number of factores increased.

In the figure, we can see that the number of latent factors that minimizes the average mean absolute error is 15. While we only show the error for the first 200 factors, our data contains 939 factors, which increases the complexity of the recommendation algorithm in space and time. By properly selecting the ideal number of factors, in this case 15, will not only decreases complexity, but also improve the quality of the recommendations. The average mean absolute error when using 15 latent factors is 0.4954. While this value is quite high, since ratings were normalized in a $[0, 1]$ scale, a manual inspection shows that recommendations are in fact quite accurate for most user profiles. In order to support this intuition, we added a small set of questions to the usability survey we will present next, that are meant to obtain human feedback regarding the provided individual recommendations.

Table 2: Summary of SUS scores obtained from survey.

Iteration	Min.	1st Qu.	Median	Mean	Std. Dev.	3rd Qu.	Max.
First	40.00	43.75	67.50	61.79	19.24	76.25	85.00
Second	47.50	62.50	77.50	73.89	16.68	90.00	95.00

5.2 Human Feedback

In order to better understand the perception of the users towards Juggle Mobile, we have elaborated a survey starting with a small guide that covered account creation as well as the core tasks of the system (rating artists, importing data from music social networks, visiting recommendations and the overall features). The first part of the survey was concerned with usability. For this, we used a set of ten standard questions based on a Likert scale [10], called the System Usability Scale (SUS) [2, 16]. We clearly named this section of the survey as "Usability". In order to also obtain some feedback on the recommendations, we also prepared and added a set of four custom questions aimed at reflecting the worst and the best of our recommender system. We clearly named this section of the survey as "Artist Recommendations Quality".

Instead of massively deploying the survey, we directly asked people to answer the questions, trying to ensure a minimal investment on the survey. Repliers included colleagues with a similar background, web designers and developers, human-computer interaction experts, and music lovers. In total, there were 16 repliers in a one-month timespan, 4 female and 12 male, with ages ranging from 23 to 38, although users below 30 corresponded to 81.25% of the sample. Only one user tested the system using a smartphone, no users tested it using a tablet, and every other user tested the system using the desktop. We speculate this has to do with the fact that it is currently unnatural (or the least less comfortable) to multitask in mobile and even more to use a surveying system that is not well adapted to these devices (very small radio buttons). In fact, the only user that tested the system using a smartphone still replied in a desktop.

5.2.1 Usability

We distinguish and compare two different iterations of our system: the first was an initial stable version, and the second was an improved version based on the comments of the first repliers. Feedback was left in a large text area at the end of the survey. Surprisingly, most repliers left long and useful comments identifying bugs, sharing concerns and proposing new features. Specifically focusing on some of the most critical bugs, we decided to re-iterate and launch some fixes, as well as the most desired features. In this section, we compare the results of both iterations, showing that the test was in fact useful and enabled us to improve the overall system's usability.

The first iteration included 7 repliers, all male, with an age ranging from 23 to 38 similarly to the global sample. The second iteration included the remaining 9 repliers, 4 female and 5 male, with ages ranging from 23 to 37, also not unlike the global sample. As you can see in Table 2, there was a 19.58%[8] improvement of the average SUS score, from the first to the second iteration. The second iteration also shows a stronger agreement between users, with a 15.35%[8] lower standard deviation value. Through usability evaluation, we were able to go from a below average SUS score (less than 68) to an above average SUS score (more than 68). Specifically, we went from a grade D to a grade C. While the goal is to get usability to a grade A, since at that point users are more likely to recommend the product to their friends, given this is a demonstrable prototype, we were quite pleased with the improvements.

5.2.2 Recommendations

At the end of the usability survey, we included a custom set of questions to assess the perceived quality of the recommendations by the users:

Q1 How good did you consider the recommended artists, based on the information you've given the system? (1 to 10)

[8] We used relative change to calculate this percentage.

Q2 How many artists did you consider to be completely misplaced in the recommendations? (max. 30)

Q3 How many artists did you consider to be great recommendations? (max. 30)

Q4 How many of the artists that were unknown to you were able to capture your attention enough to listen to later? (max. 30)

With these questions, we expected to account for the overall quality (**Q1**), the worst (**Q2**), the best (**Q3**) and the discoveries (**Q4**). The user distributions of the scores for each question showed an inconsistency in overall quality, with a tendency for lower values; the number of completely misplaced recommendations varied between 0 and 20, with the user distributions of the scores linearly decreasing from lower to higher values; the number of great recommendations varied between 0 and 10, with the user distributions of the scores also linearly decreasing from lower to higher values, but showing a stronger tendency for lower values; the number of unknown artists that captured the user's attention varied between 0 and 6, without showing a strict pattern.

To conclude, there were few completely misplaced recommendations, but also few great, or unknown and captivating recommendations. According to these results, our system didn't behave badly, but also didn't behave as good as expected. We were specially surprised by the fact that a few users found around 20 completely misplaced recommendations. However, we found two possible explanations for this that led to an inaccurate evaluation of our system. First, at the beginning of the survey, each user was asked to rate at least 10 artists before answering the questions. Some of the users, however, didn't rate a single artists or even imported artists from Facebook or Last.fm, which led them to evaluate the recommendations for a profile filled with zeros, since we didn't constrained the recommendations to only be displayed after a certain profile size was achieved. Secondly, from the first iteration to the second, we noticed that the virtual machine where we had the demo running wasn't correctly handling concurrency. Thus, we added caching to the recommendations list. Given some users didn't follow the steps we provided and visited this list before rating any artists or after doing this but before importing Facebook or Last.fm artists, the recommendations were cached for 5 minutes and displayed wrong results that led the users to reply with lower scores.

Table 3: Correlation between all the questions and the user profile size.

	Q1	**Q2**	**Q3**	**Q4**	**P**
Q1	1.0000	-0.4535	0.6051	0.3488	0.0343
Q2	-0.4535	1.0000	-0.2992	-0.3064	**-0.2467**
Q3	0.6051	-0.2992	1.0000	0.0565	0.0693
Q4	0.3488	-0.3064	0.0565	1.0000	**0.2843**
P	0.0343	**-0.2467**	0.0693	**0.2843**	1.0000

To further support our first hypothesis, we present in Table 3 the correlation matrix for the score variables of each question in the recommendations survey as well as the profile size, depicted by the **P** letter. As we can see, there is a slight negative correlation between the number of completely misplaced recommendations (**Q2**) and the profile size (**P**). On the other hand, with a similar intensity, there is a slight positive correlation between the number of unknown artists that were able to capture the user's attention (**Q4**) and the profile size (**P**). While the absolute values of these correlations are only around 30, this still leads to believe that larger profiles should result in better recommendations.

We also performed some aggregations on the data, analyzing for instance the overall quality perceived by the surveyed users, depending on age and gender. We noticed a peak for people of ages 26 and 27, independent of gender, which might lead us to believe that our target audience would be young adults. Then we analyzed the same overall quality variable, but now depending on the profile size. Although we would need more data to obtain a stronger conclusion, we noticed a slight evidence of medium sized profiles resulting in larger quality scores. The score was low for small profiles, peaked for medium profiles and decreased again for large profiles. We hypothesize that this happens because users looking for variety fed the system with a large but very narrow set of tastes, creating a biased profile (e.g. towards a specific musical genre) that resulted in a recommendations list with a very low deviation from the user's main tastes.

6 Summary and Conclusions

We have presented the Juggle Mobile responsive web application to recommend music to individuals and groups of nearby people. Our contributions focused on showing the complete process of building a recommender system, from data gathering to system engineering, including the detailed construction of our recommender engine and the evaluation of the system based on validation and human feedback. Besides this contribution from the informations system standpoint, we also present the simple, but to our knowledge, novel idea of creating a group profile based on a balanced rating aggregation methodology that priorizes users with smaller profiles in the process of recommendation to counteract the strong contribution of users with large profiles. This avoids an unfair recommendation list that is comparable to the individual recommendations of the user with the largest profile. The idea is that a group recommendation should always equally consider the tastes of their members to make sense in its inherent context.

We also showed how to evaluate a recommender system by first validating it and then using the system usability scale to assess usability together with a small set of questions to assess the quality of the actual recommendations. This brought several small concerns to our attention, regarding for instance the importance of immediately reflecting changes to the user profile in the recommendations, or even the generic importance of user feedback during the development process of a recommender system.

6.1 Future Work

As future work, we would like to improve the artist rating module to evenly provide random artists over different genres and popularities, thus avoiding a situation where the user is constantly getting an unpopular (and probably unknown) artist to rate, or a sequence of artists in a same genre that the user doesn't listen to and thus will not rate. We believe that by controlling some of this randomness we will provide a more engaging experience to the user, which in turn will result in a richer profile and better recommendations.

Regarding the sample size of our survey, we would like to extend this to a larger number of participants, after introducing some constrains in our system to avoid users from evaluating the recommender system without having provided enough information about their tastes beforehand.

We would also like to work on an evaluation framework capable of assessing the quality of recommendations to groups, in a scenario where a group corresponds to a cohesive community (e.g. close friends, work colleagues, Black Sabbath fans, etc.).

Acknowledgments

This work has been financed by Laboratório SAPO/U.Porto under a research grant for the Juggle project.

References

[1] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4):573–595, 1995.

[2] J Brooke. SUS: A 'Quick and Dirty' Usability Scale. In *Usability evaluation in industry*, pages 189–195. CRC Press, 1996.

[3] Columbia Department of Statistics. Week 7: hunch.com, Recommendation Engines, SVD, Alternating Least Squares, Convexity, Filter Bubbles. http://columbiadatascience.com/2012/10/18/week-7-hunch-com-recommendation-engines-svd-alternating-least-squares-convexity-filter-bubbles/, 2013. [Accessed: Nov. 11, 2013].

[4] Andrew Crossen, Jay Budzik, and KJ Hammond. Flytrap: Intelligent Group Music Recommendation. In *Proceedings of the 7th international conference on Intelligent user interfaces (IUI 2002)*, pages 184–185, 2002.

[5] Oskar Van Deventer, Joost de Wit, Jeroen Vanattenhoven, and Mark Guelbahar. Group recommendation in an Hybrid Broadcast Broadband Television context. In *In Proceedings of the Workshop on Group Recommender Systems: Concepts, Technology, Evaluation (GroupRS), at the 21st Conference on*

User Modeling, Adaptation and Personalization (UMAP 2013), 2013.

[6] Mike Folk, Albert Cheng, and Kim Yates. HDF5: A file format and I/O library for high performance computing applications. In *Proceedings of Supercomputing*, volume 99, 1999.

[7] Sascha Herr, Andreas Rösch, Christoph Beckmann, and Tom Gross. Informing the design of group recommender systems. *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (CHI 2012)*, page 2507, 2012.

[8] Anthony Jameson and Barry Smyth. Recommendation to groups. In *The Adaptive Web*, pages 596–627. Springer Berlin Heidelberg, 2007.

[9] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[10] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.

[11] Judith Masthoff. Group Modeling: Selecting a Sequence of Television Items to Suit a Group of Viewers. *User Modeling and User-Adapted Interaction*, 14(1):37–85, February 2004.

[12] M O'Connor, Dan Cosley, JA Konstan, and John Riedl. PolyLens: A recommender system for groups of users. In *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work (ECSCW 2001)*, pages 16–20, Bonn, Germany, 2002.

[13] Auste Piliponyte, Francesco Ricci, and Julian Koschwitz. Sequential Music Recommendations for Groups by Balancing User Satisfaction. In *In Proceedings of the Workshop on Group Recommender Systems: Concepts, Technology, Evaluation (GroupRS), at the 21st Conference on User Modeling, Adaptation and Personalization (UMAP 2013)*, 2013.

[14] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor, editors. *Recommender Systems Handbook*. Springer US, Boston, MA, 2011.

[15] Badrul Sarwar, G Karypis, J Konstan, and J Riedl. Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. In *Proceedings of the 5th International Conference on Computer and Information Science*, 2002.

[16] Jeff Sauro. Measuring Usability with the System Usability Scale (SUS). http://www.measuringusability.com/sus.php, 2011.

[17] Barry Smyth, Evelyn Balfe, Jill Freyne, Peter Briggs, Maurice Coyle, and Oisin Boydell. Exploiting Query Repetition and Regularity in an Adaptive Community-Based Web Search Engine. *User Modeling and User-Adapted Interaction*, 14(5):383–423, April 2005.

[18] Yu Zhiwen, Zhou Xingshe, and Z Daqing. An adaptive in-vehicle multimedia recommender for group users. In *2005 IEEE 61st Vehicular Technology Conference (VTC 2005)*, pages 2800–2804, 2005.