Collaborative Filtering for Mobile Application Recommendation with Implicit Feedback

Beatriz Paula
Caixa Mágica Software
Instituto Superior Técnico
Lisbon, Portugal
beatriz.paula@caixamagica.pt

João Coelho
Caixa Mágica Software
Instituto Superior Técnico
Lisbon, Portugal
joao.coelho@caixamagica.pt

Diogo Mano
Caixa Mágica Software
Faculdade de Ciências
Lisbon, Portugal
diogo.mano@caixamagica.pt

Carlos Coutinho
Caixa Mágica Software
Iscte - Instituto Universitário de Lisboa
Information Sciences, Technologies and Architecture
Research Center (ISTAR-IUL)
Lisbon, Portugal
carlos.coutinho@caixamagica.pt

João Oliveira
Iscte - Instituto Universitário de Lisboa
Information Sciences, Technologies and Architecture
Research Center (ISTAR-IUL)
Instituto de Telecomunicações (IT)
Lisbon, Portugal
joao.p.oliveira@iscte-iul.pt

Ricardo Ribeiro
Iscte - Instituto Universitário de Lisboa and INESC-ID
Lisbon, Portugal
ricardo.ribeiro@iscte-iul.pt

Fernando Batista
Iscte - Instituto Universitário de Lisboa and INESC-ID
Lisbon, Portugal
https://orcid.org/0000-0002-1075-0177

***Abstract*—This paper introduces a novel dataset regarding the installation of mobile applications in users devices, and benchmarks multiple well-established collaborative filtering techniques, leveraging on the user implicit feedback extracted from the data. Our experiments use 3 snapshots provided by Aptoide, one of the leading mobile application stores. These snapshots provide information about the installed applications for more than 4 million users in total. Such data allow us to infer the users activity over time, which corresponds to an implicit measure of interest in a certain application, as we consider that installs reflect a positive user opinion on an app, and, inversely, uninstalls reflect a negative user opinion. Since recommendation systems usually use explicit rating data, we have filtered and transformed the existing data into binary ratings. We have trained several recommendation models, using the Surprise Python scikit, comparing baseline algorithms to neighborhood-based and matrix factorization methods. Our evaluation shows that SVD-based and KNN-based methods achieve good performance scores while being computationally efficient, suggesting that they are suitable for recommendation in this novel dataset.**

***Index Terms*—Recommender System, Implicit Feedback, Collaborative Filtering**

I. Introduction

Recommendation systems are extremely valuable tools for companies. With personalized suggestions, these systems allow users to overview a small and relevant number of suggestions when too many items are available and exploring all possible options is difficult, hence being beneficial tools to businesses in the areas of e-commerce [1] and entertainment [2]. Leading companies in such areas, including Amazon [3], Youtube [4] and Netflix [5], resort to recommendation systems.

There are several approaches to recommendation systems, typically based on two distinct strategies: collaborative filtering, and content-based filtering. Considering an user-centered formulation, the former recommends items to a user based on other users with similar preferences, and is the approach explored in this work. The latter considers features of the items to be recommended (e.g., for musical items, the artist, genre and language), recommending similar items to those a given user liked in the past.

One of the biggest challenges of collaborative filtering is the lack of explicit ratings data. Specifically, in the case of mobile application stores, users have contact with much more applications than the ones they leave a review on. This limits the recommendation models to an extremely sparse training data. Therefore it is necessary to infer users' likes and dislikes from their activity with implicit feedback.

The goal of this work is to benchmark a novel dataset provided by Aptoide,[1] a mobile application store with over 300 million users and 1 million apps, containing three daily snapshots of the installed applications for over 4 million users in total. We made this dataset publicly available[2] for reproducibility and future research. This is a unique (to our knowledge) dataset that allows to profile mobile application

[1]https://aptoide.com
[2]https://apprecommender.caixamagica.pt/resources/

preferences based on their user installation patterns, whereas most datasets containing users mobile app feedback do not uniquely identify each user, and the ones that we have found of a much smaller scale.

First, we thoroughly characterized this dataset, and described a rating extraction technique based on installs/uninstalls. Then, several collaborative filtering models, implemented in the Surprise Python library, were trained and evaluated on the inferred users' ratings from the dataset. We aim to answer the following research questions with reference to the mobile application domain:

RQ1. Can install and uninstall history be used as a representation of users preferences?

RQ2. What is the performance of several Collaborative Filtering models when using Implicit Feedback?

This paper is organized as follows: Section II presents the related work on Recommendation Systems, more specifically on Collaborative Filtering approaches. Section III presents the format of the dataset being used. Section IV details the procedure we took to solve our research questions, with a detailed analysis of our dataset and of the Surprise library models used. Section V presents and analyses the results of our models. And, finally, section VI draws conclusions and presents several possible directions for future work.

II. Related Work

Collaborative filtering (CF), first used in [6], is based on the insight that if two users have rated several items similarly, due to the relative stability of people's preferences, they will probably rate other items similarly, as well, in the future. There are two main techniques of collaborative filtering: neighborhood-based models and latent feature models.

Neighborhood-based models are one of the first and most basic approaches of CF, as they are a very direct implementation of its idea. There are two types of nearest neighbours algorithms: user-user and item-item. The user-user approach [7] finds users who have rated the same items similarly, and uses their ratings to recommend new items to the current user. The first step of these algorithms is to chose a similarity function which quantifies the similarity between user tastes based on their ratings, which can be classified as correlation-based similarities, such as the Pearson correlation, the constrained Pearson correlation, Spearman rank correlation and Kendall's τ correlation [8], [9]; or cosine-based similarity, such as vector cosine similarity and adjusted cosine similarity [10]. After computing the similarities, the next step is to define the set of k nearest neighbors, where k is an hyperparameter which studies have found best to be set between 20 to 60 for better performance and lower computational cost [11]. Finally, to predict the rating of user u to item i, we simply take the weighted average of the ratings of item i from the neighbors of user u, where the weight terms correspond to the similarity between user u and its neighbors. Analogously, the item-item approach recommends similar items to the ones the user has previously rated positively, by first finding the k nearest neighbors of item i and, secondly, taking the weighted average of user u ratings on those similar items [12]. This latter approach is still widely-used in modern day systems due to its low computational costs and competitive performance [13]. Since there are usually much more users than items in these applications, items, on average, commonly have much more ratings than the number of ratings per user. This results in stable similarity scores between items, allowing the precomputation of these values and resulting in a drastic speedup during prediction.

Alternatively, latent feature models address the idea that different items can have common features, and users can be interested in particular aspects of each item. These models try to explicitly profile each user, p_u, according to their preferences to these latent features, and profile each item, q_i, in respect to the presence those same features, as vectors of dimension k. The preference of user u towards item i can then be computed as the dot product between these two vectors, which will correspond to a higher value when the user prefers the same features that are relevant in that item. Singular Value Decomposition (SVD) algorithms use the single value decomposition of the ratings matrix R, as the product of three matrices: P and Q^T. The matrix P has $|U| \times k$ dimensions and corresponds to the user-feature preferences and Q is a $k \times |I|$ matrix corresponding to the item-feature relevancy. Algebraically, P and Q are orthogonal and give an approximation to the ratings matrix, and SVD is only defined when R is complete. However, most entrances of the ratings matrix are unknown. In practice, we use singular value decomposition to define an optimization problem, such as in [14], where we learn P and Q to minimize the error between the R matrix and its approximation PQ^T, while ignoring the missing rating values. It is common to use stochastic gradient descent or alternating least square during training [15]. Other common algorithms which use this latent factor technique are very similar to this one, such as NMF and SVD++ [16].

Collaborative filtering models were initially designed and are most suitable for explicit ratings, a high quality feedback where users directly report their opinion on an item by attributing it, for example, a star rating such as Netflix's previous 5 star ratings [15], or their current binary system of like/dislike. However, this kind of data, although accurately reflective of users' opinions, is usually sparse due to users' reluctance to rate every item they have tried. To solve this lack of data, recommendation systems started leveraging implicit feedback [17], which indicates users' opinions based on their activity history (such as purchase history, click history, search patterns), leading to, usually, a binary rating indicating confidence, contrary to explicit ratings which indicate preference [18]. In this latter work, a neighborhood-based model and latent factor model were developed with the implicit feedback derived from the data collected in a 4 week period from a digital television service, where television programs were recommended based on the programs watched by the users. The training dataset was generated by setting a confidence rating, r_{ui}, of 1 if the user u watched more than half of a program i throughout the sampling period, and a rating of 0 otherwise. Their models yielded much better results than the

baseline approach of recommending the most popular results, which is still very powerful. Other works, such as [19], [20], explore different ways to derive implicit feedback. In [19], similarly to what is done in this project, several collaborative filtering models are developed with the Surprise library [21], leveraging data from a video game platform. They converted total playing time per game to a 1 to 5 rating system, instead of a more common binary approach, through the Python *cut()* and *rank()* functions. In [20] an integration of implicit feedback to matrix factorization (MF) framework is explored. They used the popular Movielens-100k and Movielens-1M datasets that use a five-level integer rating system. On top of this explicit data, they derived three different implicit feedbacks and used it to improve the explicit MF model. The first implicit feedback is implicit user relationships which reflect user' similarities based on their preferences; the second is rated records of items which reflect implicit influence among users for the same item, since users can observe previous ratings when deciding their own rating; and, lastly, the third implicit feedback is the positive attitude, which reflects a user feeling on an item, which is positive when a user gives 3 or more points to a movie, and negative otherwise.

This specific application of recommendation systems to mobile application stores is a relevant research topic due to the continuous increase of existing apps. Numerous works take into account specific characteristics of these items, such as versions and access permissions. In [22], a factorization machine model was developed by leveraging the interaction information between different feature views, such as description, permissions and category. Another work [23] takes into account that, with each version, the updates done to an application can significantly change them which can result in the interest of new users, sometimes making previous versions relevant to them. Other works focus on the security risks that come from the applications ability to access user's sensitive information, such as [24] which recognizes that users may have specific privacy preferences for each app category, and [25] that recommends applications based on popularity and security features. Other recent works extract app functionalities from their description and reviews, and leverage them for recommendation. In [26] the authors retrieve hidden item topics from user reviews, and profile each app with the probability of topic distribution as a representation of their latent features. In [27] app functionalities are extracted from their respective description that are then used in a graph-based approach to predict new functionalities the user might be interested and to recommend candidate apps that contained them.

Lastly, our work introduces and analysis a real-world dataset of applications installed on users devices from February to December 2020. A similar analysis of user-app interactions has been made in [28] which focus on the application usage information of over 4100 users and the context of their activity, including time of day and location. Related datasets have also been provided in other works such as in [29], where the two distinct datasets provided contain 1390 and 3691 app reviews classified according to their Software engineering's maintenance task (such as user experience, feature request, bug report and rating) retrieved from several mobile application stores. Similarly, in [30], a much larger dataset is supplied with over 280,000 user reviews of 395 applications as well as some code quality metrics for each application, including number of classes and depth of inheritance tree. Finally, in [31] a context-aware application usage dataset with over 90 thousand entries is provided, containing information regarding 957 users and 4082 apps. This latter dataset was the only one containing a unique user identifying, making it the only one possible to profile users according to their mobile application tastes, similarly to what we accomplish in this work, although at a much smaller scale.

III. Dataset

This section describes the dataset, provided by Aptoide, that has been used in our experiments, including the filtering and transformation process. In order to use collaborative filtering approaches, explicit ratings had to be extracted. As such, the process of converting the implicit information based on user activity into ratings, is also addressed.

A. Preliminary analysis

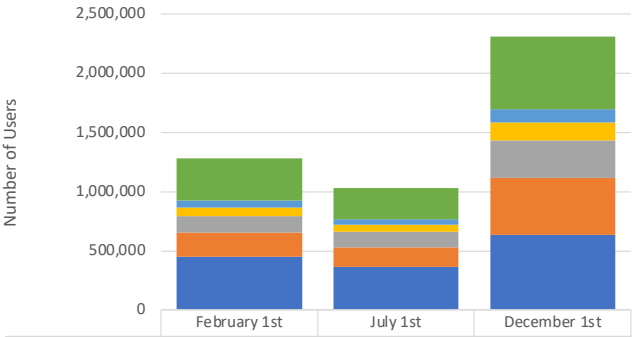
Our data comprise three snapshots collected from distinct moments throughout the year of 2020: the first is from February, 1st, the second is from July, 1st, and the third is from December, 1st. Each one of the snapshots contains a list of users who had showed some activity on that given day, such as installing, updating or uninstalling an application. The dataset contains information about the user activity, and a list of applications currently installed in the user's device. All the three snapshots account for a total of about 4.5 million users, where the first snapshot shows the activity for about 1.3 million users, the second snapshot shows the activity for about 1 million users, and the third snapshot reports the activity for about 2.3 million users.

Figure 1 shows an excerpt of the data that corresponds to a user entry. Each user entry contains an *origin* field, which shows the type of user activity, the *timestamp* of the data retrieval, the *language* set on the device, the *Aptoide user id*, which has been masked to comply with Aptoide's Data Protection Policy, alongside the user *email*, which has been removed. Finally, the *apps* field contains a list of the applications currently installed on the user's device. Each element of this list has an id field for applications from Aptoide, a package, which reflects the application name, an optional *store* field, and sometimes the number of downloads for that application.

While the dataset corresponds to three days of user activity only, it provides interesting insights for the universe of the Aptoide users and available apps. Figure 2 presents the language distributions by the users on each one of the snapshots. The most represented languages in the user devices are: English (32.49%), Brazilian Portuguese (19.0%), Spanish (13.2%), Mexican Spanish (6.4%), and French (4.9%).

```
{'origin': 'APPS_UPDATES',
 'version': 1,
 'meta': {'timestamp': '2020-02-01 12:34:56'},
 'data': {'language': 'pt_BR',
  'user': {'hash': 'AnNHezXqXM4HdLS_LmB_gMoDSj
      ...',
   'email': None,
   'aptoide_uid': '91a7155c3ed5f04b7e6a16720...',
   'stores': [1966380],
   'aptoide': {'package': 'cm.aptoide.pt',
    'vercode': 9682,
    'md5sum': '182a335603b15d3c442dfa1e0...'},
   'user_agent': 'aptoide-9.8.0.0;SM-J600F(
       j6lteub);v8.0.0;armv8l;0x0;id:617eb...;;',
   'country': ''},
  'apps': [
   {'signature': 'BA:14:17:46:D7:04:B9:6E:...',
    'enabled': True,
    'package': 'com.android.calllogbackup',
    'vercode': 26},
   {'signature': '38:91:8A:45:3D:07:19:93:54:F8:
       B1:...',
    'enabled': True,
    'package': 'com.google.android.partnersetup',
    'vercode': 26},
...
   {'signature': 'BA:14:17:46:D7:04:B9:6E:D4:DB
       :...',
    'enabled': True,
    'package': 'com.android.providers.
        userdictionary',
    'vercode': 26}
   ]}}
```

Fig. 1. Example of a user entry, from one of the snapshots.



	February 1st	July 1st	December 1st
Other	355,581	262,999	609,611
French	59,227	47,723	113,562
Spanish (Mexico)	71,679	60,883	151,736
Spanish	141,533	129,669	318,829
Portuguese (Brazil)	203,997	168,985	476,110
English	449,465	363,845	638,719

Fig. 2. Set of Language by users on their devices on each snapshot.

Fig. 3 shows the number of occurrences of users who have a certain number of installed applications for each one of the snapshots. The greatest number of occurrences of the last snapshot are due to the larger number of records, which roughly corresponds to the sum of the first two snapshots. Nonetheless, while in the first two snapshots, most of the user devices (about 70%) include at most 3 applications provided

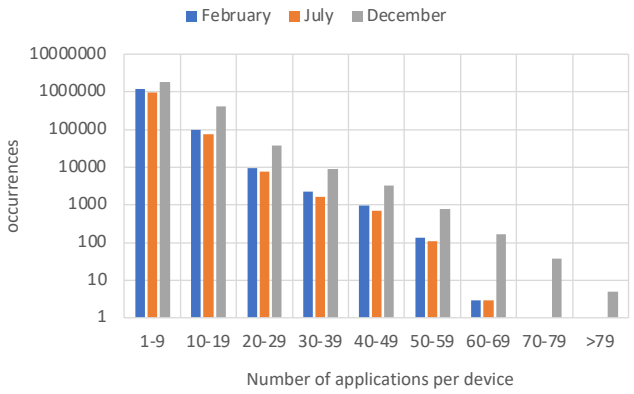


Fig. 3. Number of Apps per user over time.

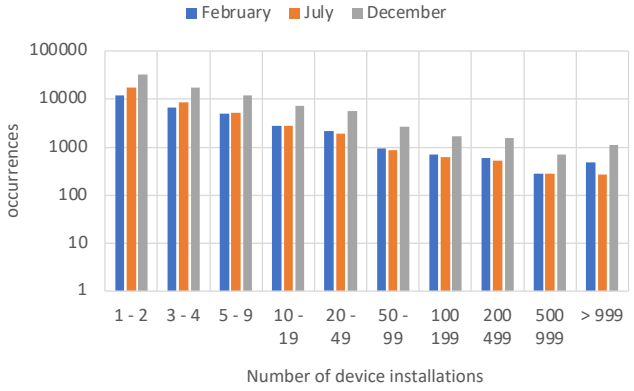


Fig. 4. Number of applications installed on a given number of devices.

by Aptoide, and only about 8% of the user devices contain 10 or more applications, such statistics are quite different for the last snapshot, where only about 32% of the devices include 3 or less applications provided by Aptoide, and about 20% of the user devices contain 10 or more applications. Such results show a significant tendency for the adoption of applications provided by Aptoide.

The three snapshots contain data about 31372, 38700, 83137 distinct applications, respectively, but the vast majority was found in only a few of the user devices. Figure 4 shows the number of occurrences of a given number of device installations, for each one of the snapshots. These results depend from the number of records involved, which is also one of the main reasons for the bigger numbers corresponding to the last snapshot.

Finally, we have analysed the most popular applications. Table I shows the most popular applications for each one of the snapshots, disregarding the Aptoide system applications, together with the proportion of users that have the respective application installed on their device. We have observed an increasing adoption of the Aptoide store over the time, and the appearance of new applications in the second and third snapshots.

TABLE I
MOST POPULAR APPLICATIONS AND RESPECTIVE FREQUENCY.

App	February	July	December
WhatsApp Messenger	7,1%	6,9%	16,1%
Messenger	1,0%	3,7%	14,2%
Facebook	2,5%	7,6%	11,6%
Gboard - the Google Keyboard	5,2%	5,5%	7,7%
Instagram	3,2%	2,9%	11,2%
Gmail	5,1%	5,0%	5,4%
Netflix	2,0%	2,1%	5,8%
Youtube TV	2,6%	3,3%	5,2%
Google Duo	0,9%	1,3%	4,5%
Facebook Lite	2,1%	2,0%	4,3%
Microsoft Word	1,1%	0,9%	3,9%
Discord	0,3%	0,4%	3,7%
Snapchat	1,2%	1,1%	3,1%
VLC for Android	1,6%	1,2%	2,0%
Microsoft PowerPoint	0,9%	1,0%	3,0%

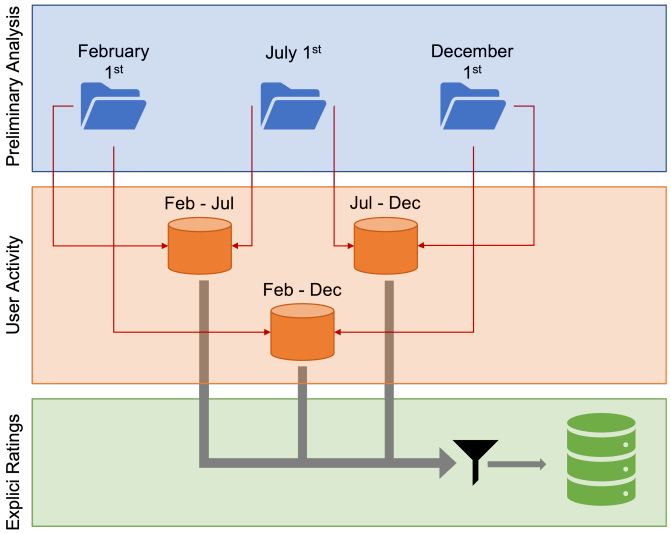


Fig. 5. Analysis and transformation of Aptoide's snapshots.

B. User Activity

Only users that are present in more than one snapshot are relevant for this work, since their activity can be inferred, to a certain degree, based on the differences between the corresponding lists of installed applications. The first and the second snapshots contain about 67,389 common users, the second and third snapshots contain about 65,521 common users, and the first and third snapshots contain about 47,763 common users. It is also interesting to note that the intersection of the three snapshots result in 25,234 common users, which suggest that these snapshots cover only a very small portion of the Aptoide users. Figure 5 illustrates the process of creating the final dataset used for training and evaluating our models. We have started by creating three temporary datasets, containing activity information about the overlapping users between a pair of origin and destination snapshots. In a second

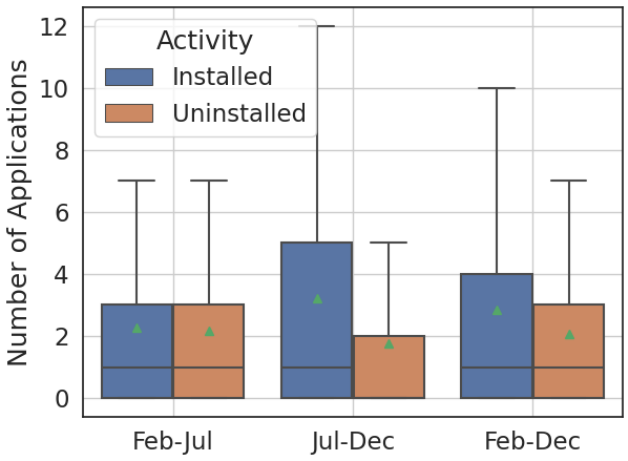


Fig. 6. Distribution of the number of installed and uninstalled applications per user for each time period. The green triangle represents the mean value.

stage, we have merged the three temporary datasets, containing about 140 thousand entries, into a big dataset. Each entry in the dataset contains 3 lists that that can be used to derive the state of both the origin and destination snapshots: i) a list of applications installed in the origin snapshot, ii) list of installed applications, and iii) list of uninstalled applications. Finally, we have discarded all the entries where the list of installed applications was empty, which resulted into a final dataset containing 112,281 relevant entries.

We assessed the users' activity patterns within the three different dates, as depicted in the box plots in Figure 6, in which the outliers were removed. Firstly, between February and July, the majority of users installed and uninstalled no more than 1 Aptoide application, more specifically 62.6% and 65,1% of users, respectively. On average, users installed 2.28 and uninstalled 2.17 applications in this time period. The highest numbers of installed and uninstalled applications from a single user were, respectively, 56 and 50. Between July and December, the majority of the users, more specifically 52.2%, installed 1 Aptoide application, and 56.4% of users did not uninstall any applications. On average, users installed 3.22 and uninstalled 1.77 applications in this time period. The highest number of installed and uninstalled applications from a single user was 64 and 52. Finally, between February and December, the majority of the users installed and uninstalled no more than 1 Aptoide application, more specifically 55.3% and 65.2% of users, respectively. On average, users installed 2.85 and uninstalled 2.06 applications in this time period. The highest numbers of installed and uninstalled applications from a single user were, respectively, 71 and 56.

C. Explicit Ratings dataset

In order to quantify the preferences of the users, we have defined a rating score based on the activity of the users. For a given user, if an application was maintained or installed it was scored as 2, and if the user uninstalled the application it was scored as 1. This process resulted into a set of 3,170,780

scores for a set of 74,850 applications. However, in order to increase the significance of our results, while also reducing the computational complexity of our models, we have filtered out all the applications with less than 100 ratings and removed user entries with less than 20 ratings, which resulted in a final dataset containing 2,102,052 scores, involving 48,863 users and 3,181 different applications.

IV. Recommendation Models

This section presents the methods used to develop the recommendation models, namely Slope One [32], Matrix Factorization-based [33], [34], and Neighborhood-based [12], that were used in the scope of this work. All the recommendation models were built using the well-known Surprise Library [21], commonly used for building and analyzing recommender systems in Python. The reminder of this section overviews each one of these models.

A. Baseline models

To better analyse our work we also leveraged two baseline models based on simple methods. The first one is implemented with the *NormalPredictor* surprise class which generates a random prediction, $\hat{r}_{ui}$ based on a normal distribution $N(\hat{\mu}, \hat{\sigma}^2)$ of the training data, $\mathrm{R}_{\mathrm{train}}$, where $\hat{\sigma}$ and $\hat{\mu}$ are calculated through Maximum Likelihood Estimation, as seen in the following equations:

$$\hat{\mu} = \frac{1}{|\mathrm{R}_{\mathrm{train}}|} \sum_{r_{ui} \in \mathrm{R}_{\mathrm{train}}} r_{ui} \ , \tag{1}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in \mathrm{R}_{\mathrm{train}}} \frac{(r_{ui} - \hat{\mu})^2}{|\mathrm{R}_{\mathrm{train}}|}} \ . \tag{2}$$

Although this algorithm is a good starting point for a baseline, it is simply a random number generator and does not take into account the user and the item bias that better resemble the preferences of each user, which we will consider in Collaborative Filtering.

Our second baseline model is the same baseline estimate used in [35] and takes these biases into consideration, by predicting the user rating for a given application, $\hat{r}_{ui}$, through the Equation 3, where b_u and b_i correspond to the user and application bias. To estimate these values, we minimized the regularized squared error in Equation 4, considering λ as the regularization parameter, and using Alternating Least Squares with the default values of the *BaselineOnly* surprise class.

$$\hat{r}_{ui} = \mu + b_u + b_i \ , \tag{3}$$

$$\sum_{r_{ui} \in \mathrm{R}_{\mathrm{train}}} (r_{ui} - (\mu + b_u + b_i))^2 + \lambda(b_u^2 + b_i^2) \ . \tag{4}$$

B. Slope One

Slope One [32] is a simple, yet accurate, collaborative filtering algorithm. The Surprise Library implementation of the method takes into account the average difference between the ratings of users who have at least one application in common. The predictions can be computed as follows:

$$\hat{r}_{ui} = \mu_u + \frac{1}{\mathrm{R}_i(u)} \sum_{j \in \mathrm{R}_i(u)} \left(\frac{1}{\mathrm{U}_i \cap \mathrm{U}_j} \sum_{u \in \mathrm{U}_i \cap \mathrm{U}_j} (r_{ui} - r_{uj}) \right) \ . \tag{5}$$

In the previous equation, U_i and U_j are the sets of users for which implicit ratings (1 or 2) have been drawn for applications i and j, respectively, and $\mathrm{R}_i(u)$ is the set of items rated by u that were also rated by users that rated item i.

C. Neighborhood based models

Another approach we used was a nearest-neighborhood algorithm to find neighborhoods of similar users, based on their ratings. These neighborhoods have a maximum of k neighbors and to find this similarity between users, the model computes the Mean Square Difference (MSD) between ratings of the same items as seen in Equations 6 and 7, where I_u corresponds to the list of items rated by user u. The addition of 1 in the latter equation is to enforce a non 0 denominator.

$$\mathrm{MSD}(u, v) = \frac{\sum_{i \in \mathrm{I}_u \cap \mathrm{I}_v} (r_{u_i} - r_{vi})^2}{|\mathrm{I}_u \cap \mathrm{I}_v|} \ , \tag{6}$$

$$\mathrm{sim}(u, v) = \frac{1}{1 + \mathrm{MSD}(u, v)} \ . \tag{7}$$

We used the *KNN Basic* class, which uses the KNN algorithm to determine the neighborhoods $\mathrm{N}_i(.)$ and, afterwards, computes the rating predictions as the weighted sum seen in Equation 8 used in [12].

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathrm{N}_i(u)} \mathrm{sim}(u, v) \cdot r_{vi}}{\sum_{v \in \mathrm{N}_i(u)} \mathrm{sim}(u, v)} \tag{8}$$

D. Matrix Factorization

The last class of tested algorithms was Matrix Factorization, where the rating prediction is computed based on Equation 9, where q_i is the corresponding item column of the items' latent factors matrix Q, and p_u is the corresponding user column of the users' latent factors matrix P. To determine these matrices we tried two different algorithms.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \ . \tag{9}$$

The first method we used was Single Value Decomposition (SVD) [36], which was popularized in the Netflix Prize [33]. We leveraged the the Surprise prediction class *SVD*, minimizing the following error function:

$$\sum_{r_{ui} \in \mathrm{R}_{\mathrm{train}}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \ . \tag{10}$$

Stochastic Gradient Descent (SGD) was used to update the variables b_u, b_i, p_u, q_i. Let γ be the learning rate and λ a regularization factor. The updates can be computed as follows:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(r_{ui} - \hat{r}_{ui} - \lambda b_u) \ , \\ b_i &\leftarrow b_i + \gamma(r_{ui} - \hat{r}_{ui} - \lambda b_i) \ , \\ q_i &\leftarrow q_i + \gamma((r_{ui} - \hat{r}_{ui}) \cdot q_i - \lambda q_i) \ , \\ p_u &\leftarrow p_u + \gamma((r_{ui} - \hat{r}_{ui}) \cdot p_u - \lambda p_u) \ . \end{aligned} \tag{11}$$

Another method used was Non-negative Matrix Factorization (NMF) [34]. We used the unbiased version, in which it sets the same prediction as in 9, but with b_u and b_i with null value. This algorithm also uses SGD, but, at each step, the latent factors matrices are updated as follows:

$$p_u \leftarrow p_u \cdot \frac{\sum_{i \in I_u} q_i \cdot r_{ui}}{\sum_{i \in I_u} q_i \cdot \hat{r}_{ui} + \lambda_u |I_u| p_u} \ , \tag{12}$$

$$q_i \leftarrow q_i \cdot \frac{\sum_{u \in U_i} p_u \cdot r_{ui}}{\sum_{u \in U_i} q_i \cdot \hat{r}_{ui} + \lambda_i |U_i| p_u} \ . \tag{13}$$

V. Results

This section presents the achieved evaluation results. Since we introduced a novel dataset, we now show the results for the previously described well-established collaborative filtering techniques on top of our implicit binary ratings. For evaluation metrics, we consider the Precision@k (P@k), the Recall@k (R@k), and the F1-score@k (F1@k), which were computed as follows:

$$\mathrm{P@}k = \frac{1}{|\mathrm{U_T}|} \sum_{i=1}^{|\mathrm{U_T}|} \frac{|\mathrm{Rel}_i \cap \mathrm{Rec}_{k,i}|}{|\mathrm{Rec}_i|} \ , \tag{14}$$

$$\mathrm{R@}k = \frac{1}{|\mathrm{U_T}|} \sum_{i=1}^{|\mathrm{U_T}|} \frac{|\mathrm{Rel}_i \cap \mathrm{Rec}_{k,i}|}{|\mathrm{Rel}_i|} \ , \tag{15}$$

$$\mathrm{F1@}k = \frac{2 \ \mathrm{P@}k \ \mathrm{R@}k}{\mathrm{P@}k + \mathrm{R@}k} \ . \tag{16}$$

In the previous equations, $\mathrm{U_T}$ is the test set of users, $\mathrm{Rec_{k,i}}$ is the set of k recommendations for the i^{th} user, and $\mathrm{Rel_i}$ is the set of applications deemed relevant for the i^{th} user. Given our implicit ratings, the recommendation threshold was set to 1.7, since rating predictions close to 1 symbolize a high probability of a negative user opinion of an application.

All the models were trained with 5-fold cross validation with the *KFold* Surprise model iterator. We will now be comparing the results of each model produced when tuning their parameters. Table II contains the best results for each algorithm. The *NormalPredictor* algorithm, which statistically predicts the ratings from a normal distribution estimated from the training data as seen in Equations 1 and 2, showed the worst result, as expected, with a F1@20 of 0.674. The *BaselineAlgo* presented much better results, as it actually accounts for user and item bias as seen in Equation 3, achieving a F1@20 of 0.845.

The *Slope One* algorithm was the only one, besides the *NormalPredictor*, yielding worse results than the *BaselineOnly*, with an F1@20 of 0.827, as it is a very simple implementation of collaborative filtering. Its computational cost was also higher than the baseline, but much lower than our remaining models.

The neighborhood based model *KNNBasic* produced the best F1 results, with a maximum value of 0.892 in F1@20, but at an extremely high computational cost. We used Surpsrise's default parameters and we were unable to train models with a number of neighbors *k* higher than 10 with our resources. The model with this aggregation parameter set to 5 had a slightly better P@20 of 0.886 and worse R@20 of 0.856 and F1@20 of 0.871, when compared to the model with the aggregation parameter set to 10, which yielded a P@20 of 0.882, a R@20 of 0.856 and F@20 of 0.892.

For the *SVD* algorithm we used Surprise's default parameter values and a learning rate of 0.2 during training to compare our models when varying the number of latent factors between 10, 20, 50, 100 and 200, and the number of epochs between 10, 25, 50, 75, 100 and 200, as seen in figure 7. As expected, a higher number of epochs produced better results, as well as a higher number of latent factors, since they allowed a more rigorous profile of our users and applications, at a computational cost. This model produced the highest P@5 of 0.913, which was obtained with 200 latent factors and 200 epochs.

Finally, the *NMF* algorithm produced very similar results to the *SVD*, and even produced the highest recall value of 0.971 in R@20, but it was much more difficult to train as it is highly dependent on initial values. As an example of the impact the initialization has on this recommendation system, by changing the bounds of the random initialization of factors from (0.0, 1.0) to (0.0, 0.2), the F1@10 for the *NMF* model with 25 epochs and 100 factors changed from 0.0005 to 0.7583. Apart from these parameters, we used Surprise's default parameters and compared our models when varying the number of latent factors between 10, 20, 50, 100 and 200, and the number of epochs between 20, 50, 100 and 200. The F1@10 results from this algorithm can be seen in figure 8, where all the models were initialized with values between 0.1 and 0.4, as these produced the best scores.

VI. Conclusion

This work introduces a novel dataset provided by Aptoide that reports the installed applications for over 4 million users in three different daily snapshots during 2020. We analysed the behavior of the users that appear in the three snapshots, inferring implicit binary ratings for applications based on installs and uninstalls. We have compared six well-established collaborative filtering techniques on top of the binary ratings. These models were trained using a 5-fold cross validation with an 80/20 training/test dataset split, and evaluated using the Precision@k, Recall@k, and F1-score@k metrics.

The presented findings found that for this dataset, the *SVD*-based model achieves the higher precision results, and the *NMF*-based achieves the best recall ones. However, at an

TABLE II
BEST RESULTS FOR EACH ALGORITHM, EVALUATED WITH P@5,10,20, R5,10,20, AND F1@5,10,20.

Algorithm	Training time [s]	Precision			Recall			F1-score		
		@5	@10	@20	@5	@10	@20	@5	@10	@20
NormalPredictor	39	0,786	0,786	0,786	0,526	0,579	0,590	0,630	0,667	0,674
BaselineOnly	56	0,869	0,869	0,859	0,698	0,806	0,831	0,774	0,837	0,845
SlopeOne	105	0,858	0,850	0,849	0,673	0,780	0,806	0,754	0,813	0,827
KNNBasic	5493	0,895	0,884	0,882	0,754	0,875	0,903	**0,818**	**0,880**	**0,892**
SVD	3917	**0,913**	**0,905**	**0,903**	0,719	0,827	0,853	0,804	0,864	0,877
NMF	611	0,832	0,821	0,819	**0,762**	**0,931**	**0,971**	0,795	0,873	0,888

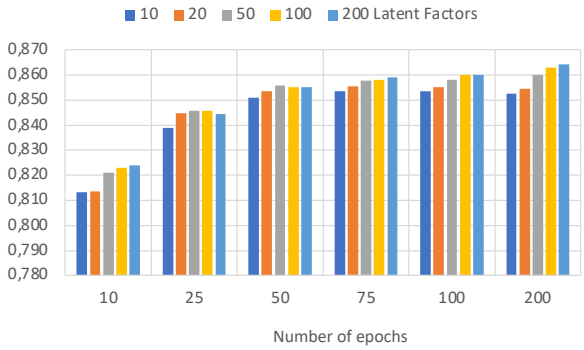


Fig. 7. F1@10 performance of the SVD models.

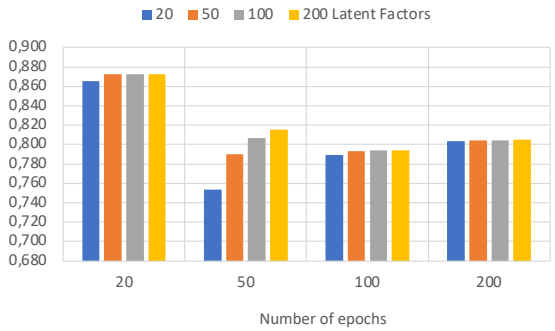


Fig. 8. F1@10 performance of the NMF models.

higher computational cost, the *KNNBasic* model achieves a better trade-off between both, yielding the higher F1-scores.

This way, we were able to successfully benchmark this dataset for recommendation, presenting the results for the most well-known techniques, thus allowing future experiments with different techniques. However, due to our computational limitations, we were not able to fully leverage this extensive dataset, since we excluded a lot of applications with few ratings, leading us not to address the cold start problem.

Future directions include the study of semantic relations between applications, in order to better detect similar apps that might not be that popular [37]. Also, recent work has studied the benefits of the usage of auto-encoders for collaborative filtering, and such approaches can be considered leveraging our dataset and implicit feedback [38]–[40]. Additionally, we only took into consideration the installs and uninstalls from users, but it could be interesting to implement an hybrid model, as in [41], that combines demographic information (such as age, location) in an attempt to increase the accuracy of user similarity. Finally, the chosen binary rating makes the overall task easier, and this may have impacted the overall performance. Other possible approach to a more comprehensive characterization of users' profiles may take into account the change of users' likes over time, as explored in [42].

ACKNOWLEDGMENTS

This work was supported by PT2020 project number 39703 (AppRecommender) and by national funds through FCT, Fundação para a Ciência e a Tecnologia, under projects UIDB/50021/2020 and UIDB/04466/2020.

REFERENCES

[1] J. B. Schafer, J. A. Konstan, and J. Riedl, "E-commerce recommendation applications," *Data Min. Knowl. Discov.*, vol. 5, no. 1/2, pp. 115–153, 2001. [Online]. Available: https://doi.org/10.1023/A:1009804230409

[2] R. Zhou, S. Khemmarat, and L. Gao, "The impact of youtube recommendation system on video views," in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, M. Allman, Ed. ACM, 2010, pp. 404–410. [Online]. Available: https://doi.org/10.1145/1879141.1879193

[3] B. Smith and G. Linden, "Two decades of recommender systems at amazon.com," *IEEE Internet Comput.*, vol. 21, no. 3, pp. 12–18, 2017. [Online]. Available: https://doi.org/10.1109/MIC.2017.72

[4] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, S. Sen, W. Geyer, J. Freyne, and P. Castells, Eds. ACM, 2016, pp. 191–198. [Online]. Available: https://doi.org/10.1145/2959100.2959190

[5] R. M. Bell and Y. Koren, "Lessons from the netflix prize challenge," *SIGKDD Explor.*, vol. 9, no. 2, pp. 75–79, 2007. [Online]. Available: https://doi.org/10.1145/1345448.1345465

[6] D. Goldberg, D. A. Nichols, B. M. Oki, and D. B. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, 1992. [Online]. Available: https://doi.org/10.1145/138859.138867

[7] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, J. B. Smith, F. D. Smith, and T. W. Malone, Eds. ACM, 1994, pp. 175–186. [Online]. Available: https://doi.org/10.1145/192844.192905

[8] J. S. Breese, D. Heckerman, and C. M. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998*, G. F. Cooper and S. Moral,

Eds. Morgan Kaufmann, 1998, pp. 43–52. [Online]. Available: https://dl.acm.org/doi/10.5555/2074094.2074100

[9] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53, 2004. [Online]. Available: https://doi.org/10.1145/963770.963772

[10] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

[11] J. L. Herlocker, J. A. Konstan, and J. Riedl, "An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms," *Inf. Retr.*, vol. 5, no. 4, pp. 287–310, 2002. [Online]. Available: https://doi.org/10.1023/A:1020443909834

[12] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, Eds. ACM, 2001, pp. 285–295. [Online]. Available: https://doi.org/10.1145/371920.372071

[13] M. D. Ekstrand, F. M. Harper, M. C. Willemsen, and J. A. Konstan, "User perception of differences in recommender algorithms," in *Eighth ACM Conference on Recommender Systems, RecSys '14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, A. Kobsa, M. X. Zhou, M. Ester, and Y. Koren, Eds. ACM, 2014, pp. 161–168. [Online]. Available: https://doi.org/10.1145/2645710.2645737

[14] S. Funk, "Netflix update: Try this at home." [Online]. Available: http://sifter.org/~simon/journal/20061211.html

[15] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009. [Online]. Available: https://doi.org/10.1109/MC.2009.263

[16] D. K. Bokde, S. Girase, and D. Mukhopadhyay, "Role of matrix factorization model in collaborative filtering algorithm: A survey," *CoRR*, vol. abs/1503.07475, 2015. [Online]. Available: http://arxiv.org/abs/1503.07475

[17] D. W. Oard, J. Kim *et al.*, "Implicit feedback for recommender systems," in *Proceedings of the AAAI workshop on recommender systems*, vol. 83. WoUongong, 1998, pp. 81–83.

[18] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*. IEEE Computer Society, 2008, pp. 263–272. [Online]. Available: https://doi.org/10.1109/ICDM.2008.22

[19] R. Bunga, F. Batista, and R. Ribeiro, "From implicit preferences to ratings: Video games recommendation based on collaborative filtering," in *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2021, Volume 1: KDIR, Online Streaming, October 25-27, 2021*, R. Cucchiara, A. L. N. Fred, and J. Filipe, Eds. SCITEPRESS, 2021, pp. 209–216. [Online]. Available: https://doi.org/10.5220/0010655900003064

[20] Y. Hu, F. Xiong, D. Lu, X. Wang, X. Xiong, and H. Chen, "Movie collaborative filtering with multiplex implicit feedbacks," *Neurocomputing*, vol. 398, pp. 485–494, 2020. [Online]. Available: https://doi.org/10.1016/j.neucom.2019.03.098

[21] N. Hug, "Surprise: A python library for recommender systems," *J. Open Source Softw.*, vol. 5, no. 52, p. 2174, 2020. [Online]. Available: https://doi.org/10.21105/joss.02174

[22] T. Liang, L. Zheng, L. Chen, Y. Wan, P. S. Yu, and J. Wu, "Multi-view factorization machines for mobile app recommendation based on hierarchical attention," *Knowl. Based Syst.*, vol. 187, 2020. [Online]. Available: https://doi.org/10.1016/j.knosys.2019.06.029

[23] J. Lin, K. Sugiyama, M. Kan, and T. Chua, "New and improved: modeling versions to improve app recommendation," in *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*, S. Geva, A. Trotman, P. Bruza, C. L. A. Clarke, and K. Järvelin, Eds. ACM, 2014, pp. 647–656. [Online]. Available: https://doi.org/10.1145/2600428.2609560

[24] H. Yin, W. Wang, L. Chen, X. Du, Q. V. H. Nguyen, and Z. Huang, "Mobi-sage-rs: A sparse additive generative model-based mobile application recommender system," *Knowl. Based Syst.*, vol. 157, pp. 68–80, 2018. [Online]. Available: https://doi.org/10.1016/j.knosys.2018.05.028

[25] R. C. Jisha, R. Krishnan, and V. Vikraman, "Mobile applications recommendation based on user ratings and permissions," in *2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018, Bangalore, India, September 19-22, 2018*. IEEE, 2018, pp. 1000–1005. [Online]. Available: https://doi.org/10.1109/ICACCI.2018.8554691

[26] K. Lin, Y. Chang, C. Shen, and M. Lin, "Leveraging online word of mouth for personalized app recommendation," *IEEE Trans. Comput. Soc. Syst.*, vol. 5, no. 4, pp. 1061–1070, 2018. [Online]. Available: https://doi.org/10.1109/TCSS.2018.2878866

[27] X. Xu, K. Dutta, A. Datta, and C. Ge, "Identifying functional aspects from user reviews for functionality-based mobile app recommendation," *J. Assoc. Inf. Sci. Technol.*, vol. 69, no. 2, pp. 242–255, 2018. [Online]. Available: https://doi.org/10.1002/asi.23932

[28] M. Böhmer, B. J. Hecht, J. Schöning, A. Krüger, and G. Bauer, "Falling asleep with angry birds, facebook and kindle: a large scale study on mobile application usage," in *Proceedings of the 13th Conference on Human-Computer Interaction with Mobile Devices and Services, Mobile HCI 2011, Stockholm, Sweden, August 30 - September 2, 2011*, M. Bylund, O. Juhlin, and Y. Fernaeus, Eds. ACM, 2011, pp. 47–56. [Online]. Available: https://doi.org/10.1145/2037373.2037383

[29] A. Al-Hawari, "A dataset of mobile application reviews for classifying reviews into software engineering's maintenance tasks using data mining techniques," Mendeley Data, 2019.

[30] G. Grano, A. D. Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: a dataset for software evolution and quality improvement," in *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2017, Paderborn, Germany, September 5, 2017*, F. Sarro, E. Shihab, M. Nagappan, M. C. Platenius, and D. Kaimann, Eds. ACM, 2017, pp. 8–11. [Online]. Available: https://doi.org/10.1145/3121264.3121266

[31] L. Baltrunas, K. Church, A. Karatzoglou, and N. Oliver, "Frappe: Understanding the usage and perception of mobile app recommendations in-the-wild," *CoRR*, vol. abs/1505.03014, 2015. [Online]. Available: http://arxiv.org/abs/1505.03014

[32] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, vol. 5, 02 2007.

[33] Y. Koren, R. M. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[34] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Trans. Ind. Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014. [Online]. Available: https://doi.org/10.1109/TII.2014.2308433

[35] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, pp. 1:1–1:24, 2010. [Online]. Available: https://doi.org/10.1145/1644873.1644874

[36] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[37] J. Kim, S. Kang, Y. Lim, and H. Kim, "Recommendation algorithm of the app store by using semantic relations between apps," *J. Supercomput.*, vol. 65, no. 1, pp. 16–26, 2013. [Online]. Available: https://doi.org/10.1007/s11227-011-0701-6

[38] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, and S. I. Nikolenko, "Recvae: A new variational autoencoder for top-n recommendations with implicit feedback," in *Proceedings of the International Conference on Web Search and Data Mining*, 2020.

[39] H. Steck, "Embarrassingly shallow autoencoders for sparse data," in *Proceedings of The World Wide Web Conference*, 2019.

[40] D. Liang, R. G. Krishnan, M. D. Hoffman, and T. Jebara, "Variational autoencoders for collaborative filtering," in *Proceedings of the World Wide Web Conference*, 2018.

[41] Y. Afoudi, M. Lazaar, and M. A. Achhab, "Intelligent recommender system based on unsupervised machine learning and demographic attributes," *Simul. Model. Pract. Theory*, vol. 107, p. 102198, 2021. [Online]. Available: https://doi.org/10.1016/j.simpat.2020.102198

[42] G. Xu, Z. Tang, C. Ma, Y. Liu, and M. Daneshmand, "A collaborative filtering recommendation algorithm based on user confidence and time context," *J. Electr. Comput. Eng.*, vol. 2019, pp. 7 070 487:1–7 070 487:12, 2019. [Online]. Available: https://doi.org/10.1155/2019/7070487