Popularity Does Not Always Mean Triviality: Introduction of Popularity Criteria to Improve the Accuracy of a Recommender System

Roberto Saia*, Ludovico Boratto, Salvatore Carta
Università di Cagliari, Dipartimento di Matematica e Informatica, Cagliari, Italy.

* Corresponding author. Tel.: +390706758755; email: roberto.saia@unica.it
Manuscript submitted September 5, 2015; accepted December 17, 2015.
doi: 10.17706/jcp.12.1.1-9

Abstract: The main goal of a recommender system is to provide suggestions, by predicting a set of items that might interest the users. In this paper, we will focus on the role that the popularity of the items can play in the recommendation process. The main idea behind this work is that if an item with a high predicted rating for a user is very popular, this information about its popularity can be effectively employed to select the items to recommend. Indeed, by merging a high predicted rating with a high popularity, the effectiveness of the produced recommendations would increase with respect to a case in which a less popular item is suggested. The proposed strategy aims to employ in the recommendation process new criteria based on the items' popularity, by measuring how much it is preferred by users. Through a post-processing approach, we use this metric to extend one of the most performing state-of-the-art recommendation techniques, i.e., SVD++. The effectiveness of this hybrid strategy of recommendation has been verified through a series of experiments, which show strong improvements in terms of accuracy w.r.t. SVD++.

Key words: Collaborative filtering, algorithms, metrics.

1. Introduction

In order to provide effective suggestions in terms of the good or services offered by a company, a *recommender system* plays an essential role, since it is able to filter the user preferences, suggesting them only the items that could be interesting. The identification of these items is based on a *prediction* task, than infers the interest of a user toward an item not yet evaluated, to derive if it is worth recommending [1]. Most of the strategies used to generate the recommendations are based on the so-called *Collaborative Filtering* (CF) approach [2] which is based on the assumption that users have similar preferences on an item if they have already rated other items in a similar way [3]. In recent years, the *latent factor models* have been adopted in CF approaches with the aim to uncover latent characteristics that explain the observed ratings [4]. Some of the most common approaches of this type are those that exploit the *neural networks* [5], the *Latent Dirichlet Allocation* [6], but especially, those that exploit a model induced by the factorization of the user-item ratings matrix [7] (i.e., the matrix that reports the ratings given to items by the users). Among these last approaches, the state of the art is represented by SVD++ [8], the Koren's version of the *Singular Value Decomposition* (SVD) [9], which exploits the so-called *latent factor model* and presents good performance in terms of accuracy and scalability [7], [10]. Although SVD++ provides excellent performance, it does not take into account the factor of popularity of the items that are

recommended. This might lead to an underperformance of a recommender system, in case the same score is predicted for multiple items. Indeed, the system is not able to discriminate them on the basis of their popularity, so there is the risk to recommend those unpopular, which are less likely to be preferred by users. The popularity of the items is an aspect that has been widely studied in the recommender systems literature. While their ability to identify items of potential interest to users has been recognized, some limitations have been highlighted. The most important of these is that the recommendations made according to popularity criteria are trivial, and do not bring considerable benefits neither to users, nor to those that offer them goods or services. This happens when using the so-called *non-personalized model* [11], a naive approach of recommendation that does not take into account the user preferences, because it always recommends a fixed list with the most popular items, regardless of the target user. On the other hand, however, recommending less popular items adds novelty (and also serendipity) [12] to the users, but it usually is a more difficult task to perform.

This work aims at improving the recommendations produced by the SDV++ approach, by considering the items' popularity. Our strategy involves a balanced use of an index of item popularity, which is based on the positive feedbacks of the users. This index is then applied within the boundaries of a recommendation list, generated through a state-of-the-art approach based on the *latent factor model* (the so-called SVD++ approach [8]), instead of using the entire dataset. This way of proceeding allows us to exploit the popularity metrics to perform a *fine-tuning* of the recommendations generated by a recommendation strategy at the state of the art, which does not take into account the item popularity, thus reducing the triviality of the final result. The contributions of our work are the following:

- Definition of the Domain Popularity Index (DPI), a metric able to evaluate the preferences of the users about an item;
- Creation of the PBSVD++ algorithm, which extends the capabilities of SVD++, adding to it the capability to evaluate the item popularity;
- Experimentation on three real-world datasets, to evaluate the capability of popularity to increase the number of effective recommendations, with respect to a state-of-the-art approach that does not employ it.

In the rest of this paper, we first introduce the literature related with the proposed strategy (Section 2), continuing to define the adopted notation and the problem definition (Section 3), the implementation details of our proposal (Section 4). Finally, we complete the paper with the description of the performed experiments (Section 5), ending with some concluding remarks and future work (Section 6).

2. Related Work

This section presents two concepts closely related with our work.

Non-personalized Models. The recommender systems based on the so-called *non-personalized model* [11], propose to all users the same list of recommendations, without taking into account their preferences. This static approach is usually based on two algorithms, the first of them (TopPop), operates by suggesting the most rated items (i.e., those most popular), while the second (MovieAvg), works by suggesting the highest rated items (i.e., those most liked). The exclusive use of the non-personalized models leads toward the absence of two important characteristics that a recommender system should have, i.e., novelty and serendipity [13]. Novelty occurs when a system is able to recommend unknown items that a user might have autonomously found, while the serendipity happens when it helps the user to find a surprisingly interesting item that a user might not have otherwise found, or if it is very hard to find.

Latent Factor Models. The type of data with which a recommendation system operates is typically a sparse matrix where the rows represent the users, and the columns represent the items. The entries of this

matrix are the interaction between users and items, in the form of ratings or purchases. The aim of a recommender system is to infer, for each user *u*, a ranked list of items, and in literature many of them are focused on the rating prediction problem. The most effective strategies in this field exploit the so-called *latent factor models*, but especially, the *matrix factorization* techniques [7]. Other CF ranking-oriented approaches that extend the matrix factorization techniques, have been recently proposed, and most of them use a ranking oriented objective function, in order to learn the latent factors of users and items [14]. SVD++ [8], the Koren's version of the *Singular Value Decomposition* (SVD) [9], is today considered one of the best strategies in terms of accuracy and scalability. In [16]-[18], the problem of modeling semantically correlated items was tackled, but the authors consider a temporal correlation and not the one between the items and a user profile.

3. Notation and Problem Definition

The mathematical notation used in this work, and the problem statement, are recalled in the following.

3.1. Notation

We are given a set of users $U = \{u_1, \ldots, u_N\}$, a set of items $I = \{i_1, \ldots, i_M\}$, and a set *V* of values used to express the user preferences (e.g., *V*=[1, 5] or *V*={*like, dislike*}). The set of all possible preferences expressed by the users is a ternary relation $P \subseteq U \times I \times V$. We denote as $P_+ \subseteq P$ the subset of preferences with a positive value (i.e., $P_+ = \{(u,i,v) \in P \,|\, v \geq \bar{v} \vee v = like\}$), where $\bar{v}$ indicates the mean value (in the previous example, $\bar{v} = 3$). Moreover, we denote as $I_+ = \{i \in I \,|\, \exists (u,i,v) \in P_+\}$ the set of items for which there is a positive preference, and as $np_{i,u} = |(u,i,v) \in P_+|, i \in I, \forall u \in U$ the number of positive preferences expressed by all users *u* for an item *i*. We also denote as $I_u = \{i \in I \,|\, \exists (u,i,v) \in P \wedge u \in U\}$ the set of items in the profile of a user *u*, and as $R_u = \{u \in U \wedge R \subseteq I\}$, the set of items *i* recommended to a user *u*. The set of items *i* without the items already evaluated by the user *u* (i.e., those in I_u) is denoted as $\hat{I}_u \subseteq I$.

3.2. Problem Definition

We consider the function $f : U \times I \rightarrow V$, adopted to predict the ratings for the not evaluated items with the SVD++ recommender system. Our aim is to define, for each item, a Domain Popularity Index *DPI(i)* that represents the popularity of the item with respect to the others in the dataset (in terms of positive evaluations given by the users to it). The Domain Popularity Index DPI of an item not evaluated by a user will be employed to build a score α. Our objective is to generate a list of recommended item i^* such that:

$$i^* = \underset{j \in \hat{I}_u}{\operatorname{argmax}} \, f(u, j) + \alpha \tag{1}$$

4. Integrating Popularity in the Recommendation Process: Algorithm

In this section, we present the steps made to generate the recommendations based on the proposed Popularity-based SVD++ (PBSVD++) strategy, starting from the definition of the popularity index employed by the approach, and ending with the implementation of our novel algorithm.

4.1. Items Popularity Definition

In this section, we introduce and formalize the popularity index employed of our approach. The value of the Domain Popularity Index (DPI) for an item $i \in I$, with $DPI \in [0,1]$, $np_{i,U}$ represents the number of positive preferences expressed by all users *U* for the item *i*. It is calculated as shown in equation 2.

$$DPI(i,U) = \frac{np_{i,U}}{\sum_{j \in I} np_{j,U}} \tag{2}$$

DPI is an important indicator, because it provides a global measure of the preferences expressed for an item by all users.

4.2. PBSVD++ Algorithm

We exploit the DPI index previously presented, in order to modify the result of the SVD++ approach. The index is employed in Algorithm 1, where we build a value α. Given a set of recommendations R_u, addressed to a user $u \in U$, the final rating $\rho_{i,u}$ assigned to each item $i \in R_u$ by our algorithm, is composed by the $rating_{i,u}$ calculated through the SVD++ approach, normalized in a continuous range from 0 to 1, and denoted as *STD(i,u)*, added to the α (also normalized in a continuous range from 0 to 1), built by employing the *DPI* index, as shown in Equation 3. The final rating assigned to an item is then in the range from 0 to 2.

$$\begin{aligned} &\rho_{i,u} = STD(i,u) + \alpha \\ &\text{with } STD(i,U) = \frac{\text{rating}_{i,u}}{\sum_{j \in U} \text{rating}_{j,u}} \text{ and } \alpha = \frac{DPI(I,U)}{\sum_{j \in U} DPI(j,U)} \end{aligned} \tag{3}$$

The new rating $\rho_{i,u}$, assigned to an item *i* for a user *u* takes into account, in a balanced way, its domain popularity, and this produces a substantial change in the canonical SVD++ ranking during the recommendation process, changing the performance of the recommender system. Algorithm 1 implements the operations described above. It takes as input the training set *s* (used by the SVD++ approach, in step 3, to build the latent factor model), the user *u* to whom address the recommendations, and the number *n* of these.

Algorithm 1. PBSVD++

Input: *s*=Training set, *u*=User, *n*=Recommendations
Output: *L* = List of *n* recommendations

```
1.  procedure GETPBSVDRECS(s,u,n)
2.  x=GetNumOfNotEvaluatedItems(u)
3.  I=GetSvdRecs(s,u,x)
4.  t=0
5.  for each i in I do
6.      if (SvdRating(i) + 1) > SvdRating(i0) then
7.          R ← i
8.          t+=GetDPI(i)
9.      end if
10. end for
11. for each r in R do
12.     rating=(SvdRating(r)/SumAllSvdRatings(R))
13.     α = GetDPI(r)/t
14.     SetNewRating(r,rating+α)
15. end for
16. L = GetRecsDescOrdered(R, n)
17. Return L
18. end procedure
```

After the number *x* of potential items to recommend to the user *u* has been obtained (step 2), we calculate through the standard SVD++ approach, for the user *u*, a set *I* of *x* recommendations based on the training set *s* (step 3). In the steps from 5 to 10, we select from *I* only the elements *i* that are possible

candidates for the recommendations based on the proposed approach. They are those items in which a modification of the score, by adding to the original rating of SVD++ the value of α (parameter calculated in the step 13, whose value is in the range from 0 to 1), could alter the rank proposed by SVD++. For this reason, the candidates are only the items to which, adding at most 1, we get a value higher than that of the item with the maximum SVD++ score (i.e., the first element i_0). We use this process also to calculate (in step 8) the sum of the DPI weight, related to all the items $i \in I$. Starting with this set *R* of candidate items, in the steps from 11 to 17, we alter the SVD++ score of each item $i \in I$, following equation 3, after which we return a list *L* of *n* recommendations, composed by the items with the highest scores.

5. Experiments

In this section, after the definition of the experimental environment and of the adopted datasets' characteristics, we describe the strategy and metrics used, concluding with the presentation and discussion of the experimental results.

5.1. Experimental Setup

The environment for this work is based on the Java language, with the support of the Apache Mahout (https://mahout.apache.org) Java framework to implement the state-of-the-art approach that we compare our novel approach with. In order to evaluate the proposed strategy, we perform a series of experiments on three different real-world datasets, which represent a quite standard benchmark in the context of the recommender systems: the first one is the dataset Yahoo! Webscope R4 (http://webscope.sandbox.yahoo.com), which contains a large amount of data related to users preferences expressed by the Yahoo! Movies community; the others two are extracted from the dataset Movielens 10M (http://grouplens.org/datasets/movielens/), composed by the data collected over various periods of time, on the MovieLens web site. The first set of experiments provides a general overview of the results obtained by comparing the performance of a recommender system, where we have implemented the new PBSVD++ algorithm, with those of a canonical system based on the SVD++ algorithm. The second set of experiments shows in more detail the results previously summarized, analyzing them through the precision and recall metrics.

5.2. Datasets

In order to evaluate the proposed strategy, we perform a series of experiments on three different real-world datasets, extracted by two quite standard benchmarks in the context of the recommender systems: Yahoo! Webscope R4 and Movielens 10M.

Yahoo! Webscope (R4). This dataset contains a large amount of data related to users preferences expressed on the Yahoo! Movies community that are rated on the base of two different scales, from 1 to 13 and from 1 to 5 (we use the latter). The training data is composed by 7,642 users, 11,915 movies/items, and 211,231 ratings. All the users in the training set have rated at least 10 items and all items are rated by at least one user. The test data is composed by 2,309 users, 2,380 items, and 10,136 ratings. There are no test users/items that do not also appear in the training data. All the users in the test set have rated at least one item and all items have been rated by at least one user. The items are classified in 20 different classes (genres), and it should be noted that an item may be classified with multiple classes.

Movielens 10M. The second dataset used in this work is composed by 71,567 users, 10,681 movies/items, and 10,000,054 ratings. It was extracted at random from MovieLens (a movie recommendation website). All the users in the dataset had rated at least 20 movies, and each user is represented by a unique ID. The ratings of the items are based on a *5-star* scale, with *half-star* increments. In this dataset the items are classified in 18 different classes (movie genres), and also in this case each item

may be classified with multiple classes (genres). Since the Movielens 10M dataset does not contain any textual description of the items, to obtain this information we used a file provided by the Webscope (R4) dataset, which contains a mapping from the movie IDs used in the dataset to the corresponding movie IDs and titles used in the MovieLens dataset. Using the script provided with the Movielens 10M dataset, we split up the whole dataset in two different datasets with exactly 10 ratings per user in the test set. Both training sets are composed by 69,878 users, and 9,301,274 ratings, with 10,667 movies/items in the first one, and 10,676 movies/items in the second one. Each test dataset contains 69,878 users, and 698,780 ratings, with 3,326 movies/items in the first one, and 5,724 movies/items in the second one. From each of these datasets, we take in account a subset of 20,000 users.

5.3. Strategy

We compare the proposed recommendation strategy with the state-of-the-art approach SVD++. The Mahout framework, used to implement it, in addition to the training set requires two additional parameters: the number of target features and the number of training steps to run. The first parameter would be equivalent to the number of involved genres, thus we have set this value to 20 for the Yahoo dataset, and to 18 for the Movielens datasets. Regarding the second parameter, we use the value 15, as indicated in the *SVD++* reference paper [8].

5.4. Metrics

Here, we present the metrics used during the experiments.

Precision and Recall. The performance measures adopted to evaluate our approach, comparing the set of recommendations generated by our strategy and the set of those generate by the canonical approach of recommendation with the real user preferences stored in the test set, are the *precision* and the *recall*, and metrics [15]. Given two sets X_u and Z_u, where X_u denotes the set of recommendations performed for a user u, and Z_u the set of the real choices of the user u in the test set, these metrics are defined as shown in Equation 4.

$$\begin{aligned} \text{precision}(X_u, Z_u) &= \frac{|Z_u \cap X_u|}{|X_u|} \\ \text{recall}(X_u, Z_u) &= \frac{|Z_u \cap X_u|}{|Z_u|} \end{aligned} \tag{4}$$

Metrics Evaluation. In order to compare the results of the two approaches of recommendation (i.e., our approach based on the PBSVD++ algorithm, and the canonical none, based on SVD++), we calculate the previous metrics, presented in Equation 4, for each group of n performed recommendations (denoted as @n, with n={2, 4,..., 20}), subtracting from the values obtained by our approach those obtained by SVD++.

In this way, a positive value denotes that our approach improves the standard one, while a negative value denotes that our approach worsens the standard one. Denoting as X_n the set of n recommendations generated by our strategy, as Y_n the set of n recommendations generated by the canonical SVD++ strategy, and as Z_n the set of n real user preferences stored in the test set, we define the measures shown in Equation 5.

$$\begin{aligned} p-\text{variation}@n &= \text{precision}@n(X_n, Z_n) - \text{precision}@n(Y_n, Z_n) \\ r-\text{variation}@n &= \text{recall}@n(X_n, Z_n) - \text{recall}@n(Y_n, Z_n) \end{aligned} \tag{5}$$

5.5. Experimental Results

Here, we report the results of the experiments presented in Section 5.1.

Performance Overview and Details. The result presented in Fig. 1 shows the general performance of the proposed strategy, in the context of the three considered real-world datasets. It indicates the percentage of times in which we have done better, or have done worse than SVD++ (respectively, *B* and *W*). The overall results show the good performance of our approach with all three datasets.

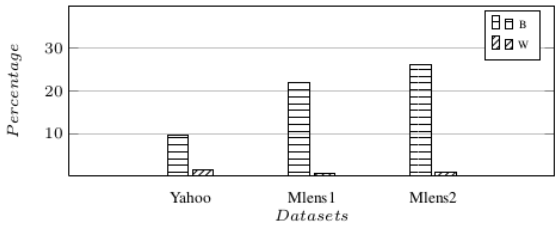
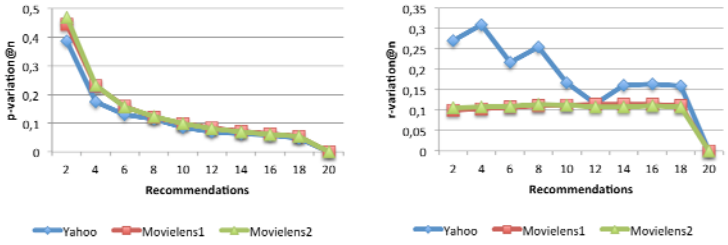


Fig. 1. Results overview.

In the second set of experiments we compare the performance of a recommender system where we have implemented the PBSVD++ algorithm, with those of the canonic recommender system based on the SVD++ algorithm. We evaluate the results in terms of *p-variation@n and r-variation@n*, as described in Section 5.4. As we can observe in the graphs in Fig. 2, the results are quite similar for all the three considered datasets, apart in the *recall@n* measure, which reports a difference between the Yahoo and Movielens results. This happens because the script provided by Movielens places a fixed number of ratings per user in the test set (10). This does not happen in the Yahoo dataset, which builds a test set with a variable number of items for each user. Since the *recall@n* metric has as denominator the number of items in the test set, this number is fixed for the Movielens dataset (hence, the results are more "flat"), and variable for the Yahoo dataset (this leads to the variable results in Fig. 2(b)).

The overall results, presented in Fig. 2, show that our strategy outperforms the canonical one, using all metrics, except when we test the maximum number of recommendations (i.e., 20). This is an obvious aspect, since the algorithm PBSVD++ operates in the domain of the SVD++ recommendations, recalculating their ratings: therefore, when we consider the entire domain, the results of SVD++, and PBSVD++, will always be identical.



(a) (b)

Fig. 2. Experimental results.

Discussion. The performed experiments, presented in Section 5, prove that our strategy, based on the novel PBSVD++ algorithm, is able to improve the results of a canonical recommender system based on the SVD++ algorithm. As we can observe, this happens with any number of recommendations, except the case in which the maximum number of these is generated, for the obvious reason explained in the previous section. When evaluating these results, we can observe that the maximum value of positive variation for a metric is 1 (which represents a 100% improvement w.r.t. SVD++). Therefore, our results suggest important improvements, thinking that the Netflix prize was based on a 10% improvement in terms of accuracy. This

proves that is possible to improve a state-of-the-art approach such as SVD++, by using its output as an input domain, in order to perform a *fine-tuning* based on the popularity of the involved items.

6. Conclusions and Future Work

In this paper we proposed a novel form of recommendation, which integrated the information about item popularity into a state-of-the-art approach. The performed experiments have shown both the validity of the adopted index, and its ability to improve the performance of the SVD++ approach. In future work, we will extend our approach, by adding new metrics able to evaluate the item popularity, in the context of systems that operate within more than one domain of goods/services, trying to parameterize both the popularity aspect of each item, and their interconnections between different operative domains. We will also study the introduction of others metrics of popularity, e.g., based on the geographic or demographic information.

Acknowledgment

This work is partially funded by Regione Sardegna under project SocialGlue, through PIA - Pacchetti Integrati di Agevolazione "Industria Artigianato e Servizi" (annualità 2010), and by MIUR PRIN 2010-11 under project "Security Horizons".

References

[1] Ricci, F. F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor. (Eds.), *Recommender Systems Handbook* (pp. 1–35). Springer.

[2] Karypis, G. (November 5-10, 2001). Evaluation of item-based top-n recommendation algorithms. *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management* (pp. 247–254). Atlanta, Georgia.

[3] Su, X. & Khoshgoftaar, T. M. A survey of collaborative filtering techniques. *Adv. Artificial Intelligence*, 2009.

[4] Koren, Y., & Bell, R. M. (2011). Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor. (Eds.), *Recommender Systems Handbook* (pp. 145–186). Springer.

[5] Georgiev, K., & Nakov, P. (2013). A non-iid framework for collaborative filtering with restricted boltzmann machines. *Proceedings of the 30th International Conference on Machine Learning*: *Vol. 28* (pp. 1148–1156).

[6] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, *3*, 993–1022.

[7] Koren, Y., Bell, R. M., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer*, *42(8),* 30–37.

[8] Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Las Vegas, Nevada, USA.

[9] Billsus, D., & Pazzani, M. J. (1998). Learning collaborative information filters. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 46–54).

[10] Bennett, J., Elkan, C., Liu, B., Smyth, P., & Tikk, D. (2007). Kdd cup and workshop 2007. *SIGKDD Explor. Newsl.*, *9(2),* 51–52.

[11] Cremonesi, P., Koren, Y., & Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. *Proceedings of the 2010 ACM Conference on Recommender Systems* (pp. 39–46).

[12] Ge, M., Delgado-Battenfeld, C., & Jannach, D. (2010). Beyond accuracy: Evaluating recommender systems by coverage and serendipity. *Proceedings of the 2010 ACM Conference on Recommender*

Systems (pp. 257–260).
[13] Iaquinta, L., de Gemmis, M., Lops, P., Semeraro, G., Filannino, M., & Molino, P. (2008). Introducing serendipity in a content-based recommender system. *Proceedings of International Conference on Hybrid Intelligent Systems* (pp. 168–173).
[14] Koren, Y., & Sill, J. (2011). Ordrec: An ordinal model for predicting personalized item rating distributions. *Proceedings of the 2011 ACM Conference on Recommender Systems* (pp. 117–124).
[15] Baeza-Yates, R. A., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
[16] Stilo, G., & Velardi, P. (2009). Time makes sense: Event discovery in twitter using temporal similarity. *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies: Vol. 2* (pp. 186–193).
[17] Stilo, G., & Velardi, P. (2014). Temporal semantics: Time-varying hashtag sense clustering. *Knowledge Engineering and Knowledge Management*, *8876,* 563–578.
[18] Stilo, G., & Velardi, P. (2015). Efficient temporal mining of microblog texts and its application to event discovery. *Data Mining and Knowledge Discovery*.

Roberto Saia is a Ph.D. at the Department of Mathematics and Computer Science of the University of Cagliari. He got a master degree in computer science at the same university. His current research activity is focused on the development of techniques and algorithms able to improve the effectiveness of the user profiling and item recommendation.

Ludovico Boratto is a research assistant at the University of Cagliari, Italy. He graduated with full marks and honor and received his PhD in 2012 at the same university. His research focuses mainly on recommender systems and data mining in social networks.

Salvatore Carta received a PhD in electronics and computer science from the University of Cagliari in 2003. He is assistant professor in computer science at the University of Cagliari since 2005. Recently, he has focused on topics related to the social Web, ubiquitous computing and computational societies.