Serendipity: A New Recommender System

S. A., J.-G. B., X. P.-C.,
L. P., S. N., C. R. and N. T.
ENS Cachan

November 17, 2016

1 Abstract

Recommender systems seek to predict what users will like. What we want to do here is to develop a new recommendation system that will help the user discover new items.

Serendipity is a fuzzy concept. But here serendipity means that the system will try to make the users discover new items by recommending items the user don't know and that are not always in his tastes, thus taking a risk to broaden the user's knowledge while trying to remain relevant.

We study what a "bias" is, and we will try to distinguish two concepts, that we called diversity, and serendipity.

2 Introduction

Recommender systems

What we want to do here is to develop a new recommendation system that will help the user discover new items. Here a recommendation system is any system used to do researches, such as Amazon or Google, and not only a system created to sell items. The goal is to have an Outside-The-Box system [1], i.e. a sytem which is not based on popularity (contrary to most of the recommendation systems) but on what we call serendipity. Here serendipity means that the system will try to make the users discover new items by recommending items the user don't know and that are not always in his tastes, thus taking a risk to broaden the user's knowledge while trying to remain relevant. This is because if we only give the user items close to want he already know they may not be relevant because he may already know them.

Furthermore we will try to reduce the long tail phenomena [3], which is the fact that very popular items tend to get recommended and non-popular are not, thus reducing the diversity of the items. Here diversity means that all items are recommended, and not only the most popular ones. One of the points here is that there is usually a lot of informations available, and a recommendation system needs to be helpful to the user by giving him what he looks for, but we want to be careful to not always give the same answer because these informations may influence the reader. Thus if the system always give the same answer to everyone, there will be a standardization of knowledge, and we will lose in diversity.

Thus we want to introduce a bias in our system. But it will not be a commercial bias, aimed to sell items but a bias aiming to extend knowledge or culture. We want the user to discover new things that will please him and we don't want to hamper small producers (like artists, bloggers...). Keeping that objective in mind we also want the user to find what he want even if it's close to his own tastes, thus we will let the users choose a serendipity level, enabeling him to either explore the unknown, or either to find something he already likes.

Here we will regroup the objects, i.e. the items, in blobs, i.e. groups of objects having attributes in common. Then using the knowledge we have on the user we define the the relative appreciation, which is the proportion of items having one attribute that the user likes, among all the objects he likes. We also define the relative percolation which is the proportion of like an atribute has received (through object having this attribute) over the number of total likes. We use the user's keywords and serendipity level (i.e. if he wants to discover new objects or not) to choose the blobs that are the most relevant and then we pick objects in them and return them to the user. We then tested it on a movie base containing 100000 ratings from 943 users on 1682 items.

In the first part of this paper we describe our model, then we explain our algorithm and in the last parts we show our tests.

State of the art

3 Model

3.1 Definitions

Recommender systems seek to predict what users will like. Some recommender systems are build as system answering queries : Most recommender systems introduce some kind of bias above this relevancy criterion : it can be a bias for trendy content, sponsorised content and so on.
We study what a "bias" is, and more specifically, two biases : diversity and serendipity.

Serendipity is a fuzzy concept. It can refer either to the ability to make discoveries by accident, to an instance of such an accident or an occurence of such discoveries.
In our system, we translate it as giving the user a result not expected, but still enjoyed.
To do so, the user will get suggestions about things which were not queried nor liked until then, but not too far from what the user likes.

Whereas serendipity is a bias focused on the users, diversity is a bias with a more general approach.
There are two kind of diversity :

- The user's behaviour diversity. Does the user only search for objects of the same kind ?
- The users' diversity. Do the users (as a group) focus only on a small part of the objects ?

Considering an attribute (for instance, the producer of the content), we want to avoid the Pareto principle : where 95% of users consume the content made by 5% of the producer pool.
To do so, naturally, the diversity's bias shifts suggestions toward rarely visited genres or producers.

3.2 Model

The model is fairly simple, there are two distinct sets :

- The object set. Objects are records of attributes which can have either string or number values. The objects can be anything : jokes, songs, books, potential romantic partners etc.
- The set of users. A user gives ratings to objects and make queries.

The user makes queries on the objects, and get an ordered set of objects, corresponding to the most relevant objects with regard to the queries.
In our model, a bias is simply a re-ordering of this original result.

Using a distance between objects based on their attributes, we make blobs. Objects are in the same blob if they are related.
Blobs are different from clusters, as they do not form a partition on the object set : objects will often belong to several blobs.
These blobs are then used to compute the serendipity bias :

- When we don't have many ratings from an agent, we pick blobs distant from the query's results to simulate serendipity.
- When we have many ratings from an agent, we compute the preferences' centers of the agent. These are the most searched and rated blobs. As such, giving results distant from these blobs, but that still might be liked by the agent is our definition of serendipity.

3.3 Indicators

To compute the diversity bias, we use two indicators :

The relative knowledge The relative knowledge standard deviation is defined for a user, with respect to an attribute (for instance, the genre of a piece of music).
We define the relative knowledge of an attribute value as the number of positively rated objects with such attribute value divided by the number of consulted objects.
For instance, if the user has liked 10 jazz songs, and has liked 500 songs, then the jazz relative knowledge of this user is $10/500 = 2\%$.
For the user's diversity, we are interested in reducing its standard deviation : which means that the user rates positively songs of genres he rarely (even never) listens to. To do so, we give priority to the objects with a low relative knowledge.

The relative percolation The relative percolation is the dual of the relative appreciation : it is defined for an attribute value (for instance, jazz).
It is defined as the number of likes from user for objects of with this attribute value divided by the total number of likes. If there are 100.000 likes for jazz songs over 10.000.000 likes, then the relative percolation of jazz is 1%.
For the users' (as a group) diversity, we are interested in reducing its standard deviation : which means that we won't have every user listening to pop-rock music. To do so, we give priority to the objects with a low relative percolation.

The difference These two indicators may look similar, or the same. It is explained by the fact that, with only one user, (or if all users were identical),

they would be indeed equal.
However, these two visions of diversity are different. Let's say we only introduce a bias with respect to the relative knowledge : if a user only listened to skweee chiptune, then, he might be suggested through this bias to listen to pop-rock music (if it was predicted that he would like it), even if it is anti-diversity.

4 Implementation

Serendipity has been implemented in *C++*. The open-source code is available at: *www.github.com/BisounoursArcEnCiel/Serendipity.git*.

Classes

Each object, agent or blob has an integer identifier. Creators are attributes of objects. For the blobs graph, an edge is drawn between two blobs if and only if the blob distance between these two blobs is inferior to a certain limit distance.

Implemented distances

Like any other recommender system, Serendipity is based on the agent's (that is, the user's) preference over objects. The agent a's preference over an object o $pref(a, o)$ is defined by an integer, either -1 (dislike), 0 (no opinion) or 1 (like). It can be extended to a whole blob b thanks to this intuitive formula:

$$pref(a, b) = \sum_{o \in b} pref(a, o)$$

We first need a measure to quantify the intrinsetric and user-independant distance between two objects. Let d_1 be this distance, that is the somehow "difference" between the attributes of two objects o_1 and o_2 :

$$d_1(o_1, o_2) = \sum_{a,string} \delta_{a^{o_1} \neq a^{o_2}} + \sum_{a,integer} |a^{o_1} - a^{o_2}|$$

We also need to compare blobs. Rather than extending the previous distance, which could be valid, for sake of complexity, we chose the following definitin. Let $d_2(b_1, b_2)$ be the distance between two blobs b_1 and b_2 :

$$d_2(b_1, b_2) = |b_1 \triangle b_2|$$

However, the role of a recommender system is to reply to a user request (that is, a certain set of keywords and elements). Here, a request is a certain set of integers a_{i_j}, a_{i_j} being the integer associated to attribute number i_j, and strings s_{ii} (see next section). Let $s \in attr(o)$ mean s is one of the string attributes of objetc o. The distance d_3 between a blob b and a request r is:

$$d_3(b,r) = \frac{1}{|b|} \times \sum_{o \in b} \sum_j |a_{i_j} - a^o_{i_j}| + \sum_i \delta_{s_i \in attr(o)}$$

Generating blobs over the set of objects

Since we want blobs not to form a partition of the set of objects (a given object can belong to multiple blobs), *SERENDIPITY* uses an extension of the *K-Means* algorithm. We will introduce a threshold t such as, if o_i is an object belonging to blob b_i resulting from the application of the *K-Means* algorithm:

$$\forall i, j, i \neq j, \forall o_i, o_j, d_2(o_i, o_j) \leq t \Rightarrow o_i, o_j \in b_i \cap b_j$$

Algorithm 1 Generating the set of blobs

INPUT: Obj the set of objects, t a lower bound on the minimal distance between two objects of two different blobs, d_2 a distance between two objects
OUTPUT: Blobs the set of blobs over *Obj*
Clust result of *K-Means*
for each object o in *Object* **do**
  for each object $o_1 \neq o$ in *Cl* **do**
    if $d_2(o, o_1) \leq t$ and o and o_1 do not belong to the same cluster **then**
      Add o to Cl_{o_1}
    end if
  end for
end for
return *Clust* = *Blobs*

Numbering the nodes of the blobs graph according to the level of serendipity

In order to determine the suitable blobs according to the user-adapted level of serendipity, we need to number the nodes of the blobs graph, using preferences of the considered agent over the given set of blobs. Two methods have been designed, and we can switch from the first one to the second when the number of reviews of the considered agent is great enough, which can be quantified by this coefficient:

$$k(a) = \frac{\sum_{o,object} |pref(a,o)|}{|\{o,object\}|}$$

Almost-zero knowledge approach

A first method is based on the blobs graph internal structure, and so the intrinsetric similarity between blobs (independant from the agent's preferences). This method is useful when little is known about the agent.

We first determine *centers of the agent's domain* as the blobs b that maximize the agent a's preference (of course there may exist several so-called *centers*):

$$b \in \text{argmax} \;\; \text{pref}(a, b') \mid b' \text{ blob} \;\; = \textit{Centers(a)}$$

Let $d_{edges}(b, b')$ be the number of edges of a minimal path between b and b' in the blobs graph (well-defined since the blobs graph is non-oriented). Then we define the *label of a blob* respect to agent a as an non-negative integer such as 0 is the first level of serendipity, where no serendipity is required, and obtained by the following recursive formula:

$$\forall b \text{ blob, label}(b) = 0 \text{ if } b \in \textit{Centers(a)}$$
$$\text{else label}(b) = \min \;\; d_{edges}(b, c) \mid c \in \textit{Centers(a)}$$

If we want the levels of serendipity to have a certain width $w > 0$:

$$\forall b \text{ blob, } \textit{label}(b) = 0 \text{ if } b \in \textit{Centers(a)}$$
$$\text{else } \textit{label}(b) = E(\tfrac{1}{w} \times \min \;\; d_{edges}(b, c) \mid c \in \textit{Centers(a)} \;) \text{ where E}(x) \text{ is the greatest integer inferior to } x.$$

If we want to avoid the farest blobs, we can have the label to get infinite value when the edge distance is superior to a certain integer.

Preference-based approach

A second method is more based on the agent a's preferences, and will be more relevant when the agent has reviewed a lot of objects. We first define the *centers of the agent's domain* as in the previous section. Let $w > 0$ be the desired width of the level of serendipity. Then we define the *label of a blob* respect to agent a with the following formula:

$$\forall b \text{ blob, label}(b) = 0 \text{ if } b \in \textit{Centers(a)}$$
$$\text{label}(b) = i \text{ where } i \text{ is the smallest integer such as } \min |\text{pref}(a, b) - \text{pref}(a, c)|$$
$$c \in \textit{Centers(a)} \;\; \leq i \times w$$

The same alternatives as in previous section can be applied.

Giving recommendations to the agent

When the user (associated with a certain agent) enters a request, the latter is processed into a set of integers and strings. Then blobs are sorted according to their level of serendipity (using the labels previously defined) and only those

that are relevant enough to the request (using the *Blob-request* distance) are selected. Then objects are selected among the objects in the selected blobs, and returned to the user.

Sorting blobs

In order to select the suitable blobs, we fix the maximum value M of discrepancy (that is, the maximum distance) between a relevant blob and the request. Then we select the blob b if and only if $d_3(b, K) \leq M$. The level of serendipity is user-defined, so, among the selected blobs, we only keep those having the corresponding level of serendipity.

Choosing an object in the selected blobs

After having selected the blobs with the method described above, we need to recommend the objects to the user with a certain order, knowing the user will less probably check the last objects of those recommended than the first ones. We can order them randomly, but it would not solve the problem of promoting the less popular objects and of offering diversity. Hence we suggest two other methods, one focused on avoiding the most popular blobs, the other on choosing specific blobs (see **Model** section).

- Avoiding popular nodes

  For each object in the selected nodes of the blobs graph (that is, the selected blob), we compute the values of *Relative Knowledge* rk and *Relative Percolation* rp described in the **Model** section. We then sort the objects in growing lexicographical order of (rp, rk).

- Choosing specific nodes

  Having a set of *Creators* selected, we first display the objects having these creators in their attributes, and then the other objects.

5 Results on test databases

About the test databases

We used the *Movielens* database. It is made up of user ratings (1-5) for movies. The data set contains **100000** ratings from **943** users on **1682** items, each user having rated at least 20 items. Each item is described by its id, its

title and a series of genre it is classified as. Each user is described by its id and its ratings.

Results

Unfortunately, the implementation is not yet completed and we could not test our model on real databases.

6 Conclusion

Conclusion

We designed a recommender system that, except for the construction of the blobs, does not use optimization and Machine Learning algorithms, contrary to most of existing recommender systems. The model seems quite robust, and to solve most of the problems quoted in the previous sections. Unfortunately, we could not test our implementation.

Discussion

It would be interesting to add user feedback to change the numbering of the blobs graph for a certain user. Also, we need to get rid of the multiple parameters needed in the current implementation, for fear of the overlearning phenomenon.

References

[1] Zeinab Abbassi, Sihem Amer-Yahia, Laks VS Lakshmanan, Sergei Vassilvitskii, and Cong Yu. Getting recommender systems to think outside the box. In *Proceedings of the third ACM conference on Recommender systems*, pages 285–288. ACM, 2009.

[2] R Alexander Bentley, Carl P Lipo, Harold A Herzog, and Matthew W Hahn. Regular rates of popular culture change reflect random copying. *Evolution and Human Behavior*, 28(3):151–158, 2007.

[3] Erik Brynjolfsson, Yu Jeffrey Hu, and Michael D Smith. The longer tail: The changing shape of amazon's sales distribution curve. *Available at SSRN 1679991*, 2010.

[4] Robin Burke. Integrating knowledge-based and collaborative-filtering recommender systems. In *Proceedings of the Workshop on AI and Electronic Commerce*, pages 69–72, 1999.

[5] Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

[6] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[7] Humbert Lesca and Marie-Laurence Caron. *Veille stratégique: créer une intelligence collective au sein de l'entreprise*. 1995.

[8] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Aaai/iaai*, pages 187–192, 2002.

[9] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

[10] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[11] Florent Verschelde. Musique et internet: vers une décentralisation de la culture?