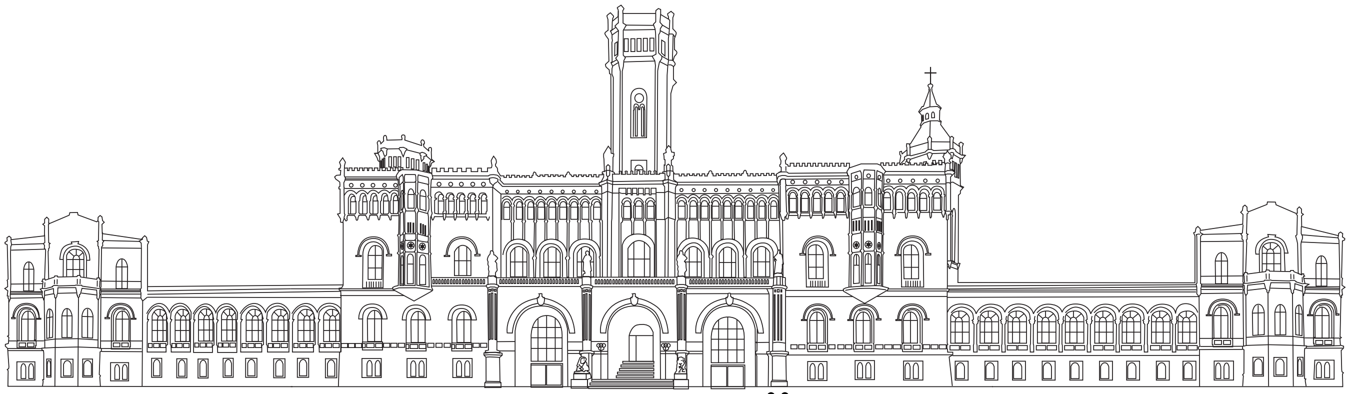


LEIBNIZ UNIVERSITÄT HANNOVER

FAKULTÄT FÜR ELEKTROTECHNIK UND INFORMATIK

# Collaborative Filtering Ensemble for Personalized Name Recommendation

**Bernat Coma-Puig**

***MASTER'S THESIS***

L3S Research Center, Leibniz University Hannover, Germany
coma-puig@l3s.de
bernat.coma@est.fib.upc.edu

Advisor: Dr. Ernesto Diaz-Aviles
First reviewer: Prof. Dr. Wolfgang Nejdl
Second reviewer: Jun.-Prof. Dr. Robert Jäschke

Leibniz Universität Hannover

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

## Acknowledgmements

We would like to thank Asmelash Teka and Rakshit Gautam for their valuable feedback. This work is funded, in part, by the L3S IAI research grant for the FizzStream Project. Bernat Coma-Puig is sponsored by the EuRopean Community Action Scheme for the Mobility of University Students (ERASMUS).

# Abstract

Out of thousands of names to choose from, picking the right one for your child is a daunting task. In this thesis, our objective is to help parents make an informed decision while choosing a name for their baby. To this end, we follow a recommender system approach and explore different methods for given name recommendation. Our final approach combines, in an ensemble, the individual rankings produced by simple collaborative filtering algorithms in order to produce a personalized list of names that meets the individual parents' taste.

Our experiments were conducted using real-world data collected from the query logs of Nameling (www.nameling.net), an on-line portal for searching and exploring names, which corresponds to the dataset released in the context of the ECML PKDD Discovery Challenge 2013. Our approach is intuitive, easy to implement, and features fast training and prediction steps.

This thesis explains all the research done by us. Unlike the publication of this research published in the conference (that basically explains how do we build our recommendation system for recommending given names), in this document we extend our explanation, adding some background knowledge, and we also explain other relevant information (such as those algorithm that did not perform well, as well as some ideas that would improve our results that were not implemented).

Additionally, we explain other approaches presented in the ECML PKDD Discovery Challenge 2013. The reader of this thesis, with this document, will have an extensive information of our work and, therefore, will have insightful information to solve the task of recommending given names.

# contents

# 1. Introduction

There are many considerations when parents are deciding on a name for their child. Many parents choose to name their babies after a grandparent, other relative, or a close friend. Some others pick names from the actors or actresses of their favourite soap opera. Cultural and societal rules, the meaning of the name, family's traditions, or religious beliefs also play an important role in many countries at the time of choosing a given name for a baby.

This is indeed a daunting task for the parents and their decision will mark the child for the rest of his or her life. The given name should be unique, making the bearer stand out from the crowd, but at the same time it should also avoid embarrassment of being the source for nicknames,humiliating initials, or annoying email addresses[1].

From thousands of names to choose from, how do parents pick the right one for their baby? In this work, we present an approach to help parents to deal with this information overload problem. We document all the studies done while doing this thesis, the algorithms used and our final implementation to recommend names to the parents. At the end, we take a recommender systems approach and show how an ensemble of simple collaborative filtering algorithms can help users to find given names that match their needs from a big pool of names.

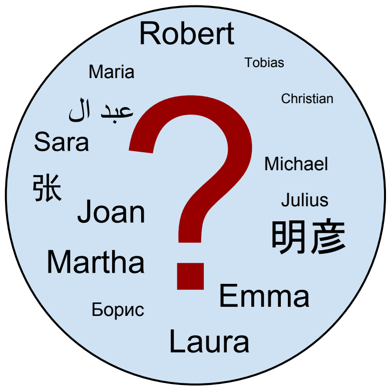


**Fig. 1.** The task of choosing a name can be complicated for the parents.

The task of recommending names using recommender systems is new, so practically there was no background studies in this field. For this reason, an extensive study in several aspects have been done to detect the characteristics of this problem (like the bests algorithms to use, or the way the data needs to be treated to remove the unnecessary information that can only introduce noise). All the conclusions of our research are explained in the document.

[1] such as the one of our friend *H. Thomas Eatons*, who has the (unfortunate) email address of `eatonsht@<anonymized>.com` :) .

The main contribution of this work is an intuitive approach for the task of given name prediction that is easy to implement, and that features fast training and prediction steps. Our study shows that, in this particular task, our ensemble of simple collaborative filtering building blocks performs significantly better than state-of-the-art latent factor models.

## 1.1. The Challenge

We conducted this study in the context of the European Conference on Machine Learning and Principles of Knowledge Discovery in Databases Discovery Challenge 2013[2] (ECML PKDD 2013). This thesis documents all the work done and the final approach of team "cadejo" on the off-line phase of the challenge. Our solution was on the top 5 in terms of performance, and our paper submitted was accepted for publication and presentation at the workshop.

The dataset used, the assessment metric for the recommendations and the properties of the evaluation data, as well as other restrictions that are imposed from the challenge remain in the thesis work. All this information is explained below.

## 1.2. The Dataset

The dataset provided for the off-line challenge that we use in this thesis is based on the query logs of *nameling* (`www.nameling.net`), an online portal for searching and exploring names. The collection comprises the time period from March 6th, 2012 to February 12th, 2013.

| | |
|---|---|
| interactions | 515,848 |
| users | 60,922 |
| names | 50,277 |

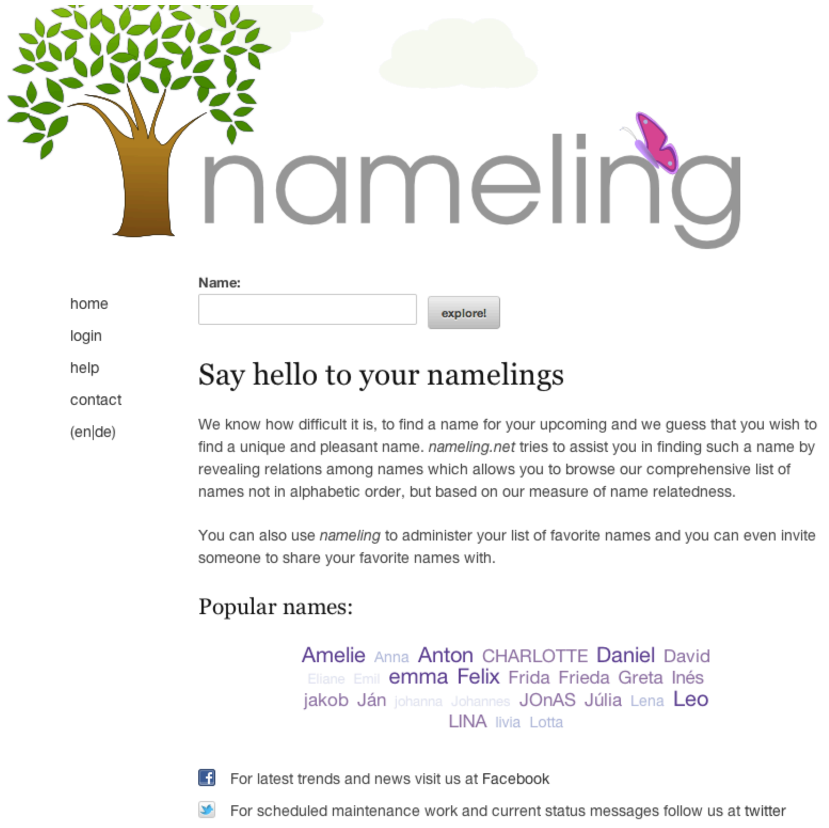
**Table 1. Information of the number of interactions (i.e. activities), different names and different users in the dataset.**

Figure 4a shows the frequency of names per user. We can observe that it corresponds to a characteristic graph of a long-tail distribution, where few names tend to concentrate a large number of users. The frequency of users per given name is shown in Figure 4b.

There are 5 different types of interactions within the dataset, which are described as follows:

1. ENTER_SEARCH: the user explicitly writes a name in the search field of Nameling's website in order to search for it.

[2] `http://www.kde.cs.uni-kassel.de/ws/dc13/offline/`



Name:

explore!

home

login

help

contact

(en|de)

Say hello to your namelings

We know how difficult it is, to find a name for your upcoming and we guess that you wish to find a unique and pleasant name. *nameling.net* tries to assist you in finding such a name by revealing relations among names which allows you to browse our comprehensive list of names not in alphabetic order, but based on our measure of name relatedness.

You can also use *nameling* to administer your list of favorite names and you can even invite someone to share your favorite names with.

Popular names:

Amelie Anna Anton CHARLOTTE Daniel David Eliane Emil emma Felix Frida Frieda Greta Inés jakob Ján johanna Johannes JOnAS Júlia Lena Leo LINA livia Lotta

For latest trends and news visit us at Facebook

For scheduled maintenance work and current status messages follow us at twitter

**Fig. 2. Main page of www.nameling.com.**

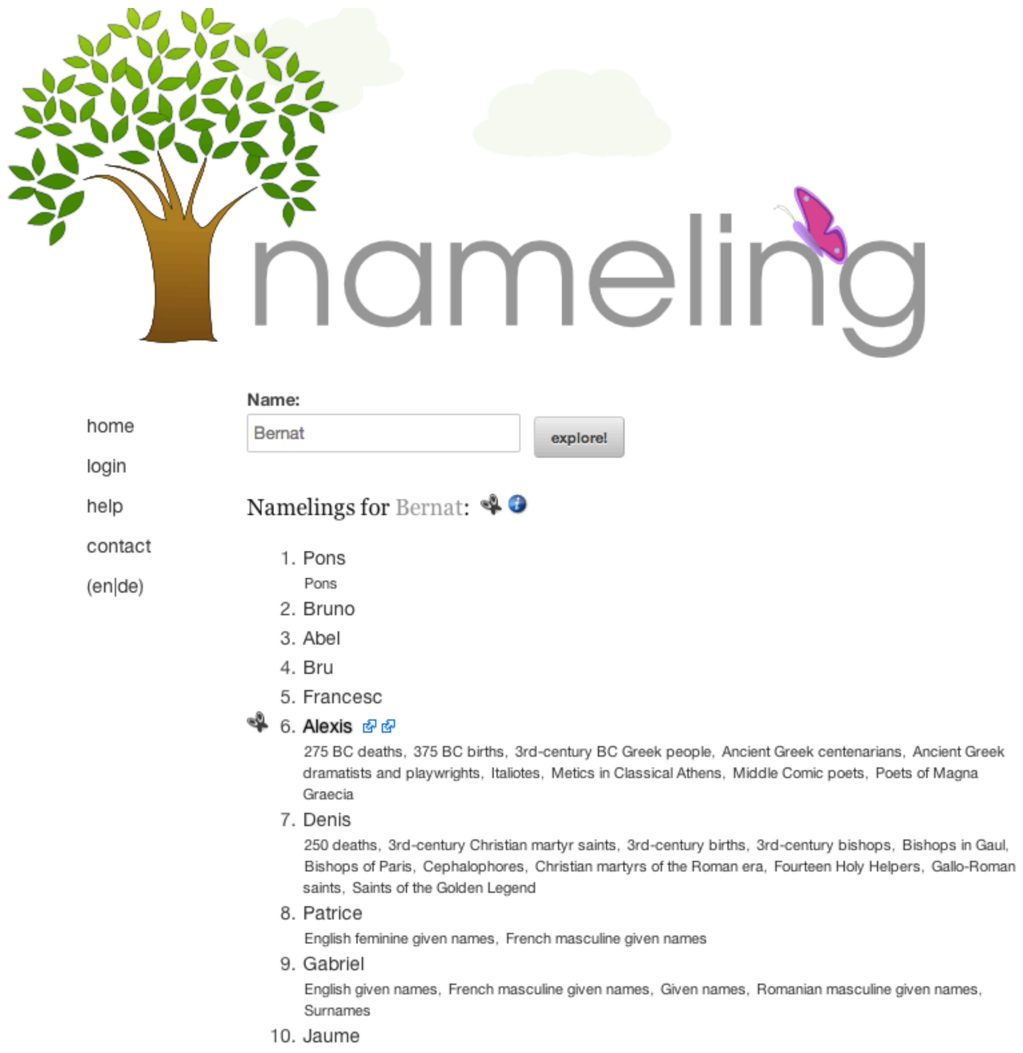
2. LINK_SEARCH: the user clicks on a name of showed at Nameling's website, following a link to a search result page.

3. NAME_DETAILS: the user requests more detailed information of a name.

4. LINK_CATEGORY_SEARCH: Wherever available, a name is categorized according to the corresponding Wikipedia article. Users may click on such a category link to obtain all names in the corresponding category.

5. ADD_FAVORITE: the user saves the name in his list of favourite names.

In addition to these datasets, there is a list of valid or *known names* provided by the organizers of the challenge which contains 36,436 given names, 3 files (one file for the German names, another one for the English names and the last one for the French names) containing a list of similar names for each name according to the Nameling's

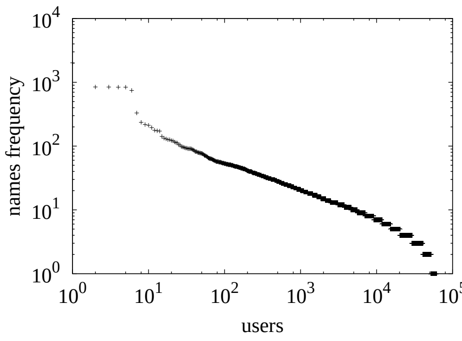


Namelings for Bernat:

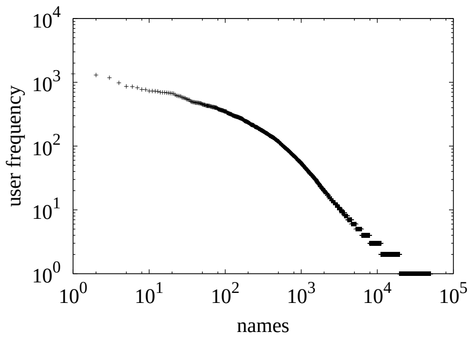
1. Pons
   Pons
2. Bruno
3. Abel
4. Bru
5. Francesc
6. Alexis
   275 BC deaths, 375 BC births, 3rd-century BC Greek people, Ancient Greek centenarians, Ancient Greek dramatists and playwrights, Italiotes, Metics in Classical Athens, Middle Comic poets, Poets of Magna Graecia
7. Denis
   250 deaths, 3rd-century Christian martyr saints, 3rd-century births, 3rd-century bishops, Bishops in Gaul, Bishops of Paris, Cephalophores, Christian martyrs of the Roman era, Fourteen Holy Helpers, Gallo-Roman saints, Saints of the Golden Legend
8. Patrice
   English feminine given names, French masculine given names
9. Gabriel
   English given names, French masculine given names, Given names, Romanian masculine given names, Surnames
10. Jaume

**Fig. 3. When a name is searched (in our case ”Bernat”), a list of names that Nameling consider similar to it are shown (”Pons”, ”Bruno”, etc.). If one of this names is clicked, a LINK_SEARCH is computed. In case we click the butterfly icon next to a name, the name is added as a favourite name (ADD_FAVORITE). If we click in one of the categories of a name (for example, the category ”250 deaths” under the name Denis), we are doing a LINK_CATEGORY_SEARCH. In case we click de information icon next to the search field, we are doing a NAME_DETAIL search of the searched name.**

similarity metric, and a third document with an approximated geographic position for some users.



(a) **Frequency of names per given user.**



(b) **Frequency of users per given name.**

**Fig. 4. Frequency of users and names.**

## 1.3. The Task

Our goal in this thesis is to present how to recommend to a user names for a child based on its preferences (in this case, names the user has already searched).

The recommender system's quality is evaluated with respect to the set of names that users have entered directly into Nameling's search field. The rationale to restrict the evaluation to ENTER_SEARCH activities is that all other user activities are biased towards the lists of names which were displayed to Nameling users.

The names we need to recommend are hidden by taking from the training data the chronologically last two, which had directly been entered into Namelings search field (i.e., ENTER_SEARCH activity) and which are also contained in the list of known names as detailed in the challenge description[3]. These names can not a favourite name (ADD_FAVORITE activity) for the user.

The assessment metric for the recommendations is Mean Average Precision at a cut-off of 1000 (MAP@1000) [6]. That is, for each test user look up the left-out names and take the precision at the respective position in the ordered list of recommended names. These precision values are first averaged per test user and than in total to yield the final score.

$$MAP := \frac{\sum_{q=1}^{Q} AveP(q)}{Q}. \tag{1}$$

MAP@1000 means that only the first 1,000 positions of a list are considered. Thus it might happen that for some test users one or both of the left out names do not occur in the list of recommendations. These cases will be handled as if the missing names were ranked at position 1001 and 1002 respectively.

[3] `http://www.kde.cs.uni-kassel.de/ws/dc13/faq/`

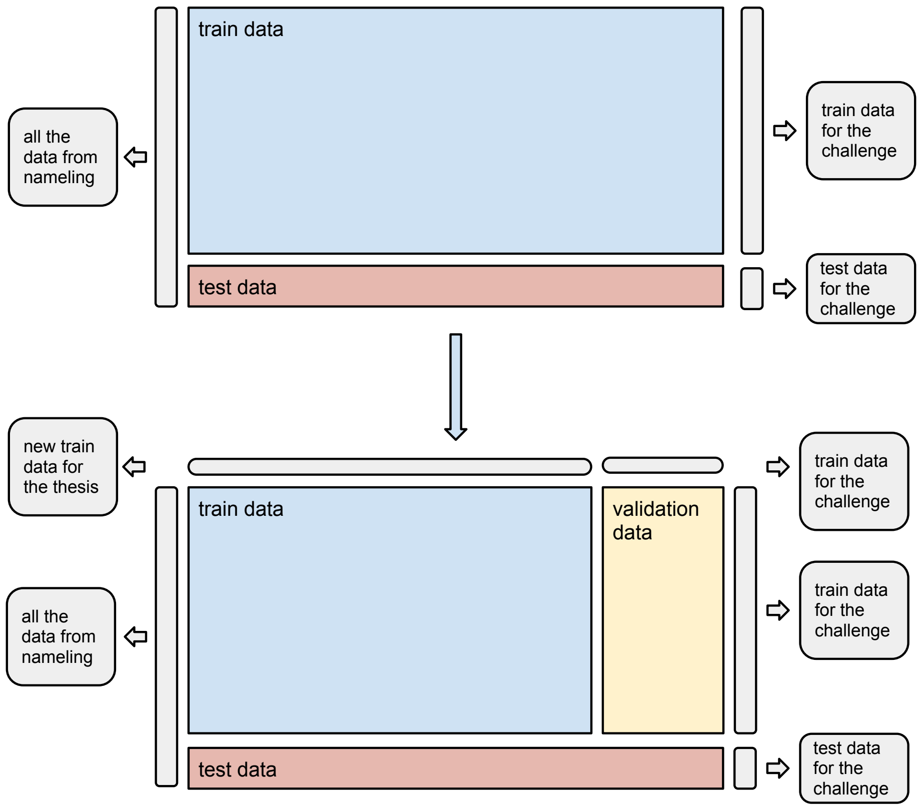

train data
all the data from nameling
train data for the challenge
test data
test data for the challenge
new train data for the thesis
train data
validation data
train data for the challenge
all the data from nameling
train data for the challenge
test data
test data for the challenge

**Fig. 5. Visual example of a test user search before splitting the data. To train our methods, the given training data was split again, creating a smaller training data and a validation data, that is going to be our ”test data”.**

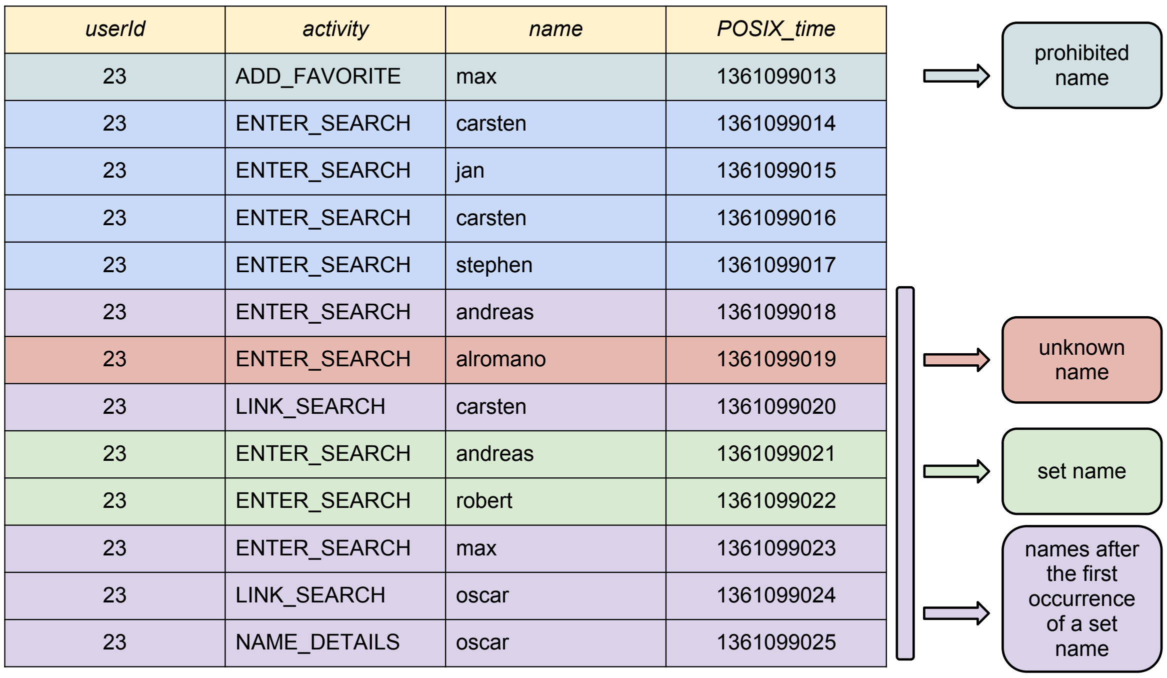
## 1.4. The test and the validation data for the experiments

As said before, the test set is built by taking from the training data the chronologically last two names which had directly been entered into Nameling's search field. But this process has several restrictions:

- Are only considered for the evaluation those names which had not previously been added as a favourite name by the user.
- All the remaining user activity after the (chronologically) first evaluation are discarded.
- Is required at least three activities per user to remain in the data set

An example to make it clear is seen in Figure 6. First of all, ”alromano” is not a known name for Nameling, so this name can not be considered as a possible name for the validation test. Also we can notice that the last user's search using the activity

ENTER_SEARCH is ”max”, but is also included as an ADD_FAVORITE name (the first activity of the user), and for this reason the name can not be included in the set name. Finally, the last two names searched using the ENTER_SEARCH are ”andreas” and ”robert” (these are going to be the test names). With the two test names selected, the next step is to remove all the user activity after the first appearance of a test name in users searches. In this case, all the data from the first appearance of ”andreas” is removed. Finally, what we receive (what is going to be our training data) is Figure 7.

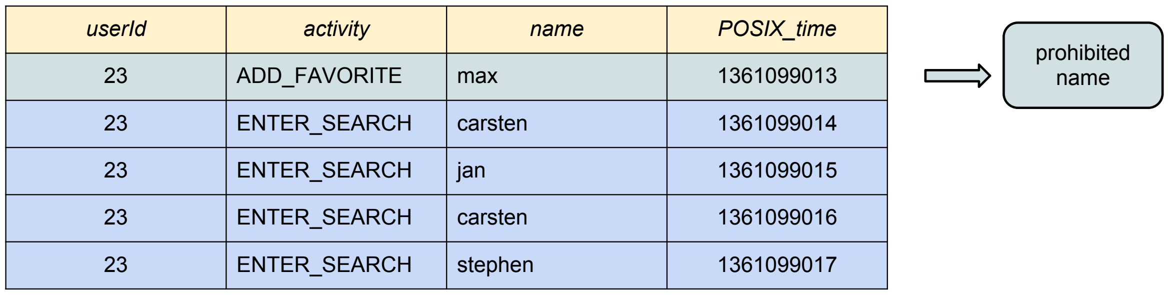


| userId | activity | name | POSIX_time |
|---|---|---|---|
| 23 | ADD_FAVORITE | max | 1361099013 |
| 23 | ENTER_SEARCH | carsten | 1361099014 |
| 23 | ENTER_SEARCH | jan | 1361099015 |
| 23 | ENTER_SEARCH | carsten | 1361099016 |
| 23 | ENTER_SEARCH | stephen | 1361099017 |
| 23 | ENTER_SEARCH | andreas | 1361099018 |
| 23 | ENTER_SEARCH | alromano | 1361099019 |
| 23 | LINK_SEARCH | carsten | 1361099020 |
| 23 | ENTER_SEARCH | andreas | 1361099021 |
| 23 | ENTER_SEARCH | robert | 1361099022 |
| 23 | ENTER_SEARCH | max | 1361099023 |
| 23 | LINK_SEARCH | oscar | 1361099024 |
| 23 | NAME_DETAILS | oscar | 1361099025 |

**Fig. 6. Visual example of a test user search before splitting the data. We can see that ”alromano” is not a known name for Nameling, and ”max” can not be a set name because of is already added as a favourite name by the user. For this reason, ”andreas” and ”robert” are going to be the hidden names.**

## 1.5. Data Preprocessing and Validation Set

In our study we could not find a clear mechanism on how to exploit activities of type LINK_CATEGORY_SEARCH, and therefore we drop such interactions from the dataset. We also concentrate only on names which appear as part of at least one interaction and that were also present in the known names list. In total our experiments consider a total number of 260,236 user-name pair interactions, from $|\mathcal{U}| = 60,922$



| userId | activity | name | POSIX_time |
|---|---|---|---|
| 23 | ADD_FAVORITE | max | 1361099013 |
| 23 | ENTER_SEARCH | carsten | 1361099014 |
| 23 | ENTER_SEARCH | jan | 1361099015 |
| 23 | ENTER_SEARCH | carsten | 1361099016 |
| 23 | ENTER_SEARCH | stephen | 1361099017 |

**Fig. 7. Visual example of a user search after splitting the data. As seen in Figure 6, ”andreas” and ”robert” are the test names, so all the searches after the first appearence of a hidden name is removed. We receive for training only the first 5 activities.**

different users and $|\mathcal{I}| = 17,467$ unique names. The mean of names per user is 4.35, the median is 3 names per user, with a minimum 1 and a maximum of 1670 names per user. All the names were converted to lower-case to avoid case-sensitive distinction.

In the challenge, we work with all the train data from Nameling, and the test names are hidden from us. However, for the experiments for the challenge and also for the thesis, the hidden names are needed to be known to be able to calculate the performance. For this reason, we split again the train data to obtain a smaller train data and a validation set to evaluate our results, using the same script used for the challenge (provided by Nameling). A visual explication is shown in Figure 5.

The script to split the data gives us a validation with 13,008 users and two target names. The first tests done with Bayesian Personalized Ranking Algorithm, were done using this validation file. In general terms, our 13,008 user validation performed better than the test of the challenge (around 20% better).

As is going to be explained in section 4, we changed our strategy to solve the problem, and we seized the moment to extend the validation adding the test users of the challenge not included in the validation. With this small change, the test data of the challenge was better represented in the validation data. After including the missing users in the validation, the performance of the methods tested in our validation (for the thesis) and the test set (for the challenge) were really similar (generally a difference lower than 5%).

From these 13,008 users, 2,264 are not within the 4,140 users in the test set, which are the ones we are required to give recommendations. In order to have a more representative cross-validation dataset, for each of these 2,264 users we also selected, from the remaining transactions in the training set, the last 2 names the user interacted with. Note that in this case we ignored the additional constraints imposed by the script, e.g., the type of activity.

## 1.6. Preliminaries

Before we start explaining methods, some explanation of the nomenclature is needed.

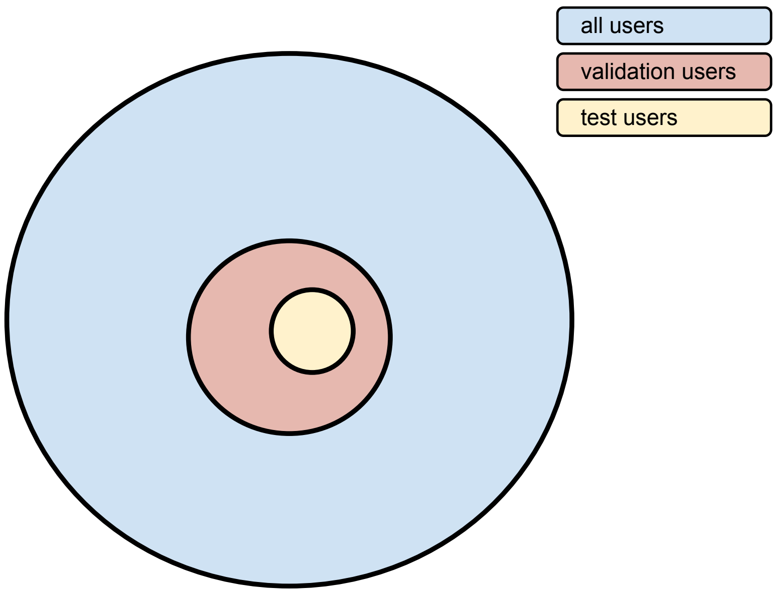


**Fig. 8. From the training set, we extracted the 2 last names from the users in the train validation that were not included in the validation set to make sure that all names of the test were included in our training validation.**

We consider the dataset as sparse matrix $\mathbf{X} = [x_{ui}]$, where we use through this paper the letter $u$ for users and $i$ for names, which corresponds to the items in a recommender system setting. We use bold letters for matrices and vectors, and non bold for scalars. The set of users and names[4] are denoted by $\mathcal{U}$ and $\mathcal{I}$, respectively. Predictions for user-item pairs are denoted as $\hat{x}_{ui}$. The set of names that the user has interacted with is written as $\mathcal{I}(u)$. The set of users, who interacted with name $i$ is $\mathcal{U}(i)$.

We use the notation $C_u(i)$ to represent the set of names co-occurring with name $i$ within $\mathcal{I}(u)$, that is, $C_u(i) := \{j \mid i, j \in \mathcal{I}(u) \land i \neq j\}$.

We denote the bag of co-occurring names for a give item $i$, as follows:

$$C(i) := \forall u \in \mathcal{U}, \{((i, j), m) \mid i \in \mathcal{I}(u) \land j \in C_u(i)\} ,$$

where $m((i, j)) : \mathcal{I} \times \mathcal{I} \to \mathbb{N}_{\geq 1}$ is a function from the set name pairs $(i, j) \in \mathcal{I} \times \mathcal{I}$ to the set $\mathbb{N}_{\geq 1}$ of positive natural numbers. For each pair of names $(i, j)$, $m((i, j))$ represents the number of occurrences of such pair in the bag, i.e., its *multiplicity*. The aggregated bag $C$ over all items corresponds to $C := \bigcup_{i \in \mathcal{I}(i)} C(i)$.

We use $S_u$ to represent the user $u$'s sequence of interactions ordered according to the corresponding time-stamp , e.g., if user $u$ searches first for name $i_1$, after that for $i_4$ and finally for name $i_2$, then his sequence $S_u$ is represented as:

[4] In this work, we use the terms “names” and “items” interchangeably.

$$S_u = i_1 \longrightarrow i_4 \longrightarrow i_2.$$

In some cases, the kind of search in $S_u$ is needed for an example of a method. In that case, the nomenclature for an item search in a $S_u$ using ENTER_SEARCH is $es_i$, using LINK_SEARCH is $ls_i$, using ADD_FAVORITE is $af_i$ and NAME_DETAILS is represented as $nd_i$.

Another nomenclature is used for the list of co-occurrence names. For example, consider three users $u_1$, $u_2$ and $u_3$, and their corresponding sequences $S$ of search actions in temporal order:

$$S_{u1} = i_1 \longrightarrow i_4 \longrightarrow i_2 \longrightarrow i_3$$
$$S_{u2} = i_4 \longrightarrow i_5 \longrightarrow i_1 \longrightarrow i_4 \longrightarrow i_3$$
$$S_{u3} = i_3 \longrightarrow i_5 \longrightarrow i_6 \longrightarrow i_7 \longrightarrow i_4$$

The bag of co-occurrences for item $i_4$, $C(i_4)$, sorted in decreasing order of multiplicity, is given by:

$$C(i_4) = \{((i_4, i_3), 3), ((i_4, i_1), 2), ((i_4, i_5), 2), ((i_4, i_2), 1), ((i_4, i_6), 1), ((i_4, i_7), 1)\}\ .$$

Some examples of methods that use the bag of co-occurrences are explained in 4.

We use the notation $CS_i$ for the list of the most similar names according to the Nameling's similarity metric of $i$. The list is given in decreasing order, meaning that the first item in the list is the most similar item. The list is represented as:

$$CS_{i1} = i_3, i_4, i_8, i_{21}, \ldots, i_{32}.$$

In this example, the most similar name to $i_1$ would be $i_3$, the second most similar would be $i_4$, and go on.

In some examples, we explain how works an algorithms giving its result (the list of names the user would receive after using this algorithm). The nomenclature for these recommended list would be $R(u)$. For example, a recommended list for $u_1$ where $i_1$, $i_2$ and $i_3$ are recommended would be:

$$R(u_1) = [i_1, i_2, i_3]\ .$$

# 2. Background

## 2.1. Recommender systems

For the last lustrums, the relation between internet and the users have change a lot. Originally, Internet was a tool for sending information between two distant points. Now, Internet has become a system where the users can purchase products, share their interests and opinions and consume culture thanks to the many services that are available.

The way the users interacts with Internet is influenced by the way they think, act and their preferences in life. That means that with the information how a user use Internet, we can create an accurate profile of him. These information is really valuable for the companies that provide these services. With the proper analysis of their data, they can understand their users, and offer something that fits everyone better.

The recommender systems is the engine that study that amount of data to be able to predict the needs and preferences of the users. There are different approaches for the recommender systems depending on several things. These systems can be classified according to:

- The technique used to extract information (Collaborative filtering vs. Content-Based filtering)
- The way a user and and item interacts (Implicit vs. Explicit data)
- How the system have to recommend the items to the users (N-items vs. Ranking)

### 2.1.1. Collaborative filtering vs. Content-Based filtering

The main idea in the recommender systems are to extract some pattern based on the data obtained that suits the users we want to recommend.

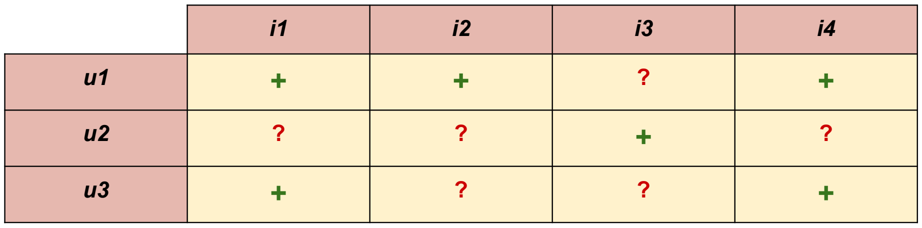
With the collaborative filtering strategy, the system uses techniques to identify collaborations and common features between users and items.

Being $u_1$, $u_2$ and $u_3$ three users, and $i_1$, $i_2$, $i_3$ and $i_4$ four items, and $Int(u)$ the list of interactions (purchases) user-item user $u$ has done:

$$Int_{u1} = i_1, i_4, i_2$$

$$Int_{u2} = i_3$$

$$Int_{u3} = i_1, i_4$$



| | i1 | i2 | i3 | i4 |
|---|---|---|---|---|
| u1 | + | + | ? | + |
| u2 | ? | ? | + | ? |
| u3 | + | ? | ? | + |

**Fig. 9. + represents an interaction between a user and an item, ? otherwise. In this case, we can see that $u_1$ and $u_3$ are quiet similar**

The collaborative filtering algorithm would detect that $u_1$ and $u_3$ are quiet similar (see Figure 9). They have in common that they have purchased $i_1$ and $i_4$. In contrast, $u_2$ purchased only $i_3$. For this reason, if a recommendation to $u_3$ is needed, it would not be unreasonable to believe that the system would recommend to him $i_2$, the item from $u_1$ that $u_3$ has not purchased yet.

On the other hand, the content based filtering uses explicit information about users and items. For example, in a music streaming service, a content based recommendation would be to recommend to every user the most popular (listened) songs from their country.

### 2.1.2. implicit vs. explicit data

When a user rates the items he purchased, the data is explicit. On the other hand, when we only have interactions between users and items without any kind of rating, the data is implicit.

The explicit data is (by and large) easier to handle. The system knows what a user purchased, but also if the user liked the item (and how much). With the implicit data, the system only knows that an interaction between an user and an item exist, but no rate is known.

As an example:

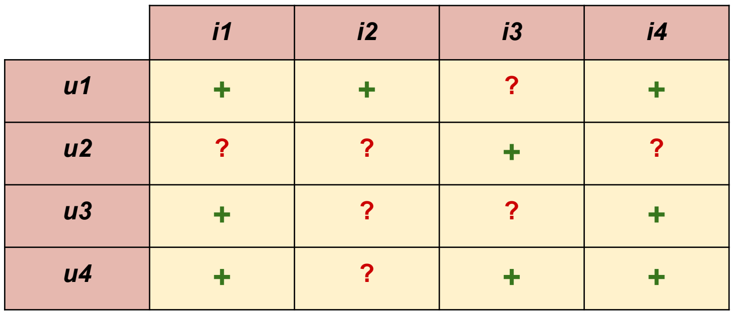
$$Int_{u1} = i_1, i_4, i_2$$

$$Int_{u2} = i_3$$

$$Int_{u3} = i_1, i_4$$

$$Int_{u4} = i_1, i_3, i_4$$

As said before, $Int(u)$ the list of interactions user-item user $u$ has done. If we want to recommend an item to $u_3$, we would have to choose between $i_2$ and $i_3$ (the items from the data that $u_3$ has not purchased). As we can see (Figure 10), the users that are similar to $u_3$ are $u_1$ and $u_4$. Both have purchased the items purchased by $u_3$, one of them have purchased $i_2$ and the other one $i_3$. With these information, we can not choose properly which one could fit $u_3$ preferences better.



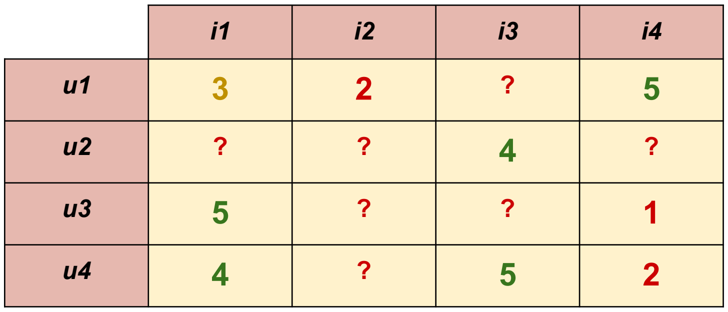
| | i1 | i2 | i3 | i4 |
|---|---|---|---|---|
| u1 | + | + | ? | + |
| u2 | ? | ? | + | ? |
| u3 | + | ? | ? | + |
| u4 | + | ? | + | + |

**Fig. 10. With this information, we can see that because of their positive interactions (represented by a +), $u_1$ and $u_4$ are more similar to $u_3$ than $u_2$.**

On the other hand, with explicit data (Figure 11), we have the additional information of the scores. As said, we can know how much an item liked a user.

In our example, now we have extra information to choose between $i_2$ and $i_3$ (or what is in this example the same thing, to choose which user $u_1$ or $u_4$ is more similar to $u_3$). We can see that the scores given by $u_3$ and $u_4$ to the items purchased in common ($i_1$ and $i_4$) are more similar that the scores given by $i_1$ and $i_3$. For this reason, now we can say that the system would recommend to $u_3$ the item purchased by $u_3$, $i_3$.

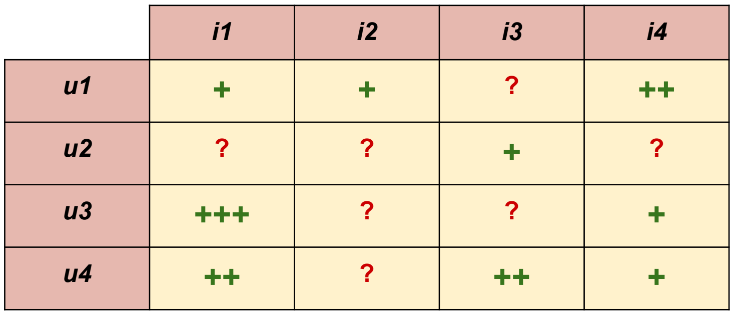
As we have seen with the examples, the explicit data gives more information than the



| | i1 | i2 | i3 | i4 |
|---|---|---|---|---|
| u1 | 3 | 2 | ? | 5 |
| u2 | ? | ? | 4 | ? |
| u3 | 5 | ? | ? | 1 |
| u4 | 4 | ? | 5 | 2 |

**Fig. 11. The scores in the explicit data allows us to say that $u_4$ is more similar than $u_1$ to $u_3$**

implicit data. This handicap for implicit data can be palliate depending on the kind of implicit interactions that we have. For example, if the data reflects how many times an user has interacted with an item or there are more than one type of interaction (just like in the Discovery Challenge, where we consider that one interaction is more important than the others), depending on the case this fact can represent a penchant, that is not an information as reliable as an score, but can determinate which item the system would recommend in case of doubt. Figure 12 is an example of these concept, where we could consider that $u_3$ is more similar to $u_4$ than $u_1$ because the first two have interacted more than once with $i_1$, when $u_1$ have interacted only once with this item. Moreover, $u_1$ has interacted twice with $i_4$, when both $u_3$ and $u_4$ have interacted with that item only once.



| | i1 | i2 | i3 | i4 |
|---|---|---|---|---|
| u1 | + | + | ? | ++ |
| u2 | ? | ? | + | ? |
| u3 | +++ | ? | ? | + |
| u4 | ++ | ? | ++ | + |

**Fig. 12. In some cases, some information extracted from the data (for example, the kind of interaction between users and items or in our example, the number of them) can give us extra information that can improve our recommendations**

### 2.1.3. N-items vs. ranking

The way the system recommends the items to a user may differ depending on the situation. In some cases, the system has to recommend an unordered amount N of items to a user. Exist other cases where the system have to recommend a ranked N items to a user, where some items are more recommended than others. The algorithm has to be adapted to it.

In case the system recommends a ranked list of items, the algorithm needs to build a structure of direct preferences between items to create a ranking, because the system needs to set ordered priorities between items. On an unranked list, the system can use other strategies to recommend N items (it is not necessary to know if one item is better than another, only if it can be recommended to the user because fits its pereferences).

In our case of recommending names, we need to find a good solution for a collaborative filtering, implicit data, ranking case.

## 2.2. The algorithms

In this work, several algorithms are tested. The methods used are all collaborative filtering algorithms. These algorithms are detailed below, classified into different groups.

### 2.2.1. Matrix Factorization Collaborative Filtering

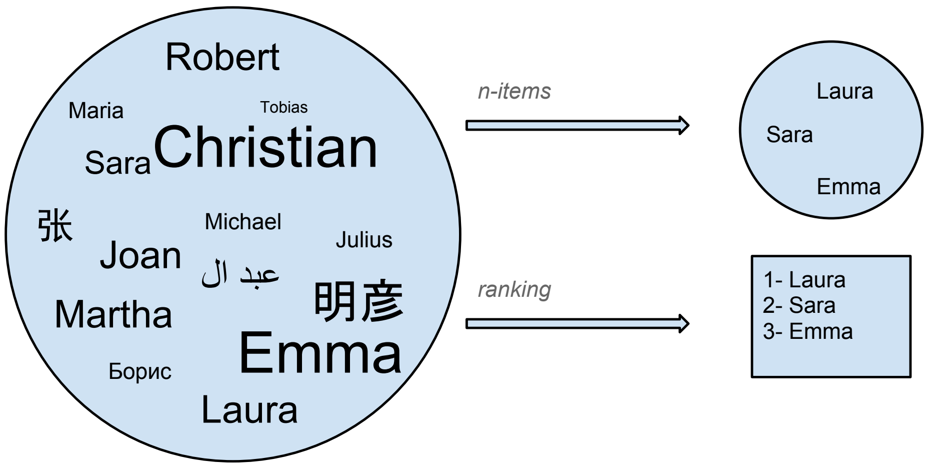


**Fig. 13. The way we have to give the recommendations conditions the algorithm. In the first case (n-items), there is no order, but in the ranking recommendation, Laura is considered a better recommendation than Emma.**

Some of the algorithms tried in this work are Matrix Factorization algorithms. The idea of these algorithms is to represent all the data (users, items and their interactions), that can be represented by a large and sparse matrix, as a product of two low-rank matrices.

In Figure 15 there is a visual explanation of this concept. The first matrix (with all the interactions between users and items) is large and sparse. For this reason, extracting information from this source can be really slow, with a lot of unnecessary data. The idea behind the Matrix Factorization is to represent all the interaction information with two smaller matrices that represent the users and the items.

Thus the representation of the data user-item created by the algorithm is the matrix M. Is obtained from the factorization of the information of the data into two matrices W and H like

$$M = WH^T$$

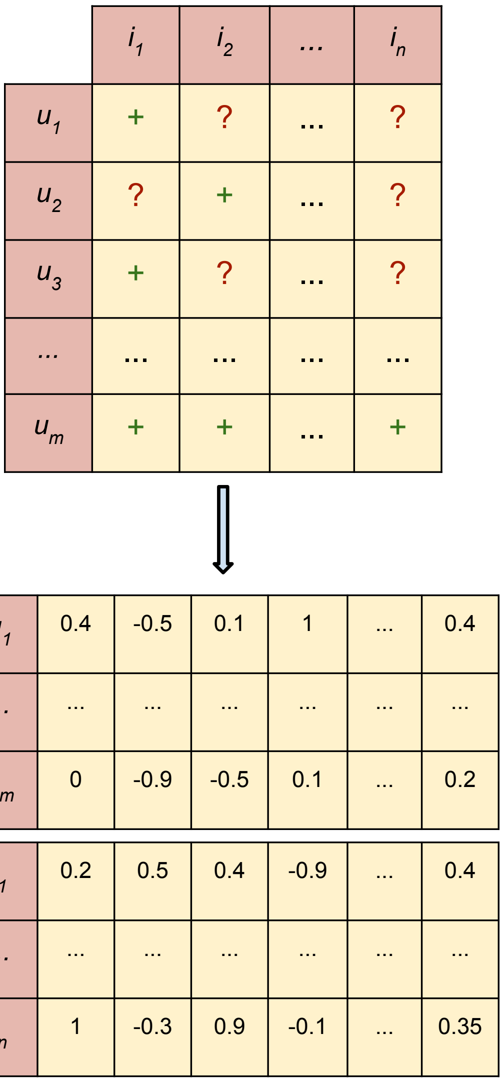
which represents abstract features of the users and the items correspondingly. To obtain the recommendations, the dot product of the vector that represents the user in W and the vector that represent an item in H gives a result. This prediction formula can be written as

$$\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if}$$

For example, imagine that $f_i$ and $f_u$ are the vectors that represent an item and an user in H and W.

$$f_{i1} = [0.2, 0.45, -1, 0]$$

$$f_{i2} = [1, -0.5, 0.2, 0]$$



|  | $i_1$ | $i_2$ | ... | $i_n$ |
|---|---|---|---|---|
| $u_1$ | + | ? | ... | ? |
| $u_2$ | ? | + | ... | ? |
| $u_3$ | + | ? | ... | ? |
| ... | ... | ... | ... | ... |
| $u_m$ | + | + | ... | + |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| $u_1$ | 0.4 | -0.5 | 0.1 | 1 | ... | 0.4 |
| ... | ... | ... | ... | ... | ... | ... |
| $u_m$ | 0 | -0.9 | -0.5 | 0.1 | ... | 0.2 |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
| $i_1$ | 0.2 | 0.5 | 0.4 | -0.9 | ... | 0.4 |
| ... | ... | ... | ... | ... | ... | ... |
| $i_n$ | 1 | -0.3 | 0.9 | -0.1 | ... | 0.35 |

**Fig. 14. The idea of the Matrix Factorization consists to factorize the matrix with the interactions into smaller matrices.**

$$f_u = [0.75, -0.1, 0, 1]$$

$i_2$ would be recommended before $i_1$ to $u_1$ because the dot product result of $u_1$ and $i_2$ is bigger than the dot product result of $u_1$ and $i_1$ ($\hat{x}_{11} < \hat{x}_{12}$, that is $0.205 < 0.8$).

### 2.2.2. Neighbourhood-Based Collaborative Filtering

Neighbourhood-Based recommendation is a classic approach for Collaborative Filtering that still ranks among the most popular methods for this kind of problem. These approaches are quite simple to describe and implement featuring important advantages such as the ability to explain a recommendation and to capture "local" relations in the data, which are useful in making serendipitous recommendations.

The basic idea of this algorithm is to label an element (item or user) based on their closest training samples in distance to it (see 15). The number of samples used to determine the classification can be predefined (the most common algorithm is the K-Neighbourhood) or based on the local density of the points (for example, the Radius-Based Neighbourhood).

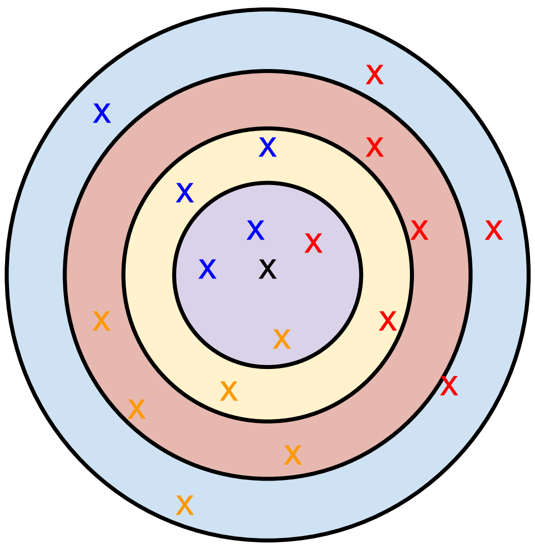


**Fig. 15. A typical visual representation of the Neighbourhood-Based CF strategy, where a user/item gets a classification depending of the information of the closest users/items (the neighbours).**

The neighbourhood algorithms can be User-Based or Item-Based. User-Based algorithms were the first neighbourhood algorithms used. These algorithms are made based on user to user similarity (how similar are the items two users have purchased). On the other hand, Item-Based algorithms, are based on the similarity between items (how similar are the items a user purchased directly with other items). Broadly, there are less items than users in a system, and the relationship between items is quite static, allowing the Item-Based algorithms to be more computational efficient and as powerful as the User-Based algorithms.

### 2.2.3. Item-to-Item Collaborative Filtering

This approach for item recommendation is based on the classic Item-Based collaborative filtering algorithm introduced by Amazon.com [20]. This kind of Collaborative Filtering algorithms work really well, with a really good performance in cases with a very large customer bases and large catalogues (like Amazon.com), being really scalable.

Rather than matching the user to similar customers, the item-to-item collaborative filtering algorithm builds a relationship between items, that is finding items that users tend to purchase together. With this approach, we avoid to compute a similarity metric between item pairs that do not tend to be chosen together by an user.

```
For each item in product catalogue, I1
    For each customer C who purchased I1
        For each item I2 purchased by customer C
            Record that a customer purchased I1 and I2
For each item I2
    Compute the similarity between I1 and I2
```

**Fig. 16.** Pseudocode in [20] for the item-to-item algorithm

In our case, we adapt the algorithm. The algorithm matches each user's interaction with a name to a set of similar names, then combines those similar items into a recommendation list.

### 2.2.4. Slope One predictor

The Slope One recommendation algorithm computes the differences of interaction intensities between items. With this information, tries to predict the interaction intensity between users and items without interactions. Having an interaction between a user *u* and an item *i*, estimate the type of interaction between the same user *u* and another item *j* based on the relationship of preference between *i* and *j* in the system. The simplest example to explain the slope one recommendation is explained in the Figure 17

This algorithm has performed well in several recommendation challenge, being easy to implement, efficient at query time, dynamically updateable and can compete against Memory-Based schemes that needs lots of memory.

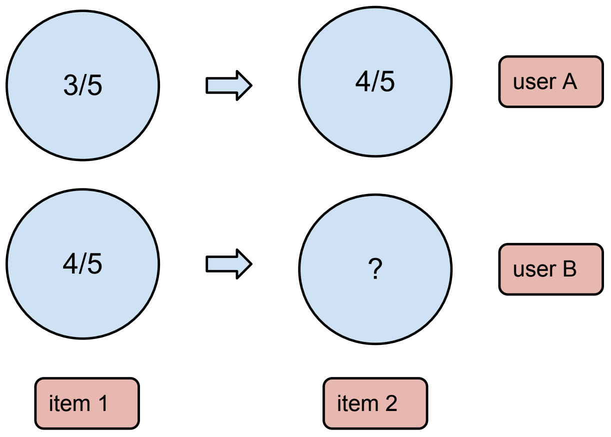


**Fig. 17. Basis of Slope One algorithm: The predicted rate of item 2 by user B would be 5/5 based on user A rates. The score difference of item 1 and item 2 for user A is 1, so we can predict that user B will score item 2 like item 1 + 1.**

### 2.2.5. Page Rank

The Page Rank algorithm was introduced in 1998 [10], and it is the algorithm behind the search engine Google.

The idea behind this algorithm is to rate how important is an item in the system based on the mentions of the other items in the system. In case of web pages, a web page is going to be well-ranked when a lot of pages have a link to that page. Moreover, these“votes” will be weighted based on the popularity. That means that a reference from a popular item (an item referenced lots of times) is more important than a less popular item.

A visual example to understand the basic idea is shown in Figure 18.

This algorithm is designed to scale easily, and is efficient in both space and time. As is explained in [10], the algorithm was built to be used in the search engine, and its implementation tries to avoid bottlenecks in CPU, memory access and memory capacity.

## 2.3. The metric: MAP

The metric used to calculate the performance is the *Mean Average Precision*. This metric gives as a result the precision in a ranked list of recommendations. The equation to calculate the performance is

$$MAP := \frac{\sum_{q=1}^{Q} AveP(q)}{Q}, \tag{2}$$

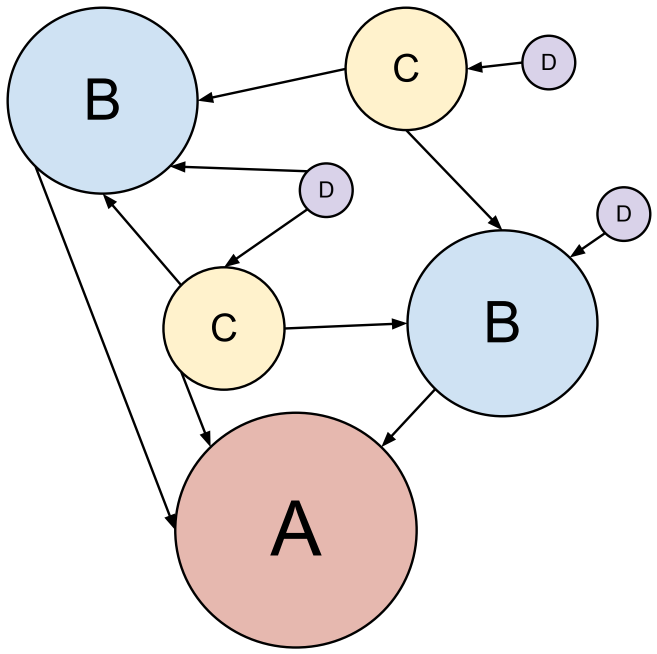


**Fig. 18. Visual example of the idea behind Page Rank. An item is ranked depending on how many items refer to it and the importance of them. For example, A is the most important item (is the bigger one) because is one of the items with more votes (represented with arrows) and those who have voted for A are quite important (both B and one C)**

The precision $P(q)$ of a recommended name is based on its position in the recommended list. $Q$ are the number of recommendation list (in our case users) we need to recommend.

In our work, in case the names to recommend are not included in the 1000 names list, they will be considered as if they were included in the 1001st and in the 1002nd position. The best way to illustrate this metric is using an example:

Imagine we have to recommend 2 different names to two users $(u_1, u_2)$. Our recommendation list could be

$$R(u_1) = [i_3, i_5, i_{436}, \ldots, i_7, i_{200}]\ .$$
$$R(u_2) = [i_1, i_{500}, i_6, \ldots, i_{43}, i_{21}]\ .$$

Imagine that the names that have to be recommended for $u_1$ are $i_{436}$ and $i_7$, that are in position 3 and 999 respectively. In that case, the MAP would be (1/3 + 2/999)/2 = 0.167668. On the other hand, the names expected by $u_2$ would be $i_1$ and $i_{543}$. The first recommended name is $i_1$, but the second name is not in the list. The performance in that case would be 1/1 = 1, but it would be incongruous because only one name is

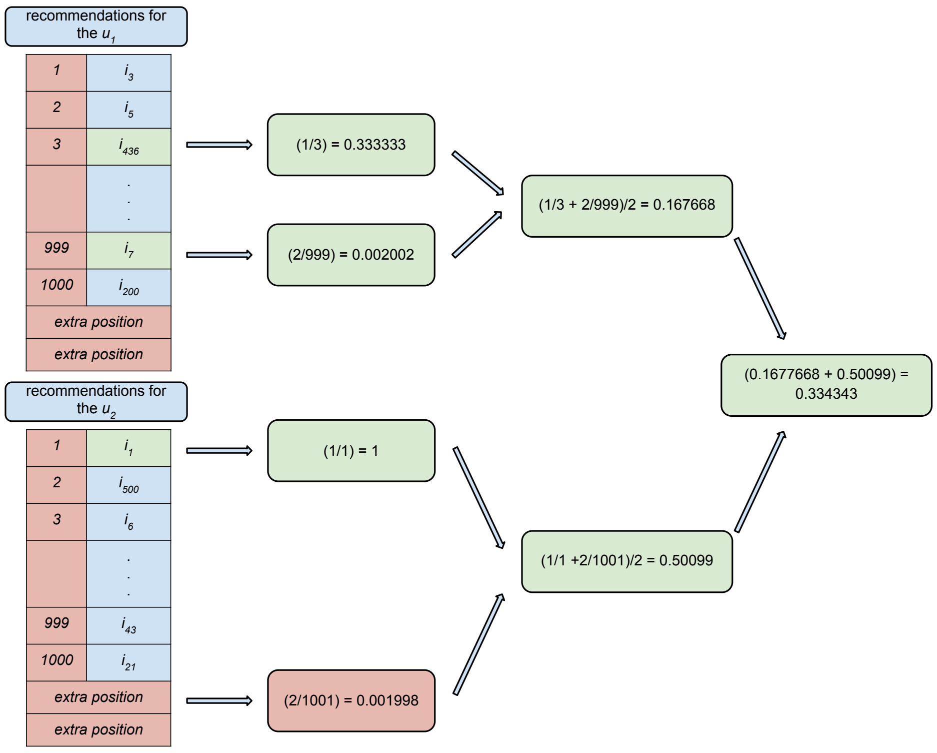


**Fig. 19. Representation of mean average precision in our work.**

recommended. For this reason, the names that are not hit in the list are considered in the 1001st and 1002nd position. In $u_2$, the performance would be (1/1 +2/1001)/2 = 0.500999. Finally, the average performance would be 0.334343. In Figure 19 there is a graphical representation of this process.

# 3. Related Work

Although top-*N* recommender systems have been studied in depth, the particular task of recommending given names is rather new. Lately, some papers have been published trying to solve this problem, most of them from the researchers that are involved in the Nameling website. In this chapter, we are going to check out that documents, especially those papers accessible in the website of the ECML PKDD'13 Discovery Challenge that have provided us a lot of information useful for the work. The two papers that we are going talk about are written by Folke Mitzlaff and Gerd Stumme.

Not only the papers that are going to be explained in this section have been useful for us. A lot of papers, investigations and documents have been read and used to complete this thesis. However, we are going to explain here only the documentation related directly to the given names recommendation task.

The other collaborative filtering algorithms and research applied in this thesis are referenced in this document when is necessary.

## 3.1. Onomastics 2.0

This work ([22]) shows how basic approaches of social network analysis, information retrieval and data mining can be applied for discovering relations among names.

This document has an extensive study of the co-occurrence networks, that is when two names occurres at the same time in a document or a file. In this paper, they use as a source to create the co-occurrence network the web page "Wikipedia". The experiments are done using two kind of co-occurrence networks, one for given names and one for city names. The results show the importance of co-occurrence networks for the recommendation task, and its importance to create relationships between names.

## 3.2. Recommending given names

This document ([23]) focuses on the recommending given names task. The research in this paper is less theoretical than the other one, or at least it has practical purpose (the name search engine "Nameling", used in the Discovery Challenge, was built after this research).

In this paper, several recommender systems approaches like User Based and Item Based Collaborative Filtering, PageRank and some Matrix Factorization Methods are tested.

One of the most interesting parts of this document is its approach of PageRank used to recommend names. The NameRank algorithm introduced in this paper adapts FolkRank [17] for name recommendation, showing promising results. The algorithm basically solves a personalized version of PageRank [10] per user in the system, over a graph of names, which does not scale gracefully to large-scale data.

# 4. Empirical evaluation

Collaborative Filtering (CF) algorithms are best known for their use on e-commerce Web sites, On-line Social Networks, or Web 2.0 video sharing platforms, where they use input about a user's interests to generate a (ranked) list of recommended items. For this reason, this is not strange that we are going to try to solve the task of recommending names using Collaborative Filtering algorithms. In this section, we describe all the Collaborative Filtering models which are tried in our work.

Our strategy changed during our tests. For this reason, this section will be divided in two groups. The first part will explain the tests using the Bayesian Personalized Ranking (4.1). The second part will explain all the other methods tried to create a recommender ensemble (4.2).

As explained in 1.5, the first tests are done using the validation with only 13008 users. As said before, we noticed that 2,264 test users in the challenge were not included in the first validation. To obtain a better representation of the test data in the second part of the experiments, we tried to add manually those missing users. Most of them are included, but there are a few users (around 60) from the test data that are not included in the extended validation data, because they do not have enough interactions to obtain two hidden names (and the script provided by the challenge to obtain the performance needs two recommendations per user).

## 4.1. Bayesian Personalized Ranking

Our first strategy to solve the challenge of recommending names was to use the Bayesian Personalized Ranking. This algorithm, introduced by [25], is an optimization criterion and learning algorithm for personalized ranking, which is derived from a Bayesian analysis of the problem. This learning algorithm can be applied to the two state-of-the-art recommender models of Matrix Factorization and Adaptative-KNN. In our case, we used the Matrix Factorization model.

$$M = WH^T$$

**Fig. 20. As already said in 2.2.1, the matrix M with all the interactions information into smaller matrices that represents the users and the items.**

The users and the items in this learning algorithm are represented by 2 matrices (see 20). Every row of a matrix describes using k dimensionality the latent features of a user or item. The basic idea of the BPR learning process is to compare positive feedbacks of item-users against negative feedbacks, and update the matrices. The learning model is based on stochastic gradient descent with bootstrap sampling. The matrices ($\Theta$) that represents the users and the items are initialized randomly, and their updates are performed with the equation

$$\Theta \longleftarrow \Theta + \alpha(\frac{e^{-\hat{x}_{uij}}}{1 + e^{-\hat{x}_{uij}}} \frac{\delta}{\delta\lambda} \hat{x}_{uij} + \lambda_{\Theta}\Theta) \tag{3}$$

where $\hat{x}_{uij}$ is the estimator defined as $\hat{x}_{ui}$ - $\hat{x}_{uj}$. The prediction of a pair user-item is

$$\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{1}^{k} w_{uf} \cdot h_{if} \tag{4}$$

and $\lambda$ is the regularization value for the users and the items and $\alpha$ is the regularization rate. The possible values for $\frac{\delta}{\delta\lambda}\hat{x}_{uij}$ in the matrix factorization case are defined in Table 3.

The Bayesian Personalized Ranking allows to be configured changing the values of $\alpha$ (regularization rate), $\lambda$ (the normalization rate), $k$ (the number of latent values that represent a user and an item) and the *epochs* (times the loop is done). Depending on the data that is wanted to be represented with matrices, the best configuration of the values may differ in each case, so some test are always needed to be done to find the best values that fit the problem.

| **method** | **performance** |
|---|---|
| Random names | 0.001873 |
| Most popular names | 0.027843 |

**Table 2. Performance of the Random and Most Popular names recommendations. Generally, the performance in the Random Names recommendation performs less than 0.002 in MAP@1000. 0.001873 is the performance in one of the random solutions computed.**

| | | |
|---|---|---|
| $(h_{if} - h_{jf})$ | if | $\theta = \omega_{uf}$ |
| $\omega_{uf}$ | if | $\theta = h_{if}$ |
| $\omega_{uf}$ | if | $\theta = h_{jf}$ |
| 0 | else | |

**Table 3. Possible values for the derivatives in the Matrix Factorization BPR.**

**Algorithm 1** BPR LEARNING ALGORITHM

```
Input:
1: procedure learnBPR($D_S$, $\Theta$)
2:     initialize $\Theta$
3:     repeat
4:     draw $(u, i, j)$ from $D_S$
5:     $\Theta \longleftarrow \Theta + \alpha(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \frac{\delta}{\delta\lambda}\hat{x}_{uij} + \lambda_\Theta \Theta)$
6:     until convergence
7:     return $\hat{\Theta}$
8: end procedure
```

The tests done using the Bayesian Personalized Ranking learning method are divided into two groups. The first one explains the results of the BPR approach using only a Matrix Factorization recommender model, and the second one tries to boost the performance of the algorithm using the Ranklib library.

To compare the results of the BPR algorithm, we consider as a baseline recommendation the top 1000 most searched names. Its performance is 0.027843 (as a reminder, our validation data has 13,008 users). In the test, the performance of the same baseline approach is 0.0263. To keep it in perspective, the random recommendation performs less than 0.002 (Table 2) in the validation set.

### 4.1.1. Matrix Factorization approach

Our first attempt with this algorithm was to recommend a list of names to the test users using (only) the learning algorithm to create the model. With the learned model, with two matrices that represents the users and the item, the recommendation list would be created with the dot product of the vectors.

The first test using BPR were executed only to get a first impression of a general performance of the algorithm. In these cases, the parameters were *epochs* = 5, $k$ = 64, $\alpha$ = [0.1, 0.01, 0.001], and $\lambda$ = [0, 0.1, 0.01, 0.001]. The data used to train were not always the same every time. We tried to train using all the known names (36,436), the known names in the train data (17,467), and sometimes less names (such training the model with only the top 10,000 names), because we wanted to know the behaviour of the algorithm changing the number of items (for example, if a very popular name can introduce noise, an unpopular name give valuable information, etc.). The results, in all the cases, were bad (lower than recommending the top 1,000 names). For these reason, we did some specific test using only the known names existing in the train data and *epochs* = 15 to be sure of the low performance of this algorithm. Table 4 has all the performances.

In these new tests, the performance of the BPR were still really low. This is not a surprise, because this poor performance is also observed in at least another paper for this kind of task ( [23]). However, more test were done to confirm this. In these test we modified the $k$ parameter and the *epochs* (the times the algorithm is repeated until convergence).

| lr | regU | epochs | k | MAP@1000 |
|---|---|---|---|---|
| 0.001 | 0 | 15 | 64 | 0.001916 |
| 0.001 | 0.001 | 15 | 64 | 0.001789 |
| 0.001 | 0.01 | 15 | 64 | 0.001679 |
| 0.001 | 0.1 | 15 | 64 | 0.001614 |
| 0.01 | 0 | 15 | 64 | 0.014992 |
| 0.01 | 0.001 | 15 | 64 | 0.014572 |
| 0.01 | 0.01 | 15 | 64 | 0.012687 |
| 0.01 | 0.1 | 15 | 64 | 0.001589 |
| 0.1 | 0 | 15 | 64 | 0.015533 |
| 0.1 | 0.001 | 15 | 64 | 0.017394 |
| 0.1 | 0.01 | 15 | 64 | 0.012196 |
| 0.1 | 0.1 | 15 | 64 | 0.018090 |

**Table 4. Recommendation performance for every Bayesian Personalized Recommender test in terms of MAP@1000.**

| lr | regU | epochs | k | MAP@1000 |
|---|---|---|---|---|
| 0.1 | 0.1 | 15 | 64 | 0.018090 |
| 0.1 | 0.1 | 15 | 96 | 0.001789 |
| 0.1 | 0.1 | 30 | 96 | 0.022404 |
| 0.1 | 0.1 | 35 | 64 | 0.018627 |
| 0.1 | 0.1 | 20 | 64 | 0.021869 |
| 0.1 | 0.1 | 20 | 80 | 0.021380 |
| 0.1 | 0.1 | 15 | 160 | 0.019148 |
| 0.1 | 0.001 | 20 | 64 | 0.015154 |
| 0.1 | 0.001 | 15 | 96 | 0.016438 |
| 0.1 | 0.1 | 20 | 96 | 0.019905 |
| 0.1 | 0.001 | 20 | 96 | 0.014468 |

**Table 5. Recommendation performance for every Bayesian Personalized Recommender test in terms of MAP@1000.**

In spite of some better performance (see Table 4.1.1), the performance were substantially worse than our baseline recommendation. For this reason, we consider that the Bayesian Personalized Ranking learning algorithm is not appropriate for this kind of challenge, but maybe it could be its learned model useful with the ranklib library.

### 4.1.2. The ranklib approach

In spite of the results using BPR were worse than a simple top frequency recommendation list, we considered that we could take profit of the learned model using it with the Ranklib library.

Ranklib is a Java library of learning to rank algorithms. Currently, eight different algorithms have been implemented, and also several retrieval metric are available, but in our work we are going to use only the non-derivative optimization algorithm Coordinate Ascent and the metric MAP.

The reason why to use this library was to boost the performance of our recommendation adjusting the recommendation list to the metric used in the challenge (mean average precision) to increase the final performance. Giving to the library a file to train with a list of interactions with the structure $score-qid-data$ (where *score* is the weight we give to the interaction, *qid* is the id of the user and *data* is the representation of the interaction in a vector, for example the representation of the item), it creates an optimized model from the data that if being used in the test data, it would boost (in theory) the final performance of the recommendation done.

```
25 qid:1 1:0.1 2:-0.32 3:0.0 4:-0.54 5:0.454 6:1 7:-0.9 8:0.21
```

**Fig. 21. Example of an interaction in a file for the ranklib. 25 would be the score, the id of the user would be 1 and item searched is represented in a vector of 8 dimensions.**

We did several test to detect the best way to improve the results. We tried to compute the score as the kind of activity, giving the maximum value the ENTER_SEARCH activity, as well as using the score as the number of interactions between a user and an item and also the total number of users searched a name. Also, we tried to represent in a vector not only the item in the interaction, but the item and the user (concatenating both vectors). In other words, different methods to represent the interactions of the users and the items were tested in the library.

Although lots of test were done, none of them performed well. The performance were just slightly better than the random performance in all the cases (around 0.002). We also tried to improve the results adding negative interactions of popular names to the file used to create the model (with the corresponding score of 0, trying to state not only the names a user was interested, but the names the user did not show preference), but it did not work neither. For this reason, none of the tests done are explained extensively in this document.

In theory, the idea was promising, but we got terrible results. In principle, learning the latent factors using this library was a good idea. We did various tests, trying to represent the data from different points of view. We considered that the performance we obtained was extremely low, so some research would be interesting to do to detect the problem in our case. The performance was extremely low, so either the idea was not as good as we thought or something wrong were done. Nevertheless, we knew that the Bayesian Personalized Ranking was not the best algorithm to face this task, so we changed our strategy.

## 4.2. Methods to ensemble

After our bad results using BPR, we changed our strategy. We wanted to try other algorithms. With those that would perform better, we would create an ensemble of the methods to boost the final performance (in case it was possible). The idea would be that with the ensemble we could boost the performance, and counteract bad recommendations of an algorithm. A better explanation of this process is explained in 4.3.

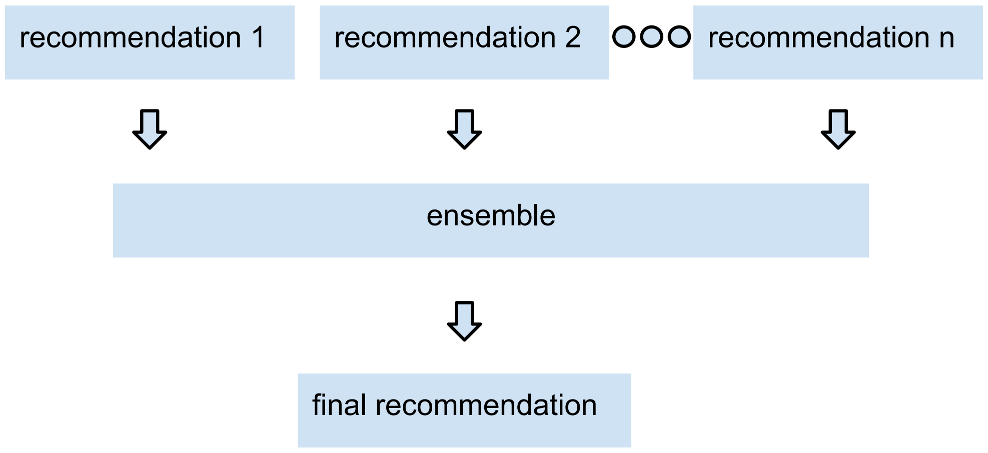


**Fig. 22. Graphical representation of the ensemble process.**

Some of the methods used are extracted from other works, and some other are new built after the valuable information from other sources. All this information is explained in the corresponding section of the algorithm.

All the N2N* models, the PageRank and the Cosine method are implemented in Python, using the numeric libraries of NumPy and SciPy[5]. For the User-Based models, the Item-Based models and Slope One, we used the Java implementation provided by Apache Mahout [6].

Here are the algorithm tested and their performance. The performance of the top 1000 names is, with the new validation set (explained before), slightly better (0.028138). Remember that the top 1000 recommendation in the test data performs 0.0263.

### 4.2.1. PageRank

[5] http://www.scipy.org/.
[6] http://mahout.apache.org/ .

| Model | Description | MAP@1000 |
|---|---|---|
| m11 | Page Rank | 0.026483 |

**Table 6. Recommendation performance in terms of MAP@1000 for the individual model m11.**

The idea of using a version of the PageRank algorithm for recommending given names is shown in [23] with good results. In that paper, an algorithm is introduced for recommending names (what is called NameRank, PPR+), which in turn is an evolution of the PPR (Preferential Page Rank), an algorithm based on the original PageRank [10]. Instead of creating a relationship structure between web pages (the basic idea behind the PageRank), PPR+ creates a structure of relation between names.

This evolution of PageRank reduce the possible problems of the frequency effects, subtracting the global PageRank score (called *PR*) from the preferential PageRank score.

The problem that we had trying to obtain a result with this algorithm is that it is really slow (or better said the time complexity is extremely high). We contacted the authors of the paper that explains the algorithm ([23]) and they confirmed us that it can take several days (at least) to compute all the relations between items. For this reason, we changed our mind and we used the original PageRank algorithm (that is explained in 2.2.5).

With this algorithm we obtain a ranked top 1000 names list. The difference between this list and a simply frequency list is that PageRank adds the weight factor explained in 2.2.5. We consider this algorithm is more comprehensive that a typical top 1000 names list. Despite all, the results of this algorithms are really similar with the baseline performance from 4.1. For this reason, even performing slightly lower in our tests than the top 1000 names, PageRank is going to be our new baseline algorithm for the ensemble.

***tests and performance:***

This algorithm has a moderate performance. As we can see in Table 6, the performance is lower than most of the methods that are going to be used in the ensemble. In spite of that, this algorithm is the only one that ensures a recommendation of 1000 names for the user. For this reason, it will be included to the ensemble to make sure that the final recommendation will have 1000 names.

### 4.2.2. Name-to-Name Collaborative Filtering

This approach for name recommendation using a Name-to-Name Collaborative Recommendation algorithm is based on the classic Item-Based collaborative filtering algorithm introduced by Amazon.com [20]. The idea is to extract points in common, relationship between names based on users' searches. In our case, we try to identify those names that have been searched together by (at least) one user.

To determine the most similar match for a given name, the algorithm constructs a bag of co-occurring names across all user interactions in the collection. The rationale behind this algorithm is that there are many names that do not co-occur in any of the user's name transactions ($\mathcal{I}(u)$) (because they are names from different languages or even alphabets), thus the approach is efficient in terms of processing time in memory, since there is no need to compute similarities over all possible pairs of names in the collection.

To compute the final recommendation list, the algorithm finds names similar to each of the ones in the user's set of names $\mathcal{I}(u)$, aggregates those co-occurring names, and then recommends the most popular or correlated names. This computation is very quick, depending only on the number of names the user has interacted with.

The Name-to-Name algorithm is summarized in Algorithm 2.

Different flavours of this approach are tested. Next, the explanation and the results of them.

#### 4.2.2.1. Name-to-Name Frequency algorithm

In this version of the algorithm, we randomly select a name $i$ for a given test user $u$ via the `getRandomName(·)` procedure specified in Algorithm 2, where the chance for a name to be chosen is proportional to how often user $u$ has interacted with it, which adds a positive bias towards those names that are more searched by the user. Furthermore, we also bias the selection of the co-occurring name $j$ (`getRandomCoName(·)` procedure in Algorithm 2) towards the multiplicity of the pair $(i, j)$.

*Example.* To illustrate this approach, consider the following example. Our dataset consists of five users, $u_1 \dots u_5$, and our task is to predict a recommendation list of names for user $u_1$. The sequence of interactions for user $u_1$ denoted as $S_{u1}$ (cf. Section 1.6) is given by

$$S_{u1} = i_4 \longrightarrow i_1 \longrightarrow i_4.$$

and for the other four users, their corresponding sequences are:

$$S_{u2} = i_1 \longrightarrow i_4 \longrightarrow i_3$$

$$S_{u3} = i_4 \longrightarrow i_5 \longrightarrow i_1 \longrightarrow i_4 \longrightarrow i_3$$

$$S_{u4} = i_3 \longrightarrow i_6 \longrightarrow i_7 \longrightarrow i_4$$

$$S_{u5} = i_1 \longrightarrow i_5 \longrightarrow i_2$$

then, the bags of co-occurrences for the names in $S_{u1}$, i.e., $i_4$ and $i_1$, sorted in decreasing order of multiplicity, are given by:

$$C(i_4) = \{((i_4, i_1), 3), ((i_4, i_3), 3), ((i_4, i_5), 1), ((i_4, i_6), 1), ((i_4, i_7), 1)\}.$$

$$C(i_1) = \{((i_1, i_4), 3), ((i_1, i_3), 2), ((i_1, i_5), 2), ((i_1, i_2), 1)\}.$$

Using the N2N-Freq shown in Algorithm 2, we first chose one name from user $u_1$'s names (i.e., from $\mathcal{I}(u_1)$), and the name's corresponding bag of co-occurrences. Let us assume that $i_4 \in \mathcal{I}(u_1)$ and its respective bag $C(i_4)$ are chosen. The first item to be included in the list of recommendations is $i_3$ ($i_1$ would not be chosen because $i_1 \in \mathcal{I}(u_1)$).

In the next iteration, consider that $C(i_4)$ is selected again, given that it has a higher probability to be picked due to the frequency of item $i_4$ in the sequence $S_{u1}$. In this case, $i_5$ would be the item chosen to be included in the list of recommendations.

In the third iteration, the list selected is $C(i_1)$, then the first item to be selected is $i_2$. Note that there are no more items from $C(i_1)$ that can be included in the list. Then, $R(u_1)$ is filled up using items from $C(i_4)$.

Finally, the list of recommendations for $u_1$ corresponds to:

$$R(u_1) = [i_3, i_5, i_2, i_6, i_7] \ .$$

***tests and performance:***

Two different versions of the co-occurrence bag are tested with this approach.

The first test uses a bag of co-occurrence created using the activities ENTER_SEARCH, LINK_SEARCH, ADD_FAVORITE and NAME_DETAILS. The second test uses a bag of co-occurrence created using only ENTER_SEARCH. To better understand this:
*Example.* Being $S_{u1}$ the sequence of interactions of $u_1$ and $es$, $ls$, $af$ and $nd$ the interactions ENTER_SEARCH, LINK_SEARCH, ADD_FAVORITE and NAME_DETAILS respectively. If $S_{u1}$ is

$$S_{u1} = es_{i1} \longrightarrow es_{i4} \longrightarrow af_{i2} \longrightarrow ls_{i4} \longrightarrow es_{i3} \longrightarrow es_{i5} \longrightarrow nd_{i6}$$

the bag of co-occurrences for the name $i_1$, using only the interactions from $u_1$ would be, in case we use all 4 activities,

$$C(i_1) = \{((i_1, i_4), 2), ((i_1, i_2), 1), ((i_1, i_3), 1), ((i_1, i_5), 1), ((i_1, i_6), 1)\},$$

but if we use only the ENTER_SEARCH interactions, the bag of co_occurrence names for $i_1$ would be

$$C(i_1) = \{((i_1, i_4), 1), ((i_1, i_3), 1), ((i_1, i_5), 1)\}.$$

As we can see, some items in the first list are not included in the second one, or are included fewer times. For example, $i_2$ is only included in the first list (because it is added using the ADD_FAVORITE activity), and the $i_4$ it is only included once in the second list (it is searched using ENTER_SEARCH and LINK_SEARCH). The performance of these two flavours are shown in Table 7.

The reason why a version of this algorithm using only ENTER_SEARCH to create the bag of co-occurrence is tested is that the hidden names in the test (see 1) are searched using this activity. For this reason, it is not unreasonable to believe that this interactions can give more valuable information than the others activities.

| Model | Description | MAP@1000 |
|---|---|---|
| m0 | N2N-Freq | 0.033449 |
| m1 | N2N-Freq-ES | 0.033430 |

**Table 7. Recommendation performance in terms of MAP@1000 for the individual models m0 and m1.**

The name-to-name algorithm explained performs really well (in fact, the two versions are those that perform better in our tests). In spite of the performance does not differ based on the co-occurrences list used (the bag created using ENTER_SEARCH, ADD_FAVORITE, LINK_SEARCH and NAME_DETAILS performs only slightly better than the bag of co-occurrences created only with ENTER_SEARCH), both approach are interesting to use further in our ensemble process. In our test, to avoid a possible infinite recursion (or better said to avoid a extremely big time consumption), a maximum number of iterations is applied (max_iterations in Algorithm 2). In our tests for this algorithm, we only do 3 iterations of the algorithms.

#### 4.2.2.2. Name-to-Name Time algorithm

In this case, the names $\mathcal{I}(u)$ are not selected biased towards frequency of user interactions, but towards recency. That is, the firsts names included in the recommendation list are those that co-occur with the last searches of the test user. The goal of this model is to capture the latest user preferences as input to compute the recommendations (it is not strange to believe that a search of a user is somehow conditioned on the last search).
*Example.* Using this algorithm, with $S_{u1}, S_{u2}, S_{u3}, S_{u4}\ S_{u5}, C(i_4)$ and $C(i_1)$ from the example given for 4.2.2.1. Using this algorithm, biased towards recency, all selectable items from $C(i_4)$ are going to be chosen first. The firsts items would correspond to $i_3$, $i_5$ and $i_6$. From $C(i_1)$ the selectable items are $i_7$ and $i_2$. The recommendation list corresponds to:

$$R(u_1) = [i_3, i_5, i_6, i_7, i_2]\ .$$

***tests and performance:***

In this case, the two versions of the co-occurrence bag explained in 4.2.2.1 are tested too. Moreover, two other versions are tried, one excluding the top 5 names (in terms of frequency) to create the bag of co-occurrences and the other one excluding the top 10 names (both cases using the bag of co-occurrences that uses all 4 activities). This rationale behind this model is to get a more specific list of names, avoiding the names that are extremely popular. To clarify how it works, here is an example.
*Example.* Being $S_{u1}$ the search in temporal order of $u_1$, $i_1$ the most popular name in the system, $i_{10}$ the second most popular name, and $C(i)$ the bag of co-occurrences of item $i$, the recommendations for $u_1$ if

$$S_{u1} = i_2 \longrightarrow i_1 \longrightarrow i_4$$

and the bag of co-occurrences for $i_1$, $i_2$ and $i_4$ are

$$C(i_1) = \{((i_1, i_4), 4), ((i_1, i_2), 3), ((i_1, i_6), 3), ((i_1, i_7), 3), ((i_1, i_5), 2), ((i_1, i_10), 1), ((i_1, i_12), 1)\}.$$

$$C(i_4) = \{((i_4, i_1), 3), ((i_4, i_2), 2), ((i_4, i_8), 1), ((i_1, i_7), 1)\}.$$

$$C(i_2) = \{((i_2, i_3), 3), ((i_2, i_1), 2), ((i_1, i_{10)}), 1\}.$$

We know that $i_1$ is one of the top popular names that we discard, so its co-occurrences names would not be considered. That means the list of recommendation would be for $u_1$

$$R(u_1) = [i_8, i_7, i_3, i_{10}]\ .$$

As we see, $C(i_1)$ is not considered because it is one of the most popular names in the data. That does not mean that a popular name can not be recommended, as we see in $R(u_1)$ (we are recommending $i_{10}$).

As we can see in Table 8, these approaches of Name-to-Name Collaborative Filtering performs really well, but the results are slightly lower than the frequency versions. We can see that N2N-Time performs slightly better than N2N-Time-ES (the same thing happens between m0 and m1). We can also see that we reach the best performance when the top 5 names are not included. The difference is really small, but provides information that can be useful for future researches. In this case there is also an iteration limit in the algorithm. In our tests for this method, if it is not possible to recommend all 1000 names with the names the user has search, we recommend the names from the bag of co-occurrence of the names that are going to be recommended (so only 2 iterations are done).

| Model | Description | MAP@1000 |
|---|---|---|
| m2 | N2N-Time | 0.032296 |
| m3 | N2N-Time-ES | 0.032008 |
| m4 | N2N-Time-NoTop5 | 0.032526 |
| m5 | N2N-Time-NoTop10 | 0.032455 |

**Table 8. Recommendation performance in terms of MAP@1000 for the individual models m2, m3, m4 and m5.**

### 4.2.3. Slope One

In spite of the fact that this approach could be interesting because of the singular way to do the recommendation, its bad performance did not allow us to us it in the ensemble.

**Algorithm 2** Name-to-Name CF

```
Input:
    Target user $u \in \mathcal{U}$. Recommendations will be computed for this user;
    $\mathcal{I}(u) \subset \mathcal{I}$: set of names that the user has interacted with;
    $C$ : bag of co-occurring names;
    $N \in \mathbb{N}$: size of the recommendation list;
    max_iterations: maximum number of iterations.
Output: Recs($u$): ranked list of recommendations for user $u$.

 1: procedure GetRecommendations($u$, $\mathcal{I}(u)$, $C$, $N$, max_iterations)
 2:     Recs($u$) ← $\emptyset$
 3:     Recs($u$) ← Name-to-Name($u$, $\mathcal{I}(u)$, $C$, $N$, Recs($u$), max_iterations)
 4:     while |Recs($u$)| < $N$ do
 5:         Recs($u$) ← Name-to-Name($u$, Recs($u$), $C$, $N$, Recs($u$), max_iterations)
 6:     end while
 7:     return sort(Recs($u$))
 8: end procedure

 9: procedure Name-to-Name($u$, $\mathcal{I}'(u)$, $C$, $N$)
10:     while |Recs($u$)| < $N$ and $t$ < max_iterations do
11:         $i$ ←getRandomName($\mathcal{I}'(u)$)
12:         $((i, j), m)$ ←getRandomCoName($C(i)$)
13:         if $j \notin$ Recs($u$) and $j \notin \mathcal{I}'(u)$ then
14:             Recs($u$) ← Recs($u$) ∪ $\{(j, m)\}$
15:         end if
16:         $t \leftarrow t + 1$
17:     end while
18:     return Recs($u$)
19: end procedure
```

As already explained in 2.2.4, this algorithm computes the differences of interactions activities between items. With this information, tries to predict how much a user would like an item that has not interacted yet.

In this case, we did use mahout to implement this algorithm. The Java code would be

```
DataModel model = new FileDataModel(new File(”data.txt”));
Recommender recommender = new SlopeOneRecommender(model);
Recommender cachingRecommender = new CachingRecommender(recommender); List¡RecommendedItem¿
recommendations = cachingRecommender.recommend(user, N);
```

where the file used to create the DataModel is a file of binary interactions between users and items. User would be the user that we want to recommend, and N the number of recommended items.

***tests and performance:***

This algorithm does not perform well, maybe because of the data used (binary data

| Model | Description | MAP@1000 |
|---|---|---|
| m6 | Slope One | 0.020344 |

**Table 9. Recommendation performance in terms of MAP@1000 for the individual model m6.**

without scores, that could be more difficult to find the patterns this algorithm needs), but it can not be assured because it is just a simple supposition. As we see in Table 17, the performance is really low. This algorithm is not going to be included in the ensemble.

### 4.2.4. Neighborhood-Based Collaborative Filtering

In our work we have tried several Neighbourhood-Based Collaborative Filtering algorithms.

On one side, we tried the User-Based algorithm approach of the Neighbourhood-Based CF, and also the Item-Based algorithm approach. On the other side, we have tried two different similarity metric, the Tanimoto coefficient for binary metric and likelihood similarity. So we have tested four different combinations of neighbourhood solutions.

As already explained before in 2.2.2, the User-Based algorithm tries to identify similarities between users based on their purchases. Item-Based algorithm tries to identify similarities directly between items. For this reason, in this section we are not going to repeat again the theory of the neighbourhood algorithm, and we are going to explain only the different ways to compute the similarity.

$$T(A,B) = \frac{A \cap B}{A \cup B}$$

**Fig. 23.** Tanimoto Coefficient. The similarity is computed like the division of items purchased in common and all the different items purchased by the users (A and B)

The first one that is going to be explained is the Tanimoto coefficient. It uses a ratio of the intersection set to the union set as a measure of similarity. That is the division between the elements in common and all the elements of the two concepts that we are trying to compare. An example to better understand this is explained below.
*Example.*Being $u_1$, $u_2$ and $u_3$ three users of the data, and $i_1 \ldots i_5$ all the items in the system,

$$S_{u1} = i_1 \longrightarrow i_4 \longrightarrow i_4 \longrightarrow i_5.$$

$$S_{u2} = i_5 \longrightarrow i_1 \longrightarrow i_2 \longrightarrow i_1 \longrightarrow i_2 \longrightarrow i_3.$$

$$S_{u3} = i_3 \longrightarrow i_4$$

This example is represented in Figure 24. In Figure 23 there is explanation of the Tanimoto coefficient.

| | i1 | i2 | i3 | i4 | i5 |
|---|---|---|---|---|---|
| u1 | + | ? | ? | + | + |
| u2 | + | + | + | ? | + |
| u3 | ? | ? | + | + | ? |

**Fig. 24.** Visual representation of the interaction to explain the Tanimoto coefficient example explained in 4.2.4

In our case, using the Tanimoto Coefficient, $T(u_1, u_2)$ is 2/5, because $u_1$ and $u_2$ have two items in common out of five items. On the other hand, $T(u_1, u_3)$ is 1/5. For this reason, we consider that $u_1$ is more similar to $u_2$ than $u_3$.

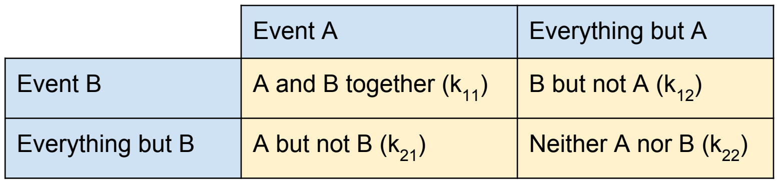
The log-likelihood is more complicated to explain than the Tanimoto similarity. It does not calculate the similarity based on if two events have been searched together, For this reason, this similarity is going to be explained along with the mahout code implementation.

Having two elements (called A and B), the main step is to compute the number of times an event occurs to one element (or does not occur) when the other event occurs to the other element (or does not occur). For example, if we want to know the similarity between two users (A and B), the events would be the items searched by them, so we need to count the items they both have searched, the items only one of them has searched and the item that have not been searched neither by A nor B. A visual explication is shown in Figure 25 where

- $k_{11}$ is the number of times the two events occurred together
- $k_{12}$ is the number of times the second event occurred without the first event
- $k_{21}$ is the number of times the first event occurred without the second event
- $k_{22}$ otherwise

After that, it is really simple to get the log-likelihood Ratio:

```
public static double logLikelihoodRatio
```



| | Event A | Everything but A |
|---|---|---|
| Event B | A and B together ($k_{11}$) | B but not A ($k_{12}$) |
| Everything but B | A but not B ($k_{21}$) | Neither A nor B ($k_{22}$) |

**Fig. 25. Occurrency count of the events necessary to calculate the Log-Likelihood similarity**

```
        (long k11, long k12, long k21, long k22)
{
        Preconditions.checkArgument
                (k11 >= 0 and k12 >= and k21 >= 0 and k22 >= 0);
        double rowEntropy = entropy(k11, k12) + entropy(k21, k22);
        double columnEntropy = entropy(k11, k21) + entropy(k12, k22);
        double matrixEntropy = entropy(k11, k12, k21, k22);
        if (rowEntropy + columnEntropy $>$ matrixEntropy):
                return 0.0;
        return 2.0 * (matrixEntropy - rowEntropy - columnEntropy);
}
```

where the entropy between two values is calculated like:

```
private static double entropy(long a, long b)
{
        return xLogX(a + b) - xLogX(a) - xLogX(b);\\
}
```

where the entropy between four values is calculated like:

```
private static double More ...entropy(long a, long b, long c, long d)
{
        return xLogX(a + b + c + d)
                - xLogX(a) - xLogX(b) - xLogX(c) - xLogX(d);
}
```

and the xLogX of a value is calculated like:

```
private static double xLogX(long x)
{
        return x == 0 ? 0.0 : x * Math.log(x);
}
```

The Log-Likelihood tries to find associations between events that could show relationships between items. The different between this similarity metric and others (like Tanimoto) is that this one also uses the events that do not happen to compute the similarity.

| Model | Description | MAP@1000 |
|---|---|---|
| m7 | UB_Tanimoto | 0.023921 |
| m8 | UB_logli | 0.028365 |

**Table 10. Recommendation performance in terms of MAP@1000 for the individual model m7 and m8.**

#### 4.2.4.1. User-Based Collaborative Filtering

The User-Based Neighbourhood Collaborative Filtering have been tested in our research using both Tanimoto Coefficient and Log-Likelihood similarity. The idea of the User-Based Collaborative Filtering is to find the similarities between users based on what they have searched.
We used the mahout library. Here the code for the Tanimoto case:

```
DataModel model = new FileDataModel(New File(”data.txt”));
UserSimilarity userSimilarity = TanimotoCoefficientSimilarity(model);
UserNeighbourhood neighbourhood = new
        NearestNUserNeighbourhood(100,userSimilarity ,model);
Recommender recommender = new
        GenericBooleanPrefUserBasedRecommender
                (model,neighborhood,userSimilarity);
CachingRecommender cachingRecommender = new
        CachingRecommender(recommender);
List$<$RecommendedItems$>$ recommendations =
        cachingRecommender.recommend(user ,N);
```

In case of the Log-Likelihood, the second line must be

```
UserSimilarity userSimilarity = new LogLikelihoodSimilarity(dataModel)
```

***tests and performance:***

The User-Based CF algorithms tested do have a moderate performance , but they are good enough to be considered for the ensemble. One of them (the one that uses Log-Likelihood similarity) is good enough to be included in the ensemble by its own. Merging both of them we could boost the performance enough to obtain a recommendation clearly better than the baseline.

#### 4.2.4.2. Item-Based Collaborative Filtering

Also in the Item-Based case, we have used the Tanimoto Coefficient and Log-Likelihood similarity. In this case (the Item-Based CF) we find similarities directly between items.

| Model | Description | MAP@1000 |
|---|---|---|
| m9 | IB_Tanimoto | 0.020638 |
| m10 | IB_logli | 0.019143 |

**Table 11. Recommendation performance in terms of MAP@1000 for the individual model m9 and m10.**

Just like the User-Based case, we also used Mahout. Here is the code:

```
DataModel model = new FileDataModel(New File("data.txt"));
ItemSimilarity itemSimilarity = TanimotoCoefficientSimilarity(model);
GenericBooleanPrefItemBasedRecommender recommender = new
        GenericBooleanPrefItemBasedRecommender(model, userSimilarity);
Recommender cachingRecommender = new CachingRecommender(recommender);
List$<$RecommendedItems$>$ recommendations =
        cachingRecommender.recommend(user,N);
```

In case of the Log-Likelihood, the second line must be

```
ItemSimilarity itemSimilarity = new LogLikelihoodSimilarity(dataModel)
```

***tests and performance:***

Both the Tanimoto and the log likelihood version of the Item-Based algorithms do not perform well (Table 11). As we will see later, the Item Based approaches will not be considered to create the recommendation.

In case of the User-Based algorithms, the log likelihood performs better than UB algorithms using the Tanimoto coefficient.

```
DataModel model = new FileDataModel(New File("data.txt"));
UserSimilarity userSimilarity = TanimotoCoefficientSimilarity(model);
UserNeighbourhood neighbourhood = new
        NearestNUserNeighbourhood(100,userSimilarity,model);
Recommender recommender = new
        GenericBooleanPrefUserBasedRecommender
                (model, neighborhood, userSimilarity);
CachingRecommender cachingRecommender = new CachingRecommender(recommender);
List$<$RecommendedItems$>$ recommendations =
        cachingRecommender.recommend(user,N);
```

### 4.2.5. KNN based on Cosine Similarity

Part of the data given by nameling, as said in 1.2, were 3 files with some information about similarity between names. For every name in English, German and French, a list of the top 100 more similar names are given. This method recommends the names that are more similar to those names searched by the user. The recommendation is ordered towards recency (like 4.2.2).
*Example.* Being $S_u$ the search in temporal order of $u$, and $Con_i$ the list of the most similar names of $i$

$$S_{u1} = i_2 \longrightarrow i_1$$
$$Con_{i1} = i_{c10}, i_{c11}, \ldots, i_{c1n}$$
$$Con_{i2} = i_{c20}, i_{c21}, \ldots, i_{c2m}$$

where $i_{c10}$ (as an example to clarify the explanation) is the first item most similar to $i_1$ in the cosine similarity list, $i_{c11}$ the second one, etc., the recommendation for $u_1$ would be

$$R(u_1) = [i_{c10}, i_{c11}, \ldots, i_{c1n}, i_{c20}, i_{c21}, \ldots, i_{c2m}] \ .$$

Notice that the cosine files are divided by language, so the cosine similarity list length of a name after merging these 3 files may differ. If a name only exist in one language, we would have 100 similar names (just as said before), but in case a name appears in two language, the list length would be something between 100 names and 200 names (depending on whether they have common names), and if a name appears in all three lists, the list length of similar names would be maximum 300 names. For example, if a name is both French and English (so it appears in the French list and in the English one), and in both French and English lists they have a similar name in common (that appears in both list), the name is not included twice. Instead of that, is only added once with the bigger similarity value.

| Model | Description | MAP@1000 |
|---|---|---|
| m6 | UB-T | 0.023921 |
| m7 | UB-LL | 0.028365 |

**Table 12. Recommendation performance in terms of MAP@1000 for the individual model m6 and m7.**

***tests and performance:***

This method did not perform well at all due to its characteristics. In so many cases there where not enough names to fill up all the 1000 recommendations for every user (not even close).

Despite all, this files, with the proper processing, could have given a lot of useful information. The lack of time did not allow us to investigate a proper way of using these similarities.

## 4.3. Ensemble and Results

Our solution to the challenge consists of an ensemble of individual rank estimates of a set of collaborative filtering algorithms, a method that has shown to improve the quality of the recommendations like in [9].

The rationale behind this idea is to use all the good predictions from the recommendation list, and try to compensate the bad recommendations. That is that with the proper ensemble, a name recommended in one list from one algorithm would be penalized when it is not recommended in any other list, but a name that different approaches detect as a good candidate would be included also in the ensemble recommendation.

Since the estimate values of our models, $\hat{x}$, can be in different scales, we do not combine their values directly, but rather we use their rank estimates. Formally, the ensemble of the rank estimates of $l$ models is given by:

$$\hat{x}_{ui}^{\text{rank}} := \sum_{l} \alpha_l \cdot \frac{1}{\text{rank}(\hat{x}_{ui}^{l})} \ , \tag{5}$$

where $\alpha_l$ is a weight associated to the predictors of model $l$ and $\text{rank}(\hat{x}_{ui}^{l})$ is the rank position within the $l$th ranked list corresponding to the estimate value $\hat{x}_{ui}^{l}$. That is, the combined estimate $\hat{x}_{ui}^{\text{rank}}$ corresponds to the weighted reciprocal rank of the individual models.

An example is shown in Figure 26.

In this section, we detail the collaborative filtering models used to do the ensemble, the procedure done to join the results of the different models and the performance boost achieved by our ensemble.

The ensemble of our solution consists of 9 of the collaborative models that we have tried. The models that are going to be used (explained in 4.2) are resumed as follows.

**[N2N-Freq]** is the Name-to-Name Collaborative filtering with the bias to those names more searched.

**[N2N-Freq-ES]** is the Name-to-Name Collaborative filtering with the bias to those names more searched (just like N2N-Freq) that uses only the ENTER_SEARCH activities to create the bag of co-occurrences.

**[N2N-Time]** is the Name-to-Name Collaborative filtering with the bias towards recency.

**[m3 – N2N-Time-ES]** is the Name-to-Name Collaborative filtering with the bias towards recency (just like N2N-Time) that uses only the ENTER_SEARCH activities

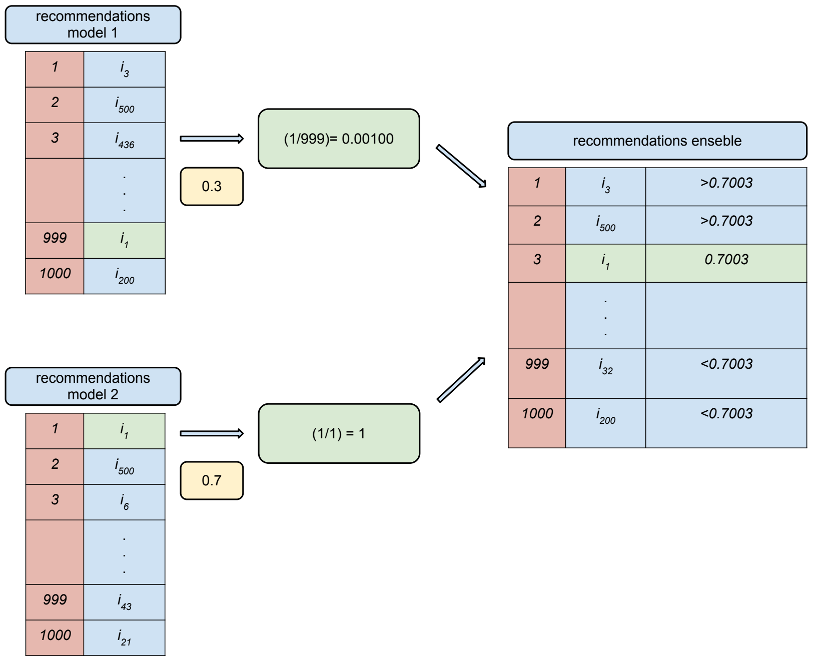


**Fig. 26. A visual example of an ensemble of 2 models with weight 0.3-0.7.**

to create the bag of co-occurrences.

**[N2N-Time-NoTop5]** is the Name-to-Name Collaborative filtering with the bias towards recency (just like N2N-Time). However, the top 5 most searched names are not considered to create the bag of co-occurrences.

**[N2N-Time-NoTop10]** is the Name-to-Name Collaborative filtering with the bias towards recency (just like N2N-Time). However, the top 10 most searched names are not considered to create the bag of co-occurrences.

**[UB-T]** is a user-based collaborative filtering algorithm [13] using Tanimoto coefficient for binary feedback as similarity metric [29]. We used a neighborhood of size 100[7].

**[UB-LL]** is a user-based model that uses likelihood as similarity metric. As in the previous model, we also used a neighbourhood of size 100 in this case.

**[PR]** This model corresponds to PageRank [10] computed on the graph of co-occurring names. This is a non-personalized recommendation algorithm biased to the most

[7] Observe that we did not optimize for this parameter.

| Model | description |
|---|---|
| m0 | N2N-Freq |
| m1 | N2N-Freq-ES |
| m2 | N2N-Time |
| m3 | N2N-Time-ES |
| m4 | N2N-Time-NoTop5 |
| m5 | N2N-Time-NoTop10 |
| m6 | UB-T |
| m7 | UB-LL |
| m8 | PR |

**Table 13. Reference names of the used methods for the ensemble**

popular items. We used this algorithm to “fill up” recommendation lists with less than 1000 names per user.

The models that are not used in the ensemble (also explained in 4.2) are:

**[Slope one]** is the recommendation algorithm that computes the differences of interaction activities to predict the intensity between users and items without interaction.

**[IB-Tanimoto]** is the item-based model that uses Tanimoto coefficient to compute the similarity between items.

**[IB-LL]** is the item-based model that uses Log-Likelihood to compute the similarity between items.

**[Cosine]** is the algorithm that recommends names based on the list of similar names provided by Nameling.

As said in 4.2, all the N2N* models, the PageRank and the Cosine method were implemented in Python, using the numeric libraries of NumPy and SciPy[8]. For the User-Based models, the Item-Based models and Slope One, we used the Java implementation provided by Apache Mahout [9].

Table 13 summarizes the individual performance of these models. The reference names of the models where added. From now on, the reference names of the used methods will be from m0 to m8, and the not used methods for the ensemble will be the methods from m9 to m12 (Table 13 and 14).

[8] `http://www.scipy.org/`.
[9] `http://mahout.apache.org/`.

| Model | description |
|---|---|
| m9 | Slope one |
| m10 | IB-Tanimoto |
| m12 | IB-LL |
| m12 | Cosine |

**Table 14. Reference names of the not used methods for the ensemble**

| Model | Description | MAP@1000 |
|---|---|---|
| m0 | N2N-Freq | 0.033449 |
| m1 | N2N-Freq-ES | 0.033430 |
| m2 | N2N-Time | 0.032296 |
| m3 | N2N-Time-ES | 0.032008 |
| m4 | N2N-Time-NoTop5 | 0.032526 |
| m5 | N2N-Time-NoTop10 | 0.032455 |
| m6 | UB-T | 0.023921 |
| m7 | UB-LL | 0.028365 |
| m8 | PR | 0.026483 |
| **Final ensemble** | | **0.036766** |

**Table 15. Recommendation performance in terms of MAP@1000 for the individual models and the final ensemble.**

## Engineering the Final Ensemble

We compute the final ensemble by first combining different *flavours* of the same approach, and then combining the resulting ranked lists as explained before. Figure 33 illustrates the assembling process.

All weights (the $\alpha$'s in Equation 5) were determined experimentally based on the performance achieved by the (sub-)ensembles in our cross validation set. First, we combined them without weight (the same weight for all the algorithms), and then some other test were done with slightly modifications of the weights, to determine how to determine the best configurations of weights (see 28 for a visual explanation of this process).

The following are the explanation of the combinations.

**combination of the N2N-Freq* algorithms:**
The models m0 and m1 (the N2N-Freq models) have a very similar performance. For this reason, is not strange that the best way to combine them is using the same weight for both recommender.

The improvement is really small because of their similarity.

**combination of the N2N-Time* algorithms:**

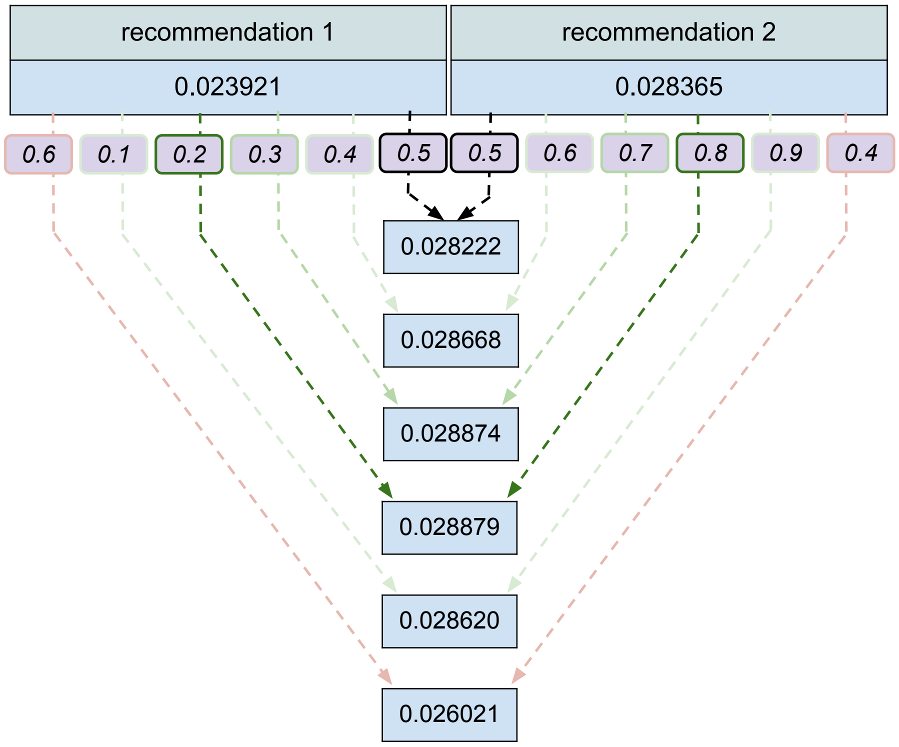


**Fig. 27. A visual example how we proceed to find the best way to weight the ensemble. In that case, is easy to see that *recommendation 2* will get more weight, and the process shows the proper weight (in that case, 0.8 to the *recommendation 2* and 0.2 to the *recommendation 1*)**

| Model | Description | MAP@1000 |
|---|---|---|
| m0 | N2N-Freq | 0.033449 |
| m1 | N2N-Freq-ES | 0.033430 |
| **Result of the ensemble** | | **0.034134** |

**Table 16. Recommendation performance in terms of MAP@1000 for the models m0 and m1 and the ensemble.**

The models m2, m3, m4 and m5 (the N2N-Time models) have a very similar performance too. For this reason, the combination will be again with the same weight every model.

The improvement is small because of their similarity, but bigger than the m0-m1 ensemble improvement. This is because (in our opinion) the files are more different be-

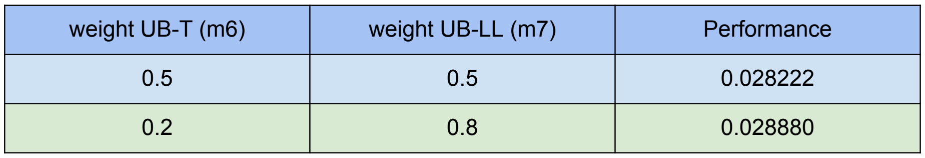
| Model | Description | MAP@1000 |
|---|---|---|
| m2 | N2N-Time | 0.032296 |
| m3 | N2N-Time-ES | 0.032008 |
| m4 | N2N-Time-NoTop5 | 0.032526 |
| m5 | N2N-Time-NoTop10 | 0.032455 |
| **Result of the ensemble** | | **0.036078** |

**Table 17. Recommendation performance in terms of MAP@1000 for the models m2, m3, m4 and m5 and the ensemble.**

tween them and moreover we have more files to combine.

**combination of the UB* algorithms:**

The models m6 and m7 (the UB models) are quite different. Using different similarity, the performance of the log-likelihood version of the User-Based algorithm performs remarkably better than the User-Based version using Tanimoto coefficient. For this reason, the log-likelihood will receive a bigger weight.



| weight UB-T (m6) | weight UB-LL (m7) | Performance |
|---|---|---|
| 0.5 | 0.5 | 0.028222 |
| 0.2 | 0.8 | 0.028880 |

**Fig. 28. The best combination of weights for maximize the performance is giving a 0.8 the User-Based algorithm using Log-likelihood and a 0.2 for the User-Based algorithm using Tanimoto coefficient.**

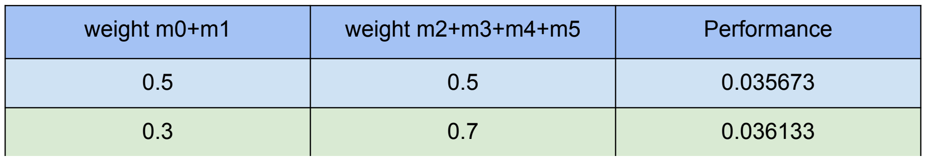
| Model | Description | MAP@1000 |
|---|---|---|
| m6 | UB-T | 0.023921 |
| m7 | UB-LL | 0.028365 |
| **Result of the ensemble** | | **0.028880** |

**Table 18. Recommendation performance in terms of MAP@1000 for the models m6 and m7 and the ensemble.**

In this case, we combined the two methods with different weights (0.8 for the m6 and 0.2 for the m7).

**combination of the 2 N2N ensembles:**

Having the ensembles for the N2N-Freq* and the N2N-Time* models, the next step was to combine them.



| weight m0+m1 | weight m2+m3+m4+m5 | Performance |
|---|---|---|
| 0.5 | 0.5 | 0.035673 |
| 0.3 | 0.7 | 0.036133 |

**Fig. 29. The best combination of weights for maximize the performance is to give a 0.7 the N2N-Time ensemble and a 0.3 for the N2N-Freq ensemble.**

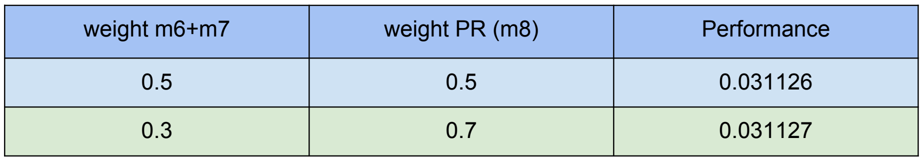
| Model | Description | MAP@1000 |
|---|---|---|
| m0+m1 | N2N-Freq* ensemble | 0.034134 |
| m2+m3+m4+m5 | N2N-Time* ensemble | 0.036079 |
| **Result of the ensemble** | | **0.036133** |

**Table 19. Recommendation performance in terms of MAP@1000 for the models m0+m1 and m2+m3+m4+m5 and the ensemble.**

The result of the ensamble is just slightly better than the N2N-Time* ensamble. But as we can see, there is still an improvement in the performance.

**combination of the UB ensemble with the PageRank model:**
As said before, the PageRank has a moderate performance, but it is really useful for the ensemble because is the only one that has for sure 1000 recommended names for all the users. Adding this model in the combination, we can ensure that all the users will receive in the final ensemble 1000 names. In Figure 30, we can see that the best way to combine the UB ensemble and the PR algorithm is giving to the first one a weight of 0.6 and the second one a weight of 0.4.

**final ensemble. Combination of the N2N ensemble with the UB-PR ensemble:**
In that case, the best way to mix the ensemble methods were giving to N2N method a weight of 0.8 and the UB-PR method a weight of 0.2. This is because the better performance of the N2N methods, that outperforms the others methods.

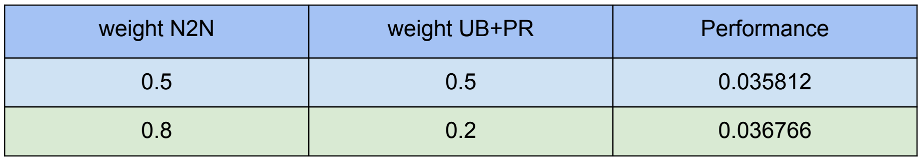


| weight m6+m7 | weight PR (m8) | Performance |
|---|---|---|
| 0.5 | 0.5 | 0.031126 |
| 0.3 | 0.7 | 0.031127 |

**Fig. 30. As we can see, the PageRank recommendation is not improving our performance, but it is useful for us because we ensure that the final ensemble will have 1000 names.**

| Model | Description | MAP@1000 |
|---|---|---|
| m6+m7 | UB ensemble | 0.028880 |
| m8 | PR | 0.026483 |
| **Result of the ensemble** | | **0.031127** |

**Table 20. Recommendation performance in terms of MAP@1000 for the models m6+m7, m8 and the ensemble.**



| weight N2N | weight UB+PR | Performance |
|---|---|---|
| 0.5 | 0.5 | 0.035812 |
| 0.8 | 0.2 | 0.036766 |

**Fig. 31. The final ensemble gives a bigger weight to the N2N ensemble because of its better performance.**

| Model | Description | MAP@1000 |
|---|---|---|
| m0+m1+m2+m3+m4+m5 | N2N ensemble | 0.036133 |
| m6+m7+m8 | UB-PR ensemble | 0.031127 |
| **Result of the ensemble** | | **0.036766** |

**Table 21. Recommendation performance in terms of MAP@1000 for the models m0 and m1 and the ensemble.**

**Resume:**

The way we decided to combine the methods were joining those methods that were more similar.

We found out that the best way to combine the N2N-Freq* (m0 and m1) methods were giving the same weight because of their similar performance. The same happened

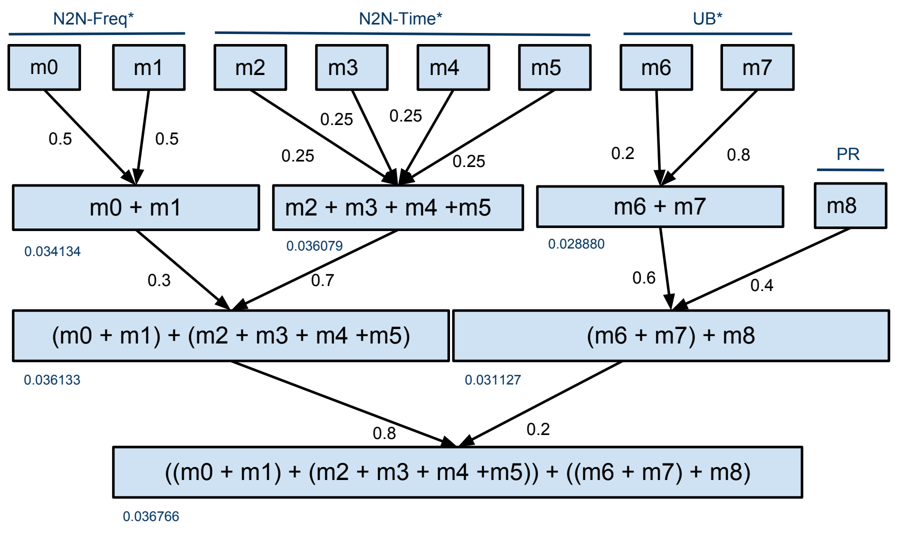


**Fig. 32. Final ensemble. The $\alpha$ weights for the partial model ensembles are indicated next to the corresponding arrow. The symbol '+' indicates the assembling of the models. The MAP@1000 for the corresponding sub-ensembles are shown below the respective boxes.**

| Model | Performance |
|---|---|
| Baseline method (PageRank) | 0.026483 |
| Ensemble method | 0.036766 |

| | |
|---|---|
| Improvement | ~40% |

**Fig. 33. Final ensemble. Compared with the baseline algorithm (PageRank), our ensemble method improve the results a 40%**

with the N2N-Time* methods. In this ensemble, we saw that the ensemble performance were really better in comparison of the performance of every single N2N-Time* methods.

The best way to combine the UB methods were with a weight of 0.8 to the Log-Likelihood method and a 0.2 to the Tanimoto coefficient User-Based algorithm. It is not strange because the UB-LL outperforms the UB-T method. The ensemble result was combined with the PageRank method. The UB model had a weight of 0.6 and the PR model a weight of 0.4.

In this point of the ensemble process, the next step was to combine the N2N-Time and the N2N-Freq method. In spite of the fact is that the single N2N-Freq* methods (m0 and m1) were better than the single N2N-Time* methods (m2, m3, m4 and m5), the ensemble method for the N2N-Time methods (m2+m3+m4+m5) were better than the ensemble N2N-Freq method (m0+m1), so the weights were set in 0.7-0.3.

Finally, we combined the N2N method (m0+m1+m2+m3+m4+m5) with the UB-PR method (m6+m7+m8) giving to the first one a weight of 0.8, and to the second one a weight of 0.2 because of the better performance of the first method. The final ensemble outperforms all the single method performance.

The Item-Based Collaborative Filtering methods (both the log-likelihood and the Tanimoto coefficient versions), the cosine method and the Slope One method were not included (Table 14) due to their low performance.

# 5. Discussion and Future Work

## 5.1. Summary

As we can see in our work, the low values of MAP@1000 obtained by our approach on this dataset give an idea of how difficult the problem of recommending given names is.

Given the evaluation design of hiding the last two names the user interacted with, models that capture the latest user preferences, e.g., from the session information, tend to work well for us.

Neighbourhood-based algorithms perform worse than item-to-item co-occurrences. Within the item-based and user-based variants, we observed that results from item-based collaborative filtering were inferior to the ones achieved by the user-based models, and therefore we did not consider them in the ensemble.

One of CF's most successful techniques are low dimensional linear factor models, that assume that user preferences can be modelled by only a small number of latent factors. One of such methods is matrix factorization, which has been effectively used for the rating and item prediction task [18].

We conducted extensive experiments using state-of-the-art CF algorithms based on matrix factorization. In particular, we evaluated the performance of BPR [25] and RMFX [14] for the challenge's task, but we found that the performance achieved was only at the level of a baseline predictor that recommends the most popular names. This poor performance of matrix factorization models has been also observed by Folke et al. [23]. For this reason, the Bayesian Personalized Ranking was not included in the final ensemble.

We also learned a name-to-name similarity matrix from the co-occurring names adjacency via optimizing a ranking criteria, as suggested in [25], the results were also discouraging. In spite of that, the name-to-name algorithms that we introduce (N2N-Freq and N2N-Time) are those that perform better.

Furthermore, we also tried to optimize directly for MAP following a Learning to Rank framework suggested in [15] and [7]. This approach learns the latent factors for users and items, and then applies standard Learning to Rank to optimize for a desired metric. Our results did not reach the level of the baseline predictor of most popular names.

Given this performance, we did not include any latent factor model in our ensemble. Why the results achieved using latent factor models, for this particular task of name prediction, are inferior to the ones obtained with simple methods? In our future research, we plan to explore this question more in detail.

In spite of the moderate performance of all the classic Collaborative Filtering algorithms, the name-to-name algorithm performs substantially better than our baseline algorithm (PageRank). Moreover, with a proper combination of several algorithm we can get a fairly good recommender system.

## 5.2. Thoughts not applied and possible improvements

In spite of a lot of algorithms are tested in our work, we had several ideas that were not included because of different reasons. One of the ideas that we wanted to implement in our solution was to take advantage of explicit information of the users and the names, and apply some kind of Content-based filtering. This data would allow us to apply some bias to the results based on this information, or work only with those names that in theory were candidates to be chosen by the user, removing all the extra names that would give only noise.
In our opinion, the language and the genre of a name was the key to boost even more our results. Our intention was to apply some bias based on this information. If the system was able to detect the genre and the language of the names a user was interested on based on their searches, the recommendation would be done according to that information. The problem for us was that we needed some gazetteers (list of names per language and names per genre) that we did not have. To solve this problem we created our own data with some sources founded in internet, but were not really useful due to the information source were not extremely reliable. Technically, the results with a good gazetteer would be promising, but to create a good gazetteer, we needed some time that we did not have in that moment.

Another way to improve our results would have been using the geological position of the user. In case of a user is from Germany, we would consider a bias to names in German (or popular names in Germany, like Turkish name because of the immigration). The problem for us was that Nameling gives partial information of their users (we only know the geographic position of some of them), and most the users were from Germany and Austria, so we were not able to apply a good bias for several countries.

So we consider that with these gazetteers the performance would have increased. Having information about the country where the user is doing the search, the idiom of the names a user was searching and the genre, lots of names would be automatically discarded.

Moreover, one of the biggest problems that we had was to give a recommendation list to those users that did not have any search yet (or just a few). If we had known the nationality (or at least the place where the user is doing the search) of all the users without enough information we could adapt our first recommendation with a content-based recommendation (like giving the list of the most popular names in its country). That would be really interesting for us because it would allow us to apply a specific recommend list for every user (instead of a general one) even if he has not done any search yet (having a good solution for the cold recommendation).

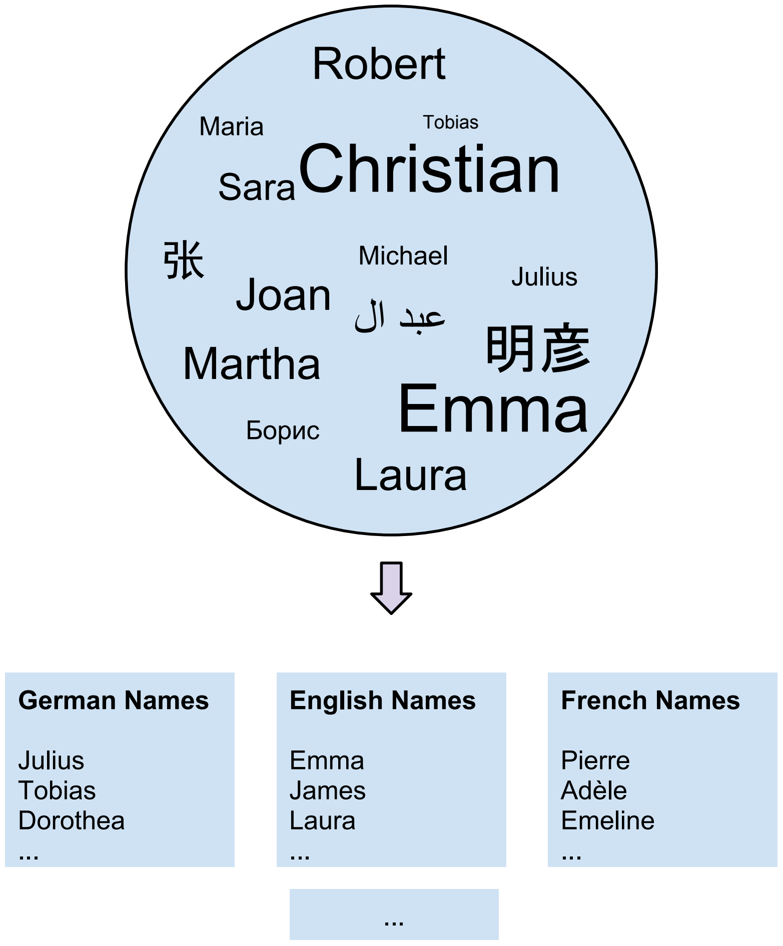


**Fig. 34. Having gazetteers of names by language would have allowed us to apply some bias to the results that would allow us to improve even more the ensemble.**

## 5.3. Conclusion and Future Work

### 5.3.1. Outlook

Some interesting approaches were used in the challenge to recommend given names. Here there is a small resume of what the participants presented in the workshop.

The paper presented in [5] bases its recommendation on exploiting the contextual information available (e.g., time and location). They use two state-of-the-art recommender systems (item-based Collaborative Filtering and Association Rules Based), and they weight the result obtained adding the contextual information. It is also interesting how they pre-processed the data available. Instead of removing those names that were not valid, they replaced them with the closest valid name (the idea of this process is to

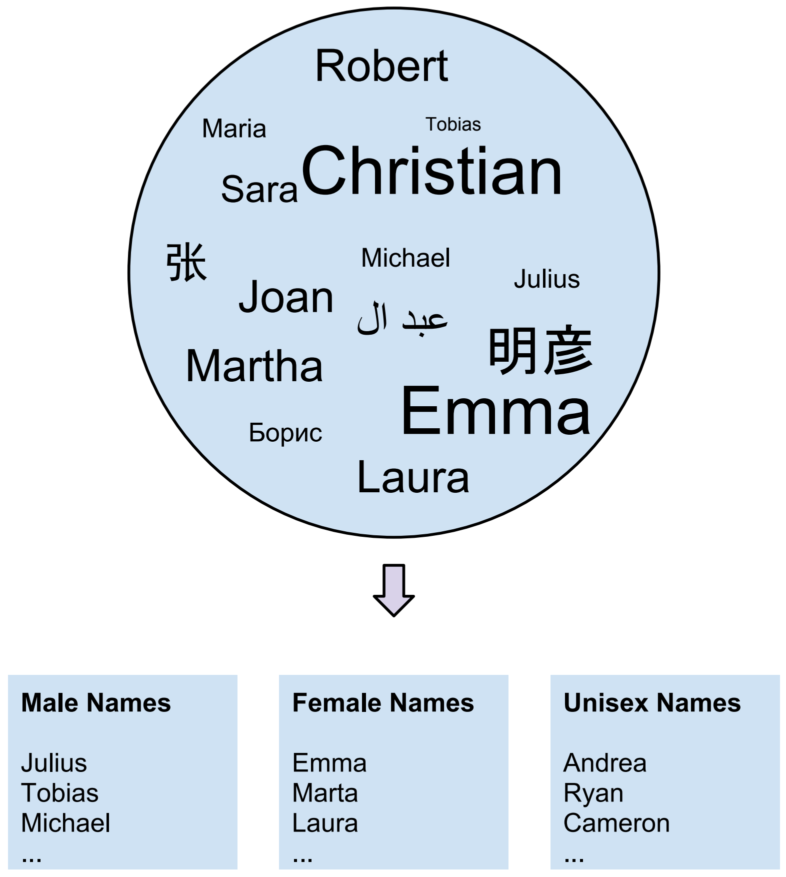


**Fig. 35. Having gazetteers of names by genre would have allowed us to apply some bias to the results that would allow us to improve even more the ensemble.**

correct a mistyped name by the user). Moreover, they removed the singleton sessions ("*users only access only one item on a website and then leave it*") that can not be used in the item-based Collaborative Filtering technique.

Another approach presented in [24] creates an hybrid recommendation that mix the results obtained from a collaborative filtering recommendation, a content-based recommendation and a popularity-based one. The popularity recommendation list is created using the most frequent searched names list (following the restrictions of the challenge). The collaborative filtering approach calculates de similarity between users using a neighbourhood algorithm. The content-based algorithm uses the Soundex algorithm to find names that have phonetic similarity.

The approach from [27] combines multiple recommendation list, that is they introduced an approach based on dyadic factors, which several information from the names (like the length or the gender). Also information from the users (the demographic factor) are considered to create the final recommendation.

The paper presented by I.Bayes and S.Rendle ([8]) differs from the others because no meta data (neither word similarity nor geographic information) is used to create the recommendation. Instead of that, they use Factorization Machines, which are a model class that combines the advantages of support Vector Machines (SVM) with factorization models. This partial recommendation is complemented with syntactical similarity information to create the final recommendation list.

The last approach presented in [19] was an approach based on association rules. Moreover, a similarity-weighted adjust is applied to boost the performance.

As we can see, in most of the cases the approaches are created combining algorithms for recommending systems that are already known and some kind of bias to the result based on content-based information of the data of the challenge (for example information of the names or geographic information of the users).

One of the most interesting ideas presented in these papers was the phonetic similarity factor used in one paper to find names that sounded similar. As we can see, most of the users used somehow the content-based information related to the language of the name or the country of the user. But it is not strange to believe that if you like a name, you would like those names that sound similar to the first one. It is a very simple yet also very powerful idea.

Finally, just to mention that all the papers gave the final recommender result combining several algorithms. This make us to believe that problem is really difficult to solve and that an ensemble approach is really promising.

### 5.3.2. Conclusions

In this work, we used an ensemble of several algorithm for personalized ranked recommendation of given names. Because of the low performance that the Collaborative Filtering algorithms performed, our solution tries to boost it trying to obtain information from all the algorithms that we tried that seemed that could find some patterns between the names the user search. As we can see, the ensemble worked improving the performance as expected.

We also found that the co-occurring name information was a key component for the Name-to-Name algorithms used in our ensemble. The approaches using the co-occurring name information were those that performed better.

Our method used to do our final recommendation is intuitive and simple to implement, and does not suffer from the scalability issues as previous methods introduced for this task. This fact allow us to believe that our method to recommend names can be improved increasing the complexity of the algorithms, or creating a specific version of our recommendation system for a specific case.

We plan to further explore this interesting challenge in order to help parents deciding what is the best name for their baby.

# 6. Bibliography

## References

1. Apache Mahout, 2013. URL `http://mahout.apache.org/`.
2. Collaborative Filtering. URL `//http://en.wikipedia.org/wiki/Collaborative_filtering`.
3. Numpy, 2013. URL `http://numpy.org/`.
4. recsyslab, 2013. URL `https://github.com/Foolius/recsyslab`.
5. M. Aurelio-Domingues, R. Marcondes-Marcacini, S. Oliveira-Rezende, and G. E.A.P.A.-Batista. Improving the recommendation of given names by using contextual information. In *15th Discovery Challenge ECML-PKDD 2013*, 2013.
6. R. Baeza-Yates, G. Navarro, and N. Ziviani. *Modern Information Retrieval*. Addison-Wesley, 2nd edition, 2011.
7. S. Balakrishnan and S. Chopra. Collaborative ranking. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 143–152, New York, NY, USA, 2012. ACM.
8. I. Bayer and S. Rendle. Factor models for recommending given names. In *15th Discovery Challenge ECML-PKDD 2013*, 2013.
9. R. M. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize. Technical report, AT&T Labs, 2007.
10. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.
11. L. Clevesy. Slope-one Recommender - Exclusive Article from Mahout in Action, 2013. URL `http://books.dzone.com/articles/slope-one-recommender`.
12. M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22.

13. C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In Ricci et al. [26], pages 107–144.
14. E. Diaz-Aviles, L. Drumond, L. Schmidt-Thieme, and W. Nejdl. Real-time top-n recommendation in social streams. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 59–66, New York, NY, USA, 2012. ACM.
15. E. Diaz-Aviles, M. Georgescu, and W. Nejdl. Swarming to rank for recommender systems. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 229–232, New York, NY, USA, 2012. ACM.
16. T. Dunning. Surprise and coincidence, 2013. URL `http://tdunning.blogspot.de/2008/03/surprise-and-coincidence.html`.
17. A. Hotho, R. Jäschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. In Sure and Domingue [28], pages 411–426.
18. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug. 2009.
19. B. Letham. Similarity-weighted association rules for a name recommender system. In *15th Discovery Challenge ECML-PKDD 2013*, 2013.
20. G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.
21. F. Mitzlaff and G. Stumme. Ranking given names. In M. Marathe and N. Contractor, editors, *Proceedings of the 1st ASE International Conference on Social Informatics*, pages 185–191. IEEE computer society, 2012.
22. F. Mitzlaff and G. Stumme. Onomastics 2.0 - the power of social co-occurrences, 2013.
23. F. Mitzlaff and G. Stumme. Recommending given names, 2013.
24. J. R.-J. Rafael Glauber, Angelo Loula. A mixed hybrid recommender system for given names. In *15th Discovery Challenge ECML-PKDD 2013*, 2013.
25. S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.
26. F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.
27. D. Schäfer and R. Senge. Nameling discovery challenge collaborative neighborhoods. In *15th Discovery Challenge ECML-PKDD 2013*, 2013.
28. Y. Sure and J. Domingue, editors. *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings*, volume 4011 of *Lecture Notes in Computer Science*, New York, NY, USA, Jan. 2006. Springer.
29. T. Tanimoto. "ibm internal report 17th nov.", 1957.