Factor Models for Recommending Given Names

Immanuel Bayer and Steffen Rendle

University of Konstanz, 78457 Konstanz, Germany,
{immanuel.bayer, steffen.rendle}@uni-konstanz.de

Abstract. We describe in this paper our contribution to the ECML PKDD Discovery Challenge 2013 (Offline Track). This years task was to predict the next given names a user of a name search engine interacts with. We model the user preferences with a sequential factor model that we optimize with respect to the Bayesian Personalized Ranking (BPR) Optimization Criterion. Therefore we complement the sequential factor model with prefix smoothing in order to explicitly model syntactical similarity.

Keywords: factor model, given names, recommender system

1 Introduction

We describe in this paper our contribution to the ECML PKDD Discovery Challenge 2013 (Offline Track). This years task task was to recommend given names to user of the name search engine "Nameling"[1] based on their historical search and clicking behavior. We interpret the problem as a classical recommender problem and use a purely statistical approach based on the user history. No meta data such as word similarity lists or geographic information for the users are used.
We use a factorized personalized Markov Chain (FPMC) [4] model in order to capture the user specific name preferences. The Bayesian Personalized Ranking (BPR) Optimization Criterion [3] is used to learn the latent variables of this model. We complement this factor model with syntactical similarity information by applying prefix smoothing to the name ranking.

2 Terminology and Formalization

Let $U = \{u_1, u_2, \ldots\}$ be the set of all users and $N = \{n_1, n_2, \ldots\}$ the set of all names. We further use $T = \mathbb{N}$ to identify user action over time.

The scoring function

$$\hat{y} : U \times T \times N \to \mathbb{R} \tag{1}$$

returns the estimated preference of a user U at time T for a name N. To obtain the user specific ranking for every user $u \in U$, each name is scored – i.e. $\hat{y}(u, t, n_1)$, $\hat{y}(u, t, n_2)$, etc. is computed – and the names are sorted by their score. We further define $D \subseteq U \times T \times N$ as the training set of available names which have been observed in the logs.

3 Sequential Factor Model

In order to extract training samples from the user activities, we first defined four indicator functions. These are then used to encode the training samples as sparse real valued features that can be used in a factorization machine. Finally, we give a formal definition of our prefix smoothing approach.

3.1 Indicators

Our model assumes that the name preference of a user u at time t can be explained by:

1. The ID of the user: u.
2. The ID of the name: n.
3. The **l**ast name selected by the user: $l : U \times T \to N$.
4. The **h**istory of all names selected by the user up to time t: $h : U \times T \to \mathcal{P}(N)$.

Besides these four indicators, the model should also take into account all interactions between indicators. E.g. the interaction between name n and last name $l(u, t)$ would model the effect for choosing name n if the name $l(u, t)$ has been selected before. In total, the first three indicators correspond to a personalized Markov chain [4]. The forth indicator can be seen as a Markov chain with long memory where all the history is aggregated into a single set.

3.2 Factorization Machine

The number of pairwise interactions between variables is high and cannot be estimated reliably with standard parametrization. Thus, we use a factorized parametrization [4] which allows to estimate parameters even in highly sparse data.

The ideas described so far can be realized with a factorization machine [2]. For this purpose, the four indicator variables are translated into a sparse real valued feature vector $\mathbf{x} \in \mathbb{R}^p$ with $p = |U| + |N| + |N| + |N|$ many predictor variables. The standard encoding described e.g. in [2] is used.

For example, for a case (u, t, n) let the values of the four indicators be:

1. user ID: 0,
2. name to rank: *Anna*,
3. last name selected by user: *Jana*,
4. history of all names selected by the user up to time t: {*Petra, Annabelle, Maria*}.

This can be encoded as a real valued feature vector of the form

$$\mathbf{x}(u,t,n) = (\underbrace{1,\ldots,0}_{|U|},\underbrace{0,1,0,\ldots}_{|N|},\underbrace{0,\ldots,1,0,\ldots}_{|N|},\underbrace{0,0.33,\ldots,0.33,\ldots,0.33,\ldots 0}_{|N|}). \tag{2}$$

The factorization machine (FM) model [2] of order $d = 2$ can be applied to the generated feature vector $\mathbf{x}$ and reads

$$\hat{y}^{\text{FM}}(\mathbf{x}(u,t,n)) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f} v_{j',f} \tag{3}$$

Here, $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^p, V \in \mathbb{R}^{p \times k}$ are the model parameters, $k \in \mathbb{N}$ is the size/ dimensionality of the latent space. Thus, the model has one feature vector $\mathbf{v}_i$ for each variable x_i.

Empirically inspecting the generated recommendations with this model shows (see Table 2) that the semantic meaning of names is found: e.g. if a user searches mainly for female names, female names are recommended; if the user selects typically short names, short ones are recommended, etc. The reason for the success is that the model automatically finds latent features $\mathbf{v}_n \in \mathbb{R}^k$ for each name n which describe its characteristics. Such characteristics could be gender, name length, etc.

3.3 Prefix Smoothing

In general, the proposed model can express any kind of pairwise relation between names[1]. However, under small data sizes, the model might have problems to find relations between infrequent names. To make an example, the model can express and will automatically learn that *Anna* and *Anne* are syntactically similar or that *Farid* and *Behrouz* are semantically similar (both Persian masculine names) if the data logs are large enough. However, the size of the observed logs is limited and the model cannot learn all these relations reliably – especially not for infrequent names.

To overcome the problem, we inject some syntactical relations manually into the model. We create indicators stating that two names (name n and last name $l(u,t)$) share a prefix of length m – we consider prefix lengths of $m \in \{1,2,3,4,5,6\}$. We add these indicators about syntactical similarity to the FM model mentioned above:

$$\begin{aligned}\hat{y}(u,t,n) := \hat{y}^{\text{FM}}(\mathbf{x}(u,t,n)) \\ + \sum_{m \in \{1,2,3,4,5,6\}} z_m \, \delta(\text{prefix}(n,m) = \text{prefix}(l(u,t),m)),\end{aligned} \tag{4}$$

where δ is the indicator function – i.e. $\delta(b) = 1$ if b is true – and $\text{prefix}(s,m)$ returns the prefix of string s of length m.

The final model is slightly more complex and considers also syntactical similarity with the next-to-last name:

$$\begin{aligned}\hat{y}(u,t,n) := \hat{y}^{\text{FM}}(\mathbf{x}(u,t,n)) \\ + \sum_{t' \in \{t,t-1\}} \sum_{m \in \{1,2,3,4,5,6\}} z_{m,t'} \, \delta(\text{prefix}(n,m) = \text{prefix}(l(u,t'),m)).\end{aligned} \tag{5}$$

[1] Note that semantic or syntactical similarities are also just pairwise relations.

Please note that the idea of prefix smoothing – i.e. the indicator $\delta(\text{prefix}(n, m) = \text{prefix}(l(u, t), m))$ – can be used directly in the FM by enlarging the feature vector $\mathbf{x}$. Using this representation, the parameters $z_{m,t}$ are part of the $\mathbf{w}$ parameters of the original FM. Extending the prefix smoothing to longer prefixes as well as taking into account names earlier then $t-1$ could further improve the model.

4 Learning

We have a lot of positive samples if we assume that the user likes names he interacts with but we know little about other names. Encoding all names the user didn't interact with as negative samples can introduce wrong user preferences since we can not distinguish between a name the user knowingly ignored and a name the user has never seen. We avoid this problem by using a pairwise loss function.

4.1 Optimization Criterion

The model parameters of the FM are learned by discriminating between previously selected and unselected names. This optimization criterion has been proposed for item recommendation as BPR (*Bayesian Personalized Ranking*) [3] and has been used in several other recommendation tasks including sequential recommendation [4]. For the task of name recommendation it reads:

$$\text{BPR-OPT} := \sum_{(u,t,n)\in D} \sum_{n_2\in(N\setminus\{n\})} \ln \sigma(\hat{y}(u,t,n) - \hat{y}(u,t,n_2)) - \lambda_\Theta ||\Theta||^2 \quad (6)$$

where $\hat{y}$ is the FM (using the predictor variables encoded in the real valued vector $\mathbf{x}$) and Θ is a vector containing the model parameters $V, \mathbf{w}, w_0$.

4.2 Algorithm

The standard BPR algorithm [3] is a stochastic gradient descent (SGD) algorithm. The algorithm samples first a positive observation $(u, t, n) \in D$ and then a negative name n_2 uniformly from $N \setminus \{n\}$. A gradient step is done on this pairwise comparison. In our implementation, instead of using a uniform distribution for negative names, the names are sampled approximately proportional to their expected rank.

Even though FM parameters and prefix smoothing could be learned jointly with BPR, for simplicity[2] we learned only the FM parameters with BPR and selected the prefix smoothing parameters (only 12 parameters) manually.

[2] The reason for separating both parts were only of practical matter because we reused an existing implementation.

4.3 Ensembling Factor Models

The BPR algorithm is a point estimator and returns one ranking. We consider uncertainty in the ranking by running the learning algorithm three times, each time with a different sampling hyperparameter. While training each model, we select[3] in total 20 iterations for which we predict the ranking each – i.e. we have 20 (slightly) different scores $\hat{y}(u, t, n)$ for each triple (u, t, n). The final scoring function is an unweighted average of the 20 scoring functions.

5 Experimental Results

In this section, we first present our scores from the official leaderboard and then discuss the latent features learned by our model on randomly selected samples.

5.1 Evaluation on Holdout Set

We separate the training data into a new validation and training set using the splitting script provided. This gives us a training set with 60.922 user and a test set with 13.008 user. We use the first 3.000 user from the new test set to calibrate our models.

5.2 Results

Table 1 shows the results we achieved on the official leaderboard. The table contains also the scores that we obtained on our validation set. The score difference between validation and challenge data might be partially explained by the different ensemble size that we used in the two evaluations. We ensemble 20 rankings for the last submitted model (as described in section 4.3) but an ensemble of 100 rankings for our validation set evaluation. After some last minute changes we had only time to select 20 rankings for the final submission.

Table 1. Results as reported on the challenge leaderboard and the official evaluation script (modified MAP@1000).

Model	validation	challenge
most-popular-item	0.0294	0.0259
FM-ens (with prefix smoothing)	0.0550	0.0472

[3] The selected iterations were chosen close to the best iterations on the holdout set, i.e. slightly before and after the best iteration.

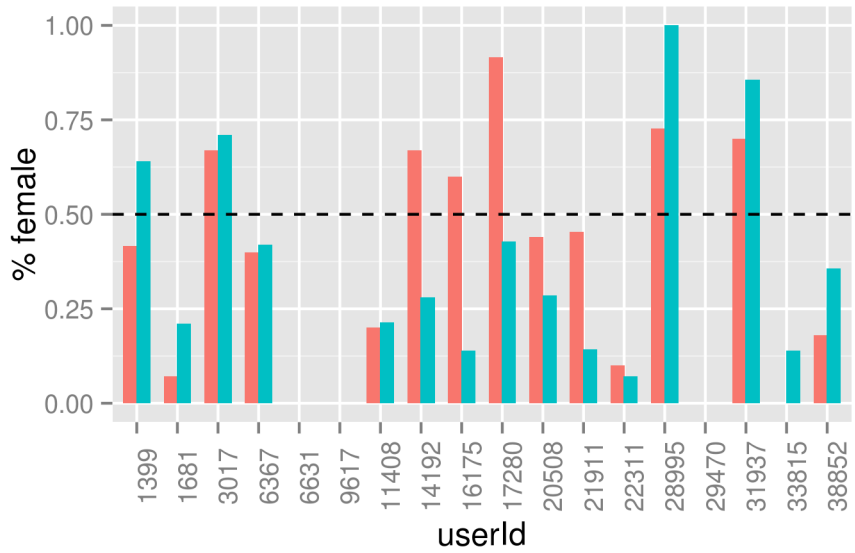

Fig. 1. Each bar pair represents the percentage of female names in the user history $h(u,t)$ (left) and the top 14 recommended names (right). The users evaluated here are the same as listed in Table 2. Similar height of both bars (left and right) for one user indicates that the user-preferred ratio of female to male names is closely reflected in the recommendations.

5.3 Ranking Analysis

In this section, we demonstrate on samples that our model is able to represent name similarities and user preferences through its latent variables. We illustrate this on name characteristics such as gender or length because the effects of those items are easy to recognize. More subtle effects might also be captured by our model but an casual inspection on a small sample might not suffice to identify them.

We select a subset of users from the challenge test set that have at least 30 and at most 50 activities (This is done to avoid users without history and to save space by avoiding user with a very large history). From this set, we randomly select 18 users and list them in table 2. The first line $h(u,t)$ for each user lists all names in his user history. Duplicates and names that are not used in the competition (not listed in namelist.txt) have been removed. The second line lists the top 14 recommendations generated by our FM model[4].

[4] The number of listed recommendations and the number of randomly selected users are adjusted to the available space.

Table 2. Top 14 recommended names, for a random selection of users. Top14 shows the list of recommended names, where the leftmost name is the top recommended one. $h(u,t)$ are all names in the user history up to the last available time t. The last name selected by the user is pried in bold.

userId	set	names
3017	$h(u,t)$	tanja, birgit, antje, karin, heide, kevin, maria, arthur, , mohammed, celine, cem, **mia**
	top14	emma, anna, julia, johanna, michael, katharina, marie, thomas, sophie, eva, alexander, christian, charlotte, lena
1681	$h(u,t)$	roman, boris, leon, robin, david, stikar, krasimira, julian, fabian, erik, henrik, **bastian**
	top14	emma, daniel, alexander, thomas, anna, david, christian, andreas, michael, sebastian, paul, jan, julia, kevin
9617	$h(u,t)$	tobias, lasse, niklas, malte, anton, **justus**
	top14	malte, jan, arne, ole, mats, nils, tim, lars, mika, finn, ben, paul, matti, linus
22311	$h(u,t)$	jakob, karin, paula, charlotte, luise, miriam, rosa, caroline, olivia, **marie**
	top14	emma, pauline, greta, anna, luisa, julia, lena, marie, paul, johanna, mathilda, ida, maria, frieda
38852	$h(u,t)$	alma, reto, christoph, philipp, paul, peter, remo, karl, sarah, lukas, **alex**
	top14	emma, anna, alexander, julia, katharina, johanna, sebastian, lena, michael, marie, hannah, maximilian, jakob, elisabeth
17280	$h(u,t)$	ida, lotta, lena, sara, charlotte, emma, greta, nora, leo, leonie, emilie, **mara**
	top14	emil, greta, paul, emma, theo, frieda, anna, oskar, anton, paula, jakob, ben, moritz, julia
1399	$h(u,t)$	verena, kay, anni, eva, effi, friedrich, friedolin, xavier, james, paulchen, resi, **paul**
	top14	emma, anna, michael, thomas, christian, stefan, mia, alexander, andreas, markus, marie, daniel, greta, paul
21911	$h(u,t)$	mia, claudia, jasmin, nicolas, niklas, andris, leinis, lara, lavina, ben, **kilian**
	top14	emma, sophie, laura, julia, anna, mia, leon, lena, david, lea, sarah, luca, sophia, lisa
29470	$h(u,t)$	lasse, kjell, bodo, enno, benno, luis, mads, mats, **caspar**
	top14	louis, henri, emil, ben, levi, finn, paul, leo, ole, max, jonas, anton, till, noah
11408	$h(u,t)$	clemens, paul, anton, matthias, **isabel**
	top14	emma, anna, emil, jakob, alexander, andreas, thomas, julia, michael, moritz, felix, peter, jonas, johannes
28995	$h(u,t)$	noa, luitgard, heidrun, yeni, nova, zoe, naemi, sune, ruben, cano, **jon**
	top14	mila, emma, luca, mia, laura, lilith, mara, sarah, leonie, lea, julia, anna, sophie, sandra
6367	$h(u,t)$	jonas, anna, esther, anja, maris, linda, lorenz, lennard, jannes, **gerhard**
	top14	emma, anna, paul, julia, emil, greta, alexander, jakob, david, daniel, jonas, katharina, johanna, anton
33815	$h(u,t)$	raphael, yannick, timon, jol, rafael, neo, nearchos, iason, jason, **phineus**
	top14	daniel, alexander, thomas, david, sebastian, emma, christian, michael, anna, andreas, leon, jonas, kevin, elias
16175	$h(u,t)$	mirka, liane, irina, vincent, **carlos**
	top14	alexander, thomas, daniel, michael, andreas, christian, david, peter, sebastian, anna, emma, martin, markus, maximilian
6631	$h(u,t)$	tim, ellis, lino, nino, paul, leon, leonard, **tobi**
	top14	luis, ben, luca, finn, noah, jonas, max, julian, leo, lukas, tom, maximilian, jan, mika
14192	$h(u,t)$	jasmin, maja, matthias, felix, emilia, **luisa**
	top14	emma, paul, leon, anna, daniel, thomas, sebastian, moritz, alexander, simon, christian, julia, mia, max
31937	$h(u,t)$	ida, kalle, melanie, mara, katja, merle, nele, juno, linda, **marlen**
	top14	greta, emma, lotta, lina, mia, anna, ella, emil, frieda, lotte, charlotte, lasse, frida, julia
0508	$h(u,t)$	luisa, anna, bettina, emilia, yngwie, oskar, emil, albert, **frederick**
	top14	anton, paul, jakob, emma, moritz, theo, johann, karl, julia, julius, greta, paula, felix, jonathan

5.4 Discussion of Learned Effects

Even though the predictive strength is best jugged by the leaderboard score, we want to give an impression of the kind of relationships that our model has learned from the training data. Please note that the rankings presented here are without prefix smoothing. This means that the model had no information on how long or which characters a certain name contains since names are only represented by unique identifiers.
Our model learns to distinguish between **female** and **male** names and recognizes if a user prefers male or female names. For users that have very strong preferences, such as user 6631, 9617, 29470, 3017, 28995, the recommendations that are accordingly balanced (see Table 2 and Figure 1). For users with very few user activities this becomes less reliable as can be seen on users; 16175 and 14192.
The model also learned if a name is **long** (user 3017, 38852, 16175) or very **short** (user 9617, 29470). It also recognizes if a user prefers **infrequent** names User 28995 for example has *kjell* and *enno* in his history and gets names like *levi* and *finn* recommended. **Double consonant** names are surprisingly frequent in the recommendations for user 31937, while names with **similar prefixes** have similar ranks for user 16175 (*martin, markus, maximilian*) or user 20508 (*julia, julius*) and also *lotte* and *charlotte* are ranked next to each other for user 31937. A more detailed analysis of the learned ranking could reveal more information on how people judge the similarities of given names.

6 Conclusion

In this paper, we have shown that a Factorization Machine [2] is well suited for the task of recommending given names. When the model parameters are optimized with respect to the personalized ranking criterion BPR-Opt [3], the latent variables are able to express name preferences such as name length and gender. Is important to note that this information is not part of the model input. Being able to learn this characteristic is useful if this information is not readily available.

The data used in this competition were strongly dominated by German speaking users (according to the user location obtained from the ip addresses) but our approach is supposed to work equally well with data that contain names from different alphabets or a user base that contains strong regional preferences without any modification.

We like to point out the close relation of given name recommendation to tasks such as movie recommendation where factor models have been very successful. The latent variables in this competition represent here syntactic and semantic similarity instead of movie related characteristics like genres or common actor. We have further shown that for infrequent names and small data sets the injection of regularizing information such as syntactical similarity does improve the prediction quality. We however expect that the benefit of information injection diminishes with the size of available data.

7 ACKNOWLEDGMENTS

We gratefully acknowledge funding by *Baden-Württemberg Stiftung*.

References

1. Mitzlaff, F., Stumme, G.: Namelings - discover given name relatedness based on data from the social web. In Aberer, K., Flache, A., Jager, W., Liu, L., Tang, J., Guret, C., eds.: SocInfo. Volume 7710 of Lecture Notes in Computer Science., Springer (2012) 531–534
2. Rendle, S.: Factorization machines with libFM. ACM Trans. Intell. Syst. Technol. **3**(3) (May 2012) 57:1–57:22
3. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI 2009). (2009)
4. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing personalized markov chains for next-basket recommendation. In: WWW '10: Proceedings of the 19th international conference on World wide web, New York, NY, USA, ACM (2010) 811–820