Context-Tree Recommendation vs Matrix-Factorization: Algorithm Selection and Live Users Evaluation

Stéphane Martin,[1] **Boi Faltings,**[2] **Vincent Schickel**[3]
[1]EPFL, LIA, Switzerland, [2]EPFL, LIA, Switzerland, [3]Prediggo SA, Switzerland
stephane.martin@epfl.ch, boi.faltings@epfl.ch, schickel@prediggo.com

Abstract

We describe the selection, implementation and online evaluation of two e-commerce recommender systems developed with our partner company, Prediggo. The first one is based on the novel method of Bayesian Variable-order Markov Modeling (BVMM). The second, SSAGD, is a novel variant of the Matrix-Factorization technique (MF), which is considered state-of-the-art in the recommender literature.

We discuss the offline tests we carried out to select the best MF variant, and present the results of two A/B tests performed on live ecommerce websites after the deployment of the new algorithms. Comparing the new recommenders and Prediggo's proprietary algorithm of Ontology Filtering, we show that the BVMM significantly outperforms the two others in terms of CTR and prediction speed, and leads to a strong increase in recommendation-mediated sales. Although MF exhibits reasonably good accuracy, the BVMM is still significantly more accurate and avoids the high memory requirements of MF. This scalability is essential for its application in online businesses.

1 Introduction

Today, recommender systems are essential components of commercial websites. They contribute to the sales effort by guiding the customers to interesting products and by personalizing the shopping experience. At the same time, the nature of online traffic imposes stringent time limits on the computation of recommendations. Online businesses also expect the systems to serve a large number of customers on affordable infrastructures. It is thus crucial to develop algorithms that attract the attention of the customers while remaining efficient in computation and memory consumption.

We carried out this work in cooperation with Prediggo SA, a start-up specializing in recommendation- and search-solutions for e-commerce (https://www.prediggo.com/). Its main recommendation algorithm is a particularly efficient and scalable content-based method, the *Ontology Filtering* (OF). The purpose of this research was to extend Prediggo's portfolio of solutions with two additional algorithms, implemented in Java within its already existing framework, in order to benefit from recent advances in recommender system technology.

The first one is a novel approach based on a Markovian algorithm, the Bayesian Variable-order Markov Model (BVMM). The second one, SSAGD, is a variant of Matrix-Factorization (MF). The family of MF algorithms is considered the state of the art in the field of recommendation; they thus offer a solid baseline.

We will start with a description of the three types of algorithms. We then describe the offline tests we performed, given the many variants based on MF, to develop the best one for our purpose, resulting in SSAGD. Finally, we discuss the live evaluations we carried in production conditions on the websites of two of Prediggo's customers. These experiments highlight the excellent performance of the BVMM. By contrast, we found MF to obtain smaller CTRs and to be considerably slower as well as memory-greedy.

2 Related Work

2.1 Matrix-Factorization

Matrix-factorization is one of the most widely studied approaches to recommendation and is known to provide a very good accuracy. Many variants of its basic method have been developed since Sarwar *et al.*'s original proposal (Sarwar et al. 2000), which relied on *Singular Value Decomposition* to extract vectors of user- and item-latent preferences.

Funk (Funk 2006) pushed the idea further and avoided the roundabout computation of a SVD by directly computing the latent coefficients through regularized gradient descent. Paterek (Paterek 2007) examines a number of improvements over Funk's regularized factorization. Among them, the *asymmetric SVDs*, NSVD1 and NSVD2, which appear to work quite well in practice (Pu and Faltings 2013). We discuss the second in Section 3. Paterek also introduced the idea of adding item- and user-biases to the latent vectors (Koren, Bell, and Volinsky 2009).

Koren (Koren 2008), together with Bell (Koren and Bell 2011) developed SVD++, to incorporate several improvements over the basic SVD method. Although they work with the Netflix dataset — with implicit ratings —, they notably observe that taking into account the *implicit feedback* offered by the users improves the accuracy (Koren and Bell 2011). They do so by introducing variables expressing the mere presence or absence of ratings, in a way reminiscent of Paterek's approach.

The problem of implicit feedback, particularly relevant in our case, is also tackled in (Oard and Kim 1998) and in (Hu, Koren, and Volinsky 2008). The latter develop a model with an additional array of $0/1$-variables in order to distinguish between the ratings of the users and the confidence we can have in those ratings. The resulting cost-function forces them to resort to a policy of optimization through *Alternating Least Squares*. A similar procedure, *Alternating-Least-Squares with Weighted-λ-Regularization*, dubbed *ALSWR*, has been proposed by Zhou *et al.* (Zhou et al. 2008), although they apply it on the Netflix Dataset, with explicit ratings. Both algorithms are part of the Apache Mahout library (https://mahout.apache.org/), which we used in our preparatory tests.

2.2 Ontologies

Just like MF, Ontologies are now widespread tools in the field of recommendation (Ricci et al. 2010, *passim*), especially in relation with *content-based* methods. Noteworthy examples discussed in the literature cover the recommendation of news (IJntema et al. 2010; Cantador, Bellogín, and Castells 2008; Frasincar, Borsje, and Levering 2009), multimedia (Juszczyszyn, Kazienko, and Musiał 2010) and commercial products (Aciar et al. 2007; Lee et al. 2006).

In particular, Prediggo relies on ontologies learned through an automated process (Schickel-Zuber 2007; Schickel-Zuber and Faltings 2007), which avoids the costly intervention of human experts. (Drumond and Girardi 2008) provides an interesting overview of this topic.

2.3 Context-Trees and Sequential Models

Context-trees are tree-shaped structures used to model sets of sequences. A large number of algorithms relying on such structures exist in the literature (Begleiter, El-Yaniv, and Yona 2004), often in relation with data-compression (Rissanen and Langdon 1979; Rissanen 1983) and *arithmetic coding* (Rissanen and Langdon 1981; Cleary and Witten 1984).

Rissanen's algorithm *Context* (Rissanen and Langdon 1979; Rissanen 1983) was among the first applications. Since then, several related methods have been proposed, such as Context-Tree Weighting (Willems, Shtarkov, and Tjalkens 1995), Prediction by Partial Match (Cleary and Witten 1984), Probabilistic Suffix Trees (Bejerano and Yona 2001), etc. Begleiter *et al.* (Begleiter, El-Yaniv, and Yona 2004) offer a valuable introduction to this topic. Besides data-compression, context-trees have also found use in musicology (Pearce and Wiggins 2003), computational biology (Bejerano and Yona 2001), and linguistics (Galves et al. 2012). In particular, the Bayesian Variable-order Markov Model (BVMM), introduced first in (Dimitrakakis 2010), has been applied to the recommendation of online news on several news-sites (Garcin et al. 2014).

The notion of sequence, central to the definition of context-trees and present to some extent in all our implementations, has already been noted to be quite relevant for recommendation (Quadrana, Cremonesi, and Jannach 2018). Interestingly, the incorporation of a sequential aspect into an MF-framework by means of time-decaying variables is also applied by Twardowski (Twardowski 2016).

3 Matrix-Factorization

Matrix-factorization (MF) is a popular method of preference extraction that has received numerous applications in the field of recommender systems. It represents an instance of dimensionality reduction (Bobadilla et al. 2013), where the ratings assigned by users to some items are decomposed into vectors of latent values. Those are recombined during the recommendation stage to estimate the scores of the unrated items.

The training data consist of a *rating-matrix* $\boldsymbol{R}$, in which the value $\boldsymbol{R}_{u,i}$ at the intersection of the uth row and the ith column expresses the utility of item i for user u. In a typical scenario, only a small number of ratings are known for each user, and the problem is to estimate the unknown values. The items with the highest estimates can then be recommended.

The simplest applications of MF (Sarwar et al. 2000) decompose the rating-matrix through Singular-Value Decomposition (SVD) and truncate the resulting matrices $\boldsymbol{R} = \boldsymbol{USV}^\top$ to keep only the K most important factors. Many refinements over this idea of matrix-decomposition exist in the literature. They typically decompose the original $\boldsymbol{R}$ into two matrices of user- and item-preferences, $\boldsymbol{U}$ and $\boldsymbol{V}$, such that each row vector $\boldsymbol{U}_u$ embeds the tastes of one user u, while each column vector $\boldsymbol{V}_i$ corresponds to an item i. The product $\boldsymbol{U}_u\boldsymbol{V}_i$ yields an estimate of the utility of i for u.

The Taste component of Apache Mahout, a project of the Apache Software Foundation, offers several MF-based recommender systems. In order to ease the selection process and, in the end, to only have to implement the most promising algorithms, we fitted three of those into our system: The `RatingSGDFactorizer`, described in the Apache documentation as a "*Matrix factorization with user and item biases for rating prediction, trained with plain vanilla SGD*", the `SVDPlusPlusFactorizer`, defined as "*SVD++, an enhancement of classical matrix factorization for rating prediction*", with a citation to (Koren 2008), and the `ALSWRFactorizer`, which relies on "*Alternating-Least-Squares with Weighted-λ-Regularization*" and "*also supports the implicit feedback variant of this approach*" as in (Hu, Koren, and Volinsky 2008). RSGD thus represents a basic version of gradient-descent factorization, similar to (Funk 2006). The other recommenders are more sophisticated methods, which we would expect to perform better.

One problem posed by online traffic, especially for session-based recommenders, is the constantly changing user-base. All we know about a user is the sequence of his actions on the site. At training time, we thus create the rating matrix $\boldsymbol{R}$ by collecting all the sequences of page-views per session logged in the past, and by extracting the subsequences of length at most $D > 0$. Those subsequences constitute the rows of $\boldsymbol{R}$, with each element of a row having a value corresponding to its chronological position in the sequence. We assign a value of D to the most recently visited product, $D - \gamma$ to the last but one, $D - 2\gamma$ to the last but two, etc[1]. The items not mentioned have a value of zero.

[1]After some tests, we found that $D = 15$ as maximum sequence length and $\gamma = 0.5$ gave satisfactory results.

As a user progresses on a website, the row that matches his browsing history will change. We use a trie to keep track of the advance of each user and to find the corresponding row in the matrix of user-factors. When no row matches his current sequence s, we take the row matching the longest suffix of s. This is similar to the policy applied in a context-tree (Section 5).

Paterek (Paterek 2007) studied several variants of Funk's gradient descent algorithm (Funk 2006), among which the so-called NSVD2. The underlying idea is to compute a factorization of $\boldsymbol{R}$ through regularized gradient descent, and to construct the coefficients of a user's latent vector as the sums of the corresponding coefficients of the items rated by the user. One advantage of this method is that the vectors of the new users can be deduced from their ratings even if they appear after the training stage.

To account for the sequential nature of the data, we also implemented our own adaptation of NSVD2, the *Sequential, Session-based, Asymetric factorization by Gradient Descent*, or SSAGD. Since it follows Paterek's idea, SSAGD does not need specific user-vectors[2]. Each time a request for recommendation is received, we create an estimate of the user-vector $\boldsymbol{U}_u$ by filling each dimension with the sum of the corresponding values in the item-matrix $\boldsymbol{V}$ for all the items i in the browsing history s, weighted according to their position in s: $\boldsymbol{U}_u = \sum_{i \in s} w_i \boldsymbol{V}_i^\top$. The most recent item receives a weight w of $D\gamma$, the last but one a weight of $(D-1)\gamma$, the last but two $(D-2)\gamma$, etc.

Funk's implementation (Funk 2006) uses early stopping to avoid overfitting. The Mahout implementations work by reducing the learning rate λ after each iteration. In our case, we found that we obtained better results by reducing the learning rate each time the error rate increases. It is a well-known property of gradient descent that the error curve sometimes experiences bumps or spikes, that suddenly increase its value. When that happens, we restore the coefficients to their previous valid values and set $\lambda := 0.9\lambda$ before the next iteration. Once the error rate becomes too small, we stop the factorization altogether.

4 Ontology Filtering

Ontology Filtering (OF) (Schickel-Zuber 2007; Schickel-Zuber and Faltings 2007) was originally conceived to provide a theoretically sound basis to the use of ontologies in recommendation. Those ontologies, in the form of directed acyclic graphs, are learned by the system during a training stage run once a day over the updated catalogue. Their graph structure defines a notion of proximity and neighbourhood among the items (Schickel-Zuber and Faltings 2007), with the immediate descendants of the same parent concept being closest neighbours, the items sharing a grandparent being more removed, etc.

At recommendation time, the browsing behaviour of the users is followed by the system while they navigate on a website. Their actions are stored into session-based profiles, used in conjunction with a Multi-Attribute Utility function (MAUT), which estimates the utility of a product for a user given his profile. Each time N recommendations are requested for a user, a subset of candidates is preselected by picking the $M > N$ nearest ontological neighbours of the item currently considered by the user. This subset is then ranked by MAUT, and the best items can be returned.

5 Bayesian Variable-Order Markov Models

Since Rissanen's initial proposal (Rissanen and Langdon 1979), context-tree based algorithms have proved well suited to handle problems of *sequence prediction*. The domain of interest of such problems consists of sequences $s \in \mathscr{I}^\star$, i.e. ordered lists of items $\sigma \in \mathscr{I}$, and their aim is to estimate the probability $\mathbb{P}(\sigma|s)$ for any σ to follow a given s. We describe here a particular variant of context-tree model, the Bayesian Variable-order Markov Model (Dimitrakakis 2010), or BVMM, that has turned out to be convenient for the task of recommendation (Garcin et al. 2014).

Given a sequence s and a dataset $\mathscr{S}$ made of other previously observed sequences, a rudimentary way to assess the probability of σ to come after s would be to look into $\mathscr{S}$ for the sequences identical to s, and make a count of the items $\sigma_0, \sigma_1, \ldots$ found after them. One could then estimate $\mathbb{P}(x = \sigma|s)$ as $\approx \mathbb{N}(\sigma|s)/\sum_i \mathbb{N}(\sigma_i|s)$, with $\mathbb{N}(x|s)$ the number of occurrences of x after s.

It is however possible that the sequence s has never been observed before, and cannot be found in the dataset. Since the number of sequences is exponential in the number of items, any realistic dataset will be quite sparse when the number of items becomes too big. One solution is then to use the suffixes $s' \in \text{suff}(s)$ of s that have been actually observed. Those suffixes, each of which defines a *context*, can be naturally arranged into a tree-shaped structure, which also allows for their efficient retrieval. This *context-tree* forms the basis of a number of algorithms, that vary in the way they select the relevant suffixes and handle the corresponding counts of items (Begleiter, El-Yaniv, and Yona 2004).

Since different suffixes of the same sequence can be followed by different counts of items, the question arises of which should be selected, and how the differences should be resolved. The BVMM proceeds by aggregating the local probabilities $\mathbb{P}_{s_i}(\sigma|s_i)$ computed for each suffix s_i of s into a global probability $\hat{\mathbb{P}}(\sigma|s)$.

We associate to each node n_{s_i} a *weight* w_{s_i}, which represents the probability that n_{s_i} be the right context to determine the next item, provided no longer suffix is the right one. This weight can be interpreted as a stopping probability in the search for the right suffix. Given a sequence s, we imagine trying all the known suffixes, starting from the longest and deepest in the tree up to the empty suffix at the root. The global probabilities are then obtained by marginalization[3]:

$$\hat{\mathbb{P}}(\sigma|s) = \sum_{s_i \in \text{suff}(s)} \pi_{s_i} \mathbb{P}_{s_i}(\sigma|s_i)$$

with $\pi_{s_i} = w_{s_i} \prod_{-D \leq j < i} (1 - w_{s_j})$.

[2]The initial rating matrix is constructed as described above, with ratings decreasing with their position in the sequence.

[3]The suffixes are indexed with negative numbers from 0 at the root to $-D$ at the deepest leaves.

It can be shown (Dimitrakakis 2010) that, for any item σ and any history s, the aggregated probability can be computed with the recurrence relation

$$\hat{\mathbb{P}}(\sigma|s_{i+1}) = w_{s_{i+1}}\mathbb{P}(\sigma|s_{i+1}) + (1 - w_{s_{i+1}})\hat{\mathbb{P}}(\sigma|s_i)$$

while the weights can be updated according to

$$w_{s_{i,t+1}} = w_{s_{i,t}}\mathbb{P}(\sigma|s_i)/\hat{\mathbb{P}}(\sigma|s_i)$$

This results in the algorithm outlined in Algo. 1. If D is the maximum length allowed for the sequences, this algorithm can be updated in $O(D)$ each time a σ is observed after a sequence (Dimitrakakis 2010). It also provides the probability of the next item in $O(D)$. If our aim is to select the N most probable items from a set of size S ($S \gg N$), the recommendation will be in $O(SD)$.

Algorithm 1 The BVMM algorithm.

```
$s$ is the current context
$\sigma$ is the current observed item

**Learning stage:**
  Initialize $q := 0$
  **for each** node $n$ on $s$ from the root to the leaf**:**
      $p_n :=$ probability of $\sigma$ for $n$'s expert
      $q := w_n p_n + (1 - w_n)q$
      $w_n := w_n p_n / q$
      Update expert on $n$
  **if** $s\sigma$ is not in the tree **then**
      Add a path corresponding to $s\sigma$

**Recommendation stage:**
  **for each** item $\sigma_i$**:**
      Initialize $q_i := 0$
      **for each** node $n$ on $s$ from the root to the leaf**:**
          $p_{n,i} :=$ probability of $\sigma_i$ for $n$'s expert
          $q_i := w_n p_{n,i} + (1 - w_n)q_i$
  **return** $N$ items with highest probability $q$
```

When applied to online recommendation (Garcin, Dimitrakakis, and Faltings 2013), the items σ represent webpages, so that each sequence recapitulates the browsing history of a user. Since the catalogue of a webshop changes only slowly, the tree can be updated at relatively distant intervals. The visits are logged in a database, and used once a day or a few times a day to update the weights and the experts all at once.

6 Pre-Production Tests

We thus considered six algorithms, three third-party implementations from the Mahout library: RSGD, SVD++, and ALSWR (used in its implicit-feedback variant), and three implementations of our own: OF, BVMM, and SSAGD. To provide a minimal baseline, we included a trivial Random recommender, which picks its recommendations with uniform probability. The implementation of the modules did not create particular problems. Prediggo's software, written in Java, already uses a system of interfaces for the various policies, that can be implemented straightforwardly. The Mahout libraries were wrapped inside an *ad hoc* module, while SSAGD and the BVMM were implemented directly. The rating matrix in SSAGD was a sparse datastructure based on an array of linked lists of cell objects. The factorized vectors were originally arrays of Java `doubles`, later changed to arrays of `floats` in order to gain space.

In order to reduce their time-complexity and ensure the semantic relevancy of the output, we apply to all our algorithms the same prefiltering operation used in OF: an initial set of ≈ 400 candidates is selected by looking within the ontologies for the nearest neigbours of the item currently visited. The main algorithm is then applied on those candidates to extract the final recommendations.

We first tested all seven strategies on a dataset collected from the traffic of one of Prediggo's clients. Those tests were performed offline by feeding requests for recommendations into the system running on a local machine. The requests reproduced the chronological unfolding of the previously recorded series of 100,000 page-views. In each step, the system had to return $N = 5$ recommendations given a visit on a product by a given user.

Every 10,000 steps, the simulation was stopped and the four MF recommenders and the BVMM were updated by launching a new learning stage. Each learning stage used the clicks previously received as training data. The clicks that were to follow were of course not used at that moment. This procedure is different from the more usual method of splitting the dataset into a training set and a testing set, since each slice of data was first used for testing, then lumped into the training set. Since the evaluation was always carried on data yet unknown to the system, it nevertheless always maintains a strict separation of testing and training data.

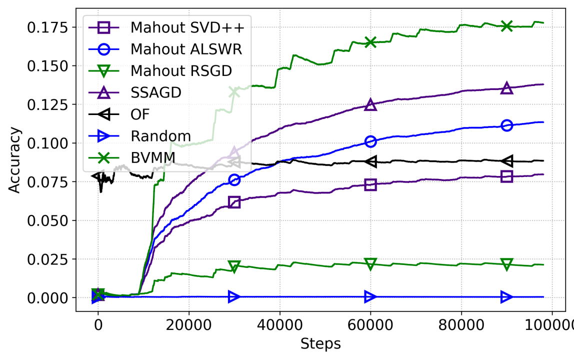

Figure 1: Offline tests, accuracy over 100,000 page-views.

Several options exist to assess the quality of a sequential recommender (Quadrana, Cremonesi, and Jannach 2018). Since we are interested in determining which of our candidates will provide the best CTR when used online, we measure accuracy as the precision over N outputs: We count a hit each time a user visits one of the items recommended in his previous step and a miss each time he does not. We then take the ratio of all the hits over the sum of hits and misses.

Fig. 1 shows the evolution of the accuracy of each recommender across the 100,000 steps. The BVMM obtains the best results with SSAGD and ALSWR coming second and

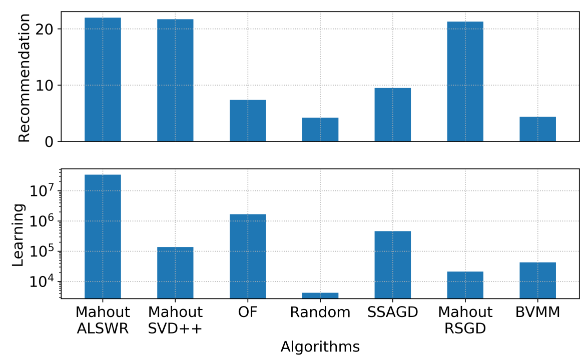

Figure 2: Offline tests, recommendation times (top) and learning times (bottom) after 100,000 page-views.

third. Fig. 2 (top) compares the mean times to compute one recommendation in ms. The three Mahout implementations are the slowest, while the BVMM is the fastest. SSAGD thus stands out as the most interesting in the MF family.

The recommendation times include an overhead covering the system's common operations. Since the random recommender has almost no work to perform on its own, its runtime can be taken as equal to this overhead. The BVMM is very close, due to the many opportunities for optimization (caching, precomputing, etc.) inherent in the algorithm.

The durations of the learning stages after 100,000 page-views (Fig. 2, bottom, in ms.) are also quite different among the algorithms: We had to use a logarithmic scale for the y-axis, due to the wide differences between the values. The ALSWR in particular suffers from prohibitively high learning times. Its training over the 100,000 clicks takes a little less than 10 hours. By contrast, SSAGD needs $\approx$ 8 minutes and the BVMM less than 1 minute. The computation of the ontologies, dependent only on the size of the catalogue, takes around 30 minutes.

Our implementations of MF are conceived to integrate the sequentiality inherent in the data — the fact that the users visit the products in sequence. This is done first, when filling the rating-matrix in the learning stage, by giving higher ratings to the most recent items. Then, in the case of SSAGD, by also over-weighting the latent parameters of the most recent items when deriving the user-vectors. Our tests showed that after 100,000 steps, SSAGD achieved an accuray of 0.14 with only the first weighting, of 0.105 with only the second, of 0.145 with both, and of 0.077 without any weighting. Obviously, both policies contribute to the accuracy.

Our offline tests thus convinced us that the Mahout implementations were not suitable for our purpose, since they do not achieve the best accuracy and appear slower than the other algorithms. By contrast, SSAGD is capable of good performance and is reasonably fast. Its close variant NSVD2 is also recognized in the literature as an excellent option (Pu and Faltings 2013). We hence selected it as representative of MF for the online experiments we will now discuss.

7 Live Evaluations

Once the best algorithms were selected, we could start preparing the production version of the new modules and organize two live A/B experiments on the websites of two of Prediggo's customers. The first one is a retailer of furniture, hightech products and home appliances, and operates a high-traffic website with a large catalogue ($\approx$ 10,000 products). The second, a webshop offering items of youth fashion, manages a smaller site with an average traffic and a more specialized catalogue ($\approx$ 1000 products). The system would be run in the usual conditions used by Prediggo to serve its customers through the *Software as a Service* (SaaS) model. We would rely on Docker to provide a lightweight virtualization, over machines running on Debian.

As the system was set up, we discovered however that the usual amount of memory allocated to each instance would not suffice. The servers were regularly crashing for lack of RAM during the preproduction launches. Earlier experiments had shown that the BVMM could run without issues on 5 GB on the same platform as the first experiment, but MF proved too greedy. We had to optimize the code further and to implement a scheduling algorithm to prevent the learning stages of MF and BVMM to occur at the same time — this is the moment when both consume the most memory. Even then, the learning stage of MF would still overwhelm the system and we had to increase the maximum memory of the JVM to 7 GB in production for the first customer. The maximum memory was left at 4.5 GB for the second.

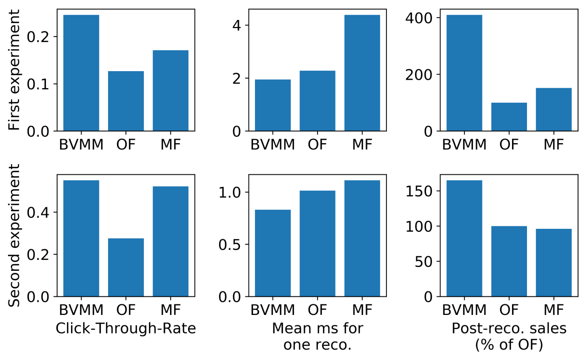

Figure 3: Online experiments: CTR, speed and sales.

In both experiments, the traffic was split into three buckets of comparable size, one for each algorithm. $\approx 34\%$ of the sessions received recommendations selected by the BVMM, $\approx 33\%$ were served by OF and $\approx 33\%$ by MF. The pages visited by the users, their clicks, their purchases and the recommendations they received were logged. The learning stages of OF and MF were scheduled early every morning when traffic is low. The rating-matrix of MF was limited to 100,000 rows, filled with the most recent sequences, and decomposed into 100 latent factors. The context-tree was limited to a size of 100,000 nodes built out of the same data. All three algorithms performed an initial step of ontological preselection as described in Section 6.

Table 1: Online experiments: Data and p-values.

1st exp.	Clicks	Impr.	CTR (%)	Sales (%)
OF	8035	6,331,286	0.127	100
BVMM	16,080	6,538,051	0.246	409.907
MF	10,807	6,314,160	0.171	152.108
	OF, BVMM	OF, MF	BVMM, MF	
p-values	≈ 0	9.23e^{-93}	1.586e^{-190}	
2nd exp.	Clicks	Impr.	CTR (%)	Sales (%)
OF	397	144,049	0.276	100
BVMM	823	149,433	0.551	164.981
MF	749	143,510	0.522	96.090
	OF, BVMM	OF, MF	BVMM, MF	
p-values	1.27e^{-31}	4.6e^{-26}	0.288	

7.1 First Experiment

The first experiment lasted from the 14th of March 2018 to the 1st of April 2018 and covered a total traffic made of $\approx$ 19,000,000 impressions of recommendations. Tab. 1 (top) gives the number of clicks with the CTRs in percent. The three p-values are computed through Fisher's exact test for each pair of algorithm. The 0-hypothesis is that they have equivalent abilities to attract clicks. The three values are small enough to confirm that this is not the case: The BVMM triggers significantly more clicks than the two others, while MF surpasses OF.

Tab. 1 also shows the amount of post-view sales (given in percent of OF for reasons of confidentiality). The values represent the aggregates of the purchases made by the users after they have been shown recommendations for the corresponding products within 24 hours. They should be interpreted with caution, since recommendations can influence purchase decisions only indirectly, among many other factors. The sample of sales registered during the experiments was also much smaller than the number of clicks, and exhibited a high variance. Additionally, the values may partly express the capacity of the recommenders to *anticipate* the purchases, not necessarily the capacity to *cause* them.

The mean time to generate one recommendation (Fig. 3, middle top) was 2.282 ms for OF, 1.946 ms for the BVMM and 4.39 ms for SSAGD. The BVMM and OF thus appear to lie in the same range of time-complexity, while MF, although still acceptably fast, is clearly slower.

7.2 Second Experiment

The second experiment was started on the 13th of March 2018 and ended on the 30th of the same month. The traffic ($\approx$ 440,000 impressions) and the number of clicks were significantly smaller than in the first experiment. Nevertheless, the p-values of Tab. 1 (bottom) confirm the superiority of the BVMM to attract clicks over OF. Although the BVMM still obtains a larger CTR, the comparison with MF is less significant. This might be due to the smaller number of products ($\approx$ 1000 against $\approx$ 10,000), which makes the task of predicting the right items easier and helps the less powerful algorithms. Although this is harder to quantify, the descriptions of the products supplied by the second company were also richer and more semantically expressive. This should favour a content-based recommender such as OF.

The differences in time-complexity follow a pattern similar to the first experiment, although they are less striking. The OF takes 1.014 ms on average to generate one recommendation, the BVMM, 0.831 ms and MF, 1.113 ms. The BVMM is still faster, as it was during the offline tests. Despite the limitations of the post-recommendation sales as a metric, they also point to significant differences among the algorithms, already visible on a relatively small site.

8 Conclusions

We discussed the implementation, testing and online evaluation of two recommender systems for e-commerce based on a context-tree model and on sequential matrix-factorization. The tests showed the value of both the BVMM and SSAGD for predicting the behaviour of online users. The BVMM in particular achieves the best CTR while remaining quite efficient. It also clearly increases the sales following a recommendation by up to four times.

This illustrates the importance of modeling the sequential nature of user-preferences. Our sequential adaptation of matrix factorization, SSAGD, doubles the accuracy in offline tests, while the explicit modeling of sequences by the BVMM increases accuracy even more. We believe that the importance of modeling sequences has been underestimated in the literature and that Markov models such as BVMM deserve more attention. The present work also suggests that carrying live evaluations systematically would bring more valuable insights and benefits than usually thought.

Significant complexity issues occur in the training of the MF model, which requires a lot of memory, and we had to expend considerable efforts of optimization to avoid crashes. While this problem does not concern the recommendation complexity, as the training stage is run only periodically, it represents a big hurdle in practical settings. Similar issues are likely to arise with neural models, which are slow to train and involve large structures of nodes. Neither OF nor the BVMM manifested a similar weakness. The same difficulties have been observed by others and explain the limited adoption of these algorithms by the industry.

The BVMM thus ensures both an excellent performance and the necessary scalability for use on commercial websites. It has since then been integrated into Prediggo's suite of algorithms and is offered to the customers as a standard recommendation strategy. Its performance on other webshops, with catalogues of $\approx$ 200,000 products, and the subsequent A/B tests we carried, confirm the results discussed above. On the other hand, the memory issues that plague MF forced Prediggo to leave it out of its product for now.

9 Acknowledgments

This work was supported by a grant from the Swiss Commission for Technology and Innovation CTI/KTI (contract 18384.1 PFES-ES).

References

Aciar, S.; Zhang, D.; Simoff, S.; and Debenham, J. 2007. Informed recommender: Basing recommendations on consumer product reviews. *IEEE Intelligent Systems* 22(3):39–47.

Begleiter, R.; El-Yaniv, R.; and Yona, G. 2004. On prediction using variable order markov models. *J. Artif. Int. Res.* 22(1):385–421.

Bejerano, G., and Yona, G. 2001. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics* 17(1):23–43.

Bobadilla, J.; Ortega, F.; Hernando, A.; and Gutiérrez, A. 2013. Recommender systems survey. *Knowledge-Based Systems* 46:109 – 132.

Cantador, I.; Bellogín, A.; and Castells, P. 2008. Ontology-based personalised and context-aware recommendations of news items. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '08, 562–565. Washington, DC, USA: IEEE Computer Society.

Cleary, J., and Witten, I. 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications* 32(4):396–402.

Dimitrakakis, C. 2010. Bayesian variable order markov models. In *JMLR*.

Drumond, L., and Girardi, R. 2008. A survey of ontology learning procedures. *WONTO* 427:1–13.

Frasincar, F.; Borsje, J.; and Levering, L. 2009. A semantic web-based approach for building personalized news services. *International Journal of E-Business Research (IJEBR)* 5(3):35–53.

Funk, S. 2006. Netflix update: Try this at home, https://sifter.org/simon/journal/20061211.html.

Galves, A.; Galves, C.; García, J. E.; Garcia, N. L.; and Leonardi, F. 2012. Context tree selection and linguistic rhythm retrieval from written texts. *Ann. Appl. Stat.* 6(1):186–209.

Garcin, F.; Faltings, B.; Donatsch, O.; Alazzawi, A.; Bruttin, C.; and Huber, A. 2014. Offline and online evaluation of news recommender systems at swissinfo.ch. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, 169–176. New York, NY, USA: ACM.

Garcin, F.; Dimitrakakis, C.; and Faltings, B. 2013. Personalized news recommendation with context trees. In *Proceedings of the 7th ACM Conerence on Recommender Systems*, RecSys '13, 105–112. New York, NY, USA: ACM.

Hu, Y.; Koren, Y.; and Volinsky, C. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, 263–272.

IJntema, W.; Goossen, F.; Frasincar, F.; and Hogenboom, F. 2010. Ontology-based news recommendation. In *Proceedings of the 2010 EDBT/ICDT Workshops*, EDBT '10, 16:1–16:6. New York, NY, USA: ACM.

Juszczyszyn, K.; Kazienko, P.; and Musiał, K. 2010. *Agent and Multi-agent Technology for Internet and Enterprise Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg. chapter Personalized Ontology-Based Recommender Systems for Multimedia Objects, 275–292.

Koren, Y., and Bell, R. 2011. *Advances in Collaborative Filtering*. Boston, MA: Springer US. 145–186.

Koren, Y.; Bell, R.; and Volinsky, C. 2009. Matrix factorization techniques for recommender systems. *Computer* 42(8):30–37.

Koren, Y. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, 426–434. New York, NY, USA: ACM.

Lee, T.; Chun, J.; Shim, J.; and goo Lee, S. 2006. An ontology-based product recommender system for b2b marketplaces. *International Journal of Electronic Commerce* 11(2):125–155.

Oard, D., and Kim, J. 1998. Implicit feedback for recommender systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, 81–83.

Paterek, A. 2007. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, 39–42.

Pearce, M., and Wiggins, G. 2003. An empirical comparison of the performance of ppm variants on a prediction task with monophonic music.

Pu, L., and Faltings, B. 2013. Understanding and improving relational matrix factorization in recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, 41–48. New York, NY, USA: ACM.

Quadrana, M.; Cremonesi, P.; and Jannach, D. 2018. Sequence-aware recommender systems. *CoRR* abs/1802.08452.

Ricci, F.; Rokach, L.; Shapira, B.; and Kantor, P. B. 2010. *Recommender Systems Handbook*. New York, NY, USA: Springer-Verlag New York, Inc., 1st edition.

Rissanen, J., and Langdon, G. G. 1979. Arithmetic coding. *IBM Journal of Research and Development* 23(2):149–162.

Rissanen, J., and Langdon, G. 1981. Universal modeling and coding. *IEEE Transactions on Information Theory* 27(1):12–23.

Rissanen, J. 1983. A universal data compression system. *IEEE Transactions on Information Theory* 29(5):656–664.

Sarwar, B. M.; Karypis, G.; Konstan, J. A.; and Riedl, J. T. 2000. Application of dimensionality reduction in recommender system – a case study. In *IN ACM WEBKDD WORKSHOP*.

Schickel-Zuber, V., and Faltings, B. 2007. Oss: A semantic similarity function based on hierarchical ontologies. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, 551–556. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Schickel-Zuber, V. 2007. *Ontology Filtering*. Phd. thesis no 3934, Swiss Federal Institute of Technology (EPFL), Lausanne.

Twardowski, B. 2016. Modelling contextual information in session-aware recommender systems with neural networks. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, 273–276. New York, NY, USA: ACM.

Willems, F. M. J.; Shtarkov, Y. M.; and Tjalkens, T. J. 1995. The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory* 41(3):653–664.

Zhou, Y.; Wilkinson, D.; Schreiber, R.; and Pan, R. 2008. Large-scale parallel collaborative filtering for the netflix prize. In Fleischer, R., and Xu, J., eds., *Algorithmic Aspects in Information and Management*, 337–348. Berlin, Heidelberg: Springer Berlin Heidelberg.