Hyper-parameter optimization for latent spaces

Veloso, B.; Carprese, L.; König, H.M.T.; Teixeira S.; Manco, G.; Hoos, H.H.; ... ; Lozano, J.A.

Hyper-parameter Optimization for Latent Spaces

Bruno Veloso[1,6](✉), Luciano Caroprese[5], Matthias König[2], Sónia Teixeira[3,6], Giuseppe Manco[5], Holger H. Hoos[2,4], and João Gama[3,6]

[1] Portucalense University, Porto, Portugal
[2] Leiden University, Leiden, The Netherlands
[3] University of Porto, Porto, Portugal
[4] University of British Columbia, Vancouver, Canada
[5] ICAR-CNR, Arcavacata, Italy
[6] LIAAD-INESC TEC, Porto, Portugal
bruno.m.veloso@inesctec.pt

Abstract. We present an online optimization method for time-evolving data streams that can automatically adapt the hyper-parameters of an embedding model. More specifically, we employ the Nelder-Mead algorithm, which uses a set of heuristics to produce and exploit several potentially good configurations, from which the best one is selected and deployed. This step is repeated whenever the distribution of the data is changing. We evaluate our approach on streams of real-world as well as synthetic data, where the latter is generated in such way that its characteristics change over time (concept drift). Overall, we achieve good performance in terms of accuracy compared to state-of-the-art AutoML techniques.

Keywords: AutoML · Hyper-parameter optimization · Latent spaces · Nelder-Mead algorithm · SMAC · Recommender systems

1 Introduction

In many application scenarios, machine learning systems have to deal with large amounts of continuous data, so-called data streams, whose properties or distribution can change and evolve. This is especially relevant for recommender systems, which usually have to deal with evolving data.

Recommender systems help users to select items among a set of choices, according to their preferences. On e-commerce platforms such as Netflix or Amazon, where the number of items to select is very large, personalized recommendation is almost mandatory.

How these recommendations are made follows one of three types of approaches: i) content-based: items are recommended based on the similarity between their features; ii) collaborative filtering: items are recommended to a user based on known preferences of similar users for those items; iii) hybrid: a

N. Oliver et al. (Eds.): ECML PKDD 2021, LNAI 12977, pp. 249–264, 2021.
https://doi.org/10.1007/978-3-030-86523-8_16

combination of the content-based and collaborative filtering. In this work, we use a collaborative filtering approach. Collaborative filtering is based on a very large sparse users × items matrix. A typical approach to deal with this huge matrix is *matrix factorization*, which constructs a user embedding matrix and an item embedding matrix, such that users are embedded into the same latent space as items [21]. Through the dot product of the matrices of these embeddings, it is then possible to obtain, by approximation, the ratings (preferences).

The hyper-parameter optimization (HPO) problem seeks to choose optimal settings for the hyper-parameters of a given machine learning system or algorithm, such that the best possible performance is obtained on previously unseen data. The literature suggests approaches such as i) grid search [17], ii) gradient search [24], iii) random search [4] and iv) Bayesian optimization algorithms [13] for hyper-parameter optimization.

In this work, we describe a method for online optimization that can automatically adapt the hyper-parameters of an embedding model to changes in the data. We employ the Nelder-Mead algorithm, which uses a set of operators to produce a potentially good configuration to be deployed. This step is repeated whenever the distribution of the data changes. To evaluate our approach, we developed a synthetic data generator capable of producing realistic data streams with characteristics that change over time (concept drift).

Specifically, we make the following contributions:

- online optimization of the hyper-parameters for matrix factorization on large data sets;
- drift detection, that is, our proposed method reacts to changes in the process producing a given data stream.

The remainder of this paper is structured into five sections. Section 2 provides a systematic literature review on latent space models, automated machine learning in the context of online hyper-parameter optimization, and streaming approaches. Section 3 formulates the problem and describes our proposed solution. Section 4 details our experiments and discusses the results we obtained. Finally, Sect. 5 draws some general conclusions and outlines directions for future work.

2 Background and Related Work

In this section, we cover work related to latent variable spaces in recommender systems, online AutoML techniques, and streaming approaches.

Latent Space Models. In this work, we focus on latent variable models, which are historically proven to work effectively in modeling user preferences and providing reliable recommendations (see, *e.g.*, [3] for a survey). Essentially, these approaches embed users and items into latent spaces that translate relatedness to geometrical proximity. Latent embeddings can be used to decompose the large sparse preference matrix [1,35], to devise item similarity [20,27], or more generally, to parameterize probability distributions for item preference [16,29,40] and to sharpen the prediction quality employing meaningful priors.

The recent literature has shifted to more complex models based on deep learning [41], which in principle could show substantial advantages over traditional approaches. For example, neural collaborative filtering (NCF) [15] generalizes matrix factorization to a non-linear setting, where users, items, and preferences are modeled through a simple multilayer perceptron network that exploits latent factor transformations. Despite this progress, collaborative filtering based on simple latent variable modeling still represents a reference framework in the are of recommender systems. Notably, recent studies [30,31] showed that carefully tuned basic matrix factorization models can outperform more complex models based on sophisticated deep learning architectures. This capability, combined with the intrinsic simplicity of the model and the underlying learning process, is the main reason why we focus on them in our work presented here.

Online AutoML. The field of AutoML is generally concerned with automatically constructing machine learning pipelines that efficiently map raw input data to desired outputs [19], such as class labels, and can therefore be seen as an extension of plain *hyper-parameter optimization (HPO)* [9]. The task of automatically constructing a machine learning pipeline is formally modeled by the *combined algorithm selection and hyper-parameter optimization (CASH)* problem, which formalizes the search for the most effective algorithm and its associated hyper-parameter configuration as a joint optimization task [22].

In principle, there are various ways to tackle both the HPO and CASH problems. One of the most prominent approaches is through *sequential model-based algorithm configuration (SMAC)* [18], which is a widely known, freely available, state-of-the-art general-purpose configuration procedure based on sequential model-based (or Bayesian) optimization. The main idea of SMAC is to construct and iteratively update a probabilistic model of target algorithm performance to guide the search for good configurations. In the case of SMAC, this so-called *surrogate model* is implemented using a random forest regressor [5].

AutoML methods have been shown to be able to efficiently assemble and configure full machine learning pipelines (see, *e.g.*, [10]), including automated data pre-processing, feature selection, and hyper-parameter optimization. However, the performance of these systems is usually evaluated in static environments, *i.e.*, on data that does not change over time.

While there exists a vast body of research on processing streams and data in the presence of drift, there has been relatively little work on *online* AutoML, *i.e.*, AutoML methods that can automatically adapt machine learning algorithms to dynamic changes in the data on which they are deployed. Recently, some first attempts have been made to extend AutoML methods to dynamic data streams [8,25]. The main idea behind online AutoML is to not only automatically build a machine learning pipeline, but also to adjust or replace it when its performance degrades due to changes in the data. There are different adaption strategies suggested in the literature, which can be broadly divided into *model replacement* and *model management* strategies [25], where the former globally replace the model with a new one and the latter updates ensemble weights of the initially learned model based on new data.

As our work presented here focuses on a specific algorithm, *i.e.*, matrix factorization, we do not configure a machine learning ensemble, but instead, seek to dynamically optimize the hyper-parameters of the embedding model.

Streaming Approaches. With the development of modern computer architectures and with the substantial increase in the acquisition of sensor data, it becomes evident that offline model training and selection will become largely obsolete in the near future. *React* is one of the first approaches to model selection that explored the emergence of new multi-core architectures to compute several models in parallel [11,12]. This model selection technique implements a tournament-type process to decide which are the most effective hyper-parameter configurations. The major drawback of this technique is the computational cost associated with running multiple model configurations in parallel.

In the literature, we can also find several works proposing incremental model selection for classification tasks. The IL-MS algorithm [23] uses a k-fold cross-validation procedure to compute additional support vector machine (SVM) models with different configurations to select the best model. This procedure is run periodically, to minimize the computational cost; however, the evaluation procedure performs a double pass over the data (offline learning), which increases computational cost. A different approach uses meta-learning and weighting strategies to rank a set of heterogeneous ensembles [32,33]. However, this strategy requires the extraction of computationally expensive meta-features, which can pose challenges for stream-based scenarios. More recently, the same authors [34] proposed a way for measuring the score of ensemble members on a recent time window and combining their votes.

The *confStream* algorithm [6,7] was developed to automatically configure a stream-based clustering algorithm using ensembles. Each ensemble has different configurations, which are periodically evaluated and changed, based on the performance on the last window. These performance observations are used to train a regression model, which suggests a set of unknown good configurations for the new ensemble models.

More recently, the problem of hyper-parameter tuning on data streams was formulated as an optimization problem [37,38], using the well-known Nelder-Mead algorithm [26] for optimizing a given loss function. This particular contribution showed to be highly versatile and was applied to three different machine learning tasks: classification [2,36], regression [37], and recommendation [39]. In the particular case of recommendation, the authors adopted an incremental matrix factorization model with a static hyper-parameter configuration, *i.e.*, the hyper-parameters are configured at the beginning of the stream and are subsequently kept unchanged. In the following, we will use embedding models in combination with a concept drift detector, based on the Page-Hinkley test [28], to restart the optimization process.

3 Hyper-parameter Optimization for Latent Spaces in Recommender Systems

We start by introducing notation to be used throughout the remainder of this paper. In the following, $u \in U = \{1, \ldots, N\}$ indexes a user and $i \in I = \{1, \ldots, M\}$ indexes an item for which a user can express a preference. Let $R_{u,i} \in \{r_{min}, \ldots, r_{max}\}$ denote the preference (rating) of user u for item i. The range $\{r_{min}, \ldots, r_{max}\}$ represents a preference rank: when $R_{u,i} = r_{min}$, user u maximally dislikes item i, while $R_{u,i} = r_{max}$ denotes maximal preference for the item. Typical ranges are $\{0, 1\}$ (implicit preference) or $\{1, ..., 5\}$.

The set of all preferences can be represented as a rating matrix R, or alternatively, as a set of triplets $R = \{(u_1, i_1, R_1), \ldots, (u_n, i_n, R_n)\}$ where $R_j = R_{u_j, i_j}$ When N and M are large, R only represents a partial and extremely small view of all possible ratings. With an abuse of notation, we shall denote by $(u, i) \in R$ the fact that there exists a triplet $(u, i, R_{u,i})$ in R. The underlying learning problem is hence to provide a reliable estimate (completion) $\hat{R}_{u,i}$ for each possible pair (u, i), given the current partial view R not containing the pair.

The standard matrix factorization framework for predicting such values assumes the following. Each user u and item i admit a representation in a K-dimensional space. We denote such representations by embedding vectors $\mathbf{p}_u, \mathbf{q}_i \in \mathbb{R}^K$, which represent the row of the embedding matrices $\mathbf{P} \in \mathbb{R}^{N \times K}$ and $\mathbf{Q} \in \mathbb{R}^{M \times K}$. Given user u and item i, the corresponding preference can be modeled as a random variable with a fixed distribution whose parameters rely on the embeddings $\mathbf{p}_u$ and $\mathbf{q}_i$. In the following, we assume that $R_{u,i} \sim \mathcal{N}(\mu, \sigma)$, where σ is fixed and $\mu = \mathbf{p}_u \cdot \mathbf{q}_i$ [35]. Other modeling choices are possible and do not substantially change the overall framework (see, *e.g.*, [29]). Finally, the learning objective can be specified as finding optimal embedding matrices $\mathbf{P}^*$ and $\mathbf{Q}^*$ that maximize the likelihood of the partial observations R, or that minimize the MSE loss:

$$\ell(\mathbf{P}, \mathbf{Q}; R) = \frac{1}{|R|} \cdot \sum_{(u,i) \in R} (R_{u,i} - \mathbf{p}_u \cdot \mathbf{q}_i)^2$$

This optimization problem can be easily solved via stochastic gradient descent, using a given learning rate η. In general, for a given R, the optimal embedding depends on both the embedding size K and the learning rate η. A proper exploration of the search space induced by these parameters enables the discovery of the most appropriate model for R.

An additional assumption that we make in this work is that the partial view R can be continuously updated. That is, either new unknown entries can be disclosed (for example, some users can express preferences for previously unseen items) or former preferences change (for example, due to change of tastes by the user, or due to a more accurate evaluation). Thus, we assume that the history of preferences produces a continuous stream of snapshots $R^{(1)}, R^{(2)}, \ldots, R^{(t)}, \ldots$, where $R^{(t)}$ represents the view of R at time t. As already mentioned, snapshots can overlap, *i.e.*, it can happen that both $R^{(n)}_{u,i}$ and $R^{(m)}_{u,i}$ exist for $m \neq n$, and $R^{(n)}_{u,i} \neq R^{(m)}_{u,i}$. We then define $\bigsqcup_{n \leq t} R^{(n)}$ as a merge among

$R^{(1)}, R^{(2)}, \ldots, R^{(t)}$ that preserves recency. That is, by denoting $\bigsqcup_{n \leq t} R^{(n)}$ as $V^{(t)}$, we define $V_{u,i}^{(t)} = R_{u,i}^{(t^*)}$, where $t^* = \max\{t \mid (u,i) \in R^{(t)}\}$. If no such t^* exists, then $V_{u,i}^{(t)}$ is undefined. The problem hence becomes: Given the current history $R^{(1)}, R^{(2)}, \ldots, R^{(t)}$, can we predict the missing entries of $\bigsqcup_{n \leq t} R^{(n)}$ and provide a reliable estimate $\hat{R}$ of the preferences at time t? More specifically, what is the embedding size K and learning rate η that produce optimal embeddings $\mathbf{P}, \mathbf{Q}$ minimizing the loss $\ell\left(\mathbf{P}, \mathbf{Q}; \bigsqcup_{n \leq t} R^{(n)}\right)$?

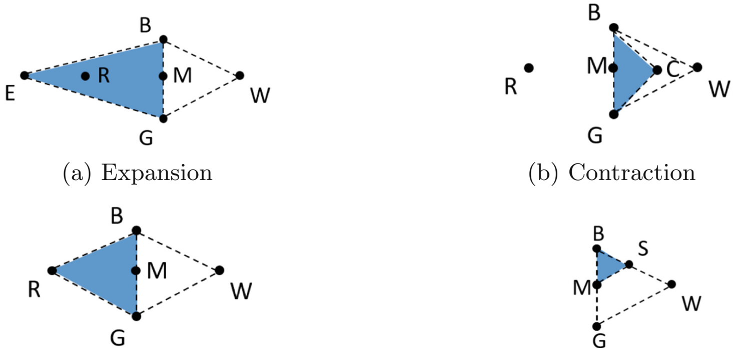
3.1 The Nelder-Mead Approach

The problem of hyper-parameter tuning we consider can be stated as follows: given a machine learning algorithm Alg with a default hyper-parameter configuration $Conf$, represented by Alg_{Conf}, and a data stream S consisting of an infinite set of mini-batches MB, the goal is to find an optimal configuration Alg^*_{Conf}, where Alg^*_{Conf} yields better performance compared to past configurations. In our situation, the model is represented by the embedding matrices $\mathbf{P}$ and $\mathbf{Q}$, $Conf$ is represented by the embedding size K and the learning rate η, and Alg is essentially stochastic gradient descent applied to the embedding matrices with learning rate η.

Here, we adopt a stream-based version of the Nelder-Mead algorithm [37] to find the optimal configuration Alg^*_{Conf}. This method consists of two phases: i) exploration, where the algorithm tries different versions of Alg^*_{Conf} to minimize a loss function, using a set of operators based on heuristics; and ii) deployment of the best Alg^*_{Conf} over the following set of MB in the stream S.

The original Nelder-Mead algorithm requires $n+1$ configurations $Conf$ to optimize a set of n hyper-parameters. Since we try to optimize the learning rate (η) and embedding size (K), we need to maintain three configurations: B (representing the configuration with the best score), W (the configuration with the worst score) and G (with a score in between B and W). The stream-based version additionally computes in parallel the M, R, E, S, C auxiliary models for the application of the Nelder-Mead operators. The underlying configurations for these models are obtained by applying four different operations to the configurations B, G, W. These are contraction, shrinking, expansion and reflection. Figure 1 illustrates how these models are obtained from B, G, W. Essentially, each operation corresponds to modifying the values of η and K, by either enlarging or narrowing it, and then devising the optimal models corresponding to these modified hyper-parameters.

In the streaming scenario, it is crucial to optimize the model incrementally. Basic stochastic gradient descent (SGD) is well-suited to incremental adaptation. This can be achieved by starting from a previous model and performing additional updating steps, possibly while exploiting the new learning rate. However, adapting the embedding dimension K poses several challenges when we try to apply the contraction or expansion operators, because it changes the structure of the model. This means that the SGD algorithm cannot exploit a previous



(a) Expansion

(b) Contraction

(c) Reflection

(d) Shrinking

Fig. 1. Basic heuristic operations.

model, and consequently has to restart the entire optimization process. To cope with this issue, we adopt two heuristic adaptation strategies:

- If any operation contracts the embedding size, we compute the new embedding matrices from the original ones by dropping some columns. The dropped columns correspond to the latent dimensions exhibiting the lowest variance values. The intuition is that these dimension do not discriminate well among users/items, and hence their contribution is likely to be redundant.
- If, on the other side, the embedding size needs to be expanded, we can keep all previous dimensions and add new ones. To add new dimensions, we create a set of vectors orthogonal to the existing latent vectors. In principle, orthogonal vectors allow for more efficient exploration of the search space by inspecting directions not covered before – for example, compared to randomly generated vectors.

The overall procedure is illustrated in Algorithm 1. It starts with three randomly generated configurations B, G, W. Then, the main cycle (lines 2–23) iterates over all time windows (denoted by t), while performing two steps:

- In the exploration phase (lines 4–11 and 17–20), the search space is explored to adapt the model with configurations coherent with the changes in the distribution of the data. To detect if a concept drift occurs, we use the Page-Hinkley test (PHt) [28] to report a concept drift if the observed prequential loss at some instant is greater than a given user-defined threshold (acceptable magnitude of change). When the mean of new data points exceeds the predefined threshold, an alert is triggered, signaling that a concept drift has been detected.
- In the exploitation phase (lines 13–16), the current best configuration is maintained, and the underlying model is optimized with respect to the current batch. Exploitation only occurs when the differences between the configurations B, G, W are negligible and, consequently, the simplex tends to converge to a single point.

Algorithm 1: Nelder-Mead Algorithm

```
   Input: A stream $R^{(1)}, R^{(2)}, \ldots, R^{(t)}, \ldots$, where each $R^{(t)}$ is a partial matrix of ratings
1  Initialize the process by randomly creating configurations $B, W, G$
2  for $t \geq 1$ do
       /* Working on chunk $V^{(t)}$ */
3      $S^{(t)} = \texttt{Sample}\left(\bigsqcup_{n \leq t-1} R^{(n)}\right)$
4      if drift is detected then
           /* Exploration is enabled */
5          Reconfigure $G, W$ by random perturbations from $B$
6          $simplex = \{B, G, W\}$
7          Train($simplex$)
8          $simplex$ = UpdateSimplex($simplex$)
9          $aux$ = UpdateAux($simplex$)
10         Train($aux$)
11     end
12     else
13         if Convergence is detected on simplex then
               /* Exploitation is enabled */
14             $simplex = \{B\}$
15             $aux = \emptyset$
16         end
17         else
18             $simplex$ = UpdateSimplex($simplex \cup aux$)
19             $aux$ = UpdateAux($simplex$)
20         end
21         Train($simplex \cup aux$)
22     end
23 end
24 Function Train(models):
25     for each model $m \in models$ do
           /* Update the current model using Gradient Descent */
26         $\mathbf{P}_m^{(t)}, \mathbf{Q}_m^{(t)}, l_m^{(t)} = \texttt{Optimize}(R^{(t)} \sqcup S^{(t)}; \mathbf{P}_m^{(t-1)}, \mathbf{Q}_m^{(t-1)}, \eta_m)$
27     end
28 Shoul Function UpdateSimplex(models):
       /* Select best, good and worst model */
29     Re-identify $B, G, W$ from models based on the current associated losses
30     return $\{B, G, W\}$
31 Function UpdateAux(models):
       /* Contraction, shrinking, expansion, reflection */
32     Generate auxiliary configurations $M, E, R, S, C$ from models
33     return $\{M, E, R, S, C\}$
```

In both situations, the training of the model (line 26) starts from the optimal embeddings for the same configuration, as computed in the preceding time window and adapted according to the previously described contraction and expansion processes. Furthermore, training is performed on the same set of data points as used in the evaluation step (line 3).

4 Empirical Evaluation

We conducted an extensive empirical evaluation of the proposed approach, with the goal of answering the following research questions:

RQ1: Does the exploitation phase of the Nelder-Mead algorithm converge to high-quality solutions?

RQ2: When drift is present in the data, how well does the Nelder-Mead approach adapt to the underlying changes?

RQ3: How does the Nelder-Mead algorithm compare to specific adaptations of well-known baseline approaches from the literature for online automatic model learning?

To foster reproducibility, we have publicly released all the data and code required to reproduce our experiments.[1]

4.1 Baselines and Evaluation Protocol

Our evaluation was performed on synthetic and real-world datasets. For the evaluation protocol, we considered the temporal data ordering and partitioned each dataset into intervals of the same size. The proposed method was evaluated with the predictive sequential (prequential) evaluation protocol for data streams [14], and we used root-mean-square error (RMSE) as our evaluation metric.

To address the research questions stated above, we considered two baselines. The first of these was aimed at verifying that our algorithm can make correct predictions whenever the entire stream is considered to be resulting from a stationary process. To this end, we considered a statically tuned matrix factorization model. As discussed in [30], a careful setup of the gradient-based algorithm for this basic model can outperform several more sophisticated approaches, including neural collaborative filtering approaches [31]. Hence, it is natural to ask whether the Nelder-Mead approach proposed in Sect. 3 is sufficiently robust to guarantee similar or better performance. To investigate this, we performed an exhaustive hyper-parameter search on the basic matrix factorization model using SMAC over the entire dataset. Hereby, we aimed to find a combination of globally optimal hyper-parameter settings, *i.e.*, settings that perform well over the entire data stream. After completion of this configuration process, these hyper-parameter settings were used to initialize a static model and to start an online training process, where each chunk was processed sequentially using stochastic gradient descent.

As a second baseline, we chose SMAC, a state-of-the-art, general-purpose algorithm configuration procedure that has been widely used for hyper-parameter optimization (see also Sect. 2). The general workflow of SMAC starts with picking a configuration Alg_{Conf} and an instance π (a sample of the data). Next, the configurator performs a run of algorithm Alg with configuration $Conf$ on instance π and measures the resulting performance. The information gleaned from such individual target algorithm runs is then used to iteratively build and update a predictive model that provides the basis for identifying promising configurations for subsequent runs, and to thus find configurations that perform well on the given training instances. Once its configuration budget (*e.g.*, the number

[1] https://github.com/BrunoMVeloso/ECMLPKDD2021.

of evaluation runs) is exhausted, SMAC returns its current incumbent $Conf^*$, *i.e.*, the best configuration found so far.

While the Nelder-Mead approach is intrinsically stream-based, the SMAC procedure is static by design. That is, SMAC uses a fixed subset of the data to find an optimal configuration, which is then deployed on the remaining data set. Here, however, we are dealing with a data stream, in which data not only arrives gradually over time, but is also expected to change. Therefore, we adapted the original SMAC procedure to employ a model replacement strategy [25], in which the optimization procedure is re-run from scratch when drift is detected to find better hyper-parameter settings $Conf^*$ and, subsequently, deploy a new instance of Alg with $Conf^*$.

The parameters of the model replacement strategy are chosen as follows. First, SMAC finds a configuration $Conf^*$ using the first chunk of the data stream; *i.e.*, the first n mini-batches, where n is the size of a chunk. It should be noted that the value of n has an impact on the effectiveness of the configuration procedure; more precisely, a larger value for n allows for better generalization, but at the same time increases computational cost, as each configuration has to be evaluated on a larger amount of data.

Once drift has been detected, we re-start the configuration procedure using the data chunk in which the drift occurred as the training set. Again, one could increase n or even re-run the configurator using all stored data up to the current batch. However, given an infinite stream of mini-batches, the computational cost incurred by the re-configuration process would grow indefinitely, making this approach infeasible for the scenario studied in this work.

4.2 Experiments on Real-World Data

In our first set of experiments, we evaluated our approach using Movielens1M[2], a standard benchmark dataset for collaborative filtering. Movielens is a time-series dataset containing user-item rating pairs along with the corresponding timestamps. The dataset comprises $6K$ users, $4K$ items, and 1 million ratings.

Figure 2 compares the results for this dataset obtained by the automatically tuned matrix factorization model using Nelder-Mead and SMAC, as well as the static matrix factorization model (trained with $K = 35$ and $\eta = 0.0337$).

The graph shows the average prequential loss, computed over a sliding window of fixed size. We can see that the proposed approach consistently outperforms the baselines after an initial burn-in period at the very beginning of the stream. Interestingly, the predictive ability of Nelder-Mead surpasses the static matrix factorization. This answers our first research question: the online version of Nelder-Mead does converge to a high-quality solution.

[2] https://grouplens.org/datasets/movielens/.

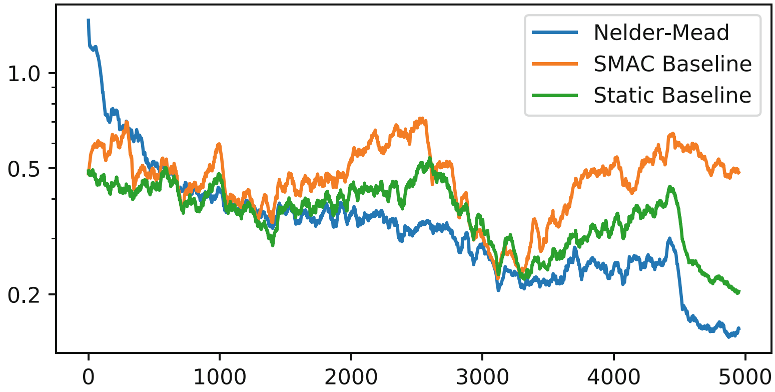


Fig. 2. Moving-average error rate for the MovieLens dataset.

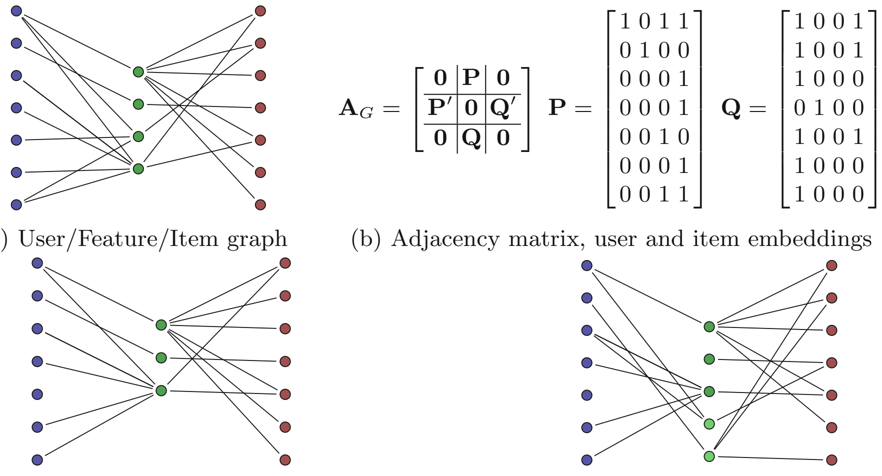
4.3 Experiments on Synthetic Data

In a second set of experiments, we carried out a careful comparison of the proposed approaches by evaluating their performance in a more controlled way. We tested our framework on datasets produced by a synthetic data generator, designed to be able to create data streams with controllable characteristics. The working principle of the generator is that ratings can be generated as the result of a stochastic process influenced by preference changes, and hence governed by evolving embedding matrices. The evolution essentially consists of the addition or removal of features.

The data generation process starts with a fixed number of users N and items M, and relies on the initial non-negative embedding matrices $\mathbf{P} \in \mathbb{R}^{N \times K_0}$ and $\mathbf{Q} \in \mathbb{R}^{M \times K_0}$, where K_0 is the initial feature size. The latent features in the data generation process resemble latent semantic topics [16]: Given a user u, the embedding $\mathbf{p}_u$ encodes the leanings of u for the given topic; in particular, $p_{u,k}$ is non-zero if there is a leaning of u for topic (feature) k. Analogously, $q_{i,k}$ represents the relatedness of item i to topic k. The generation process for $\mathbf{P}$ and $\mathbf{Q}$ proceeds in two steps. First, we build a tripartite graph $G = (V, E)$, where $V = (U, I, T)$, with U representing the set of all users, I the set of all items, and $T = \{1, \ldots, K_0\}$ the set of all features. Edges only connect topics to users or items and are generated through preferential attachment, by considering each topic in order. For a given topic k, we first sample the number of user (resp. item) neighbors n_k from a Zipf distribution with parameter α. Then, for each of these neighbors, we select a user (resp. an item) x and add the edge (x, k) to E. The element x is stochastically selected from U (resp. from I):

- with probability p, we randomly sample x with uniform probability;
- otherwise (with probability $1-p$), we sample x with probability proportional to its current degree.

The embeddings $\mathbf{P}$ and $\mathbf{Q}$ are extracted from the adjacency matrix $\mathbf{A}_G$ of G. By construction, $\mathbf{A}_G$ exhibits two main non-zero blocks, representing the edges from U to T and from I to T, respectively. $\mathbf{P}$ corresponds to the block connecting users to topics, whereas $\mathbf{Q}$ corresponds to the block connecting items to topics. Figure 3 illustrates the process. In Fig. 3a, we see the tripartite graph connecting 7 users (blue) and items (red) to 4 features (green). The corresponding adjacency matrix is shown in Fig. 3b. The two main blocks of the matrix represent both the user and item embeddings. Notice that both embeddings are represented by binary matrices. Finally, a rating for a pair (u, i) is generated by sampling from a Gaussian distribution with fixed variance and mean $\mu = (r_{max} - r_{min}) \cdot \mathbf{p}_u \mathbf{q}_i / \|\mathbf{q}_i\| + r_{min}$. The normalization of $\mathbf{q}_i$ guarantees that the maximal preference is only achieved when user and item completely overlap over the features.



$$\mathbf{A}_G = \left[\begin{array}{c|c|c} \mathbf{0} & \mathbf{P} & \mathbf{0} \\ \hline \mathbf{P}' & \mathbf{0} & \mathbf{Q}' \\ \hline \mathbf{0} & \mathbf{Q} & \mathbf{0} \end{array}\right] \quad \mathbf{P} = \begin{bmatrix} 1\,0\,1\,1 \\ 0\,1\,0\,0 \\ 0\,0\,0\,1 \\ 0\,0\,0\,1 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \\ 0\,0\,1\,1 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} 1\,0\,0\,1 \\ 1\,0\,0\,1 \\ 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 1\,0\,0\,1 \\ 1\,0\,0\,0 \\ 1\,0\,0\,0 \end{bmatrix}$$

(a) User/Feature/Item graph

(b) Adjacency matrix, user and item embeddings

(c) Feature dropping

(d) Feature expansion

Fig. 3. An example illustrating the data generation process. (Color figure online)

Within the construction scheme outlined above, concept drift can be easily modeled, by evolving the underlying tripartite graph. New ratings can be generated accordingly from the updated embedding matrices. We consider two main drifting operations here: (1) elimination of a feature and (2) expansion with an additional feature. Elimination is simple and essentially consists of removing a feature node. The corresponding connections are rewired to an existing feature, according to a preferential attachment criterion (governed by parameters p_d and p_r): Given an edge (x, k) connected to an eliminated feature k, the edge is removed with probability p_d. If not removed, it is rewired to a random feature (either chosen uniformly with probability p_r, or with probability proportional to the feature degree with probability $1 - p_r$). Figure 3c shows an example where feature 3 from Fig. 3a is eliminated.

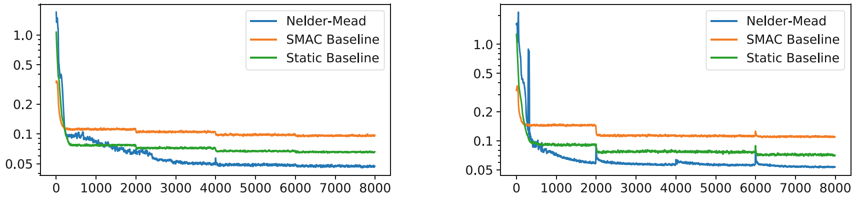


Fig. 4. Moving-average error rate for synthetic data (left: first experiment; right: second experiment; for details, see text).

The expansion follows a similar scheme: for each newly added feature, we fix the number of neighbors to connect, and select each such neighbor either randomly or by preferential attachment. The connection produces a new edge with probability p_a, and otherwise rewires one of the existing neighbor connections (with probability $1 - p_a$). Figure 3d illustrates how two new features are added and some edges are rewired.

We performed two experiments on a synthetic dataset with 10 000 users and 2 000 items. In both of these, we set $K_0 = 100$ and $\alpha = 1.1$. We generated a total of 8 000 chunks with 2 000 ratings each, with a drift every 2 000 chunks. In the first experiment, we set $p = 1$, and drift was controlled by $p_d = p_r = p_a = 1$. In the second experiment, we set $p = 0.9$, and drift was controlled by $p_d = p_r = p_a = 0.5$. In both cases, we experimented with expansion, by adding K_0 more features at each drift.

Figure 4 shows that our framework correctly detects the drift and restarts the hyper-parameter tuning process. More specifically, we observe that Nelder-Mead exhibits fast convergence and consistently adapts to all the injected occurrences of drift, which answers our second research question. Furthermore, we can notice a substantial difference between Nelder-Mead and SMAC baseline as well as the static matrix factorization model, resolving the third research question. Although the latter approach also adapts to changes, as a result of the SGD optimization, the automatic hyper-parameter optimization provided by the Nelder-Mead algorithm allows for a better adaptation to the changes triggered by the new features, and consequently yields a lower error rate. When using SMAC as an online configurator, we also find that model performance improves after each drift, indicating that the re-configuration procedure effectively accounts for the induced changes in the data. However, it does so less successfully than the Nelder-Mead approach and the static approach, which could be explained by the relatively small amount of data points SMAC is exposed to in the online setting. Possibly, the performance of this approach could be improved by increasing the chunk size n, thereby providing SMAC with a larger set of training data in the (re-)configuration procedure.

5 Conclusions

The objective of this research was to investigate and develop an online optimization approach for latent spaces in dynamic recommender systems. We have introduced an adaptation of the Nelder-Mead algorithm for data streams, which uses a simplex search mechanism, combined with a concept drift detection mechanism, to find hyper-parameter configurations that minimize the given loss function. Through experiments on real-world and artificial data sets, we have shown that the automatic selection of hyper-parameter settings has substantial impact on the outcomes of stream-based recommendations. In particular, we have demonstrated that i) our new approach achieves lower prediction error than a carefully tuned static matrix factorization model as well as the state-of-the-art configurator SMAC adapted to an online setting; ii) the concept drift detector is able to automatically trigger the search for a new optimal solution, which is an advantage when compared with static approaches.

The proposed approach can be adopted to other scenarios where an embedding model is used for predictive purposes. We plan to further explore the applicability of this optimization method in situations where the parameter space is more complex than the simple embedding size; these situations arise, for example, when using complex deep learning models requiring multiple components and specific tuning of the underlying components, such as convolutional architectures, recurrent layers or even attention models based on transformers.

Acknowledgments. The research reported in this work was partially supported by the EU H2020 ICT48 project “HumanE-AI-Net” under contract #952026. The support is gratefully acknowledged.

References

1. Agarwal, D., Chen, B.C.: LDA: matrix factorization through latent Dirichlet allocation. In: ACM International Conference on Web Search and Data Mining, pp. 91–100 (2010)
2. Bahri, M., et al.: AutoML for stream k-nearest neighbors classification. In: IEEE International Conference on Big Data, pp. 597–602 (2020)
3. Barbieri, N., Manco, G.: An analysis of probabilistic methods for top-N recommendation in collaborative filtering. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 172–187 (2011)
4. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
5. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
6. Carnein, M., et al.: confStream: automated algorithm selection and configuration of stream clustering algorithms. In: International Conference on Learning and Intelligent Optimization, pp. 80–95 (2020)
7. Carnein, M., et al.: Towards automated configuration of stream clustering algorithms. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 137–143 (2019)

8. Celik, B., Vanschoren, J.: Adaptation strategies for automated machine learning on evolving data. In: arXiv preprint arXiv:2006.06480 (2020)
9. Feurer, M., Hutter, F.: Hyperparameter optimization. In: Automated Machine Learning, pp. 3–33 (2019)
10. Feurer, M., et al.: Auto-sklearn: efficient and robust automated machine learning. In: Automated Machine Learning, pp. 113–134 (2019)
11. Fitzgerald, T., et al.: Online search algorithm configuration. In: AAAI Conference on Artificial Intelligence, vol. 28 (2014)
12. Fitzgerald, T., et al.: ReACT: real-time algorithm configuration through tournaments. In: Annual Symposium on Combinatorial Search (2014)
13. Galuzzi, B.G., Giordani, I., Candelieri, A., Perego, R., Archetti, F.: Hyperparameter optimization for recommender systems through Bayesian optimization. CMS **17**(4), 495–515 (2020). https://doi.org/10.1007/s10287-020-00376-3
14. Gama, J., Sebastião, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Mach. Learn. **90**(3), 317–346 (2013)
15. He, X., et al.: Neural collaborative filtering. In: International Conference on World Wide Web, pp. 173–182 (2017)
16. Hofmann, T.: Latent semantic models for collaborative filtering. ACM Trans. Inf. Syst. **22**(1), 89–115 (2004)
17. Hsu, C.-W., Chang, C.-C., Lin, C.-J.: A practical guide to support vector classification (2003)
18. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization, pp. 507–523 (2011)
19. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning: Methods, Systems, Challenges. Springer Nature (2019). https://doi.org/10.1007/978-3-030-05318-5
20. Kabbur, S., Ning, X., Karypis, G.: FISM: factored item similarity models for top-N recommender systems. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 659–667 (2013)
21. Karimi, R., et al.: Non-myopic active learning for recommender systems based on matrix factorization. In: IEEE International Conference on Information Reuse Integration, pp. 299–303 (2011)
22. Kotthoff, L., et al.: Auto-WEKA: automatic model selection and hyperparameter optimization in WEKA. In: Automated Machine Learning, pp. 81–95 (2019)
23. Lawal, I.A., Abdulkarim, S.A.: Adaptive SVM for data stream classification. South Afr. Comput. J. **29**(1), 27–42 (2017)
24. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. In: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 2113–2122 (2015)
25. Madrid, J.G., et al.: Towards AutoML in the presence of drift: first results. In: arXiv preprint arXiv:1907.10772 (2019)
26. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
27. Ning, X., Karypis, G.: SLIM: sparse linear methods for top-N recommender systems. In: IEEE International Conference on Data Mining, pp. 497–506 (2011)
28. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1/2), 100–115 (1954)
29. Rendle, S., et al.: BPR: Bayesian personalized ranking from implicit feedback. In: Conference on Uncertainty in Artificial Intelligence, pp. 452–461 (2009)
30. Rendle, S., Zhang, L., Koren, Y.: On the difficulty of evaluating baselines: a study on recommender systems. In: arXiv preprint arXiv:2006.06480 (2019)

31. Rendle, S., et al.: Neural collaborative filtering vs. matrix factorization revisited. In: RecSys 2020. Virtual Event, Brazil: Association for Computing Machinery (2020)
32. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Algorithm selection on data streams. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) DS 2014. LNCS (LNAI), vol. 8777, pp. 325–336. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11812-3_28
33. van Rijn, J.N., et al.: Having a blast: meta-learning and heterogeneous ensembles for data streams. In: IEEE International Conference on Data Mining. IEEE, pp. 1003–1008 (2015)
34. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: The online performance estimation framework: heterogeneous ensemble learning for data streams. Mach. Learn. **107**(1), 149–176 (2017). https://doi.org/10.1007/s10994-017-5686-9
35. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: Proceedings of the International Conference on Neural Information Processing Systems, pp. 1257–1264 (2008)
36. Veloso, B., Gama, J.: Self hyper-parameter tuning for stream classification algorithms. In: Gama, J., et al. (eds.) ITEM/IoT Streams -2020. CCIS, vol. 1325, pp. 3–13. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66770-2_1
37. Veloso, B., Gama, J., Malheiro, B.: Self hyper-parameter tuning for data streams. In: Soldatova, L., Vanschoren, J., Papadopoulos, G., Ceci, M. (eds.) DS 2018. LNCS (LNAI), vol. 11198, pp. 241–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01771-2_16
38. Veloso, B., et al.: Hyperparameter self-tuning for data streams. Inf. Fus. **76**, 75–86 (2021)
39. Veloso, B., Gama, J., Malheiro, B., Vinagre, J.: Self hyper-parameter tuning for stream recommendation algorithms. In: Monreale, A., et al. (eds.) ECML PKDD 2018. CCIS, vol. 967, pp. 91–102. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14880-5_8
40. Wang, C., Blei, D.: Collaborative topic modeling for recommending scientific articles. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 448–456 (2011)
41. Zhang, S., et al.: Deep learning based recommender system: a survey and new perspectives. In: ACM Comput. Surv. **52**(1), 1–38 (2019)