Computational Management Science
https://doi.org/10.1007/s10287-020-00376-3

ORIGINAL PAPER

Hyperparameter optimization for recommender systems through Bayesian optimization

B. G. Galuzzi[1] · I. Giordani[1] · A. Candelieri[1] · R. Perego[1] · F. Archetti[1,2]

Received: 17 February 2019 / Accepted: 2 September 2020

Abstract
Recommender systems represent one of the most successful applications of machine learning in B2C online services, to help the users in their choices in many web services. Recommender system aims to predict the user preferences from a huge amount of data, basically the past behaviour of the user, using an efficient prediction algorithm. One of the most used is the matrix-factorization algorithm. Like many machine learning algorithms, its effectiveness goes through the tuning of its hyper-parameters, and the associated optimization problem also called hyper-parameter optimization. This represents a noisy time-consuming black-box optimization problem. The related objective function maps any possible hyper-parameter configuration to a numeric score quantifying the algorithm performance. In this work, we show how Bayesian optimization can help the tuning of three hyper-parameters: the number of latent factors, the regularization parameter, and the learning rate. Numerical results are obtained on a benchmark problem and show that Bayesian optimization obtains a better result than the default setting of the hyper-parameters and the random search.

Keywords Bayesian optimization · Collaborative filtering · Hyperparameters optimization · Matrix factorization · Recommender system

1 Introduction

Recommender systems (RS) represent a critical component of B2C online services. They improve the customer experience exposing contents of which customer are still unaware and attempt to profile user preferences. More in details, an RS aims to recommend items (movies, songs, books, etc.) that fit the user's preferences, to help the user

✉ B. G. Galuzzi
bruno.galuzzi@unimib.it

1 Department of Computer Science, Systems and Communications, University of Milano-Bicocca, viale Sarca 336, 20125 Milan, Italy

2 Consorzio Milano-Ricerche, via Roberto Cozzi, 53, 20126 Milan, Italy

Published online: 22 September 2020

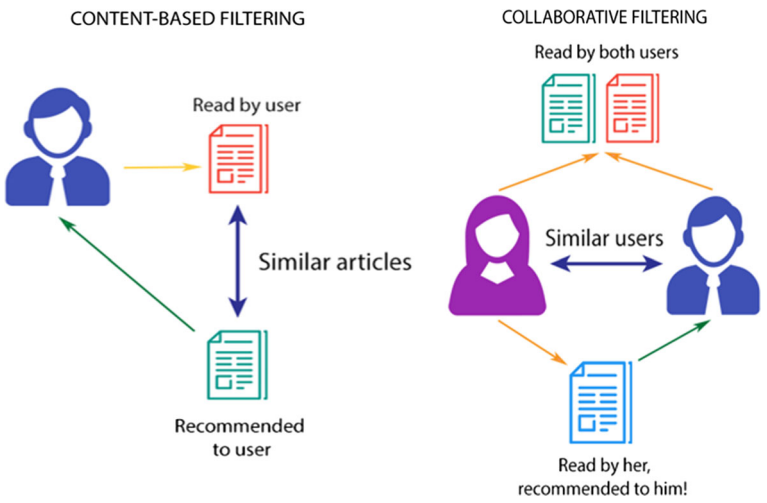

Fig. 1 Two different types of ML algorithm for RS: **a** Content-based approach, and **b** Collaborative filtering approach. Source: https://towardsdatascience.com/brief-on-recommender-systems-b86a1068a4dd

in selecting items from a large set of choices. Example of applications can be found in many fields, among which movies (Koren et al. 2009), music (Lee et al. 2010), books (Crespo et al. 2011), e-commerce (McNally et al. 2011) and active stock selection (De Rossi et al. 2019). The idea behind an RS is that providing personalized suggestions significantly increasing the likelihood of a customer making a purchase compared to un-personalized ones. Personalized recommendations have huge importance where the number of possible items is large such as in e-commerce related to art (books, movies, music), fashion, food, etc. Some of the major participants in e-commerce (Amazon), movie streaming (Netflix), and music streaming (Spotify) successfully apply recommender systems to deliver automatically generated personalized recommendations to their customers.

Machine learning (ML) algorithms in Recommender systems are typically classified into two categories (Aggarwal 2016) (see Fig. 1):

- *Content-based approaches* profile users and items by identifying their characteristic features, such as demographic data for user profiling, and product information/descriptions for item profiling;
- *Collaborative filtering approaches* (CF) identify relationships between users and items and make associations using the past user activities information to predict user preferences on new items.

The first approach is laborious because it is necessary to collect information about users/items, and it often tricky because the users must share their personal data for the creation of a database for profiling. The CF approach requires few data, basically a list of tuples containing the user ID, the item ID, and the rating done by the user to that item. Moreover, the CF algorithms are more flexible in that they can be applied to RS independently of the domain of application.

In this paper, the authors focus on CF, in which the basic data structure is the *rating matrix*, formed by any possible user-item combination. The problem is also called the *matrix completion problem,* because it is based on an incompletely specified matrix of values (the users' past rated items), and the aim is to predict the remaining values (the predicted user' rates on new items) using some learning algorithm. The main challenge in designing CF methods is that real-world databases are mostly sparse (many unknown entries), but the unknown ratings are predictable because the known ratings are often highly correlated across various users or items.

Two types of methods are commonly used in the CF framework (Cacheda et al. 2011): the *memory-based methods* and *model-based methods*. The memory-based methods, or *neighborhood-based CF algorithms*, were among the earliest collaborative filtering algorithms, in which the ratings of user-item combinations are predicted based on their neighborhoods (users similar to a target user or items similar to a target item). They are based on the fact that similar users display similar patterns of rating behavior (user-based) or similar items receive similar ratings (item-based). An example of neighbourhood method is the k-Nearest neighbours (Yeehuda 2010). These methods are simple to implement, and the resulting recommendations are often easy to explain. On the other hand, memory-based algorithms do not work very well with sparse rating matrices: they scale poorly with the number of dimensions, and their predictions are not accurate for user/item matrix with few ratings.

The model-based methods are an alternative approach that try to predict the ratings by characterizing both items and users using a certain number of parameters inferred from the rating patterns. More in details, they are based on the assumptions that the preferences of a user can be inferred from a small number of hidden or *latent factors*. The most successful realizations of latent factor models are based on matrix factorization (Salakhutdinov and Mnih 2008; Koren et al. 2009). These approaches are considered superior to classic nearest-neighbour techniques for producing recommendations and learn the latent factors within an optimization framework. This corresponds to a low-rank approximation of the rating matrix, with the assumption of correlations between rows (or columns) to guarantee the dimensionality reduction of the matrix itself.

The RS becomes a minimization problem, in which we want to determine two low-rank matrices whose product is as close as possible to the rating matrix. The associated error function depends on the number of latent factors. Moreover, a regularization term is added to reduce both the non-linearity effect and the overfitting problem. Finally, if we solve this problem by means of stochastic gradient descent (Bottou 2010), we must set also the learning rate related to the descent direction.

These three hyper-parameters (i.e., number of latent factors, regularization term and learning rate) must be tuned through an optimization procedure, usually called *hyper-parameter optimization*.

The objective function maps any possible hyper-parameter configuration to a numeric score quantifying the quality of the matrix factorization. This score is computed by dividing the known entries of the rating matrix in two sets: a training test and a test set. For each hyper-parameter configuration, the stochastic gradient method is applied on the training set and the score is obtained as validation on the test set. This validation schema represents a randomize time-consuming black-box optimization

problem. Therefore, an efficient global optimization strategy is mandatory to obtain the best possible configuration using a small number of function evaluations.

Recently, *Bayesian optimization* (BO) (Shahriari et al. 2015; Frazier 2018) is becoming one of the most widely adopted strategies for global optimization of multi-extremal, and expensive-to-evaluate objective functions related to, e.g., sensor networks (Garnett et al. 2010), drug design (Meldgaard et al. 2018), time-series forecasting (Candelieri et al. 2018a), inversion problems (Perdikaris and Karniadakis 2016; Galuzzi et al. 2018), and robotics (Olofsson et al. 2018).

An example of the application of BO for RS is, e.g., in Dewancker et al. (2016) and Cano (2019). In the first case, the authors use a Bayesian optimization web-service, called *SigOpt*, and show that BO outperforms random search in tuning three hyper-parameters of the *Alternating Least Squares algorithm* (Udell et al. 2016) to estimate the entries of the rating matrix. In the second case, the authors show the advantage in tuning the two hyper-parameters related to the KNN method to find the top five items recommended. Finally, alternative use of BO for RS can be found in Vanchinathan et al. (2014), where the challenge of ranking recommendation lists based on click feedback by efficiently encoding similarities among users and among items is considered. In this case, the Gaussian Process is used to model the elements of the rating matrix directly.

In our paper, we want to use BO for the hyper-parameter optimization related to the parameters of the stochastic gradient descent to find the best possible configuration, in terms of the learning rate, number of latent factors, and regularization parameter. We describe the mathematical formalization of the problem and we show an example of application of BO on a benchmark dataset, the MovieLens-100k, to find the best possible configuration of the hyper-parameters.

The rest of the paper is organized as follows. Section 2 introduces the problem definition and Sect. 3 describes the Bayesian optimization algorithm. A benchmark application is presented in Sect. 4, and Sect. 5 we present the conclusions.

2 The problem definition

In the most general framework, a CF problem is based on the definition of two sets (Takács et al. 2009):

- The set of users $U = \{u_1, u_2, \ldots, u_M\}$, where M is the number of users;
- The set of items $I = \{i_1, i_2, \ldots, i_N\}$, where N is the number of items.

Each user expresses its judgement, or *rating*, $r \in X$, where typical rating values can be binary or integers from a given range. The set of all the ratings given by the users on the items can be represented as a partially specified matrix $\mathbf{R} \in \mathbb{R}^{M \times N}$, where its entries r_{ui} express the possible ratings of user u for item i. Usually, each user rates only a small number of items, thus the matrix elements are known in a small number of positions $(u, i) \in \mathrm{S}$, with $|\mathrm{S}| \ll min\{M, N\}$(see Fig. 2).

	Item 1	Item 2	Item 3	...	Item n
User 1	2	3	?	...	5
User 2	?	4	3	...	?
User 3	3	2	?	...	3
...	...	...	...	...	...
User m	1	?	5	...	4

Fig. 2 Example of rating matrix

Usually, the dataset S is divided in a training set S_{Tr} and a test set S_{Te} with $S_{Tr} \cap S_{Te} = \varnothing$ and $S_{Tr} \cup S_{Te} = S$, and the aim of CF is to create a prediction of the elements of S_{Te} using only the knowledge of S_{Tr}, minimizing some error functions:

$$error = \sum_{(u,i) \in S_{Te}} \| r_{ui} - \hat{r}_{ui} \| \tag{1}$$

where $\widehat{r}_{ui} = \widehat{r}_{ui}(S_{Tr})$ denotes the prediction on r_{ui}, and is obtained as a function of the training set S_{Tr}. The typical used error function is the root mean square error (RMSE),

$$\text{RMSE} = \sqrt{\frac{1}{|Te|} \sum_{(u,i) \in S_{Te}} \left(r_{ui} - \hat{r}_{ui} \right)^2} \tag{2}$$

2.1 The matrix factorization algorithm

MF algorithms are the most successful ones for RS, and represent the state-of-the-art since they have shown their superiority in terms of predictive performance and runtime in numerous publications (Koren et al. 2009; Takács et al. 2009). The idea behind MF techniques is to approximate the matrix R as the product of two matrices (Fig. 3):

$$\boldsymbol{R} \approx \boldsymbol{P} \cdot \boldsymbol{Q} \tag{3}$$

where $\boldsymbol{P}$ is a $M \times K$ and $\boldsymbol{Q}$ is a $K \times N$ matrix. The first matrix is called the user-feature matrix, the second one is called the item-feature matrix, and K represent the number of latent factors in the given factorization. Typically, $K \ll min\{M, N\}$, and both **P** and **Q** contain real numbers, even when **R** contains only integers.

The aim of CF approach based on matrix factorization is to minimize the error function on the training set S_{Tr}, as a function of the matrixes $(\boldsymbol{P}, \boldsymbol{Q})$. The prediction $\widehat{r}_{ui}$ in Eq. (1) is expressed by:

$$\hat{r}_{ui} = \sum_{k=1}^{K} p_{uk} q_{ki} \tag{4}$$

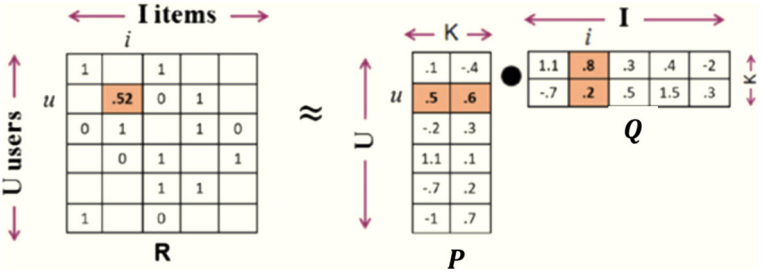

Fig. 3 Example of matrix factorization of the rating matrix in user features matrix and the item feature matrix

where p_{uk} and q_{ki} denote the elements of P and Q, respectively. Therefore, the optimization problem becomes

$$\left(\boldsymbol{P}^*, \boldsymbol{Q}^*\right) = \underset{(\boldsymbol{P}, \boldsymbol{Q})}{\arg\min} \left[\frac{1}{2} \sum_{(u,i) \in S_{Tr}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} \right)^2 \right] \tag{5}$$

After the minimization of Eq. (5) is done, one can estimate the RMSE on the test set S_{Te}:

$$\text{RMSE} = \sqrt{\frac{1}{|S_{Te}|} \sum_{(u,i) \in S_{Te}} \left(r_{ui} - \sum_{k=1}^{K} p^*_{uk} q^*_{ki} \right)^2} \tag{6}$$

The total number of variables of the optimization problem of Eq. (5) depends on the number of latent factors K. Generally, increasing the number of latent factors K improves the quality of the solution of Eq. (5) but can cause an overfitting problem. Basically, with too much freedom (too many free parameters), the model starts fitting well the training data but not generalizing well the unseen test data. A common way to avoid overfitting is to apply a regularization term to Eq. (5), obtaining a new but similar optimization problem

$$\left(\boldsymbol{P}^*, \boldsymbol{Q}^*\right) = \underset{(\boldsymbol{P}, \boldsymbol{Q})}{\arg\min} \left[\frac{1}{2} \sum_{(u,i) \in S_{Tr}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki} \right)^2 + \frac{1}{2} \lambda \sum_{k=1}^{K} \left(p_{uk}^2 + q_{ki}^2 \right) \right] \tag{7}$$

where $\lambda \geq 0$ is the regularization factor.

Besides, a simple split in the training set and test set does not guarantee robustness of the results. A more suitable procedure, widely adopted, is the cross-validation approach. More in detail, using the *N-fold validation*, the original dataset is divided in n subsets. Then, the model prediction is computed for n times, using the nth set as test set, and the other $n - 1$ folds as the training set.

The results of this procedure can be, for example, the mean of the n different values of the RMSE on the n folds. In Fig. 4 we report an example of pseudo-code for a numeric score function, using the n-fold validation.

Input (hyper-parameter configuration $\boldsymbol{\theta_n}$, number of folds N, dataset S)

score = 0

Divide the dataset S in N folds randomly

For n=1,...,N **do**

Assume the n-th fold as test set S_{Te} and the other folds as training set S_{Tr}

Solve the optimization problem (Eq. 4) using θ_n and S_{Tr}

Evaluate the RMSE (Eq. 6) on S_{Te}

score= score +RMSE

End for

score= score/N

Output (score)

Fig. 4 Pseudo-code for the numeric score function, using the k-fold validation

2.2 The optimization problem

The optimization problem associated with (Eq. 7) represents a non-convex and multi-extremal global optimization problem. Global optimization methods could be used to solve it such as evolutionary algorithms or multi-star method. However, finding the global minimum results very hard to solve since also the high number of variables. Then, it is usually content to find only a local minimum through a local gradient method. The state-of-the-art approaches are the Alternating Least Squares and the Stochastic Gradient Descent (SGD), that is the one we consider for this work.

First, one need to compute the partial derivative of the objective function with respect to the variable p_{uk} and q_{ki}:

$$\begin{aligned}\frac{\partial}{\partial p_{uk}} &= -\sum_{(u,i)\in S_{Tr}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki} \right) \cdot q_{ki} + \lambda \cdot p_{uk} \\ \frac{\partial}{\partial q_{ki}} &= -\sum_{(u,i)\in S_{Tr}} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki} \right) \cdot p_{uk} + \lambda \cdot q_{ki} \end{aligned} \tag{8}$$

At this point, one could update the entire vector of decision variables as $\bar{VAR}=\bar{VAR} - \eta \ \bar{\nabla J}$, where $\bar{\nabla J}$ indicates the entire vector of the partial derivatives and η is the step size or *learning rate*. However, when the data size is very large, it is preferable to use the SGD, in which the update is stochastically approximated in

Input (Training Set $S \subset R$, Learning Rate η,Reg. factor λ, n° of latent factors K)

Randomly initialize matrices P and Q ;

n=0

while not(convergence) **do**

Randomly shuffle observed entries in S;

for each $(u, i) \in S$ **do**

$$e_{u,i} = \left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki}\right)$$

for each $q \in \{1 \ldots k\}$ **do** $\hat{p}_{uk} = p_{uk} + \eta \cdot \left(e_{u,i} \cdot q_{ki} - \lambda \cdot p_{uk}\right)$

for each $q \in \{1 \ldots k\}$ **do** $\hat{q}_{ki} = q_{ki} + \eta \cdot \left(e_{u,i} \cdot p_{uk} - \lambda \cdot q_{ki}\right)$

for each $q \in \{1 \ldots k\}$ **do** $p_{uk} = \hat{p}_{uk}$ *and* $q_{ki} = \hat{q}_{ki}$

end for

n=n+1

end while

Output (P, Q)

Fig. 5 Pseudo-code for the stochastic gradient descent algorithm

terms of the error in a (randomly chosen from a uniform distribution) observed entry (i, j) as follows:

$$\begin{aligned} p_{uk} &= p_{uk} + \eta \cdot \left(\left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki}\right) \cdot q_{ki} - \lambda \cdot p_{uk}\right) \\ q_{ki} &= q_{ki} + \eta \cdot \left(\left(r_{ui} - \sum_{k=1}^{K} p_{uk} \cdot q_{ki}\right) \cdot p_{uk} - \lambda \cdot q_{ki}\right) \end{aligned} \tag{9}$$

By this way, one can cycle through the observed entries in R one at a time (in random order) and updates only the relevant set of $2 \cdot K$ entries in the factor matrices rather than all $(m \cdot K + n \cdot K)$. Indeed, for each observed rating $r_{i,j}$, the error $e_{ui} = \left(r_{ui} - \sum_{k=1}^{K} p_{uk} \bullet q_{ki}\right)$ is used to update the k entries in row i of U and the k entries in the row j of Q. A typical initialization for the elements $\mathbf{P}$ and $\mathbf{Q}$ is done randomly according, e.g., to a normal distribution with mean $\mu = 0$ and small variance σ. The pseudo-code for the stochastic gradient descent method is illustrated in Fig. 5.

2.3 Tuning the hyper-parameters

The optimization algorithm presented has three hyper-parameters to set: the learning rate η, the regularization factor λ, and the number of latent factors K. Different combinations of these hyper-parameters can alter the performance of the optimization

Input (hyper-parameter space Θ, Target score function $H(\boldsymbol{\theta})$, max n° of evaluation $\boldsymbol{n_{max}}$)

Select an initial hyper-parameter configuration $\boldsymbol{\theta_0} \in \Theta$

Evaluate the initial score $y_0 = H(\boldsymbol{\theta_0})$

Set $\boldsymbol{\theta^*} = \boldsymbol{\theta_0}$ *and* $\boldsymbol{y^*} = H(\boldsymbol{\theta_0})$

For n=2,...,n_{max} **do**

Select a new hyper-parameter configuration $\boldsymbol{\theta_n} \in \Theta$ *using some optimization strategy*

Evaluate H to obtain a new numeric score $y_n = H(\boldsymbol{\theta_n})$

If $y_n < F^*$

$\theta^* = \theta_n$ and $y^* = y_n$

End if

end for

Output: $\boldsymbol{\theta^*}$ and y^*

Fig. 6 Pseudo-code for hyper-parameter optimization

algorithm significantly. If we want to know which parameter combination yields the best results, we must perform a hyper-parameter optimization.

Formally, we define the hyperparameter optimization task as follows. Let A be the target algorithm with n number of parameters to be tuned. Each hyper-parameter θ_i can be a value taken from an interval $[a_i, b_i]$ (continuous or integer) in a hyper-parameter configuration space $\Theta = [a_1, b_1] \times \cdots \times [a_n, b_n]$. Defining a performance function $H : \Theta \rightarrow \mathbb{R}^+$ that maps each possible configuration $\boldsymbol{\theta} \in \Theta$ to a numeric score, the aim of the hyper-parameter optimization is to find the best configuration $\boldsymbol{\theta}^*$ that minimizes $H(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \Theta} H(\boldsymbol{\theta}) \tag{10}$$

The objective function H is characterized by the following properties:

1. The function is time-consuming in the sense that each evaluation takes a substantial amount of time (minutes for a small dataset, hours for a bigger one);
2. The function is "black-box", that means we do not know any properties about its structure like linearity or concavity, and we do not have information about derivatives;
3. The evaluation of the function is characterized by noise. This fact is because the evaluation of the performance function depends on a series of random factors related to the stochastic gradient descent algorithm (e.g., the starting point of the gradient-based optimization) and on how the dataset is divided in training and validation set.

All these facts make the optimization problem hard to solve and require a specific procedure to obtain the best configuration in a few evaluations. In Fig. 6 we show the pseudo-code of hyper-parameter optimization.

3 Bayesian optimization for hyperparameter optimization

BO is a sequential model-based approach to solve the problem (Eq. 10) and, in general, global optimization problems based on expensive-to-evaluate *black-box functions*. The search space Θ can be a compact subset of $\mathbb{R}^d$, but the BO framework can be applied also to search spaces involving categorical or conditional inputs. Moreover, recent studies extend BO also for more complex input space, such as combinatorial structures (Baptista and Poloczek 2018) and graph structured input space (Oh et al. 2019).

3.1 A framework for Bayesian optimization

Bayesian optimization framework has two key components (see Fig. 7). The first component is a probabilistic *surrogate model*, which consists of a prior distribution that models the unknown objective function. The second component is an *acquisition function* that is optimized for deciding where to sample next. The surrogate model provides a posterior probability distribution that describes potential values for $H(\boldsymbol{\theta})$ at a candidate configuration $\boldsymbol{\theta}$. The basic idea is that each time we observe H at a new point $\boldsymbol{\theta}$, we update this posterior distribution. The most used surrogate model is the Gaussian Process (GP), which is completely specified by a mean function $\mu(\boldsymbol{\theta}) : \Theta \rightarrow R$ and a definite positive covariance function, also called kernel, $k\left(\boldsymbol{\theta}, \boldsymbol{\theta}^{'}\right) : \Theta^2 \rightarrow R$,

$$H(\boldsymbol{\theta}) \sim GP\left(\mu(\boldsymbol{\theta}); k\left(\boldsymbol{\theta}, \boldsymbol{\theta}'\right)\right) \tag{11}$$

An important property of the kernel function is that the closer two points are in the input space, the larger is the value of the kernel function. Common choices are the Gaussian kernel or the Matern kernel. Regarding the mean function, the most common choice is a constant value, $\mu(\boldsymbol{\theta}) = \mu_0$.

The BO algorithm starts with an initial set of k configurations $\{\boldsymbol{\theta}_i\}_{i=1}^{k}$ and their associated function values$\{y_i\}_{i=1}^{k}$, with$y_i = H(\boldsymbol{\theta}_i)$. At each iteration $t \in \{k+1,\ldots,N\}$, we update the GP model using the Bayes rule, to obtain posterior distribution conditioned on the current training set $S_t = \{(\boldsymbol{\theta}, y_i)\}_{i=1}^{t}$ containing the past evaluated configurations and observations. For any, potentially non-evaluated, configuration $\boldsymbol{\theta} \in \Theta$, the posterior mean $\mu_t(\boldsymbol{\theta})$ and the posterior variance $\sigma_t^2(\boldsymbol{\theta})$ of the GP, conditioned on S_k, are known in closed-form:

$$\mu_t(\boldsymbol{\theta}) = \boldsymbol{k}(\boldsymbol{\theta})^T \left[K + \tau^2 I\right]^{-1} \boldsymbol{y}, \tag{12}$$

$$\sigma_t^2(\boldsymbol{\theta}) = k(\boldsymbol{\theta}, \boldsymbol{\theta}) - \boldsymbol{k}(\boldsymbol{\theta})^T [K + \tau^2 I]^{-1} \boldsymbol{k}(\boldsymbol{\theta}), \tag{13}$$

where K is the $t \times t$ matrix whose entries are $K_{i,j} = k\left(\boldsymbol{\theta_i}, \boldsymbol{\theta_j}\right)$, $\boldsymbol{k}(\boldsymbol{\theta})$ is the $t \times 1$ vector of covariance terms between $\boldsymbol{\theta}$ and $\{\boldsymbol{\theta}_i\}_{i=1}^{t}$, $\boldsymbol{y}$ is the $t \times 1$ vector whose i^{th} entry is y_i, and τ^2 is the noise variance.

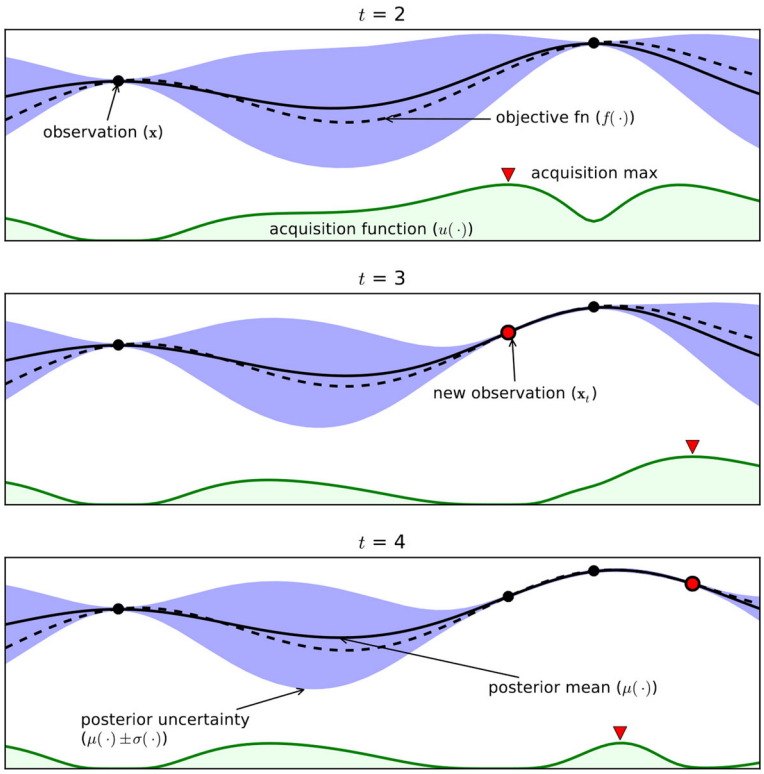

Fig. 7 Illustration of the Bayesian optimization procedure over three iterations, with the mean and confidence intervals estimated with the surrogate model (Gaussian Process) of the objective function. The acquisition functions in the lower shaded plots are showed. The figure is taken from Shahriari et al. (2015)

If a new point $\boldsymbol{\theta}_{t+1}$ is selected and evaluated to provide an observation $y_{t+1} = H(\boldsymbol{\theta}_{t+1})$, we add the new pair $\{(\boldsymbol{\theta}_{t+1}, y_{t+1})\}$ to the current training set S_t, obtaining a new training set for the next iteration $S_{t+1} = S_t \cup \{(\boldsymbol{\theta}_{t+1}, y_{t+1})\}$.

The next candidate point to evaluate is selected by solving an auxiliary optimization problem, typically of the form:

$$\boldsymbol{\theta}_{t+1} = \arg\max_{\theta \in \Theta} U_t(\boldsymbol{\theta}; S_t), \tag{14}$$

where U_t is the acquisition function to maximize. The rationale is that, because the optimization run-time or cost is dominated by the evaluation of the expensive function f, time and effort should be dedicated to choosing a useful and informative (in a sense defined by the auxiliary problem) point to evaluate. In Fig. 8 we show the pseudocode of the BO.

In BO, typical utility functions used to select the next candidate point include: the probability of improvement (Kushner 1964), the expected improvement (Mockus 1989), GP Confidence Bound (Auer 2003) (i.e., Lower Confidence Bound, LCB, in case of minimization and Upper Confidence Bound, UCB, in case of maximization),

Input (hyper-parameter space Θ, Target score function $H(\boldsymbol{\theta})$, max n° of evaluation $\boldsymbol{n_{max}}$)

Select an initial configuration $\boldsymbol{\theta_0} \in \Theta$

Evaluate the initial score $y_0 = H(\boldsymbol{\theta_0})$

Set $\boldsymbol{\theta}^* = \boldsymbol{\theta_0}$, $\boldsymbol{y}^* = H(\boldsymbol{\theta_0})$, *and* $S_0 = \{\boldsymbol{\theta_0}, y_0\}$

For n=1,...,n_{max} **do**

Select a new hyper-parameter configuration $\boldsymbol{\theta_n} \in \Theta$ *by optimizing an acquisition function* U_n

$$\boldsymbol{\theta}_{\mathrm{n}} = \arg\max_{\boldsymbol{\theta}\in\Theta} U_n(\boldsymbol{\theta}; S_t),$$

Evaluate H in $\boldsymbol{\theta_n}$ *to obtain a new numeric score* $y_n = H(\boldsymbol{\theta_n})$

Augment the data $S_n = S_{n-1} \cup \{\boldsymbol{\theta_n}, y_n\}$

Update the surrogate model

If $y_n < F^*$

$\theta^* = \theta_n$ and $y^* = y_n$

End if

end for

Output: $\boldsymbol{\theta}^*$ and y^*

Fig. 8 Pseudo-code of Bayesian optimization

or, more recently, the Knowledge Gradient (KG) (Frazier et al. 2008). Since the low cost to evaluate the acquisition function, greedy optimization strategies can be used to solve the problem (Eq. 14) such as random search, multi-star approach or genetic algorithms.

In this paper, we use the Expected Improvement (EI), that is based on the maximization of the following function:

$$U_t(\boldsymbol{\theta}; S_t) = \left(f^* - \mu_t(\boldsymbol{\theta})\right) \cdot \Phi\left(f^*; \mu_t(\boldsymbol{\theta}); \sigma_t(\boldsymbol{\theta})\right) + \sigma_t(\boldsymbol{\theta}) \cdot \mathcal{N}\left(f^*; \mu_t(\boldsymbol{\theta}); \sigma_t(\boldsymbol{\theta})\right) \tag{15}$$

where $\mathcal{N}$ and Φ are the normal distribution function and the normal cumulative distribution function, respectively. The EI has two components: the first can be increased by reducing the mean function $\mu_t(\boldsymbol{\theta})$, the second can be increased by increasing the variance $\sigma_t(\boldsymbol{\theta})$. These two terms balance the trade-off between exploitation (evaluating at points with low mean) and exploration (evaluating at points with high uncertainty).

3.2 Dealing with integer parameters

The previous framework considered for BO through GP assumes that all the variables for H are continuous. However, even if the regularization factor λ and the learning rate η are continuous, the number of latent factor K takes values in a closed subset of integers. Optimization problems involving continuous and integer/discrete variables is common in many tasks of optimizing for the hyper-parameters of machine learning systems (Snoek et al. 2012). Typical examples can be found, e.g., in an ensemble of decision trees generated by the gradient boosting algorithm, in which we must adjust

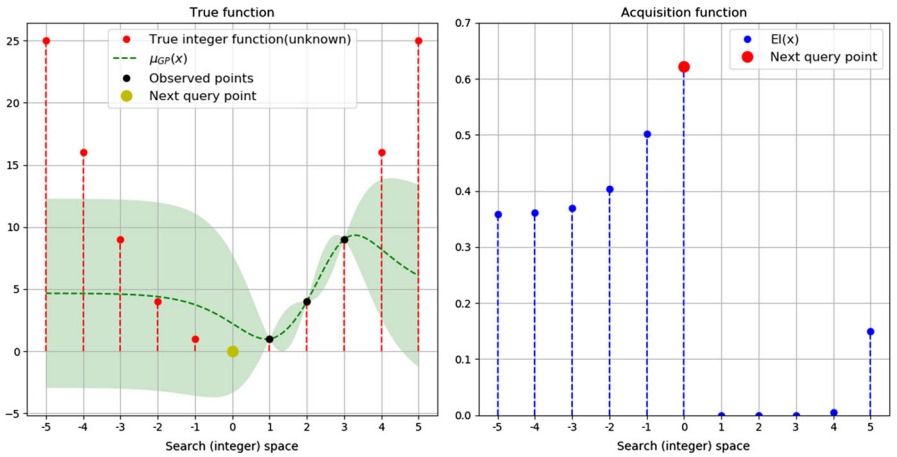

Fig. 9 The left figure shows the true integer function (red), and the approximation (mean and variance) to the original function by the GP model (green). The right figure shows the acquisition (integer) function (EI) values after the surrogate model is fitted. The next query point is coloured by red (color figure online)

the learning rate and the maximum depth of the trees, or in a deep neural network, in which we must adjust the learning rate, the number of layers and the number of neurons per layer, which can only take discrete values.

In this case, two possible approaches can be considered. In the first case, other surrogate models, different from GP, can be proposed, such as Random Forest (RF) (Tin Kam Ho 1995; Candelieri et al. 2018b), which should be, at least ideally, well suited for optimization problem with only integer variables or mixed space in which most of the variables are integer. In the second approach, that is the one we considered, the GP is used, but an approximation is done along the integer dimensions. More in details, the optimization of the acquisition function A can be done assuming all variables take continuous values. The values for the integer-valued variables are replaced by the closest integer (the naïve approach in "Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes" (Garrido-Merchán and Hernández-Lobato 2018)). Another possibility is to deal with integer-valued variables considering their discrete nature: for instance, random sampling-based optimization will sample in a finite set of integer values instead of a continuous range. This is the approach followed by the software Scikit-optimize (https://scikit-optimize.github.io/), and the one we considered. Figure 9 shows an example of optimization for a 1D integer function (red), and the approximation (mean and variance) to the original function by the GP model (green), whereas the second column shows the acquisition function (EI) values after the surrogate model is fitted. Note that only integer values are considered for the acquisition function.

4 Benchmark problem

The *MovieLens* datasets (Harper and Konstan 2015) were collected by the GroupLens Research Project at the University of Minnesota. These datasets are used as benchmark

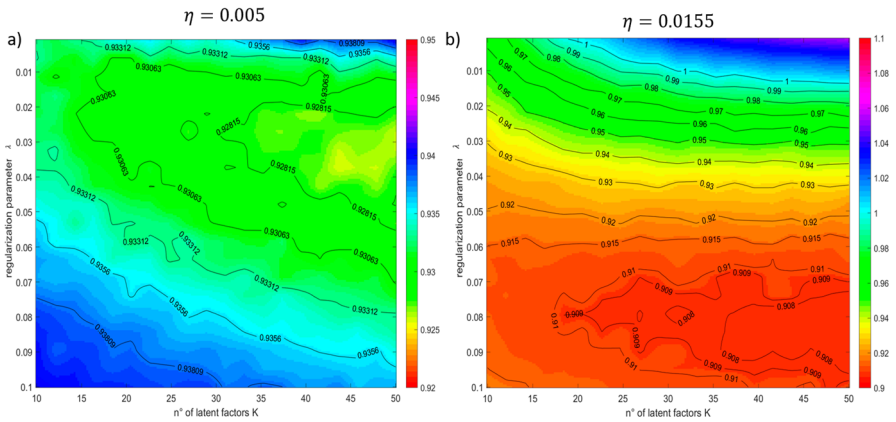

Fig. 10 Two images of the score function as a function of λ and K, fixing two different values of η

in many different works, among which (Tsai and Hung 2012; Matuszyk et al. 2016; Katarya and Verma 2017; Bogunovic et al. 2018). One of these datasets is called *MovieLens-100 k* and consists of 100,000 ratings (1–5) from 943 users on 1682 movies. Each user has rated at least 20 movies. The data was collected through the MovieLens web site (https://movielens.org) during the seven-month period from September 19th, 1997 through April 22nd, 1998.

To rating matrix associated to the problem consists of 943 rows (users), 1682 column (items) and about 100,000 known entries. To apply the procedure described in Sect. 2, we use the *Surprise* library (https://surpriselib.com/), a Python *scikit* library for recommender system. This library has a set of built-in prediction algorithms, among which the matrix-factorization algorithm with a default setting of the hyper-parameters, 0.02 for λ, 0.005 for η, and 100 for K, respectively. As performance function H, we consider a tenfold cross-validation procedure with 10 splits, from which we extract the mean RMSE on the different folds. The objective of this problem is to find the best configuration for hyper-parameters in as few iterations as possible. One iteration is defined as one call to the tenfold cross validation procedure. Using the default setting of the hyper-parameters, we obtain a value of 0.9296, where the rating error can be at most 4.

To search which hyper-parameter combination yields the best performance results, we define a configuration search space Θ with three dimensions (λ, η, K): the number of latent factors range in the integer interval {10, 100}, whereas the learning rate and the regularization parameter range in the continuous interval [0.001,0.1].

First, to analyse the complexity of the metric score function, in Fig. 10 we report two images in which we represent the score function as a function of λ and K, fixing to different value of η.

In the first case ($\eta = 0.005$) the value of the score function varies between 0.925 and 0.945. The score function decrease as a function of K, and increases as a function of λ, respectively. The worst score values are for $\lambda \approx 0$ (no regularization) and $K > 30$, and for $\lambda > 0.8$ and $K < 30$. The better score values are for $0.02 < \lambda < 0.04$ and

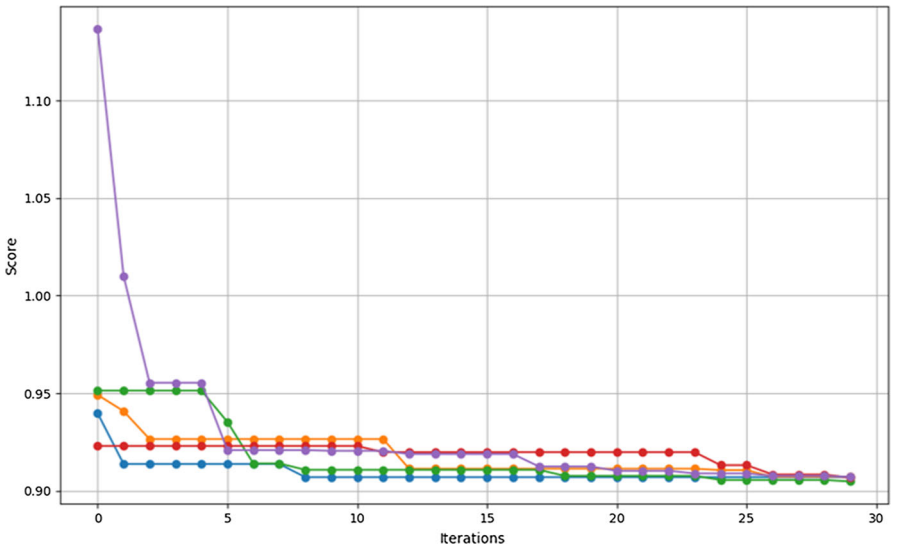

Fig. 11 Five curves representing five different runs of BO. The curves show the minimum found (y-axis) as a function of the number of iterations performed so far (x-axis)

$30 < K < 50$. In this part of the graph, we have a value of the score function less than 0.928.

In the second case ($\eta = 0.0155$), the values of the score function varies between 0.905 and 1.1. For $\lambda < 0.05$, the score function decreases as a function of λ, and increases as a function of K, respectively. In this part of the graph, we have values of the score function higher than 0.93. The worst case is when $\lambda \approx 0$ (no regularization) and $K > 20$, in which we have value of the score function over 1. Then for $\lambda > 0.05$, the score function appears less sensitive to the value of K and λ.

To apply BO we use the *gp_minimize* function from the *Scikit-Optimize* package (Head et al. 2019), based on the Scikit-Learn library (Pedregosa et al. 2012). This function uses GP as surrogate model with a Matern kernel ($\nu = 2.5$) and EI as acquisition function. The method to optimize the acquisition function is the Random Search, in which the function is evaluated on 10,000 points.

We perform BO several times using a different seed for the random generator. In Fig. 11, we report the results on five different experiments, in which we evaluate the objective function 30 times using BO (5 initial configurations chosen randomly, and 25 iterations of the BO algorithm). The plot shows the value of the minimum found (y-axis) as a function of the number of iterations performed so far (x-axis). The BO reaches a mean final value and standard deviation of 0.9064 and 0.0009, respectively, and the final values are all lower than the one obtained for the default setting. For the first iterations, the curves are different. After iteration five, the next point at which to evaluate the function is guided by the model, which is where an important decrease starts to appear. After iteration 25, all the curves seem to converge to a common value.

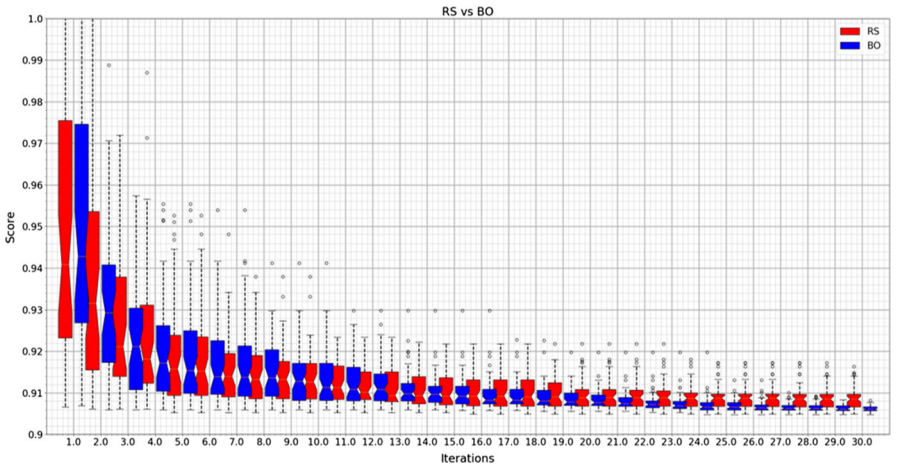

Fig. 12 Boxplot for each iterations of BO (blue) and RS (red) computed on 50 different experiments (color figure online)

To make the analysis of the BO results more robust, we perform a total of 50 different tests, and we report the results in Fig. 12 as a boxplot (minimum, first quartile, median, third quartile, and maximum) for each iteration. The performances are compared with that of Random Search (RS). As we can note, the behaviour of the two algorithms is similar in the first iterations (below 10/12 iterations). On the contrary, the performance for BO is better than RS in the last iterations: the BO reaches a mean final value and standard deviation of 0.9062 and 0.0007, respectively, whereas the RS reaches a mean final value and standard deviation of 0.9086 and 0.0026, respectively. One can test if the differences between the two distribution of BO and RS are meaningful or not at different iterations. This can be done performing the Mann–Whitney U test in which the null hypothesis is that "the two groups are sampled from populations with identical distribution". If the p-value is below 0.05, then the null hypothesis is not true, and the two groups are different. Performing a Mann–Whitney U test at iteration n° 1°, 10°, 20°, and 30°, we obtain a p value of 0.380, 0.496, 0.066, and 3.29e-09, respectively.

Then the two distributions can be considered similar in the first iterations but are different in the final ones. This means that the performance of BO and RS are similar in the first iterations, but then BO outperforms RS. This is due mainly to the capacity of BO to explore the most promising regions obtained by the associated surrogate model, that results better than RS for the exploitation phase.

Figure 13 represents the 2D scatter plot between η and λ (a), η and K (b), and λ and K(c), in which we display circles at the locations specified by the final configurations of the 50 tests for BO (blue circles) and RS (red circles). As a general result, we can see that η must be quite low and λ must be quite high, where K can take different values without altering so much the score value. We can see that many combinations of the hyper-parameters reach a low value of the score function, and that it is possible to consider a low value of K for the sake of interpretability. Indeed, the latent factors can refer, for MovieLens-100k dataset to the genre that the movie belongs to.

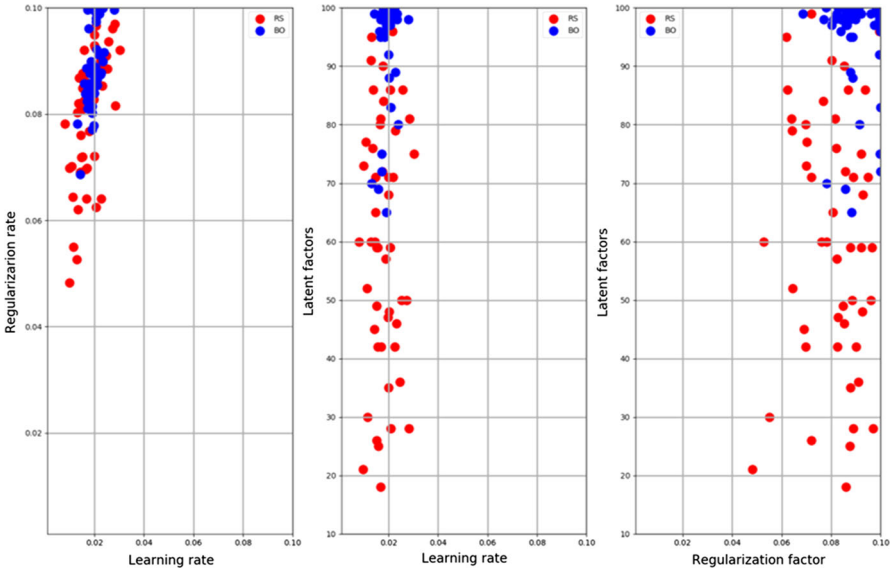

Fig. 13 2D scatter plots between η and λ (**a**), η and K (**b**), and λ and K (**c**), in which we display circles at the locations specified by the final configurations of the 50 tests for BO (blue circles) and RS (red circles)

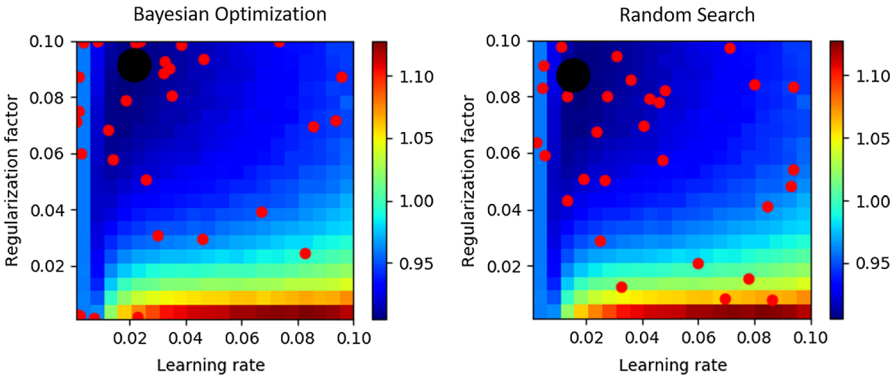

Fig. 14 Analysis of the distribution of the points chosen by BO (left) and RS (right), respectively, for two different tests. The found minimum is represented by the black point

In Fig. 14 we analyse the distribution of the points chosen by BO (left) and RS (right), respectively, for two different tests. The found minimum is represented by the black point. We report only the value of η and λ (x-axis) over the value of the objective function, fixing $K = 100$. From the figure, we can note that the distribution of points selected by BO is more focused on the most promising zone of the score function, located in the upper-left part. On the contrary, a few points are in the low part of the graph, where we have a high value of the score function.

This general behaviour is testified also in Fig. 15, in which we represent the boxplot of η (left) and λ (right) component for the group formed by all the best configurations

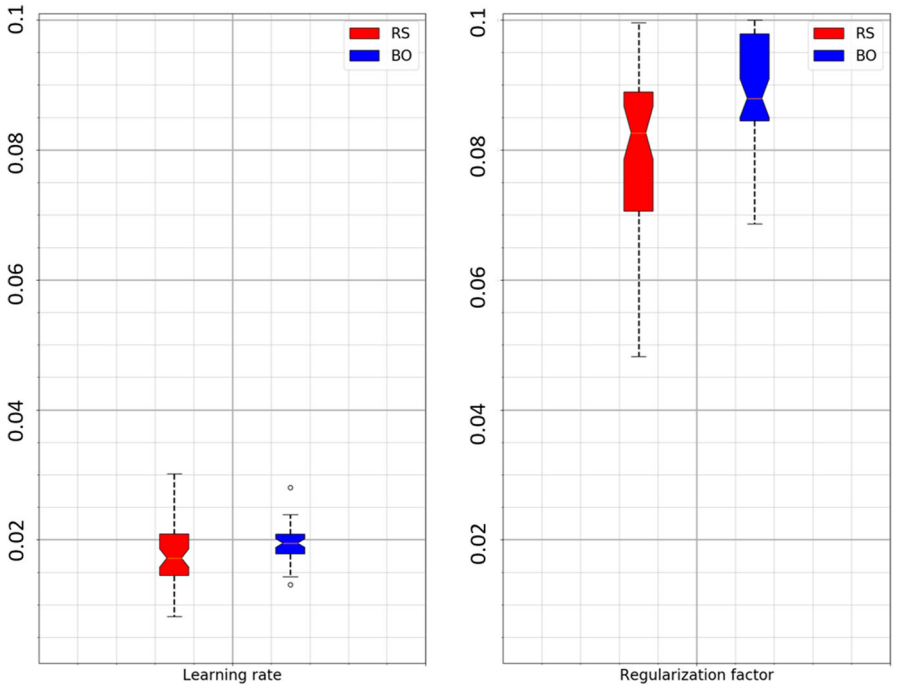

Fig. 15 boxplot of η (left) and λ (right) component for the group formed by all the best configurations obtained by the 50 tests for BO (blue) and RS (red) (color figure online)

obtained by the 50 tests for BO (blue) and RS (red). Note that BO group is more focused on the most promising zone of the score function, located in the upper-left part, with $\eta \approx 0.02$ and $\lambda \geq 0.8$.

5 Conclusions and remarks

In these years, many approaches (content or collaborative filtering) and types of ML algorithms (memory-based methods and model-based methods) have been studied to build efficient RS. Like many applications of ML, the aim of a RS is to predict the preferences of the users on new incoming data according to a huge amount of known data (the past histories of users and/or items). As a learning algorithm, we used the CF matrix-factorization method, which leads to an optimization problem that can be solved through the stochastic gradient descent algorithm. The effectiveness of this procedure for RS goes through the tuning of its hyper-parameters. Two of these, the number of latent factors and the regularization factor, are related to objective function, whereas the learning rate is related to the optimization procedure.

A default setting of these values cannot be done without any prior information about the problem but requires a hyper-parameter optimization procedure. The associated objective function, usually a score measure based on cross-validation, is noisy time-

consuming and black-box. The Grid Search method cannot assure a good precision if we use a large grid size or results too long to compute if we use a small grid size.

In this work, we have showed how the optimal hyper-parameter configuration can be obtained optimizing through BO a performance measure based on cross-validation. The success of BO strategy can be motivated by its exploration–exploitation balance: the exploration probes a larger portion of the search space to find the most promising regions, that are not yet refined, the exploitation allows of probing these promising regions in order to improve already promising solutions.

The numerical results have been obtained on a benchmark example, called Movielens-100k. The results showed that BO obtains a value of the performance metric slightly better than the RS. However, the better result obtained by BO could revealed more significant in case of complex objective or increasing the number of variables associated to different hyper-parameter optimization problem.

Funding Open access funding provided by Università degli Studi di Milano - Bicocca within the CRUI-CARE Agreement.

References

Aggarwal CC (2016) Recommender systems. Springer, Cham

Auer P (2003) Using confidence bounds for exploitation-exploration trade-offs. J Mach Learn Res 3:397–422. https://doi.org/10.1162/153244303321897663

Baptista R, Poloczek M (2018) Bayesian optimization of combinatorial structures. arXiv preprint: arXiv:180608838

Bogunovic I, Scarlett J, Jegelka S, Cevher V (2018) Adversarially Robust Optimization with Gaussian Processes. In: Advances in neural information processing systems

Bottou L (2010) Large-scale machine learning with stochastic gradient descent. In: Lechevallier Y, Saporta G (eds) Proceedings of COMPSTAT 2010. Physica-Verlag, Heidelberg

Cacheda F, Carneiro V, Fernández D, Formoso V (2011) Comparison of collaborative filtering algorithms. ACM Trans Web 5:1–33. https://doi.org/10.1145/1921591.1921593

Candelieri A, Giordani I, Archetti F et al (2018a) Tuning hyperparameters of a SVM-based water demand forecasting system through parallel global optimization. Comput Oper Res. https://doi.org/10.1016/j.cor.2018.01.013

Candelieri A, Perego R, Archetti F (2018b) Bayesian optimization of pump operations in water distribution systems. J Global Optim 71:213–235. https://doi.org/10.1007/s10898-018-0641-2

Cano A (2019) Recommender systems and hyper-parameter tuning. Towards Data Science

Crespo RG, Martínez OS, Lovelle JMC et al (2011) Recommendation system based on user interaction data applied to intelligent electronic books. Comput Hum Behav 27:1445–1449. https://doi.org/10.1016/j.chb.2010.09.012

De Rossi G, Kolodziej J, Brar G (2019) A recommender system for active stock selection. CMS. https://doi.org/10.1007/s10287-018-0342-9

Dewancker I, McCourt M, Clark S (2016) Bayesian optimization for machine learning: a practical guidebook. arXiv:161204858

Frazier PI (2018) A tutorial on Bayesian optimization. arXiv preprint arXiv:180702811
Frazier PI, Powell WB, Dayanik S (2008) A knowledge-gradient policy for sequential information collection. SIAM J Control Optim 47:2410–2439. https://doi.org/10.1137/070693424
Galuzzi BG, Perego R, Candelieri A, Archetti F (2018) Bayesian optimization for full waveform inversion. In: Daniele PSL (ed) New trends in emerging complex real life problems. Springer, Taormina, pp 257–264
Garnett R, Osborne MA, Roberts SJ (2010) Bayesian optimization for sensor set selection. In: Proceedings of the 9th ACM/IEEE international conference on information processing in sensor networks. IPSN'10. Stockholm, pp 209–219
Garrido-Merchán EC, Hernández-Lobato D (2018) Dealing with categorical and integer-valued variables in bayesian optimization with Gaussian processes. arXiv preprint arXiv:180503463
Harper FM, Konstan JA (2015) The MovieLens Datasets. ACM Trans Interact Intell Syst 5:1–19. https://doi.org/10.1145/2827872
Head T, Lueppe G, Shcherbatyi I, MechCoder (2019) Scikit-Optimize. GitHub repository
Katarya R, Verma OP (2017) An effective collaborative movie recommender system with cuckoo search. Egypt Inform J 18:105–112. https://doi.org/10.1016/j.eij.2016.10.002
Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42:30–37. https://doi.org/10.1109/MC.2009.263
Kushner HJ (1964) A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. J Basic Eng 86:97. https://doi.org/10.1115/1.3653121
Lee SK, Cho YH, Kim SH (2010) Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. Inf Sci 180:2142–2155. https://doi.org/10.1016/j.ins.2010.02.004
Matuszyk P, Castillo RT, Kottke D, Spiliopoulou M (2016) a comparative study on hyperparameter optimization for recommender systems. In: Workshop on recommender systems and big data analytics
McNally K, O'Mahony MP, Coyle M et al (2011) A case study of collaboration and reputation in social web search. ACM Trans Intell Syst Technol 3:1–29. https://doi.org/10.1145/2036264.2036268
Meldgaard SA, Kolsbjerg EL, Hammer B (2018) Machine learning enhanced global optimization by clustering local environments to enable bundled atomic energies. J Chem Phys. https://doi.org/10.1063/1.5048290
Mockus J (1989) The application of Bayesian methods. In: Dixon L, Szego G (eds) Towards global optimization. Springer, Dordrecht, pp 157–196
Oh C, Tomczak JM, Gavves E, Welling M (2019) Combinatorial Bayesian Optimization using the Graph Cartesian Product. arXiv:1902.00448v2
Olofsson S, Mehrian M, Calandra R et al (2018) Bayesian multi-objective optimisation with mixed analytical and black-box functions: application to tissue engineering. IEEE Trans Biomed Eng. 66(3):727–739
Pedregosa F, Varoquaux G, Gramfort A et al (2012) Scikit-learn: machine learning in python. J Mach Learn Res
Perdikaris P, Karniadakis GE (2016) Model inversion via multi-fidelity Bayesian optimization: a new paradigm for parameter estimation in haemodynamics, and beyond. J R Soc Interface. https://doi.org/10.1098/rsif.2015.1107
Salakhutdinov R, Mnih A (2008) Probabilistic matrix factorization. In: Advances in neural information processing systems (NIPS), pp 1257–1264
Shahriari B, Swersky K, Wang Z, et al (2015) Taking the human out of the loop: a review of Bayesian optimization. In: Proceedings of the IEEE, pp 148–175
Snoek J, Larochelle H, Adams RP (2012) Practical Bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp 2951–2959
Takács G, Pilászy I, Németh B, Tikk D (2009) Scalable collaborative filtering approaches for large recommender systems. J Mach Learn Res 10:623–656
Ho TK (1995) Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. IEEE, pp 278–282
Tsai CF, Hung C (2012) Cluster ensembles in collaborative filtering recommendation. Appl Soft Comput J 12:1417–1425. https://doi.org/10.1016/j.asoc.2011.11.016
Udell M, Horn C, Zadeh R, Boyd S (2016) Generalized low rank models. Found Trendsin Mach Learn 9:1–118. https://doi.org/10.1561/2200000055
Vanchinathan HP, Nikolic I, De Bona F, Krause A (2014) Explore-exploit in top-N recommender systems via Gaussian processes. In: Proceedings of the 8th ACM conference on recommender systems, pp 225–232

Yeehuda K (2010) Factor in the neighbors: scalable and accurate collaborative filtering. ACM Trans Knowl Discov Data (TKDD) 4:1