Journal of Big Data

RESEARCH Open Access

The role of classifiers and data complexity in learned Bloom filters: insights and recommendations

Dario Malchiodi[1,2*†], Davide Raimondi[1†], Giacomo Fumagalli[1†], Raffaele Giancarlo[3†] and Marco Frasca[1†]

†Dario Malchiodi, Davide Raimondi, Giacomo Fumagalli, Raffaele Giancarlo, and Marco Frasca have contributed equally to this work

*Correspondence: dario.malchiodi@unimi.it

[1] Department of Computer Science, Università degli Studi di Milano, Via Celoria 18, 20133 Milan, Italy
[2] CINI National Laboratory of Artificial Intelligence and Intelligent Systems (AIIS), University of Rome, 00185 Rome, Italy
[3] Department of Mathematics and CS, University of Palermo, Via Archirafi 34, 90123 Palermo, Italy

Abstract
Bloom filters, since their introduction over 50 years ago, have become a pillar to handle membership queries in small space, with relevant application in Big Data Mining and Stream Processing. Further improvements have been recently proposed with the use of Machine Learning techniques: learned Bloom filters. Those latter make considerably more complicated the proper parameter setting of this multi-criteria data structure, in particular in regard to the choice of one of its key components (the classifier) and accounting for the classification complexity of the input dataset. Given this State of the Art, our contributions are as follows. (1) A novel methodology, supported by software, for designing, analyzing and implementing learned Bloom filters that account for their own multi-criteria nature, in particular concerning classifier type choice and data classification complexity. Extensive experiments show the validity of the proposed methodology and, being our software public, we offer a valid tool to the practitioners interested in using learned Bloom filters. (2) Further contributions to the advancement of the State of the Art that are of great practical relevance are the following: (a) the classifier inference time should not be taken as a proxy for the filter reject time; (b) of the many classifiers we have considered, only two offer good performance; this result is in agreement with and further strengthens early findings in the literature; (c) Sandwiched Bloom filter, which is already known as being one of the references of this area, is further shown here to have the remarkable property of robustness to data complexity and classifier performance variability.

Keywords: Bloom filters, Learned Bloom filters, Approximate set membership, Dataset complexity

Introduction

In the past 25 years, Machine Learning (ML) radically impacted on the computing landscape, unveiling a very significant part of the applications which brought Artificial Intelligence in everyday life. Recent works show that ML can be used to design and analyze from a different perspective also some of the core components of "classical" computing field, such as algorithms and data structures. In particular, Kraska et al. [1] initiated a new research area focusing on learned data structures, currently under active development and with documented impacts on several domains, such as databases, network management [2] and Computational Biology [3]. An analogous trend concerning algorithms has been proposed by Mitzenmacher and Vassilvitskii [4]. Concerning data structures, the common theme to this new approach is that of training a classifier [5] or a regression model [6] on the input data. Then such a learned model is used as an oracle that a given "classical" data structure can use in order to answer queries with improved performance (usually w.r.t. time). To date, learned indexes have been the most studied, e.g., [1, 7–17], although rank/select data structures have also received some attention [18]. In this work, we focus on Bloom filters (BFs) [19], which represent one of the cornerstones of Big Data processing and stream analysis [20]. In short, they are special data structures expressly designed for storing a set which is large w.r.t. the available RAM resources. In a nutshell, rather than saving all the elements of the set, these structures wisely exploit hashing techniques so that it is possible to answer membership queries by using a relatively small amount of main memory. This property, which however is balanced with a controllable amount of false positives when the data structure is queried, makes Bloom filters heavily used in contexts characterized by the processing of a massive amount of data, such as for instance in detecting redundancy within a stream [21], in sequencing DNA data [22, 23], or when it is necessary to limit the number of unnecessary disk accesses in cloud-based databases [24]. As well as other data structures, also Bloom filters received attention in the realm of learned data structures, within which learned Bloom filters have been proposed to improve efficiency [25]. This is definitely natural, accounting for their pervasive use. Indeed, several variants have been proposed also for the classical version of Bloom filters, long before the appearance of their learned counterparts.

Bloom filters (BF) solve the *Approximate Set Membership* problem, defined as follows: having fixed a *universe* U and a set of *keys* $S \subset U$, for any given $\boldsymbol{x} \in U$, find out whether or not $\boldsymbol{x} \in S$. *False negatives*, that is negative answers when $\boldsymbol{x} \in S$, are not allowed. On the other hand, *false positives* (i.e., elements in $U \backslash S$ wrongly decreed as keys) are allowed, albeit their fraction (termed henceforth *false positive rate*, FPR for short) should be bounded by a given ϵ. The metrics that can be considered for evaluating any data structure solving the approximate set membership problems are: (1) the FPR, (2) the amount of RAM to be used to store the data structure, and (3) the *reject time*, that is the expected time required to reject any element not belonging to S. It is to be noted that the Bloom filter as described above assumes that U is one-dimensional, and we keep such an assumption throughout this paper. Bloom filters can be naturally extended to the multi-dimensional case with the use of suitable hash functions; however, the problem of how to design a multi-dimensional Bloom filter has not been considered and properly addressed in the scholarly literature, even in the learned setting which is considered here, e.g. [26, 27].

The first proposal for the learned version of a Bloom filter was naturally called learned Bloom filter (LBF) [1]; it is based on a binary classifier induced from data, with the aim of predicting set membership while requiring less space than a classical BF, being equal the FPR of the two structures (or vice versa, fix space budget and reduce the FPR). Such classifier is coupled with a standard BF storing the false negatives suffered from the former. When queried, the learned filter computes the prediction via the classifier and possibly uses the BF as a fallback ensuring that no false negatives on the training data are ever produced. Mitzenmacher [28] has provided a model for those filters, together with a very informative mathematical analysis of their pros/cons, resulting in new models for LBFs, and additional models have been introduced recently [29, 30]. It is important to underline that all the mentioned learned Bloom filters are static, i.e., no updates are allowed. This is the realm we are studying here, aiming at a suitable joint optimization of the aforementioned key resources. As for the dynamic case, some progress is reported in [31] in the Data Stream model of computation.

It is worth pointing out that, although they differ in architecture, each of these proposals has a binary classifier at its core. Somehow, not much attention has been devoted to the choice of the classifier to be used in practical settings, despite its centrality in this new family of filters and its role in the related theoretical analysis [28]. Kraska et al. use a Recurrent Neural Network, while Dai and Shrivastava [29] and Vaidya et al. [30] use Random Forests. These choices are only informally motivated, giving no evidence of superiority with respect to other possible ones, e.g., via a comparative analysis. Therefore, apart from preliminary investigations presented in [32, 33], the important problem of suitably choosing the classifier to be used to build a specific LBF has not been fully addressed so far. In addition to that, although the entire area of learned data structures and Algorithms finds its methodological motivation as a conceptual tool to reconsider classic approaches in a data-driven way, the role that the *complexity* of a dataset plays in guiding the practical choice of a learned data structure for that dataset has been considered to some extent for learned indexes only [15]. This aspect is even more relevant for LBFs. Indeed, as well discussed in [28], while the performance of classic BFs is "agnostic" w.r.t. the statistical properties of the input data, LBFs are quite dependent on them. In addition, it is well-known that the performance of a learnt classifier (a central component in this context) is very sensitive to the "classification complexity" of a dataset [34–37]. Nonetheless, the literature on LBFs usually describes experiments based on the processing of limited sets of benchmark data, e.g., in the realm of malicious URL detection [38] or geospatial information retrieval [27], without analyzing the sensitivity of the proposed approaches to the properties of the processed data. Such a state of the art is problematic, both methodologically and practically, for LBFs to be a reliable competitor of their classic counterparts.

Our aim is to provide a methodology and the associated software to support the design, analysis and deployment of state-of-the-art learned Boom filters with respect to specific constraints regarding their multi-criteria nature. Namely, space efficiency, false positive rate, and reject time. The present study is organized as follows. In "Classical and learned Bloom filters" section we briefly describe the original and learned variants of Bloom filters, underlining the specific roles of the hyperparameters to be tuned during the inductive phase. We illustrate in "Experimental methodology" section the methodology which we propose to LBF designers and users in need of studying the relations among the metrics used to evaluate how a (either classic or learned) filter performs on a fixed dataset, given the classification complexity of the latter, and the classifier the learned filter is based upon. "Experiments" section has the twofold aim of introducing the software platform within which we implemented the above-mentioned methodology and of describing the experiments which we carried out. In particular, having fixed the total space budget and the complexity of a dataset, we focus on the problem of choosing the most suitable classifier backing a LBF, also considering reject time as a selection criterion. In [28], a related approach uses a fixed filter, thus providing partial answers to our research question and, moreover, somehow suggesting an experimental methodology which has not been validated. The results of our experiments are discussed in "Results and discussion" section. In particular, we show that only two specific classifiers are worth to be considered when building a LBF, namely, those based on Support Vector Machines and feed-forward Neural Networks. Remarkably, they have been almost neglected in the existing literature, with the exceptions of [32, 33, 39].

Two further contributions regard: (1) an analysis of the relation among the classifier inference time and the learned BF reject time, showing that the first should not be taken as a proxy for second; (2) the study of how the complexity of a dataset impacts on the performance of several state-of-the-art learned Bloom filters, identifying a variant exhibiting higher robustness against variations in the dataset complexity and in the classifier performance. As a byproduct, "Guidelines" section provides a set of guidelines and recommendations for the use of state-of-the-art learned Bloom filter variants. Some concluding remarks end the paper.

Classical and learned Bloom filters

As mentioned in the "Introduction" section, a Bloom filter is a data structure designed in order to solve the Approximate Set Membership problem for a set S [19]. It is composed of an array $\boldsymbol{v}$ of m boolean entries and of k hash functions, $h_1, \dots, h_k$, mapping keys and non-keys onto positions within $\boldsymbol{v}$. Such functions are assumed to be *k-wise independent* [40, 41], although in practice less strict requirements are effective [19]. After the entries of $\boldsymbol{v}$ have been initialized to 0, all the keys $\boldsymbol{x} \in S$ are considered, setting $v_{h_j(\boldsymbol{x})} \leftarrow 1$, for each $j \in \{1, \dots, k\}$. At this point, the filter can be used to predict membership in the set for a generic value $\boldsymbol{x}$ by evaluating $v_{h_j(\boldsymbol{x})}$, for each j: if at least once the result is 0, the value is *rejected* (that is, it is predicted to be a non-key). Otherwise, it is classified as a key (an element of S). By construction, such predictions never incur in false negatives, although hash collisions might cause false positives. However, the rate ϵ of the latter is inversely bound by m as stated by equation (21) in [19], that formally relates reject time, space requirements, and FPR. This fact is typically used to choose the filter configuration. For instance, FPR can be minimized by suitably selecting the reject time once the filter space has been fixed. Similar relations can be exploited in order to fix m and k so as to build a BF exhibiting the most succinct configuration [25, 28]. Analogously, having fixed the FPR to ϵ and considering $n = |S|$ keys, the BF with optimal reject time can be obtained via an array of size

$$m = \frac{n}{\ln 2} \log_2(1/\epsilon) \tag{1}$$

bits, as shown in [42].

The learned variants of Bloom filters [1] can be seen as systems exhibiting the same properties of the latter, meanwhile reducing FPR or time/space resource demand. The common point to these variants is that all rely on a classifier induced from data. Their simplest implementation, which we will refer to as a LBF, is formally described as follows. Starting from a set of labeled instances $(\boldsymbol{x}, y_x)$, where $y_x = 1$ if $\boldsymbol{x} \in S$ and 0 otherwise, a classifier $C\colon U \to [0,1]$ is trained through supervised ML techniques to classify keys in S. Namely, the higher the *classification score* $C(\boldsymbol{x})$, the more likely $\boldsymbol{x} \in S$. A crisp prediction is obtained through the application of a threshold $\tau \in [0,1]$, so that $\boldsymbol{x} \in U$ is, at this stage, considered as a key if and only if $C(\boldsymbol{x}) > \tau$. This setting, however, might give rise to a non-empty set $\mathrm{FN} = \{\boldsymbol{x} \in \mathrm{S} \mid \mathrm{C}(\boldsymbol{x}) \le \tau\}$ of false negatives, which is not accepted within the Approximate Set Membership problem. Therefore, the classifier is coupled with a (classical) Bloom filter F, called the *backup filter*, used to hold the elements in FN. All queries for which the classifier produces a negative output are searched in F, swapping the prediction whenever F answers $\boldsymbol{x}$ is a key. In sum, any $\boldsymbol{x} \in U$ is predicted to be a key if the classification score $C(\boldsymbol{x})$ is greater than τ, or if $C(\boldsymbol{x}) \le \tau$ and F does not reject $\boldsymbol{x}$. In all other cases $\boldsymbol{x}$ is rejected.

An important difference between classical and learned Bloom filters is that the former have a FPR essentially independent of keys, whereas in a LBF the same quantity depends on the distribution of the instances used to query the filter itself. To this end, in the following we explicitly refer to the *empirical FPR*

$$\epsilon = \epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F \tag{2}$$

of a learned filter, where $\epsilon_\tau = |\{\boldsymbol{x} \in \overline{S} \mid C(\boldsymbol{x}) > \tau\}|/|\overline{S}|$ is the analogous empirical FPR of the classifier C on a query set $\overline{S} \subset U \backslash S$, and ϵ_F is the FPR of the backup filter. Given the previous definition of empirical FPR (2), the overall LBF has FPR equal to ϵ when the backup filter F is built ensuring $\epsilon_F = (\epsilon - \epsilon_\tau)/(1 - \epsilon_\tau)$, under the constraint $\epsilon_\tau < \epsilon$. Finally, we point out two important considerations. Firstly, choosing τ influences the values of the metrics used to evaluate a learned Bloom filter. Moreover, the dependency on data makes estimating the FPR in a learned setting no longer immediate as it is for classical filters, as highlighted in [28]. Similar considerations hold for the experimental methodology to be used to assess this FPR, which is one of the novelties proposed in this paper.

Alongside LBFs described so far, we consider the following variants.

1. Sandwiched LBFs (SLBFs) [28] are based on the insight that filtering out non-keys *before* querying the classifier can lead to an optimized space efficiency, and as a consequence to a smaller backup filter. More precisely, the provided keys are initially used to build an *initial* (still classical) BF I, and all keys in S not rejected by this filter are used to build a LBF. The overall system is queried as follows: if I rejects an element $\boldsymbol{x} \in U$, the latter is predicted as a non-key, otherwise the prediction provided for this element by the LBF is returned. The empirical FPR of a SLBF is obtained as $\epsilon = \epsilon_I\big(\epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F\big)$, being ϵ_I the FPR of the initial filter. Also in this case, the target FPR ϵ of the SLBF automatically identifies that of I. Precisely, $\epsilon_I = (\epsilon/\epsilon_\tau)(1 - |\mathrm{FN}|/\mathrm{n})$, where $|\mathrm{FN}|$ is the number of false negatives of C, under the constraint $\epsilon(1 - |\mathrm{FN}|/\mathrm{n}) \le \epsilon_\tau \le 1 - |\mathrm{FN}|/\mathrm{n}$. Also in this case, FPR, space requirement and reject time are affected by the classifier accuracy.
2. Adaptive LBFs (ADA-BF) [29] represent another variation of a LBF in which the training instances $\boldsymbol{x}$ are partitioned in g groups in function of their classification score $C(\boldsymbol{x})$. Each group is assigned a different set of hash functions (although globally using the same number of functions a LBF would use), and when membership

is to be predicted for an instance $\boldsymbol{x}$, this is done only using the hash functions of the corresponding score group. As for LBFs and SLBFs, the expected FPR is estimated empirically, though with a rather complex formula (the interested reader can refer to [29] for the mathematical details). As there are several variants of ADA-BF, in our experiments we consider the best performing one.

With the exception of another variant described in [30], for which the software is neither public nor available from the authors, our selection of learned BFs is state-of-the-art.

The induction of these learned filters requires to fine-tune some hyperparameters: LBF and SLBF have to fix the threshold τ, while ADA-BF needs the number g of groups to be set, as well as the value $\bar{c}$ representing the proportion of non-keys scores falling in two consecutive groups. "Selecting optimal classifiers and learned Bloom filters" section details how such hyperparameters have been tuned.

Experimental methodology

This section outlines the methodology which we use to design and analyse learned Bloom filters, also accounting for the inherent complexity of the dataset to be classified. In short, we generalize the approach described in [33] as follows. Starting from a dataset (either real-world or synthetically generated in function of suitable classification complexity metrics), we adopt the following pipeline: collect or generate data; use them to induce a classifier, estimate the corresponding FPR; build a learned Bloom filter based on this classifier; estimate the overall empirical FPR. Here below we detail the classifier families that we consider, as well as the above-mentioned procedure used to generate synthetic data.

The considered classifiers

We performed a preliminary experimentation phase involving an initial list of ML models—compiled without exhaustiveness pretensions—with the aim of identifying which among them were worth of further consideration, on the basis of the performance/space requirements trade-off (experiments and data about this initial step of our study are available upon request). Confirming the results of another study [32], Logistic Regression [43], Naive Bayes [44] and Recurrent Neural Networks [45] were removed from the list of classifier families to be considered, due to their poor trade-off performance. Here below we succinctly illustrate the retained models, also detailing their inference process in a context characterized by unbalanced labels, as our evaluation both considers balanced and unbalanced classification problems. We distinguish between *regular* and *key* hyperparameters of the corresponding learning algorithms, where the second category contains hyperparameters whose value affects the induced classifier space occupancy (which, when not differently specified, also acts as a proxy for classifier complexity). In light of this distinction, we only consider regular hyperparameters in the model selection phase. On the other hand, dedicated experiments are carried out w.r.t. different configurations for the key hyperparameters, with the aim of analysing the interplay among FPR, space requirements and reject time in learned Bloom filters.

In order to fix notation, we assume $U = \mathbb{R}^q$ as universe, and we refer to $D \subset U$ as a set of d labeled instances, denoting by $\boldsymbol{x} \in D$ a training instance and by $y_x \in \{0, 1\}$ its label (with a slightly different notation for SVMs, as explained here below).

Linear SVM. Classification in Linear Support Vector Machines (SVMs) for a given instance $\boldsymbol{x}$ is usually based on the optimal hyperplane $(\mathbf{w}, b)$ and the sign of $f(\boldsymbol{x}) = \mathbf{w} \cdot \boldsymbol{x} + b$. To fall in the setting defined for a LBF, we need to transform the SVM prediction into a score in [0, 1]. To this end, we use $f(\boldsymbol{x})$ as argument to a sigmoid function. The optimal hyperplane $(\mathbf{w}, b)$ is learned via maximization of the margin between the two classes, by solving

$$\begin{array}{ll} \min_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 + c\sum_{\boldsymbol{x}\in D}\xi_x \, , \\ \text{such that} & y_x f(\boldsymbol{x}) \geq 1 - \xi_x \quad \forall \boldsymbol{x} \in D \, , \\ & \xi_x \geq 0 \quad \forall \boldsymbol{x} \in D \, , \end{array}$$

where misclassification is allowed by the slack variables ξ_x and penalized using the hyperparameter $c > 0$ (note that in this case $y_x \in \{-1, 1\}$). Nonlinear SVMs might have been used here, but the need of storing the kernel matrix (e.g., containing the values of a Gaussian kernel) alongside the hyperplane parameters results in an unacceptable size for the learned filters. For the linear case we have chosen, we have only one non-key hyperparameter, namely c. When dealing with unbalanced labels, we consider the cost-sensitive SVM version described in [46].

Feed-forward NNs We also consider Feed-Forward neural networks [47, 48] accepting instances as inputs and equipped with l hidden layers, respectively having $h_1, \ldots, h_l$ hidden units (NN-$h_1, \ldots, h_l$ for short). One output unit completes the network topology. As usual, training involves the estimation of unit connections and biases from data. In this case, the h_is are key hyperparameters, while the learning rate lr acts as a regular hyperparameter to be tuned. It is worth noting that we fix the activation functions for all network units (although the former are tunable in principle), to limit the size of the already massive set of experiments. Precisely, as typically done, we use ReLU and sigmoid activations for hidden and output units, respectively. Where appropriate, label imbalance is dealt with by a cost-sensitive model variant [49].

Decision trees Additionally, we consider the Decision Tree classifier (DT) [50], a non-linear classifier based on recursive splits of the input space so as to suitably detect the regions to be assigned to the individual classes according to input data D. The set of recursive splits can be represented through a tree, where each node is associated with a feature and a threshold, used to bi-partition input data according to their value for that feature. Data are recursively split till leaf nodes, where no further split is possible. DTs are typically used to perform binary or multiclass classification, therefore leaves are associated with one of the classes. In our case instead, where a real score in [0, 1] is needed as output, the usual approach is to take the fraction of 1-labeled samples ending in a leaf node (w.r.t. the total number of samples ending in the same node) as its prediction value. The setting of this classifier is jointly selected with that of Random Forests (an extension of DTs explained in the next paragraph), to avoid the model selection for both models to become too computationally intensive. As a consequence, we fix the maximum depth and maximum number of leaves during the DT growth, and consider only one non-key hyperparameter, the minimum number δ of samples in a leaf. This hyperparameter prevents further splitting of nodes when the split would induce children containing less that δ samples, and accordingly allows to control the depth of the tree and its complexity. Although this can effectively affect the classifier size, the overall training setting of DT substantially limits the impact of this hyperparameter on the DT size.

It is worth noting here that the 'indirect' way to derive a score of a classifier inherently designed for discrete outputs might lead to 'unusual' distributions of output scores, which might produce unstable behaviors of the learned BF using it. For instance, this is the case when the number of leaves is small and/or multiple leaves contain the same fraction of 1-labeled instances (that is, they are associated with the same output score), which can cause the classifier to have only a few distinct output scores.

Random Forests Finally, we consider also Random Forests [51], shortened as RF-t to make explicit that this model is an ensemble of t classification trees. Each such tree is trained on a different bootstrap subset of D randomly extracted with replacement. Analogously, the splitting functions at the tree nodes are chosen from a random subset of the available attributes. The RF aggregates classifications uniformly across trees, computing for each instance the fraction of trees that output a positive classification. To address the case of unbalanced labels, we adopt an imbalance-aware variant of RFs [52, 53] in which, during the growth of each tree, the bootstrap sample is not drawn uniformly over D, but by selecting an instance $\boldsymbol{x}$ with probability

$$p_x = \begin{cases} \frac{1}{2|D_+|} & \text{if } y_x = 1, \\ \frac{1}{2|D_-|} & \text{if } y_x = 0, \end{cases}$$

where $D_+ = \{\boldsymbol{x} \in D | y_x = 1\}$, and $D_- = D \setminus D_+$. In this way, the probabilities of extracting a positive or a negative example are both 1/2, and the trees are trained on balanced subsets. The key hyperparameter of a RF is t, directly impacting on the classifier size. The non-key hyperparameters are selected as for DTs (see previous paragraph).

Measures of classification complexity

Several approaches can be considered when the classification complexity of a dataset is to be assessed (see [54] for a survey). In our experiments, we specifically focus on binary classification tasks, using the notation "class i", $i = 1, 2$, to refer to the two classes involved in the problem (although when less precision is needed, we resort to the classical terminology involving a *positive* and a *negative* class). Some of the measures listed in the above mentioned survey (precisely, *F1*, *T2*, or *T3*) proved to be insensitive across a variety of data synthetically generated, whereas the evaluation of other ones (mainly network- or neighborhood-based measures such as *LSC* and *N1*) required an excessive amount of RAM. Therefore we selected the *feature-based* measure *F1v* and the *class-imbalance* measure *C2*: both take values in [0, 1], and the higher the value, the more complex is the dataset. The former quantity, also called the Directional-vector Maximum Fisher's Discriminant Ratio, is defined as follows: denote, respectively, by p_i, μ_i, and Σ_i the proportion of examples belonging to class

i and the corresponding centroid and scatter matrix, so that $\boldsymbol{W} = p_1\Sigma_1 + p_2\Sigma_2$ and $\boldsymbol{B} = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^\top$ are the between- and within-class scatter matrices. In this case, $\boldsymbol{d} = \boldsymbol{W}^{-1}(\mu_1 - \mu_2)$ corresponds to the direction onto which there is maximal separation of the two classes (being $\boldsymbol{W}^{-1}$ the pseudo-inverse of $\boldsymbol{W}$), and we can define the $F1\nu$ measure as

$$F1\nu = \left(1 + \frac{\boldsymbol{d}^\top \boldsymbol{W} \boldsymbol{d}}{\boldsymbol{d}^\top \boldsymbol{B} \boldsymbol{d}}\right)^{-1}. \tag{3}$$

The second measure accounts for label balance in the dataset: letting n_i be the number of examples of class i, we have $C2 = (n_1 - n_2)^2/(n_1^2 + n_2^2)$.

Data generation

We generate synthetic data considering three parameters, a, r and ρ, which allow to tune the complexity of generated data according to the aforementioned measures. Intuitively, a controls the linearity of the separation boundary, r the label noise, and ρ the label imbalance. More precisely, in order to generate a binary classification dataset with a given level of complexity, n_1 positive and $n_2 = \lceil \rho n_1 \rceil$ negative instances (with $N = n_1 + n_2$), we proceed as follows. Let $\{\boldsymbol{x}_1, \ldots \boldsymbol{x}_N\} \subset \mathbb{R}^q$ be the set of samples, with each sample $\boldsymbol{x}_i$ having q features $x_{i1}, \ldots, x_{iq}$, and a binary label $y_i \in \{0, 1\}$. The N samples are drawn from a multivariate normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$, with $\boldsymbol{\Sigma} = \gamma \boldsymbol{I}_q$ (with $\gamma > 0$ and $\boldsymbol{I}_q$ denoting the $q \times q$ identity matrix). In our experiments we set $\gamma = 5$ so as to have enough data spread, reminding that this value however does not affect the data complexity. Without loss of generality, we consider the case $q = 2$. To determine the classes of positive and negative samples, the parabola $x_2 - ax_1^2 = 0$ is considered, with $a > 0$: a point $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ is positive ($y_i = 1$) if $x_{i2} - ax_{i1}^2 > 0$, negative otherwise ($y_i = 0$). This choice allows us to control the linear separability of positive and negative classes by varying the parameter a: the closer a to 0, the more linear the separation boundary. As a consequence, a controls the problem complexity for a linear classifier. An example of generated data by varying a is given in Fig. 1a–c. Further, to vary the data complexity even for non linear classifiers, labels are perturbed with different levels of noise: we flip the label of a fraction r of positive samples, selected uniformly at random, with an equal number on randomly selected negatives. The effect of three different levels of noise is depicted in Fig. 1d–f, where the higher the noise, the less sharp the separation boundary. The third parameter ρ is the ratio between the number of negative and positive samples in the dataset, thus it controls the $C2$ complexity measure. Higher values of ρ make the negative boundary more clear (Fig. 1g–i), while making harder training an effective classifier [55].

Experiments

This section describes the datasets and the hardware we use in our experiments, as well as the process we follow in order to select the optimal classifiers and the corresponding learned Bloom filters.

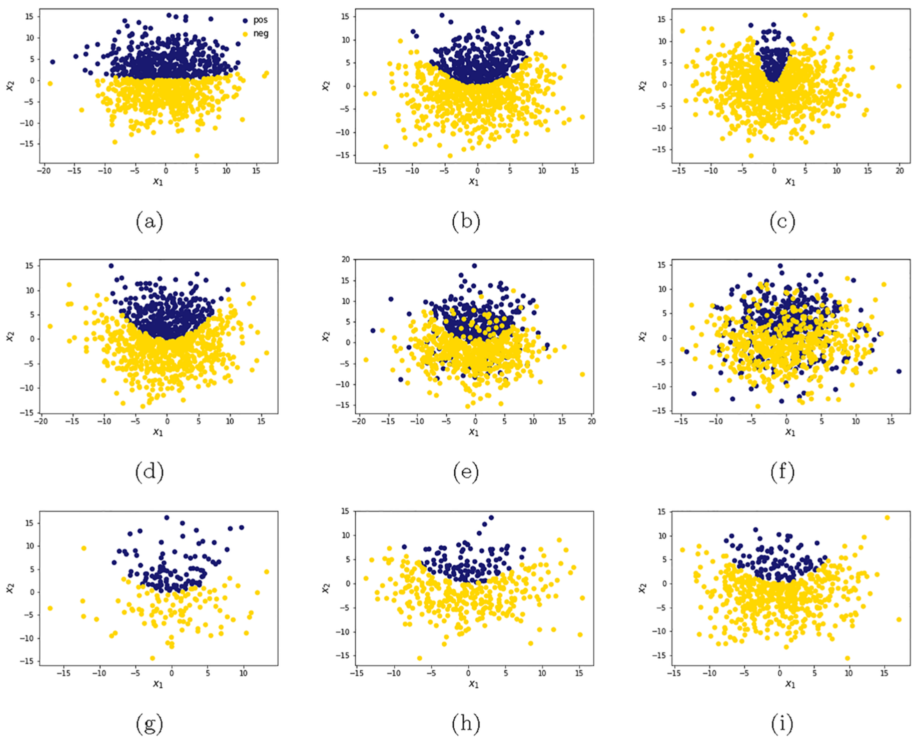

Fig. 1 Graphical representation of synthetic data: first row, parameter configuration is $n_p = 500, r = 0, \rho = 1$ and $a = 0.01$ (**a**), $a = 0.1$ (**b**), and $a = 1$ (**c**); second row $n_p = 500, a = 0.1, \rho = 1$ and $r = 0$ (**d**), $r = 0.1$ (**e**), and $r = 0.25$ (**f**); third row, $n_p = 100, a = 0.1, r = 0, \rho = 1$ (**g**), $\rho = 3$ (**h**), and $\rho = 5$ (**i**). "pos" and "neg" entries in the legend stand for positive and negative class, respectively

Table 1 Complexity of the real data

Data	F1v	C2
URL	0.08172	0.62040
DNA	0.99972	0

Datasets

We process both domain-specific and synthetically generated datasets. Concerning the first category, we use a URL dataset and a DNA dictionary. The first has been used by [29], who also kindly provided us with the dataset, as part of their experimentation on learned Bloom filters. The second dataset comes from experiments in Bioinformatics regarding the storage and retrieval of k-mers (i.e., strings of length k appearing in a given genome, whose spectrum is the dictionary of k-mers) [56] and was directly generated by us. We point out that no sensible information is contained in these datasets. With reference to Table 1, they represent two extreme cases of classification complexity: the URL dataset is *easy*, as it is simple in terms of linear separability ($F1v$), albeit exhibiting a relevant $C2$ complexity due to the label imbalance; the DNA data is *hard*, in that it has almost the maximum $F1v$ possible value, meaning that positive and negative classes are indistinguishable by a linear classifier.

The URL dataset contains 485,730 unique URLs, 80,002 *malicious* and the remaining *benign*. Seventeen lexical features are associated with each URL, which are used as the classification features. It is worth pointing out that all of the previous works on learned Bloom filters have used URL data. In this context, a Bloom filter can be used as a time- and space-efficient data structure to quickly reject benign URLs, never erroneously trusting a malicious URL although occasionally misclassifying a benign one. We adhere to this standard here.

The DNA dataset refers to the human chromosome 14, containing $n = 49{,}906{,}253$ 14-mers [56] constituting the set of our keys. As non-keys, we uniformly generate other n 14-mers from the 4^{14} possible strings on the alphabet $\{A, T, C, G\}$. Each 14-mer is associated with a 14-dimensional feature vector, whose components are the integers 0, 1, 2, 3, each associated with one of the four DNA nucleobases A, T, C, G, respectively (for instance a 14-mer TAATTACGAATGGT is coded as (1, 0, 0, 1, 1, 0, 2, 3, 0, 0, 1, 3, 3, 1)). A fundamental problem in Bionformatics, both technological [57] and in terms of evolutionary studies [58], is to quickly establish whether a given k-mer belongs to the spectrum of a genome. In this case, the Bloom filter stores the dictionary. It is worth mentioning that the use of Bloom filters in Bioinformatics is one of their latest fields of application, with expected high impact [59]. Such a domain has not been considered for the evaluation of learned Bloom filters, as we do here.

We also generate two categories of synthetic data, each attempting to reproduce the complexity of one of the domain-specific data. The first category has nearly the same $C2$ complexity of the URL dataset, i.e., it is *unbalanced*, with $n_1 = 10^5$ and $\rho = 5$. The second one has the same $C2$ complexity of the DNA dataset, i.e., it is *balanced*, with $n_1 = 10^5$ and $\rho = 1$. The choice of n_1 allows to have a number of keys similar to that in the URL data, and at the same time to reduce the computational burden of the massive set of experiments planned. Indeed, both balanced and unbalanced categories contain nine datasets, exhibiting increasing levels of $F1v$ complexity. Specifically, all possible combinations of parameters $a \in \{0.01, 0.1, 1\}$ and $r \in \{0, 0.1, 0.25\}$ are used. The corresponding complexity estimation is shown in Table 2. Consistently, $F1v$ complexity increases with a and r values, in both balanced and unbalanced settings. Noteworthy, the label imbalance slightly affects also the $F1v$ measure: in absence of label noise ($r = 0$), $F1v$ augments, likely due to the fact that $F1v$ is an *imbalance-unaware* measure;

Table 2 Complexity of the synthetic data

a	*r*	Balanced		Unbalanced	
		F1v	C2	F1v	C2
0.01	0	0.127	0.0	0.129	0.615
0.1	0	0.181	0.0	0.202	0.615
1	0	0.306	0.0	0.360	0.615
0.01	0.1	0.268	0.0	0.187	0.615
0.1	0.1	0.327	0.0	0.269	0.615
1	0.1	0.459	0.0	0.433	0.615
0.01	0.25	0.571	0.0	0.308	0.615
0.1	0.25	0.619	0.0	0.399	0.615
1	0.25	0.718	0.0	0.563	0.615

on the contrary, in presence of noise the $F1v$ complexity is barely reduced w.r.t. the balanced case. Although not immediate, the sense of this behavior might reside in what we also observe in Fig. 1g–i. That is, the boundary of negative class tends to be more crisp when ρ increases, thus mitigating the opposite effect the noise has on the boundary (Fig. 1d–f).

Hardware and software

We use two Ubuntu machines: an Intel Core i7-10510U CPU at 1.80 GHz×8 with 16 GB RAM, and an Intel Xeon Bronze 3106 CPU at 1.70 GHz× 16 with 192 GB RAM. This latter is used for experiments that require a large amount of main memory, i.e., on the DNA dataset. The supporting software [60] is written in Python 3.8, leveraging the ADA-BF public implementation provided in [61], which we extend as follows: (1) the construction of learned Bloom filters can be done in terms of the classifiers listed in "The considered classifiers" section and of the datasets illustrated in "Datasets" section; (2) SLBF is added to the already included BF models; (3) the choice of the classifier threshold τ is performed considering any number of evenly spaced percentiles of the obtained classification scores, instead than checking fixed values; (4) ranges for the hyperparameters of the learned versions of BFs can be specified by the user; (5) a main script allows to perform all experiments, rather than invoking several scripts, each dedicated to a LBF variant.

The provided implementation is built on top of standard libraries, listed in a dedicated environment file in order to foster replicability. In particular, the space required by a given classifier is computed, as typically done in these cases, using the *Pickle* module and accounting for both structure information and parameters [62], in order to obtain a fair comparison among all tested configurations. Moreover, the software is open to extensions concerning the inclusion of new datasets and/or new LBF models, thus it can be used as a starting point for further independent researches.

Selecting optimal classifiers and learned Bloom filters

Classifiers Being the classifier performance/size trade-off crucial for the learned variants of BF, we first assess the classifier generalization ability independently of the filter employing it. We adopt a threefold cross validation (CV) procedure (outer), measuring the classifier performance in terms of (a) the area under the ROC curve (AUC), and of (b) the area under the precision-recall curve (AUPRC), averaged across folds. We tune non-key hyperparameters of each model via a nested threefold CV (inner), where in each round of the outer CV, we select the optimal non-key hyperparameters through a grid search, retaining the configuration yielding the best AUPRC value. The following grids are considered: $c \in \{10^{-1}, 1, 10, 10^2, 10^3\}$ (SVM); $\delta \in \{1, 3, 5\}$ (DT and RF); $lr \in \{10^{-4}, 10^{-3}\}$ (NN). The different size of the grid across classifiers is due to the different training time of the classifier (SVM is the fastest one). The configuration of a classifier is strictly dependent on the space budget assigned to the LBF leveraging that classifier (see Table 5 discussed later on); consequently, the key hyperparameters for a given classifier, i.e., hyperparameters influencing the space occupancy, are set as follows. Recalling that no key hyperparameters exist for SVMs, we consider RFs related to two values of t, leading to a simpler and a more complex model. The choice

Table 3 Space occupancy in Kbits of selected classifiers on the considered datasets

SVM	DT	RF-10	RF-20	NN-25	NN-150,50	NN-200,75
`Synthetic Data`						
5	30.9	259.3	508.6	5.1	260.2	506.6
SVM	**DT**	**RF-10**	**RF-20**	**NN-7**	**NN-150,35**	**NN-175,70**
`URL Data`						
5.9	31	259.3	508.7	6.2	259.2	499.9
SVM	**DT**	**RF-10**	**RF-100**	**NN-7**	**NN-125,50**	**NN-500,150**
`DNA Data`						
5.8	30.9	259.5	2504	5.6	265.8	2652.3

The acronyms for all classifiers refer to the notation introduced in "The considered classifiers" section

Table 4 Average classifier inference time (across samples) in seconds. Same notations as in Table 3

SVM	DT	RF-10	RF-20	NN-25	NN-150,50	NN-200,75
`Synthetic Data`						
$1.278 \cdot 10^{-8}$	$2.651 \cdot 10^{-8}$	$4.425 \cdot 10^{-7}$	$8.968 \cdot 10^{-7}$	$8.494 \cdot 10^{-6}$	$9.257 \cdot 10^{-6}$	$1.008 \cdot 10^{-5}$
SVM	**DT**	**RF-10**	**RF-20**	**NN-7**	**NN-150,35**	**NN-175,70**
`URL Data`						
$3.730 \cdot 10^{-8}$	$6.515 \cdot 10^{-8}$	$5.815 \cdot 10^{-7}$	$9.930 \cdot 10^{-7}$	$6.825 \cdot 10^{-6}$	$7.018 \cdot 10^{-6}$	$7.198 \cdot 10^{-6}$
SVM	**DT**	**RF-10**	**RF-100**	**NN-7**	**NN-125,50**	**NN-500,150**
`DNA Data`						
$2.87 \cdot 10^{-8}$	$1.236 \cdot 10^{-7}$	$5.572 \cdot 10^{-7}$	$5.364 \cdot 10^{-6}$	$6.572 \cdot 10^{-6}$	$8.138 \cdot 10^{-6}$	$1.044 \cdot 10^{-5}$

$t = 10$, a reference already used in the literature [29], leads to a simple model, which we use on all datasets. As for the complex model, we have two subcases, depending on space budget. For the low budget case, we set $t = 20$, and the corresponding model is used for the synthetic/URL dataset. As for the high budget case, we set $t = 100$, and the corresponding model is used for the DNA dataset. Those choices, although empirical, indeed provide models with the required complexity. For completeness, we mention that other values of t could have been considered, but we have verified that those choices would not add any further insight to the perspective of our analysis. The key hyperparameters for NNs are selected so as to yield three models nearly having the same occupancy of the SVM and of the two RF models, for a fair comparison. Indeed, we concentrate on those two classifiers since they represent the two end-points in regard to the space occupancy spectrum: DTs use more space than the SVM and less than the RFs. The above-mentioned preliminary experiments have suggested, where enough space budget was available, that a two-layered topology is to be preferred to the one-layered one. Precisely, we consider the following models: NN-25, NN-150, 50 and NN-200, 75 (synthetic dataset); NN-7, NN-150, 35 and NN-175, 70 (URL dataset); NN-7, NN-125, 50, NN-500, 150 (DNA dataset). The final classifier configuration for all experiments and their space requirements are detailed in Table 3. In Table 4 we also include the average prediction time of the tested classifiers, that, jointly with results in Table 3, will be discussed in "Results and discussion" and "Guidelines" sections.

Learned Bloom filters We leverage the evaluation setting for the Bloom filter variants proposed by [29], and composed of the following steps: (1) train the classifiers using all keys and 30% of non-keys; (2) query the filter using remaining 70% of non-keys to compute the empirical FPR; (3) fix an overall memory budget of m bits the filters must use, and compare them in terms of their empirical FPR ϵ. Furthermore, we also measure the *average reject time* of filters, motivated by the fact that it can unveil interesting trends about the synergy of the filter variants and the classifier they employ. Indeed, we train any filter variant using in turn each of the considered classifiers.

The budget m is selected in relation to the desired (expected) ϵ of a classical Bloom filter, according to (1). We choose budgets differently on each dataset, since m directly depends on the key set size n. For synthetic data, having generated several datasets, we only test two different choices for the space budget m for each of them. Namely, those yielding $\epsilon \in \{0.05, 0.01\}$ for the classical Bloom filter using a bit vector of m bits. Conversely, more budget choices are tested on real datasets, since they are less numerous, namely those leading to $\epsilon \in \{0.01, 0.005, 0.001, 0.0005, 0.0001\}$. The difference between this setting and that of synthetic data is due to the following considerations. First, the dimensionality of synthetic data is 2, whereas that of real data it is 17 and 14, respectively, for URL and DNA. This makes the classifiers using real data larger than their counterparts on synthetic data. For this reason, on real data we omit the case $\epsilon = 0.05$, which yielded a too small budget. Indeed, some classifiers alone exceed the budget in this case (cfr. Table 3 for details about the space occupancy of classifiers). Moreover, having only two datasets, we can test more choices of ϵ, and accordingly better evaluate the behavior of learned Bloom filters when a smaller (expected) false positive rate is required.

Table 5 shows the obtained budget configurations for synthetic and real datasets. Even to train a learned Bloom filter we have operated a grid search to choose the optimal hyperparameters on the training data, optimizing with respect to the FPR. The following grids are utilized: (a) LBF and SLBF, 15 evenly spaced values for threshold τ in the classifier score range [0, 1]; (b) ADA-BF, the integers within [3, 15] for g, and 10 evenly spaced values in [1, 5] for $\bar{c}$ (cfr. "Classical and learned Bloom filters' section'). Importantly, the latter choice includes and extends the configurations adopted in [61] (namely, [8, 12] for g and [1.6, 2.5] for $\bar{c}$).

Table 5 Space budget in Kbits adopted on the various datasets. ϵ is the false positive rate, n is the number of keys in the dataset

Data	ϵ	Budget (Kbits)	n
Synthetic	0.05, 0.01	622, 956	10^5
URL	0.01, 0.005, 0.001, 0.0005, 0.0001	765, 880, 1148, 1263, 1530	$8 \cdot 10^4$
DNA	0.01, 0.005, 0.001, 0.0005, 0.0001	477460, 549325, 716191, 788056, 954921	$4.99 \cdot 10^7$

Results and discussion

In this section we present the results obtained from the classifier screening on both synthetic and real data, the experimental evaluation of all variants of LBFs based on those classifiers, and the relative discussion. In particular, we focus on the following questions: (1) It is possible to detect in advance, before constructing the learned BF, which classifier is best of use starting only from the classifier performance? (2) Is there a direct impact of the data complexity on the performance of a learned BF? (3) The noise in data plays a role of the performance of a learned BF? (4) Is there a relation linking the choice of a learned BF to reject time? We discuss the first question in "Performance of classifiers" section, the two subsequent ones in "Data classification complexity versus learned filters performance" section, and the last one in "Reject time" section.

Performance of classifiers

As clear from previous sections, the classifier can be seen as an oracle for a learned BF, where the better the oracle, the better the related filter, i.e., its FPR for a given fixed the space budget. Accordingly, we first discuss the performance of classifiers, tested on the datasets described in "Datasets" section, and according to the hyperparameter configurations described in "Selecting optimal classifiers and learned Bloom filters" section. Figure 2 depicts the performance of classifiers on *balanced* and *unbalanced* synthetic data, whereas results obtained on real data are shown in Fig. 3. However, it is central here to

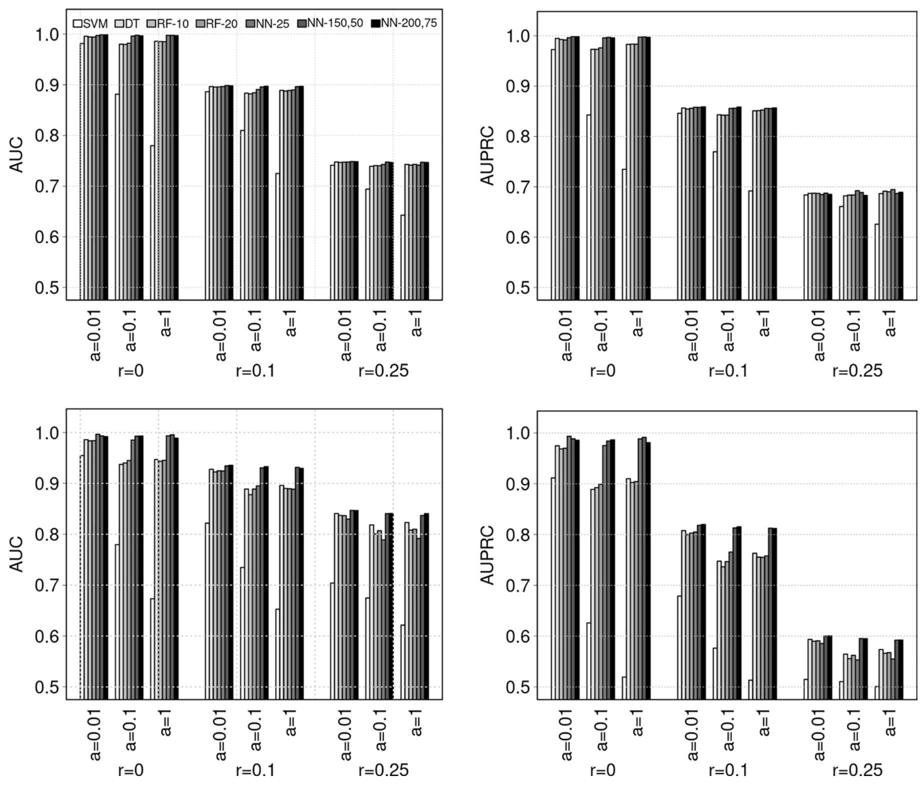

Fig. 2 Performance averaged across folds of compared classifiers on synthetic data. First row for *balanced* data, second row for *unbalanced* data. Bars are grouped by dataset, in turn denoted by a couple (a, r) expressing, respectively, the separation boundary linearity and the amount of label noise

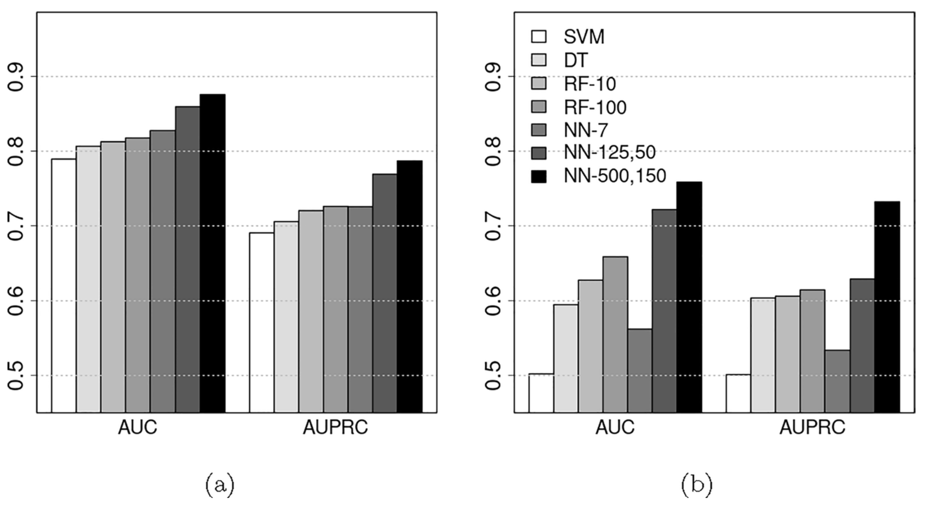

Fig. 3 Performance averaged across folds of compared classifiers on real data: **a** URL, **b** DNA

emphasize that the interpretation of such results is somewhat different than what one would do in a standard machine learning setting. Indeed, we have a space budget for the entire filter, and the classifier must discriminate well between keys and non-keys, while being substantially succinct with regard to the space budget of the data structure. Such a scenario implicitly imposes a performance/space trade-off: hypothetically, we might have a perfect classifier using less space than the budget, and on the other extreme, a poor classifier exceeding the space budget.

Overall results analysis

First, the behavior of classifiers in terms of AUC and AUPRC is coherent with what expected according to our methodology to generate synthetic data. Indeed, the SVM performance decays when the parameter a increases, being in line with the fact that it means increasing the non-linearity of the class separation boundary. Analogously, all classifiers worsen as noise r increases, which is clearly what to expect in this case. Moreover and most importantly, two main cases arise with respect to classification complexity: roughly $F1v \leq 0.35$ and $F1v > 0.35$. Being this threshold experimentally derived, the division between the two cases is not crisp. We refer to the first case as datasets 'easier and easier' to classify, for brevity 'easy', and to the second as datasets 'harder and harder' to classify, for brevity 'hard'.

Easy datasets. All classifiers perform very well on synthetic datasets with the stated complexity (except for SVMs when $a > 0.01$). Clearly, with such excellent oracles, the remaining part of a learned Bloom filter (e.g., with reference to the description of LBF, the backup filter) is intuitively expected to be very succinct.

Hard datasets. In this case, both AUC and AUPRC sensibly drop, being in some cases (SVM) not so far from the behavior of a random classifier. While in the previous case the performance of classifiers clearly yields the choice of the most succinct and faster model, here there is a trade-off to consider. Indeed, within the given space budget, at one end of the spectrum, we have the choice of a small-space and inaccurate classifier, at the other end of the spectrum we have larger and more accurate ones. As an example, for the LBF in the first case a large backup filter is required, whereas in the second one the classifier would use most of the space budget.

Preliminary observations on the classifiers to be retained

Here we address the question of how to choose a classifier to build the filter upon, based only on the knowledge of space budget and data classification complexity/classifier performance. On synthetic and URL data (Figs. 2, 3a), more complex classifiers perform just slightly better than the simpler ones, likely due to the low data complexity in these cases. At the same time, they require a sensibly higher fraction of the space budget (Table 5), and it is thereby natural to retain in these cases only the smallest/simplest variants, namely: DT, RF-10 and NN-25 (synthetic) and NN-7 (URL), in addition to SVM (we refer here, and in the rest of the paper, to the acronyms introduced in "The considered classifiers" section). Conversely, in our DNA experiments more complex classifiers substantially outperform the simpler counterparts, coherently with the fact that this classification problem is much harder (Tables 1, 2). Since the available space budget is larger in this case, all classifiers have been retained in the subsequent filter evaluation.

Performance of learned Bloom filters

In this section we: (1) explore the behavior of learned filters with respect to the data classification complexity, an aspect so far neglected in the literature (see Introduction); (2) discuss the reject time of filters with regard to their empirical FPR; (3) gain further insights about the interplay between the classifiers and the learned Bloom filter variants.

Data classification complexity versus learned filters performance

The empirical FPR of learned Bloom filters on balanced and unbalanced synthetic data (generated according to the procedure described in "Datasets" section) are respectively shown in Figs. 4 and 5, whereas Figs. 6 and 7 depict the results on URL and DNA data. In all cases, also the baseline Bloom filter is present.

Easy datasets. According to the rough definition introduced in "Overall results analysis" section ($F1v$ around 0.35 or smaller), the three/four leftmost configurations on the x-axis in Figs. 4 and 5 of synthetic data, and URL data can be considered as 'easy' data. In such cases, our results are coherent with those obtained in the literature, where ADA-BF slightly outperforms the other competitors [29], and RF-10 induces lower FPR values with regard to the baseline BF. However, such a classifier is not the best choice, since other ones induce filters with lower FPR, e.g., NN-25 for synthetic data and NN-7 for URL data. This again warns from the use of classifiers without a justification, as it has been done in previous studies.

Hard datasets. A novel scenario emerges with the increase of data complexity, i.e., when moving towards right on the horizontal axis in Figs. 4 and 5, or when considering DNA data. The performance of the filters drops more and more, in compliance with the performance decay of the subordinate classifiers ("Performance of classifiers" section), and unexpectedly the drop is faster in ADA-BF (and LBF) w.r.t. SLBF. This trend can be ascertained for instance on all synthetic data having $r > 0$ (noise injection). Unexpectedly because it represents an inversion with reference to the trend reported in the literature, where usually ADA-BF outperforms SLBF (which in

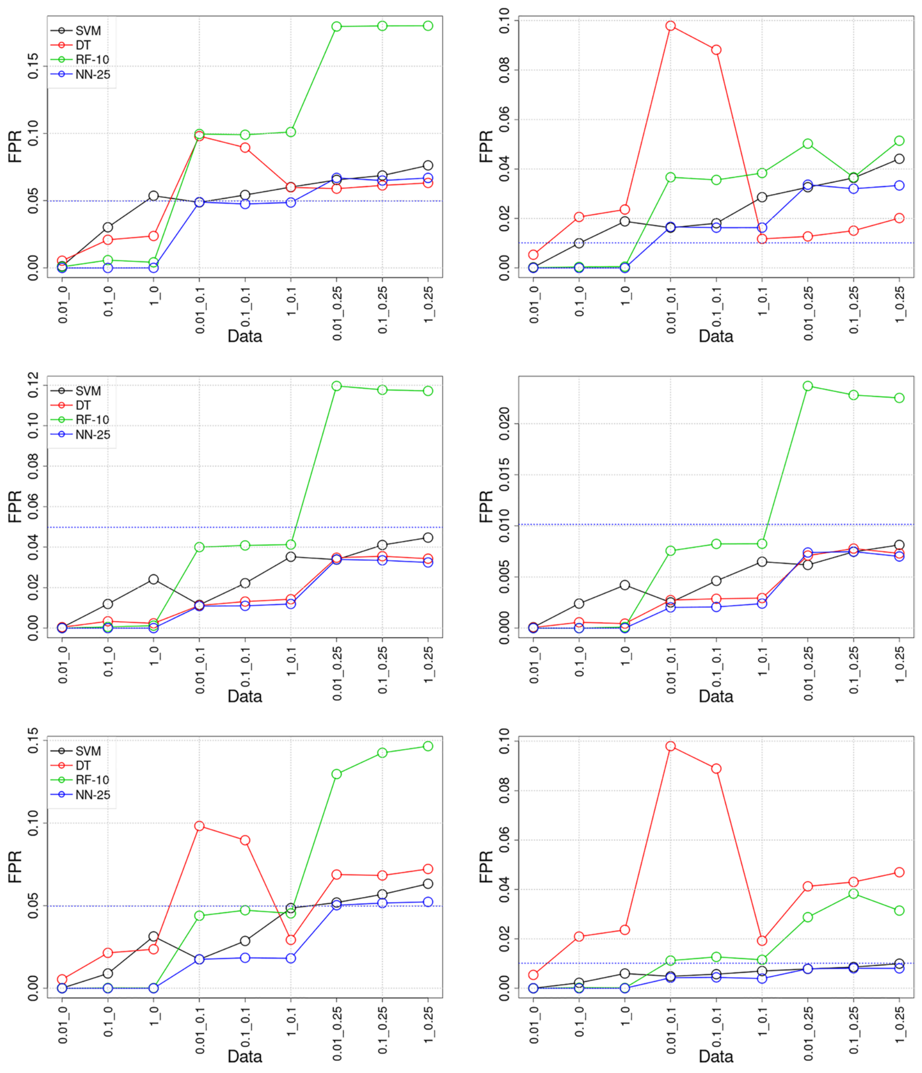

Fig. 4 False positive rates of LBF (first row), SLBF (second row), and ADA-BF (third row) attained on balanced synthetic datasets (cfr. "Datasets" section). On the horizontal axis, labels X_Y denote the dataset obtained when using $a = X$ and $r = Y$. The blue dotted line corresponds to the empirical false positive rate of the classical BF in that setting. Two space budgets m are tested, ensuring that $\epsilon = 0.05$ (left) and $\epsilon = 0.01$ (right) for the classical BF. Legends shared across rows

turn improves LBF). Our results indeed manifest that SLBFs is more robust to noise, which is likely due to a reduced dependency for SLBF on the classifier performance, favoured by the usage of the initial Bloom filter. Such a filter indeed allows the classifier to be queried only on a subset of the data.

When adopting RFs in this setting, the empirical FPR of learned filters strongly increases, and potential explanations reside in the excessive score discretization: having 10 trees, only 11 distinct scores are possible. In addition, RF space occupancy is larger (limiting in turn the space to be assigned to initial/backup filters). These results have a relevant confirmation on the very hard and large DNA dataset (Fig. 7), where LBF cannot attain any improvement with regard to the baseline BF, unlike SLBF and ADA-BF. A potential cause is in the worse performance of classifiers on

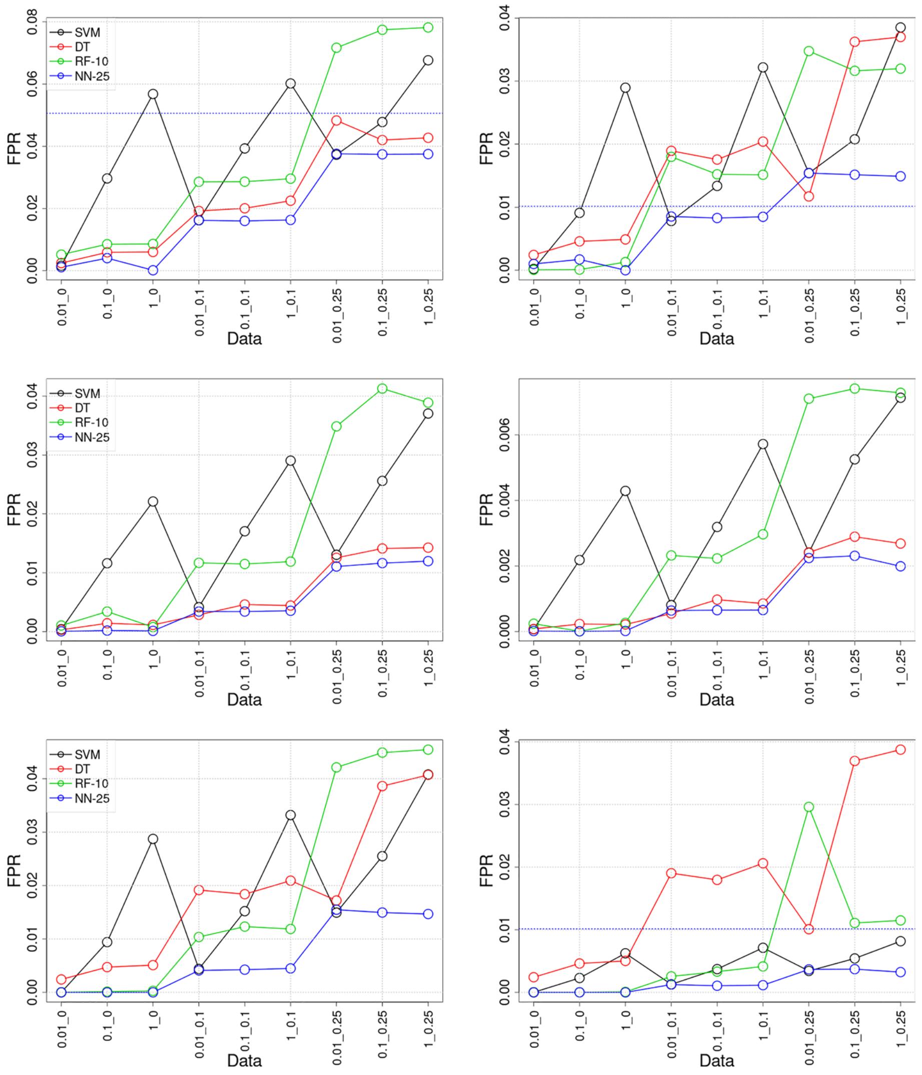

Fig. 5 False positive rates of LBF (first row), SLBF (second row), and ADA-BF (third row) attained on unbalanced synthetic datasets (cfr. "Datasets" section). On the horizontal axis, the labels X_Y denote the dataset obtained when using $a = X$ and $r = Y$. The blue dotted line corresponds to the measured false positive rate of the classical Bloom filter in that setting. Two space budgets are tested: that ensuring $\epsilon = 0.05$ for a classical Bloom filter (left), and that ensuring $\epsilon = 0.01$(right). Legends shared across rows

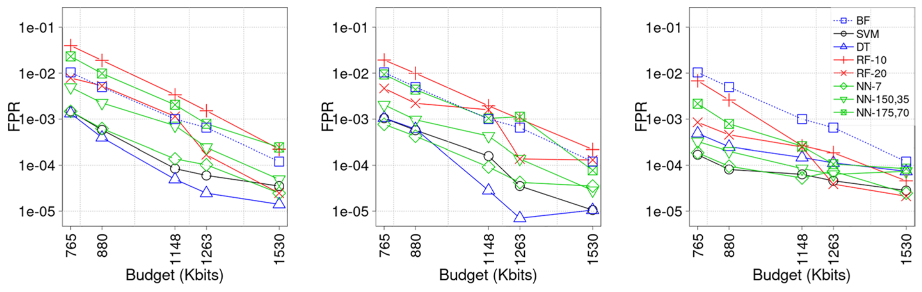

Fig. 6 Empirical false positive rate of LBF (left), SLBF (central), and ADA-BF (right) filters on URL data. On the horizontal axis the different budgets configurations. Dotted blue line represents the baseline classical Bloom filter. The legend is shared across rows

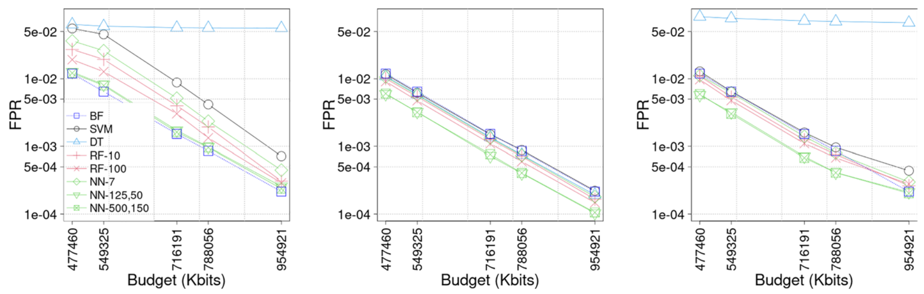

Fig. 7 Empirical false positive rate of LBF (left), SLBF (central), and ADA-BF (right) filters on DNA data. On the horizontal axis the different budgets configurations. Dotted blue line represents the baseline classical Bloom filter. Same legends as in Fig. 6

this hard dataset, differently from those obtained on synthetic and URL data, and in a too marked dependency of LBF on the classifier performance. Such a dependency is likely to be mitigated instead in the other two filter variants by the usage of the initial BF (SLBF) and by the fine-grained classifier score partition (ADA-BF). DTs even amplify this behavior: for the reasons explained in "The considered classifiers" section, its scores are strongly quantized and on this hard task we found they do not span the whole range [0, 1] (most leaf scores are concentrated around 0.5), fostering a nearly flat FPR for both LBF and ADA-BF. As a general tendency, SLBF outperforms both LBF and baseline of one order of magnitude in FPR with the same space amount, and ADA-BF when using weaker classifiers, or when a higher budget is available. We suspect the latter case is due to overfitting in the partitioning of classifier codomain ADA-BF operates, which is more deleterious when the classifier performance is not excellent, as it happens for DNA data. As a consequence, here the classifiers leading to the best empirical FPR values are the most complex, differently from hard synthetic data, where the key set was smaller (and consequently also the space budget was smaller). In particular, the best choice for these data are NN-500,150 and NN-125,50, which, as an additional motivation to employ them, can be also further compressed using specific techniques recently emerged [63]. In other words, we can conclude that too simple classifiers are useless or even deleterious on hard datasets, see for instance SVM-based filters, that never improve the baseline.

Reject time

Table 6 provides the average per-element reject time of all learned filters, taken across all the query sequences and space budgets that we have used in our experiments. They are expressed as percentage increment (or decrement) of the time required by the baseline. A first novel and interesting feature which emerges is that learned BF are sometimes faster than the baseline, which in principle is not expected, since learned variants have to query a classifier in addition to a classical BF. Our interpretation is that this can happen for two main reasons: (1) the adopted classifier is very fast and also effective, hence allowing in most cases to skip querying the backup filter; (2) the key set is very large, thus requiring a large baseline BF, whereas a good classifier can allow to sensibly drop the dimension of backup filters, thus making their invocation

Table 6 Learned Bloom filters average reject time, expressed as ratio between learned filter and baseline BF reject times (whose time in seconds is in parentheses)

Classifier	LBF	SLBF	ADA-BF
`Synthetic Data` ($1.364 \cdot 10^{-5}$)			
SVM	18.4	**6.1**	151.2
DT	−1.1	**−1.2**	1.3
RF	−11.1	**−17.5**	112.8
NN	106.9	**54.1**	159.3
`URL Data` ($3.259 \cdot 10^{-5}$)			
SVM	22.6	**3.7**	3.9
DT	**−1.4**	**−1.4**	−1.1
RF	**6.6**	7.1	9.7
NN	43.9	49.6	**35.3**
`DNA Data` ($4.817 \cdot 10^{-5}$)			
SVM	**−12.5**	−11.7	35.9
DT	−1.1	**−1.3**	1.3
RF-10	1.4	**−20.6**	32.0
RF-100	19.8	**−7.4**	40.6
NN-7	−5.0	**−12.0**	25.8
NN-125,50	−3.6	**−7.5**	25.2
NN-500,150	−1.9	**−11.6**	39.2

Positive (resp. negative) values indicate that the learned filter is slower (faster) than the baseline. The best configurations are highlighted in bold. Results are averaged across test queries and the filter space budgets considered. We remark that for DNA experiments another machine has been used w.r.t. synthetic and URL data (see "Hardware and software" section)

much faster. See for instance the case of DNA data, where most learned filters are faster that the baseline, with most classifiers.

Another intriguing behavior concerns the reject time of ADA-BF, often the worst architecture in terms of this metric. We believe it depends on the more complex procedure used in order to establish whether or not to access the backup filter—an exception is represented by DT-based filters, where, as apposite to other classifiers, the 'anomalous' score distribution moves the optimal number of groups learned towards the lowest values, and the resulting filters are faster. Indeed, such a procedure is subject to tuning, which in turn can yield less or more complex instances of the filter. As evident from our experiments, such a strategy does not always payoff.

Finally, we discuss the relationship between the inference time of the classifiers and the reject time of the learned Bloom filters, an aspect completely overlooked in the literature. In particular, from our experiments it emerges that the classifier inference time cannot be considered as a proxy for the reject time of the induced learned Bloom filter, and accordingly it is to be considered in the choice of the classifier only when classifiers exhibit very similar classification performance.

Indeed, we can observe that the order of the inference time of classifiers (Table 4) is often inverted with regard to the reject time of the corresponding filters. For instance, SVM is always the fastest classifier, but in some cases RF-based filters are faster (e.g., LBF and SLBF on synthetic data). This behavior is not so immediate to explain, and might be related to the following discussion. The accuracy of the classifier can impact on the false negative rate of the filter, and consequently on the number of false negatives to be stored in the backup filter/s (as well as on the reject time of the backup filter). Therefore, to optimize the reject time of a learned BF one can reduce the inference time of the classifier and/or reduce its false negative rate (that is, lowering the size of the backup filter). As an example, on synthetic data RFs outperform SVMs (Fig. 2), their inference time is one order of magnitude higher than SVMs (on average), but all the learned filters it induces have a lower reject time than the SVM-based counterparts (cfr. Table 6).

Guidelines

We summarize in this section our findings about the configuration of learned variants of Bloom filters, given a prior knowledge of data complexity and available space budget.

Data Complexity and Classifier Choice. With reference to Fig. 8, and recalling from "Reject time" section that the classifier inference time is not a proxy for the filter reject time, although it is natural to choose a fast classifier among several ones having comparable classification power, our recommendations are as follows. First, it must be evaluated how complex is the dataset, e.g. in our experiments *easy* ($F1v \leq 0.35$) and *hard* (otherwise). Choosing the classifier is relatively straightforward on easy datasets: independently of the space budget, it is always more convenient to select simple classifiers. We found linear SVMs to be the best choice on datasets having almost linear separation boundary ($a < 0.1$), and the smallest/simplest NNs are advisable to be used in the remaining cases of this category.

Conversely, the space budget becomes more discriminant for the classifier choice on hard datasets. Within the budget given by (1), we can distinguish two extreme cases for hard datasets: those having a relatively small set of keys, and accordingly a small budget, and those having instead a large key set and a high budget. When the budget is small, for instance like it happens in more complex synthetic data, the following considerations hold: (1) the choice is almost forced towards small (and potentially inaccurate) classifiers, being the larger ones too demanding for the available budget; (2) a (linear) SVM is to be excluded, due to the increased difficulty of the task; Therefore, among the remaining classifiers, we recommend the use of small NNs, which in our experiments were the

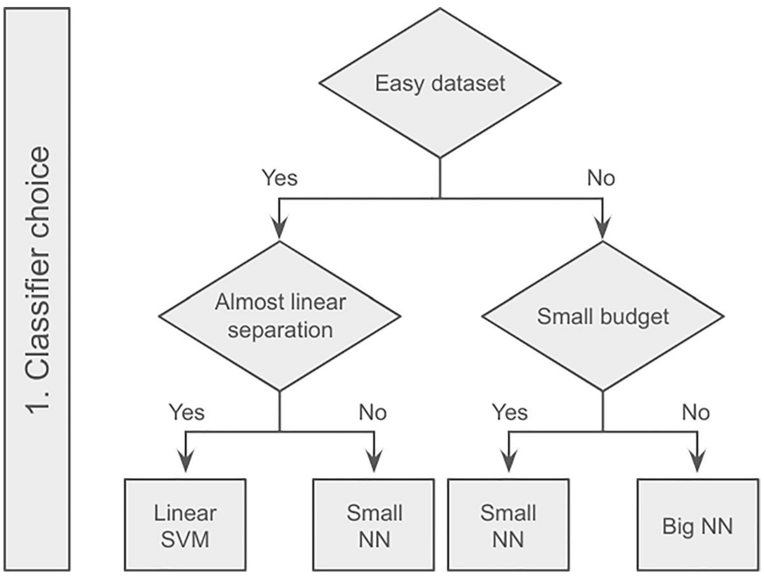

Fig. 8 Recommended guidelines for choosing the classifier in a learned BF

most succinct model among the best performing ones (cfr. Fig. 2). On the other hand, when the budget is high (as with DNA data) our findings suggest to learn more accurate classifiers, even if this requires the usage of a considerable budget fraction. Therefore, we suggest to invest a significant part of the budget in a complex model (in our experiments, the most complex NNs) that will better fit the available data. This is motivated by the following facts: (1) the gain related to higher classification abilities allows to save space when constructing the auxiliary BFs of the learnt filters, and to consequently have a smaller reject time (cfr. "Data classification complexity versus learned filters performance" section); (2) too poor classifiers induce the learned BFs to perform even worse than their classical counterpart (see Fig. 7); (3) the availability of a big amount of data allows to train complex classifiers more effectively [64].

Learned Bloom Filters Choice. With regard to the choice of the learned BF to be applied, we recommend the suggestions depicted in Fig. 9. In presence of complex or noisy data, use a SLBF, in view of its ability to exploit even classifiers having a poor performance. In the remaining situations, the required reject time becomes discriminant. Since ADA-BF often exhibits the highest reject times, if a fast learned BF is required, SLBF should be preferred over ADA-BF, otherwise ADA-BF is a better choice (confirming previous literature results).

Conclusions and future developments

The present study proposes an experimental evaluation that can guide in the design and validation of learned Bloom filters. The key point is to base the choice of the classifier to be used in a learned Bloom filter on the space budget of the entire data

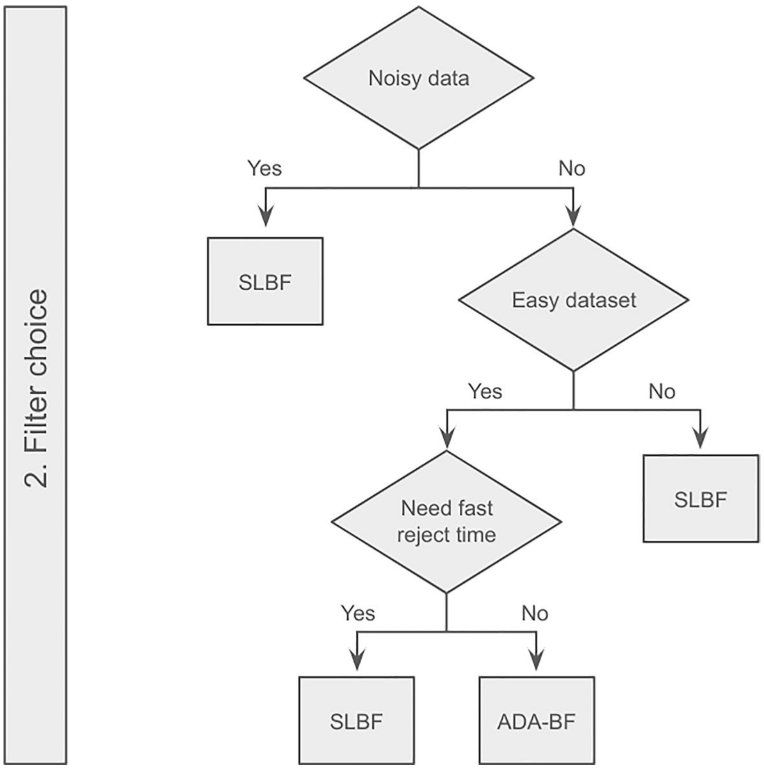

Fig. 9 Recommended guidelines for choosing the learned BF type

structure, as well as on the classification complexity of the dataset in input. Our experimental approach has unveiled behaviors of learned Bloom filters neglected by previous studies, including: (1) how robust are the current learned filters for the processing of datasets of increasing complexity: indeed, only the "easy to classify" scenario has been considered in the Literature so far; (2) how robust are the learned filters with regard to noise injection in input data: our methodology revealed that the efficiency ranking of learned Bloom filters emerged in previous studies must be revised in presence of noisy data; (3) how crucial and discriminant is the selection of the classifier in terms of false positive rate, size and reject time of the learned Bloom filter. We have summarized such novel insights within a "Guidelines" section to help practitioners in suitably designing learned Bloom filters for their applications.

A potential limitation of such results is that they might be dependent on the considered data; nonetheless, this is somehow inevitable due to the nature of learned data structures. Finally, we point out that the societal impacts of our contributions are in line with general-purpose Machine Learning technology. Natural extensions of this research are the following. As already remarked, we have complied with an experimental setting coherent with the state of the art. We can also consider the scenario in which the desired false positive rate is fixed (instead of the overall filter size) and one asks for the most succinct pair classifier-filter (instead of the lowest false positive rate). Moreover, in his seminal paper [28], Mitzenmacher has shown that learned Bloom filters can be quite sensitive to the input query distribution. Yet, no study is available to quantify this aspect. Our methodology can be extended also to those types of analysis and work in this direction is in progress. Finally, as outlined in the Introduction, the multi-dimensional case has received very little attention, and it deserves further investigations.

Acknowledgements
We would like to thank Z. Dai and A. Shrivastava for having provided us the dataset used in [29].

Authors' contributions
DM, RG and MF conceived the study, designed the experiments and discussed the results. They also were major contributors in writing the manuscript. DR, GF and MF performed the experiments. All authors read and approved the final manuscript.

Funding
This work has been supported by the Italian MUR PRIN project "Multicriteria data structures and algorithms: from compressed to learned indexes, and beyond" (Prot. 2017WR7SHH). Additional support to RG has been granted by Project INdAM—GNCS "Analysis and Processing of Big Data based on Graph Models". DM, DR, GF and MF acknowledge the support of the APC central fund of the University of Milan.

Availability of data and materials
The source code, data, and/or other artifacts have been made available at https://github.com/RaimondiD/LBF_ADABF_experiment.

Declarations

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interest
The authors declare that they have no competing interest.

Received: 11 May 2023 Accepted: 14 March 2024
Published online: 27 March 2024

References

1. Kraska T, Beutel A, Chi EH, Dean J, Polyzotis N. The case for learned index structures. In: Proceedings of the 2018 international conference on management of data. SIGMOD '18. New York: Association for Computing Machinery, 2018. p. 489–504. https://doi.org/10.1145/3183713.3196909.
2. Wu Q, Wang Q, Zhang M, Zheng R, Zhu J, Hu J. Learned bloom-filter for the efficient name lookup in information-centric networking. J Netw Comput Appl. 2021;186:103077. https://doi.org/10.1016/j.jnca.2021.103077.
3. Kirsche M, Das A, Schatz MC. Sapling: accelerating suffix array queries with learned data models. Bioinformatics. 2020;37(6):744–9. https://doi.org/10.1093/bioinformatics/btaa911.
4. Mitzenmacher M, Vassilvitskii S. Algorithms with predictions. In: Roughgarden T, editor. Beyond the worst-case analysis of algorithms. Cambridge: Cambridge University Press; 2021. p. 646–62. https://doi.org/10.1017/9781108637435.037
5. Duda RO, Hart PE, Stork DG. Pattern classification. 2nd ed. New York: Wiley; 2000.
6. Freedman D. Statistical models?: Theory and practice. Cambridge: Cambridge University Press; 2005.
7. Amato D, Lo Bosco G, Giancarlo R. Standard versus uniform binary search and their variants in learned static indexing: the case of the searching on sorted data benchmarking software platform. Softw Pract Exp. 2023;53(2):318–46. https://doi.org/10.1002/spe.3150.
8. Amato D, Giancarlo R, Lo Bosco G. Learned sorted table search and static indexes in small-space data models. Data. 2023;8(3):56. https://doi.org/10.3390/data8030056.
9. Amato D, Lo Bosco G, Giancarlo R. Neural networks as building blocks for the design of efficient learned indexes. Neural Comput Appl. 2023;35(29):21399–414. https://doi.org/10.1007/s00521-023-08841-1.
10. Ferragina P, Frasca M, Marinò GC, Vinciguerra G. On nonlinear learned string indexing. IEEE Access. 2023;11:74021–34. https://doi.org/10.1109/ACCESS.2023.3295434.
11. Ferragina P, Vinciguerra G. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. PVLDB. 2020;13(8):1162–75. https://doi.org/10.14778/3389133.3389135.
12. Ferragina P, Lillo F, Vinciguerra G. On the performance of learned data structures. Theor Comput Sci. 2021;871:107–20.
13. Kipf A, Marcus R, van Renen A, Stoian M, Kemper A, Kraska T, Neumann T. Radixspline: a single-pass learned index. In: Proceedings of the of the third international workshop on exploiting artificial intelligence techniques for data management. aiDM '20. New York: Association for Computing Machinery; 2020. p. 1–5.
14. Kirsche M, Das A, Schatz MC. Sapling: accelerating suffix array queries with learned data models. Bioinformatics. 2020;37(6):744–9.
15. Maltry M, Dittrich J. A critical analysis of recursive model indexes. Proc VLDB Endow. 2022;15(5):1079–91. https://doi.org/10.14778/3510397.3510405.
16. Marcus R, Kipf A, van Renen A, Stoian M, Misra S, Kemper A, Neumann T, Kraska T. Benchmarking learned indexes, vol. 14; 2020. p. 1–13. arXiv preprint arXiv:2006.12804
17. Marcus R, Zhang E, Kraska T. CDFShop: Exploring and optimizing learned index structures. In: Proceedings of the 2020 ACM SIGMOD international conference on management of data. SIGMOD '20; 2020; p. 2789–2792.
18. Boffa A, Ferragina P, Vinciguerra G. A "learned" approach to quicken and compress rank/select dictionaries. In: Proceedings of the SIAM symposium on algorithm engineering and experiments (ALENEX); 2021.
19. Bloom BH. Space/time trade-offs in hash coding with allowable errors. Commun ACM. 1970;13(7):422–6. https://doi.org/10.1145/362686.362692.
20. Leskovec J, Rajaraman A, Ullman JD. Mining of massive data sets. 2nd ed. Cambridge: Cambridge University Press; 2014. https://doi.org/10.1017/CBO9781139924801.
21. Almeida PS, Baquero C, Preguiça N, Hutchison D. Scalable Bloom filters. Inf Process Lett. 2007;101(6):255–61.
22. Melsted P, Pritchard JK. Efficient counting of k-mers in DNA sequences using a Bloom filter. BMC Bioinf. 2011;12(1):1–7.
23. Zhang Z, Wang W. RNA-Skim: a rapid method for RNA-Seq quantification at transcript level. Bioinformatics. 2014;30(12):283–92. https://doi.org/10.1093/bioinformatics/btu288.
24. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: a distributed storage system for structured data. ACM Trans Compute Syst. 2008;26(2):1–26.
25. Broder A, Mitzenmacher M. Network applications of Bloom filters: a survey. In: Internet mathematics, vol. 1, 2002. p. 636–646. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.98
26. Crainiceanu A, Lemire D. Bloofi: multidimensional Bloom filters. Inf Syst. 2015;54:311–24. https://doi.org/10.1016/j.is.2015.01.002.
27. Zeng M, Zou B, Kui X, Zhu C, Xiao L, Chen Z, Du J, et al. Pa-lbf: prefix-based and adaptive learned bloom filter for spatial data. Int J Intell Syst. 2023;2023.
28. Mitzenmacher M. A model for learned bloom filters and optimizing by sandwiching. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in Neural Information Processing Systems, vol. 31. Red Hook: Curran Associates; 2018. p. 1.
29. Dai Z, Shrivastava A. Adaptive Learned Bloom Filter (Ada-BF): Efficient utilization of the classifier with application to real-time information filtering on the web. In: Advances in neural information processing systems, vol. 33, Red Hook: Curran Associates, Inc.; 2020. p. 11700–11710. https://proceedings.neurips.cc/paper/2020/file/86b94dae7c6517ec1ac767fd2c136580-Paper.pdf
30. Vaidya K, Knorr E, Kraska T, Mitzenmacher M. Partitioned learned Bloom filters. In: International conference on learning representations; 2021. https://openreview.net/forum?id=6BRLOfrMhW
31. Liu Q, Zheng L, Shen Y, Chen L. Stable learned Bloom filters for data streams. Proc VLDB Endow. 2020;13(12):2355–67. https://doi.org/10.14778/3407790.3407830.
32. Fumagalli G, Raimondi D, Giancarlo R, Malchiodi D, Frasca M. On the choice of general purpose classifiers in learned Bloom filters: an initial analysis within basic filters. In: Proceedings of the 11th international conference on pattern recognition applications and methods (ICPRAM); 2022. p. 675–682.

33. Dai Z, Shrivastava A, Reviriego P, Hernández JA. Optimizing learned bloom filters: How much should be learned? IEEE Embedded Syst Lett. 2022;14(3):123–6.
34. Ali S, Smith KA. On learning algorithm selection for classification. Appl Soft Comput. 2006;6(2):119–38.
35. Cano J-R. Analysis of data complexity measures for classification. Expert Syst Appl. 2013;40(12):4820–31. https://doi.org/10.1016/j.eswa.2013.02.025.
36. Flores MJ, Gámez JA, Martínez AM. Domains of competence of the semi-naive Bayesian network classifiers. Inf Sci. 2014;260:120–48.
37. Luengo J, Herrera F. An automatic extraction method of the domains of competence for learning classifiers using data complexity measures. Knowl Inf Syst. 2015;42(1):147–80.
38. Patgiri R, Biswas A, Nayak S. deepbf: Malicious url detection using learned bloom filter and evolutionary deep learning. Comput Commun. 2023;200:30–41.
39. Malchiodi D, Raimondi D, Fumagalli G, Giancarlo R, Frasca M. A critical analysis of classifier selection in learned bloom filters: the essentials. In: Iliadis, L., Maglogiannis, I., Castro, S., Jayne, C., Pimenidis, E. (eds.) Engineering application of neural networks—24th international Conference—EAAAI/EANN 2023—León, Spain, June 14-17, 2023—Proceedings. Communications in Computer and Information Science, vol. 1826; 2023, p. 47–61. Springer.
40. Wegman MN, Carter JL. New hash functions and their use in authentication and set equality. J Comput Syst Sci. 1981;22(3):265–79. https://doi.org/10.1016/0022-0000(81)90033-7.
41. Carter JL, Wegman MN. Universal classes of hash functions. J Comput Syst Sci. 1979;18(2):143–54. https://doi.org/10.1016/0022-0000(79)90044-8.
42. Broder A, Mitzenmacher M. Network applications of Bloom filters: a survey. Internet Math. 2004;1(4):485–509.
43. Cox DR. The regression analysis of binary sequences. J R Stat Soc Ser B (Methodol). 1958;20(2):215–32.
44. Duda RO, Hart PE. Pattern classification and scene analysis. New York: Willey; 1973.
45. Cho K, van Merriënboer B, Bahdanau D, Bengio Y. On the properties of neural machine translation: Encoder–decoder approaches. In: Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. p. 103–111. Association for Computational Linguistics, Doha, Qatar; 2014. https://doi.org/10.3115/v1/W14-4012. https://aclanthology.org/W14-4012
46. Morik K, Brockhausen P, Joachims T. Combining statistical learning with a knowledge-based approach: a case study in intensive care monitoring. Technical Report; 1999.
47. Zell A. Simulation neuronaler netze. habilitation, Uni Stuttgart, 1994.
48. Haykin S. Neural networks: a comprehensive foundation. Upper Saddle River: Prentice Hall PTR; 1994.
49. Bruzzone L, Serpico SB. Classification of imbalanced remote-sensing data by neural networks. Pattern Recogn Lett. 1997;18(11):1323–8. https://doi.org/10.1016/S0167-8655(97)00109-8.
50. Breiman L, Friedman JH, Olshen RA, Stone CJ. Classification and regression trees. Boca Raton: Chapman & Hall/CRC; 1984.
51. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.
52. Van Hulse J, Khoshgoftaar TM, Napolitano A. Experimental perspectives on learning from imbalanced data. In: Proceedings of the 24th International Conference on Machine Learning. ICML '07. New York: ACM; 2007. p. 935–942. https://doi.org/10.1145/1273496.1273614
53. Khalilia M, Chakraborty S, Popescu M. Predicting disease risks from highly imbalanced data using random forest. BMC Med Inf Decis Mak. 2011;11(1):51.
54. Lorena AC, Garcia LPF, Lehmann J, Souto MCP, Ho TK. How complex is your classification problem? A survey on measuring classification complexity. ACM Comput Surv. 2019;52(5):1–34. https://doi.org/10.1145/3347711.
55. He H, Garcia EA. Learning from imbalanced data. IEEE Trans Knowl Data Eng. 2009;21(9):1263–84. https://doi.org/10.1109/TKDE.2008.239.
56. Rahman A, Medevedev P. Representation of k-mer sets using spectrum-preserving string sets. J Comput Biol. 2021;28(4):381–94. https://doi.org/10.1089/cmb.2020.0431.
57. Solomon B, Kingsford C. Fast search of thousands of short-read sequencing experiments. Nat Biotechnol. 2016;34(3):300–2. https://doi.org/10.1038/nbt.3442.
58. Chor B, Horn D, Goldman N, Levy Y, Massingham T, et al. Genomic DNA k-mer spectra: models and modalities. Genome Biol. 2009;10(10):108.
59. Elworth RAL, Wang Q, Kota PK, Barberan CJ, Coleman B, Balaji A, Gupta G, Baraniuk RG, Shrivastava A, Treangen TJ. To petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics. Nucleic Acids Res. 2020;48(10):5217–34. https://doi.org/10.1093/nar/gkaa265.
60. Raimondi D, Fumagalli G. A Critical Analysis of Classifier Selection in Learned Bloom Filters—Supporting Software. https://github.com/RaimondiD/LBF_ADABF_experiment. Last checked on May, 2023; 2023.
61. Dai Z. Adaptive Learned Bloom Filter (ADA-BF): Efficient Utilization of the Classifier. https://github.com/DAIZHENWEI/Ada-BF. Last checked on November 8, 2022; 2022.
62. Python Software Foundation: pickle—Python object serialization. https://docs.python.org/3/library/pickle.html. Last checked on May 17, 2022 (2022)
63. Marinò GC, Petrini A, Malchiodi D, Frasca M. Deep neural networks compression: a comparative survey and choice recommendations. Neurocomputing. 2022. https://doi.org/10.1016/j.neucom.2022.11.072.
64. Raudys S. On the problems of sample size in pattern recognition. In: Detection, pattern recognition and experiment design: Vol. 2. Proceedings of the 2nd All-union conference statistical methods in control theory (1970). Publ. House "Nauka".

Publisher's Note