A Critical Analysis of Classifier Selection in Learned Bloom Filters: the Essentials

Dario Malchiodi[1†][0000−0002−7574−697X], Davide Raimondi[1][0000−0001−8171−8302], Giacomo Fumagalli[1][0000−0002−2068−9293], Raffaele Giancarlo[2][0000−0002−6286−8871], and Marco Frasca[1][0000−0002−4170−0922]

[1] Dept. of Computer Science, University of Milan, Via Celoria 18, 20133 Milano, Italy
{malchiodi, frasca}@di.unimi.it,
{davide.raimondi2, giacomo.fumagalli1}@studenti.unimi.it
[2] Dept. of Mathematics and CS, University of Palermo, Palermo, Italy
raffaele.giancarlo@unipa.it
[†] Corresponding author

Abstract. It is well known that Bloom Filters have a performance essentially independent of the data used to query the filters themselves, but this is no more true when considering Learned Bloom Filters. In this work we analyze how the performance of such learned data structures is impacted by the classifier chosen to build the filter and by the complexity of the dataset used in the training phase. Such analysis, which has not been proposed so far in the literature, involves the key performance indicators of space efficiency, false positive rate, and reject time. By screening various implementations of Learned Bloom Filters, our experimental study highlights that only one of these implementations exhibits higher robustness to classifier performance and to noisy data, and that only two families of classifiers have desirable properties in relation to the previous performance indicators.

Keywords: Learned Bloom filters · Data complexity · Learned data structures.

1 Introduction

Recent studies have highlighted how the impact of machine learning has the potential to change the way we design and analyze data structures. Indeed, the resulting research area of Learned Data Structures has had a well documented impact on a broad and strategic domain such as that of Data Bases, and an analogous impact can be expected for Network Management [25] and Computational Biology [13]. More in general, as well argued in [14], this novel way to use machine learning has the potential to change how Data Systems are designed. The common theme to this new approach is that of training a Classifier [10] or a Regression Model [11] 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). In this work, we focus on Bloom

Filters (BFs) [1] which have also received attention in the realm of Learned Data Structures. Such an attention is quite natural, due to the fundamental nature and pervasive use of BFs. Indeed, many variants and alternatives for these filters have been already proposed, prior to the Learned versions [2].

Problem Statement, Performance of a Bloom Filter and a Learned Version. 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, we can have *false positives* (i.e., elements in $U \backslash S$ wrongly decreed as keys), albeit their fraction (termed henceforth *false positive rate*, FPR for short) should be bounded by a given ϵ. The key parameters of any data structure solving the approximate set membership problem are: (i) the FPR ϵ; (ii) the total space needed by the data structure; and (iii) the *reject time*, defined as the expected time for rejecting a non-key.

Kraska et al. [15] have proposed a Learned version of Bloom Filters (LBF) in which a suitably trained binary classifier is introduced with the aim of reducing space occupancy w.r.t. a classical BF, having fixed the FPR. Such classifier is initially queried to predict the set membership, with a fallback to a standard BF in order to avoid false negatives. Mitzenmacher [20] has provided a mathematical analysis for those filters and novel LBF variants, together with a discussion of their pros/cons. Additional models have been introduced recently [7,23].

The Central Role of Classifier Selection. Apart from an initial investigation [8,12], the problem of suitably choosing the classifier to be used to build a specific LBF has not been fully addressed so far. Moreover, 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 [18].

Paper contribution. Given the above State of the Art, our aim is to provide a methodology and the associated software to guide the design, analysis and deployment of Learned Boom Filters with respect to given constraints about their space efficiency, false positive rate, and reject time. In order to achieve these goals, our contributions are the following.

(1) **We revisit BFs** in their original and learned versions (Sect. 2), detailing the hyperparameters to be tuned within the related training procedures.
(2) **We propose a methodology**, which can guide both developers and users of LBFs in their design choices (Sect. 3), to study the interplay among: (a) the parameters indicating how a filter, learned or classic, performs on an input dataset; (b) the classifier used to build the LBF; (c) the classification complexity of the dataset.
(3) **Software platform and findings**: we provide a software platform implementing the above-mentioned methodology, along with important insights about the overall applicabity of LBF, as detailed next.

(a) We address the problem of choosing the most appropriate classifier in order to design a LBF having as only prior knowledge the total space budget, the data complexity, the list of available classifiers, and their inference time. A related problem has been considered in [20] with two important differences: the filter is fixed, and the obtained results supply only partial answers, leading to the suggestion of an experimental methodology, which has not been validated and it is not supported by software. Our experiments (Sect. 4) shows that among the many classifiers used in this research, only two classifiers are worth of attention. Remarkably, none of the two has been considered before for LBFs (Sect. 5).
(b) As a further contribution, we assess how the performance of State-of-the-Art BFs is affected by datasets of increasing complexity (Sect. 5). In particular, we identify a variant of Learned Bloom Filters more robust to variations of data complexity and classifier performance.
(c) We also provide user guidelines on how to use State-of-the-Art LBF solutions (Sect. 6).

2 Bloom Filters and Learned Bloom filters

A Bloom Filter [1] is a data structure solving the Approximate Set Membership problem defined in the Introduction, based on a boolean array $\boldsymbol{v}$ of m entries and on k hash functions $h_1, \ldots, h_k$ mapping U to $\{1, \ldots, m\}$. These functions are usually assumed to be *k-wise independent* [3,24], although much less demanding schemes work well in practice [1]. A BF is built by initializing all the entries of $\boldsymbol{v}$ to zero, subsequently considering all keys $\boldsymbol{x} \in S$ and setting $v_{h_j(\boldsymbol{x})} \leftarrow 1$ for each $j \in \{1, \ldots k\}$; a location can be set to 1 several times, but only the first change has an effect. Once the filter has been built, any $\boldsymbol{x} \in U$ is tested against membership in S by evaluating the entry $v_{h_j(\boldsymbol{x})}$, for each hash function h_j: $\boldsymbol{x}$ is classified as a key if all tested entries are equal to 1, and rejected (a shorthand for saying that it is classified as a non-key) otherwise. False positives might arise because of hash collisions, and the corresponding rate ϵ is inversely bound to the array size m. More precisely, equation (21) in [1] connects reject time, space occupancy and FPR, so that one can choose the configuration of the filter: for instance, given the available space, one can derive the reject time that minimizes the FPR. Analogous trade-offs [2,20] can be used to tune the hyperparameters of a BF (namely, m and k) in order to drive the inference process towards the most space-conscious solution. In particular, fixed an FPR ϵ and a number $n = |S|$ of keys, a BF ensuring optimal reject time requires an array of

$$m = 1.44n \log(1/\epsilon) \text{ bits.} \tag{1}$$

A Learned Bloom Filter [15] is a data structure simulating a BF to reduce its resource demand or its FPR by leveraging a classifier. The main components of a LBF are a classifier $C : U \to [0, 1]$ and a BF F, defined as it follows.

1. Using supervised machine learning techniques, C is induced from a labeled dataset D made of items $(\boldsymbol{x}, y_x)$, where y_x equals 1 if $\boldsymbol{x} \in S$ and 0 otherwise.

In other words, C is trained to classify keys in S so that the higher is $C(\boldsymbol{x})$, the more likely $\boldsymbol{x} \in S$. A binary prediction is ensured by thresholding using $\tau \in [0, 1]$, i.e. classifying $\boldsymbol{x} \in U$ as a key if and only if $C(\boldsymbol{x}) > \tau$.
2. Of course, nothing prevents us from having a set of false negatives $\{\boldsymbol{x} \in S \,|\, C(\boldsymbol{x}) \leq \tau\} \neq \emptyset$, thus a *backup* (classical) Bloom Filter F for this set is built. Summing up, $\boldsymbol{x} \in U$ is predicted to be a key if $C(\boldsymbol{x}) > \tau$, or $C(\boldsymbol{x}) \leq \tau$ and F does not reject $\boldsymbol{x}$. In all other cases, $\boldsymbol{x}$ is rejected.

It is important to underline that the FPR of a classical BF is essentially independent of the distribution of data used to query it. This is no more true for a LBF [20], in which such rate should be estimated from a query set $\overline{S} \subset U \backslash S$. To remark such difference, one commonly refers to the *empirical FPR* of a learned filter, which is computed as $\epsilon = \epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F$, where:

1. $\epsilon_\tau = |\{\boldsymbol{x} \in \overline{S} \,|\, C(\boldsymbol{x}) > \tau\}|/|\overline{S}|$ is the analogous empirical FPR of the classifier C on $\overline{S}$, and
2. ϵ_F is the false positive rate of the backup BF.

Hence, having fixed a target value for ϵ, the backup filter can be built setting $\epsilon_F = (\epsilon - \epsilon_\tau)/(1 - \epsilon_\tau)$, under the obvious constraint $\epsilon_\tau < \epsilon$. Within the learned setting, the three key factors of the filter are connected (and influenced) by the choice of τ. However, due to the dependency on the query set distribution, reliably estimating the FPR of a LBFs is no longer immediate, as pointed out in [20], that also suggests an experimental methodology to assess it. The latter is part of the evaluation setting proposed in this paper.

Here below we outline the main features of the LBF variants which we have considered. With the exception of the one in [23], for which the software is neither public nor available from the authors, our selection is State of the Art.

Sandwiched LBFs [20]. The Sandwiched variant of LBFs (SLBF for brevity) is based on the idea that space efficiency can be optimized by filtering out non-keys *before* querying the classifier C, requiring as consequence a smaller backup filter F. More in detail, a BF I for S is initially built and used as a first processing step. All the elements of S that are not rejected by I are then used to build a LBF as described earlier. The SLBF immediately rejects an element $\boldsymbol{x} \in U$ if I rejects it, otherwise the answer for $\boldsymbol{x}$ of the subsequent LBF is returned. The empirical FPR of the SLBF is $\epsilon = \epsilon_I\big(\epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F\big)$, where ϵ_I is the FPR of I. Here, fixed the desired ϵ, the corresponding FPR to consctruct I is $\epsilon_I = (\epsilon/\epsilon_\tau)(1 - \mathrm{FN}/n)$, where FN is the number of false negatives of C. Also in this case, the classifier accuracy affects the FPR, space and reject time, with the constraint $\epsilon(1 - \mathrm{FN}/n) \leq \epsilon_\tau \leq 1 - \mathrm{FN}/n$.

Adaptive LBFs [7]. Adaptive LBFs (ADA-BF) represent an extension of LBF, partioning the training instances $\boldsymbol{x}$ into into g groups, according to their classification score $C(\boldsymbol{x})$. Then, the same number of hash functions the backup filter of an LBF would use are partitioned across groups, and the membership for the instances belonging to a given group is tested only using the hash functions

assigned to it. Even for ADA-BF the expected FPR can only be estimated empirically, but in this case the formula is rather complicated: the interested reader can refer to [7]. We retained here the best performing variant of ADA-BF.

Hyperparameters. The Learned Bloom Filters described above have some parameters to be tuned. Namely, the threshold τ for LBF and SLBF, and two parameters g and $\bar{c}$ for ADA-BF, representing the number of groups in which the classifier score interval is divided into, and the proportion of non-keys scores falling in two consecutive groups. The details on the tuning of these hyperparameters are discussed in Sect. 4.2.

3 Experimental Methodology

In this section we present the methodology which we adopt in order to design and analyse LBFs with regard to the inherent complexity of input data to be classified, subsumed as follows. The starting point is a dataset, either real-world or generated through a procedure suitable for synthesize data in function of some classification complexity metrics. Overall, the pipeline adopted is the following: collect/generate data; induce a classifier from data and estimate its empirical FPR; construct a Learned Bloom Filter exploiting the learnt classifier, and in turn estimate its empirical FPR. The following sections review the considered classifier families and describe in depth the adopted data generation procedure.

3.1 A Representative Set of Binary Classifiers

Starting from an initial list of classifiers—without presuming to be exhaustive—we performed a set of preliminary experiments, from which we received indications about the families of classifiers to be further analyzed, based on their time performance/space requirements trade-off[3]. Namely, from the initial list, we have removed the following classifiers: Logistic Regression [5], Naive Bayes [9] and Recurrent Neural Networks [4], due to their poor trade-off performance, confirming the results of a preliminary study [12]. The remaining ones are briefly described in the following paragraphs. Since our evaluation considers both balanced and unbalanced classification problems, we also detail how their inference is managed in an unbalanced context. The hyperparameters of the corresponding learning algorithms are distinguished between *regular* and *key* hyperparameters, the latter affecting the space occupancy of the classifier. The model selection phase only involves non-key hyperparameters, while different configurations for key hyperparameters are analysed in dedicated experiments aiming at studying the interplay among FPR, space occupancy and reject time of Learned Bloom Filters.

[3] The experiments and data about this preliminary part are available upon request.

3.2 Measures of Classification Complexity and a Related Data Generation Procedure

In order to evaluate dataset complexity, several measures are available (see [16] for a survey). We specifically focus on measures suitable for binary classification tasks, and hereafter we use the notation "class i", $i = 1, 2$, to refer to one of the two classes. A preliminary analysis highlighted that some of the measures in [16] were insensitive across a variety of synthetic data, as happened, e.g., with the *F1*, *T2*, or *T3* measures, or needed an excessive amount of RAM (such as network- or neighborhood-based measures, like *LSC* and *N1*). As a consequence, we selected the *feature-based* measure *F1v* and the *class-imbalance* measure *C2*, Both measures are in $\in [0, 1]$, and the higher the value, the higher the complexity.

Data generation procedure. In order to generate a binary classification dataset of size N 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 $D = \{\boldsymbol{x}_1, \dots \boldsymbol{x}_N\} \subset \mathbb{R}^q$ be the set of samples, with each sample $\boldsymbol{x}_i$ having q features $x_{i1}, \dots, x_{iq}$, 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 $\boldsymbol{\Sigma} = \gamma \boldsymbol{I}_q$ (where $\gamma > 0$ and $\boldsymbol{I}_q$ denotes 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 $x_2 - a x_1^2 = 0$ is considered, with $a > 0$: a point $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ is positive ($y_i = 1$) if $x_{i2} - a x_{i1}^2 > 0$, 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, and ρ, instead, controls the data imbalance. Further, to vary the data complexity even for nonlinear classifiers, labels are permuted with different levels of noise: we flip the label of a fraction r of positive samples, selected uniformly and at random, with an equal number on randomly selected negatives.

4 Experiments

4.1 Data

Domain-Specific Data. We use a URL dataset and a DNA dictionary. The first has been already studied as a benchmark for Learned Bloom Filters [7], and the authors of this research kindly provided us the dataset. It contains 485730 URLs described by 17 lexical features: 80002 URLs are *malicious* (and they constitute our key set), while the remaining ones are *benign*. The DNA dictionary regards 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) [21], and was directly generated by us. More precisely, it refers to the human chromosome 14, containing $n = 49906253$ 14-mers [21] 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 $\{A, T, C, G\}$. We point out that no sensible information is contained in these datasets. We study them because they represent two extreme cases of classification complexity: the URL dataset is *easy* (*F1v*=0.08172, *C2*=0.62040), the DNA data is *hard* (*F1v*=0.99972, *C2*=0).

Synthetic Data. Using the technique described in Sect. 3.2 we 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 number of experiments planned. Indeed, both balanced and unbalanced categories contain nine datasets, exhibiting increasing levels of *F1v* complexity (see Table 1). 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 estimations are consistent, as *F1v* increases with a and r and *C2* reflects the complexities of the URL and DNA datasets respectively in the unbalanced and balanced case.

Table 1. *F1v* complexity measure of the synthetic data. The *C2* index is equal to 0.0 and 0.615, respectively, in the balanced and unbalanced case.

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

4.2 Model Selection

Classifiers. The classifier generalization ability is assessed independently of the filter employing it, via a 3-fold cross validation (CV) procedure; performance is measured in terms of the area under (i) the ROC curve (AUC), and of (ii) the precision-recall curve (AUPRC), averaged across folds. However, for synthetic data, we report only the AUPRC results, since AUC results showed very similar trends and no enough room is available. The key hyperparameters for a given classifier are set in order not to exhaust all available space budget, while non-key hyperparameters are selected through a grid search using an inner 3-fold CV on the current training set, retaining the best configuration. When possible, for each classifier, we selected different hyperparameter configurations yielding models of different complexity, from simpler to more complex ones. Finally, for a fair comparison, the key hyperparameters for NNs are selected so as to yield three models nearly having the same size of the SVM and of the two RFs models.

Table 2. Space budget in Mbits adopted on the various datasets. ϵ is the false positive rate, n is the number of keys in the dataset.

Data	$\boldsymbol{\epsilon}$	**Budget (Mbits)**	$\boldsymbol{n}$
Synth	$0.05, 0.01$	$0.62,\ 0.96$	10^5
URL	$0.01, 0.005, 0.001, 0.0005, 0.0001$	$0.76, 0.88, 1.15, 1.26, 1.53$	$8 \cdot 10^4$
DNA	$0.01, 0.005, 0.001,\ 0.0005,\ 0.0001$	$477.46, 549.32, 716.19,\ 788.06,\ 954.92$	$4.99 \cdot 10^7$

Summarizing: i) SVM has no key hyperparameters; ii) for RF, we used $t = 10, 20$ (URL, synthetic) and $t = 10, 100$ (DNA); iii) for NNs, we considered 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).

Learned Bloom Filters. The Bloom Filter variants under study are evaluated under the setting proposed in [7], that is: 1) train the classifiers on all keys and 30% of non-keys, and query the filter using remaining 70% of non-keys to compute the empirical FPR; 2) fix an overall memory budget of m bits for each filter, and compare them in terms of their empirical FPR ϵ. Each filter variant is trained leveraging in turn each of the considered classifiers. The budget m is related to the desired (expected) ϵ of a classical Bloom Filter, according to (1). Being the space budget directly influenced by the key set size n, we adopt a setting tailored on each dataset. Concerning synthetic data, as we generate numerous datasets, for each of them we only test two different choices for the space budget m: namely, those yielding $\epsilon \in \{0.05, 0.01\}$ for the classical Bloom Filter. On real datasets, we test five space budgets corresponding to $\epsilon \in \{0.01, 0.005, 0.001, 0.0005, 0.0001\}$. Table 2 contains the resulting budget configurations for all the considered datasets. To build the learned Bloom Filters variants, the hyperparameters have been selected via grid search on the training data, optimizing with respect to the FPR, according to the following setting: (a) 15 different values for threshold τ, and (b) the ranges $[3, 15]$, and $[1, 5]$ for hyperparameters g and $\bar{c}$, respectively (cfr. Sect. 2). Importantly, the latter choice includes and extends the configurations suggested in the original paper [6], namely, $[8, 12]$ for g and $[1.6, 2.5]$ for $\bar{c}$.

5 Results and Discussion

As evident from Sect. 2, the classifier can be interpreted as an oracle for a learned BF, where the better the oracle, the better the related filter, i.e., its FPR once fixed the space budget. Accordingly, it is of interest to evaluate the performance of classifiers. All classifiers described in Sect. 3.1 have been tested on the datasets described in Sect. 4.1, with the configuration described in Sect. 4.2. Figure 1 depicts the performance of classifiers on synthetic and real data. However, it is central here to 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 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. Surprisingly, from the behaviour of classifiers it emerges a crisp separation of the data in terms of complexity (cfr. Table 1), that corresponds to datasets *easy* to classify (roughly $F1v \leq 0.35$) and *hard* ($F1v > 0.35$) to classify. 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 (cfr. Fig. 1), 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 (cfr. Table 2), and it is thereby natural to retain in those cases only the smallest/simplest variants, namely: RF-10 and NN-25 (synthetic) and NN-7 (URL), in addition to SVM. Conversely, in DNA experiments, more complex classifiers substantially outperform the simpler counterparts, coherently with the fact that this classification problem is much harder. Since the available space budget is higher in this case, all classifiers have been retained in the subsequent filter evaluation.

Learned Filters Performance w.r.t. Data Classification Complexity

Easy datasets. Figure 2 reports the FPR results of learned Bloom Filters on balanced and unbalanced synthetic data, respectively, whereas Fig. 3 depicts the results on URL and DNA data. According to the definition provided above ($F1v$ around 0.35 or smaller), easy data can be associated to the three/four leftmost configurations on the x-axis in Fig. 2 of synthetic and to URL data. In these cases, we observe results coherent with the literature, where ADA-BF slightly outperforms the other competitors [7], and when using RF-10 as classifier lower FPRs are obtained with regard to the classical BF. Notwithstanding, it clearly emerges that such a classifier is not the best choice, underlining all the doubts about a selection not motivated in the original studies. For instance, on URL data there are at least two classifiers yielding a lower FPR in most cases and

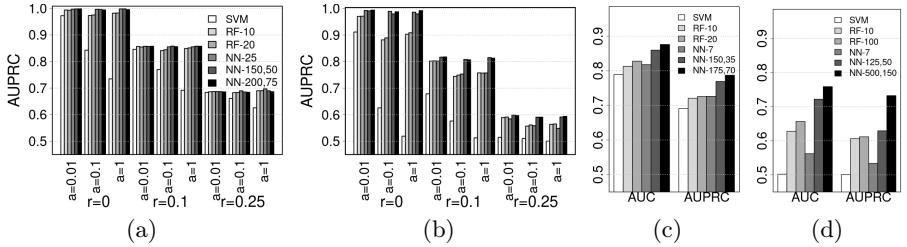

Fig. 1. Performance averaged across folds of compared classifiers on synthetic (a-balanced, b-unbalanced), URL (c) and DNA data (d). On synthetic data, bars are grouped by dataset, in turn denoted by a couple (a, r).

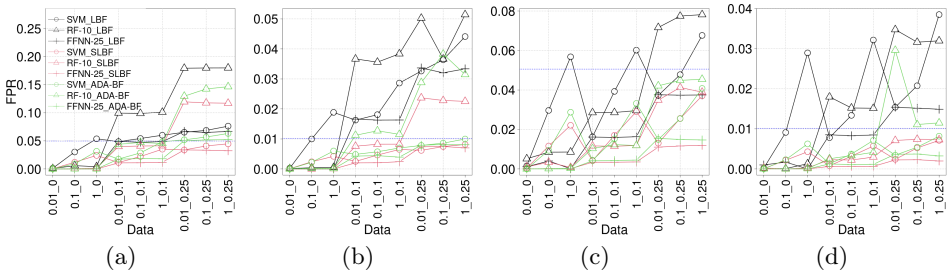

Fig. 2. False positive rates of learned filters attained on balanced (a, b) and unbalanced (c, d) synthetic datasets. On the horizontal axis, labels X_Y denote the dataset obtained when using $a = X$ and $r = Y$. The blue dotted line corresponds to the empirical false positive rate of the classical BF in that setting. Two space budgets m are tested, ensuring that $\epsilon = 0.05$ ((a), (c)) and $\epsilon = 0.01$ ((b), (d)) for the classical BF.

for all filter variants (SVM and NN-7). In additon, SVMs are much faster. NN-7 (or NN-25 for synthetic data) remains the best choice even when the separation boundary becomes less linear ($a > 0.01$), and filters induced by SVMs become less effective or even worse than the baseline BF.

Hard datasets. Our experiments show a novel scenario with the increase of data complexity, i.e., when moving towards right on the horizontal axis in Fig. 2, or when considering DNA data. We observe that the performance of the filters drops more and more, in line with the performance decay of the corresponding classifiers, and unexpectedly the drop is faster in ADA-BF (and LBF) w.r.t. SLBF. This happens for instance on all synthetic data having $r > 0$ (noise injection). We say unexpectedly since we have an inversion of the trend also reported in the literature, where usually ADA-BF outperforms SLBF (which in turn improves LBF). Indeed, SLBFs here exhibit behaviours more robust to noise, which are likely due to a reduced dependency on the classifier for SLBF, yielded by the usage of the initial Bloom Filter. Such a filter allows the classifier to be queried only on a subset of the data. Noteworthy is the behavior of filters when using RFs in this setting: their FPR strongly increases, and potential explanations are the excessive score discretization (having 10 trees we have only 11 distinct scores for all queries), and the space occupancy is larger (limiting the space that can be assigned to initial/backup filters). These results find a particularly relevant confirmation on the very hard, real-world, large, and novel DNA dataset. Here, surprisingly, the LBF cannot attain any improvement with regard to the baseline BF, differently from SLBF and ADA-BF. A potential cause can reside in the worse performance achieved by classifiers on this hard dataset, compared to those obtained on synthetic and URL data, and in a too marked dependency of LBF on the classifier performance, 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). SLBF outperforms both LBF and baseline of one order of magnitude in FPR with the same space amount, and ADA-BF when using

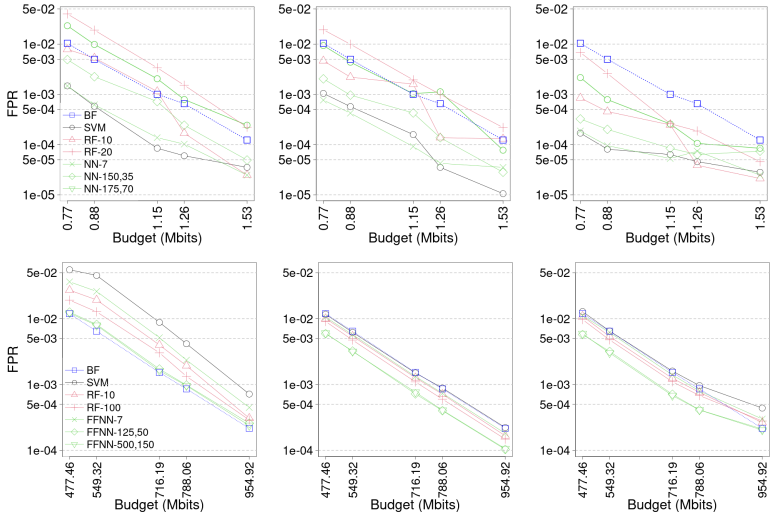

Fig. 3. Empirical false positive rate of LBF (left), SLFB (central), and ADA-BF (right) filters on URL data (first row) and DNA data (second row). On the horizontal axis the different budgets configurations. Dotted blue line represents the baseline Bloom Filter.

weaker classifiers and when a higher budget is available. This is likely due to overfitting of ADA-BF in partitioning the classifier codomain when the classifier performance is not excellent (or similarly when the data complexity is high), as with DNA data. Differently from hard synthetic data, where the key set was smaller (as consequently was the space budget), here the classifiers leading to the best FPR are the most complex, in particular NN-500,150 and NN-125,50 (which are also the top performing ones—they could be even further compressed [19]). In other words, on hard datasets simple classifiers are useless or even deleterious (SVM never improve the baseline, and in some cases they are even worse).

Reject time. The results concerning reject time analysis are provided in full in [17]. Here we bring to light that learned BF are sometimes faster than the baseline, which in principle is not expected, since they have to query a classifier in addition to a classical BF. Our interpretation is that it can happen for two main reasons: 1) the classifier is very fast and effective, allowing in most cases to skip querying the backup filter; 2) the key set is very large and it requires a large baseline BF, whereas a good classifier sensibly drops the dimension of backup filters, making their invocation much faster. See for instance the case of DNA data, where most learned filters are faster than the baseline, with most classifiers.

6 Guidelines

We summarize here our findings about the configuration of learned Bloom Filters exploiting the prior knowledge of data complexity and available space budget.

Dataset Complexity and Classifier Choice. We have roughly distinguished two main categories of data based on their complexity and the related behaviour of filters: easy dataset, having $F1v \leq 0.35$, and hard dataset, having $F1v > 0.35$. The dataset complexity emerged as a central discriminant for the filter setup. Indeed, when dealing with easy datasets, the choice of the classifier becomes quite easy, and, independently of the space budget, it is always more convenient to select simple classifiers. Specifically, our experiments designate linear SVMs as best choice for the easiest datasets (those having almost linear separation boundary), and the smallest/simplest NNs for the more complex data in this category. In addition, the classifier inference time plays an important role for this data category: although a fastest classifier does not necessarily implies a lower reject time of the corresponding filter (see [17]), when the average performance of two classifiers is close, then the inference time can be a discriminant feature for the classifier selection. But only in this case: see for instance the URL results, where the RF-10 performed just slightly better than SVMs, but although having an inference time one order of magnitude higher, the induced LBF has a lower reject time (see [17] for the relative discussion.) Surprisingly, this analysis has been overlooked in the literature. For instance, benchmark URL data falls in this category, but all previous experimental studies regarding learned BF on this data do not consider neither SVMs, nor NNs.

For hard datasets instead, the space budget is central for the classifier choice. Indeed, within the budget given by (1), on synthetic datasets, having a relative small key set and accordingly a lower budget, the choice is almost forced towards small although inaccurate classifiers, being the larger ones too demanding for the available budget. In particular, SVM is to be excluded due to the increased difficulty w.r.t. that of URL data, and for the remaining classifiers, we note that they behave very similarly (Fig. 1). Thus the most succinct ones, namely the smallest NN, are to be preferred. As opposite, when the space budget increases, as it happens for DNA data, our findings suggest to learn more accurate classifiers, even if this requires the usage of a considerable budget fraction. Indeed, the gain induced by higher classification abilities allows to save space when constructing the backup filter, to have consequently a smaller reject time, as well as an overall more efficient structure (cfr. Sect. 5). This is also motivated by the fact that to accurately train complex classifiers the sample size must be large enough [22].

Learned Bloom Filters Choice Our experiments reveal three main trends: 1) on benchmark data, that is those used also in the literature so far (URL data), ADA-BF is confirmed as more effective variant in terms of FPR, having fixed the budget; 2) however, its reject time is always and largely the highest one (see [17]), thus suggesting to exclude its usage in applications where fast responses are necessary (e.g., real-time applications). This subject includes also the classifier choice, since the most effective filters are in most cases induced by NNs, but they are also slower in terms of reject time of the counterparts induced by faster classifiers, and accordingly a trade-off FPR/reject time must be carefully investigated; 3) SLBF is the filter most robust to data noise, and the only one able to benefit more even from classifier with poor performance (cfr. synthetic

noisy and DNA results). As a consequence, SBLF is clearly the filter to choose in presence of very complex data. In particular, the point 3) is a new and quite unforeseen behaviour, emerged only thanks to the study of data complexity and relative noise injection procedure designed in this study, and which to some extent changes the ranking of most effective learned BF in practice, since most real datasets are typically characterized by noise.

7 Conclusions and Future Developments

We have proposed an experimental methodology that can guide in the design and validation of learned Bloom Filters. The key point is to base the choice of the used classifier on the space budget of the entire data structure as well as the classification complexity of the input dataset. We empirically detected two class of problems, *easy* and *hard*, and confirmed (to some extent) the results on the former one, which is the only scenario considered so far in the Literature, while the unexplored *hard* scenario revealed novel trends, and almost inverted the ranking of LBFs emerged in the easy case. 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. In addition, Learned Bloom Filters can be quite sensitive to the input query distribution. Yet, no study is available to quantify this aspect, and our methodology can be easily extended in this sense.

Acknowledgements This work has been supported by the Italian MUR PRIN project 2017WR7SHH "Multicriteria data structures and algorithms: from compressed to learned indexes, and beyond". Additional support to R.G. has been granted by Project INdAM - GNCS "Analysis and Processing of Big Data based on Graph Models".

References

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (Jul 1970)
2. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. In: Internet Mathematics. vol. 1, pp. 636–646 (2002)
3. Carter, J., Wegman, M.N.: Universal classes of hash functions. Journal of Computer and System Sciences **18**(2), 143–154 (1979)
4. 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. pp. 103–111. Association for Computational Linguistics, Doha, Qatar (Oct 2014)
5. Cox, D.R.: The regression analysis of binary sequences. Journal of the Royal Statistical Society: Series B (Methodological) **20**(2), 215–232 (1958)
6. Dai, Z.: Adaptive learned bloom filter (ada-bf): Efficient utilization of the classifier. `https://github.com/DAIZHENWEI/Ada-BF` (2022), last checked on Nov. 8, 2022

7. 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, pp. 11700–11710. Curran Associates, Inc. (2020)
8. Dai, Z., Shrivastava, A., Reviriego, P., Hernández, J.A.: Optimizing learned bloom filters: How much should be learned? IEEE Embedded Systems Letters **14**(3), 123–126 (2022). https://doi.org/10.1109/LES.2022.3156019
9. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. John Willey & Sons, New York (1973)
10. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley (2000)
11. Freedman, D.: Statistical Models : Theory and Practice. Cambridge University Press (2005)
12. 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). pp. 675–682 (2022)
13. Kirsche, M., Das, A., Schatz, M.C.: Sapling: accelerating suffix array queries with learned data models. Bioinformatics **37**(6), 744–749 (10 2020)
14. Kraska, T.: Towards instance-optimized data systems. Proc. VLDB Endow. **14**(12), 3222–3232 (oct 2021)
15. Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N.: The case for learned index structures. In: Proc. of the 2018 Int. Conf. on Management of Data. pp. 489–504. SIGMOD '18, Association for Computing Machinery, New York, NY, USA (2018)
16. Lorena, A.C., Garcia, L.P.F., Lehmann, J., Souto, M.C.P., Ho, T.K.: How complex is your classification problem? a survey on measuring classification complexity. ACM Comput. Surv. **52**(5) (sep 2019)
17. Malchiodi, D., Raimondi, D., Fumagalli, G., Giancarlo, R., Frasca, M.: A critical analysis of classifier selection in learned bloom filters (2022). https://doi.org/10.48550/ARXIV.2211.15565, https://arxiv.org/abs/2211.15565
18. Maltry, M., Dittrich, J.: A critical analysis of recursive model indexes. CoRR **abs/2106.16166** (2021), https://arxiv.org/abs/2106.16166
19. Marinò, G.C., Petrini, A., Malchiodi, D., Frasca, M.: Deep neural networks compression: a comparative survey and choice recommendations. Neurocomputing **520**, 152–170 (2023)
20. Mitzenmacher, M.: A model for learned bloom filters and optimizing by sandwiching. In: Advances in Neural Information Processing Systems. vol. 31 (2018)
21. Rahman, A., Medevedev, P.: Representation of k-mer sets using spectrum-preserving string sets. Journal of Computational Biology **28**(4), 381–394 (2021)
22. 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. Publ. House" Nauka" (1970)
23. Vaidya, K., Knorr, E., Kraska, T., Mitzenmacher, M.: Partitioned learned bloom filters. In: International Conference on Learning Representations (2021)
24. Wegman, M.N., Carter, J.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences **22**(3), 265–279 (1981)
25. Wu, Q., Wang, Q., Zhang, M., Zheng, R., Zhu, J., Hu, J.: Learned bloom-filter for the efficient name lookup in information-centric networking. Journal of Network and Computer Applications **186**, 103077 (04 2021)