On the Choice of General Purpose Classifiers in Learned Bloom Filters: An Initial Analysis Within Basic Filters

Giacomo Fumagalli[1][a], Davide Raimondi[1][b], Raffaele Giancarlo[2][c], Dario Malchiodi[1,3][d] and Marco Frasca[1,3][e]

[1]*Dipartimento di Informatica, Università degli Studi di Milano, 20133 Milan, Italy*

[2]*Dipartimento di Matematica ed Informatica, Università di Palermo, 90133 Palermo, Italy*

[3]*CINI National Laboratory in Artificial Intelligence and Intelligent Systems (AIIS), Università di Roma, 00185 Roma, Italy*

Keywords: Learned Bloom Filters, Learned Data Structures, Information Retrieval, Classification.

Abstract: Bloom Filters are a fundamental and pervasive data structure. Within the growing area of Learned Data Structures, several Learned versions of Bloom Filters have been considered, yielding advantages over classic Filters. Each of them uses a classifier, which is the Learned part of the data structure. Although it has a central role in those new filters, and its space footprint as well as classification time may affect the performance of the Learned Filter, no systematic study of which specific classifier to use in which circumstances is available. We report progress in this area here, providing also initial guidelines on which classifier to choose among five classic classification paradigms.

1 INTRODUCTION

Learned Data Structures is a novel area at the cross-road of Classic Data Structures and Machine Learning. They have been initially proposed by Kraska et. al. (Kraska et al., 2018) and have had a rapid growth (Ferragina and Vinciguerra, 2020a). Moreover, now the area has been extended to include also Learned Algorithms (Mitzenmacher and Vassilvitskii, 2020). The theme common to those new approaches to Data Structures Design and Engineering is that a query to a data structure is either intermixed with or preceded by a query to a Classifier (Duda et al., 2000) or a Regression Model (Freedman, 2005), those two being the learned part of the data structure. To date, Learned Indexes have been the most studied, e.g., (Amato et al., 2021; Ferragina and Vinciguerra, 2020b; Ferragina et al., 2021; Kipf et al., 2020; Kraska et al., 2018; Maltry and Dittrich, 2021; Marcus et al., 2020a; Marcus et al., 2020b). Rank/select data structures have also received some attention (Boffa et al., 2021).

Bloom Filters (Bloom, 1970), which are the object of this study, have also been considered, as we detail next. Historically, they have been designed to be a data structure able to solve the Approximate Membership Problem in small space (see Section 2). Due to their fundamental nature and pervasive use, many variants and alternatives have been proposed. A good review in the domain of Internet applications is provided in (Broder and Mitzenmacher, 2002).

Kraska et al. (Kraska et al., 2018) have proposed a learned version of such a filter, in which a query to the data structure is preceded by a query to a suitably trained binary classifier. The intent is to reduce space and reject time. Mitzenmacher (Mitzenmacher, 2018) has provided a model for those filters, together with a very informative mathematical analysis of their pros/cons and even an alternative proposal. Additional Learned Bloom Filters have been proposed recently (Dai and Shrivastava, 2020; Vaidya et al., 2021). It is worth pointing out that, although they differ in architecture, each of those proposals has a classifier as its central part.

Somewhat puzzling is that, although the classifier is the novel part of this new family of Filters and it is accounted for in theoretical studies (Mitzenmacher, 2018), not much attention is given to which classifier

[a] https://orcid.org/0000-0002-2068-9293
[b] https://orcid.org/0000-0001-8171-8302
[c] https://orcid.org/0000-0002-6286-8871
[d] https://orcid.org/0000-0002-7574-697X
[e] https://orcid.org/0000-0002-4170-0922

Fumagalli, G., Raimondi, D., Giancarlo, R., Malchiodi, D. and Frasca, M.
On the Choice of General Purpose Classifiers in Learned Bloom Filters: An Initial Analysis Within Basic Filters.
DOI: 10.5220/0010889000003122
In *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2022)*, pages 675-682
ISBN: 978-989-758-549-4; ISSN: 2184-4313

to use in practical settings. Kraska et al. use a Neural Network, while Dai and Shrivastava and Vaidya et al. use Random Forests. Those choices are only informally motivated, giving no evidence of superiority with respect to other possible ones, via a comparative analysis. Therefore, the important problem of how to choose a specific classifier in conjunction with a specific filter has not been addressed so far, even in one application domain.

Such a State of the Art is problematic, both methodologically and practically. Here we propose the first, although initial, study that considers the choice of the classifier within Learned Filters. Since our study is of a fundamental nature, we consider only basic versions of the Filters and six generic, if not textbook, classifier paradigms. The intent is to characterize how different classifiers affect different performance parameters of the Learned Filter. The application domain is the one of malicious URL determination, as in previous work. As per literature standard, our datasets are real and fall within the data size used by Dai and Shrivastava. However, even so, Random Forest classifiers do not seem to be competitive in our setting, needing bigger datasets. Indeed, for the continuation of this work, we plan to consider much larger datasets and include also Random Forests. Additional application domains will also be considered.

Our results provide a confirmation of some expected behaviours, like that more complex classifiers tend to yield larger space reduction when used within a learned Bloom Filter, but also some counterintuitive perspectives, including that simple linear classifiers might represent very competitive alternatives to their bigger counterparts, sometimes even better. These results make more sense if we consider that in a learned Bloom Filter classifiers come up with one or two classic Bloom Filters, calibrated in turn on the classifier performance, and accordingly it is their synergy that determines the overall space reduction, not just the classifier itself. Finally, a relevant note is also that in our experiments linear classifiers allow two orders of magnitude faster reject times.

2 APPROXIMATE SET MEMBERSHIP PROBLEM: DEFINITIONS AND KEY PARAMETERS

Definition. Given $S \subset U$ and $x \in U$, where U is the universe of keys, the *set membership* problem consists in finding a data structure able to determine if $x \in S$. The *approximate set membership* instead allows for a fraction ε of *false positives*, that is elements in $U \setminus S$ considered as members of S. No *false negatives* are allowed. That is, elements in S considered as non-members.

A Paradigmatic Example. Assume that a set of URLs is given. They can be *malicious*, i.e., websites hosting unsolicited content (spam, phishing, drive-by downloads, etc.), or luring unsuspecting users to become victims of scams (monetary loss, theft of private information, and malware installation), or otherwise labeled as *benign*. The aim is to design a time efficient data structure that takes small space and that "rejects" benign URL quickly, as they do not belong to the malicious set. On occasions, we may have false positives.

Key Parameters of the Filter. The reject time, taken as the expected time to reject a non-member of the set; the space taken by the data structure, since the cardinality of S may be very large; the false positive rate ε. Quite remarkably, Bloom provided two data structures to solve the posed problem, linking the three key parameters in trade-off bounds. The most space-efficient of those data structures goes under the name of Bloom Filter and it is outlined next, together with two of its learned versions.

2.1 Basic Classic and Learned Bloom Filters

Classic. Letting $|S| = n$, a Bloom Filter (BF) is made up by a bit array $\boldsymbol{v}$ of size m, whose elements are all initialized to 0. It uses k hash functions $h_j : U \longrightarrow \{0,1,\ldots,m-1\}$, $j \in \{1,\ldots,k\}$, which can be assumed to be perfect in theoretic studies (see (Mitzenmacher, 2018)). When a new element x of S is added, it is coded using the hash functions: each bit in $\boldsymbol{v}$ in position $h_j(x)$ is set to 1, for each $j \in \{1,\ldots,k\}$. To test if a key $x \in U$ is a member of S, $h_j(x)$ for each j is computed, and x will be rejected if there exists $j \in \{1,\ldots,k\}$ such that $v[h_j(x)] = 0$. However, when the Bloom Filter considers x as a member, it might be a false positive due to the hash collisions. The false positive rate ε is inversely related to the space usage of the bit array. More accurately, the trade-off formula connecting the key parameters of the Filter is given in equation (21) in (Bloom, 1970). Analogous trade-off formulas are also known, e.g., (Broder and Mitzenmacher, 2002; Mitzenmacher, 2018). In applications (Broder and Mitzenmacher, 2002), one usually asks for the the most space-conscious filter, given a false positive rate, being the reject time a consequence of those choices.

Learned: One Classifier and One Filter. The intent of this data structure, named Learned Bloom Filter (LBF), is to achieve a given false positive rate, as in a classic filter, but in less space, and possibly with little loss in reject time (Kraska et al., 2018). One can proceed as follows. A classifier $C : U \longrightarrow [0,1]$ is trained on a labeled dataset (x, y_x), where $x \in D \subset U$, $S \subset D$, and $y_x = 1$ when $x \in S$, $y_x = -1$ otherwise. The positive class is thereby the class of keys S. The larger $C(x)$, the more likely x belongs to S. Then, in order to have a binary prediction, a threshold $\tau \in (0,1)$ is to be fixed, yielding positive predictions for any $x \in U$ such that $C(x) \geq \tau$, negative otherwise. To avoid false negatives in the learned Bloom Filter, that is elements $x \in S$ rejected by the filter, a classic "backup" Bloom Filter F is created on the set $\{x \in S | C(x) < \tau\}$. A generic key x is tested against membership in S by computing the corresponding prediction $C(x)$: x is considered as an element of S if $C(x) \geq \tau$ or $C(x) < \tau$ and F does not reject x. x is rejected otherwise. Unlike a classic Bloom Filter, the false positive rate of a LBF depends on the distribution of a given query set $\bar{S} \in U \setminus S$ (Mitzenmacher, 2018). Hereafter, we will refer to $\epsilon_\tau = |\{x \in \bar{S} | (C(x) \geq \tau)\}|$ as the *empirical* false positive rate of the classifier on $\bar{S}$, to ϵ_F as the false positive rate of F, and to the *empirical* false positive rate of the LBF on $\bar{S}$ as $\epsilon = \epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F$. Fixed a desired ϵ, the optimal value of ϵ_F is

$$\epsilon_F = \frac{\epsilon - \epsilon_\tau}{1 - \epsilon_\tau},$$

which yields the constraint $\epsilon_\tau < \epsilon$.

It is to be pointed out that now, false positive rate ϵ, space and reject time of the entire filter are intimately connected and influenced by the choice of τ. Another delicate point that emerges from the analysis offered in (Mitzenmacher, 2018) is that, while the false positive rate of a classic Filter can be reliably estimated experimentally because of its data independence, for the Learned version this is no longer so immediate and further insights are needed. As an additional contribution, an experimental methodology is suggested in (Mitzenmacher, 2018) and we adhere to it here.

Learned: Sandwiched Classifier. The intent of this variant, named Sandwiched Learned Bloom Filter (SLBF), is to filter out most non-keys before they are supplied to the LBF. This would allow the construction of a much smaller backup filter F and an overall more compact data structure (Mitzenmacher, 2018). The specifics follow. A Bloom Filter I for the set S precedes the structure of the Learned Bloom Filter described in the previous paragraph. The subsequent LBF is constructed on the elements of S not rejected by I. Clearly, I might yield a considerable number of false positives, when a limited budget of bits is dedicated to it. A query $x \in U$ is rejected by the SLBF if I rejects it, otherwise the results of the subsequent LBF is returned. Denoted by ϵ_I the false positive rate of the filter I, the empirical false positive rate of a SLBF is $\epsilon = \epsilon_I \big(\epsilon_\tau + (1 - \epsilon_\tau)\epsilon_F\big)$. For a desired value of ϵ, the following properties hold (Mitzenmacher, 2018):

- $\epsilon_I = \frac{\epsilon}{\epsilon_\tau}\left(1 - \frac{FN}{n}\right)$, with FN number of false negatives of C;
- $\epsilon\left(1 - \frac{FN}{n}\right) \leq \epsilon_\tau \leq 1 - \frac{FN}{n}$.

As for the previous learned Bloom Filter variant, the classifier accuracy affects all the key factors of a SLBF, namely false positive rate, space and reject time. Considerations and experimental methodologies in this case are the same as in the previous case.

3 EXPERIMENTAL METHODOLOGY

3.1 General Purpose Classifiers

The classifiers used in our analysis are briefly described below. We assume each $x \in U$ is represented through a set of n real-valued attributes $A = \{A_1, \ldots, A_n\}$, and that $Y = \{-1, 1\}$ is the set of labels, with -1 denoting the negative class. The training set is made up by labeled instances $D \subset U$, in the form of couples (x, y_x), with $x \in D$ and $y_x \in Y$.

- *RNN-k*. A character-level recurrent neural network with Gated Recurrent Units (GRU) (Cho et al., 2014), having 10-dimensional embedding and k-dimensional GRU. RNN is included as baseline from literature for the same problem (Kraska et al., 2018). The parameters to be learned are the connections weights and unit biases for all the layers in the model.

  Hyperparameters. The hyperparameters here are the embedding dimension, the GRU size k, the learning and the dropout rate. No changes in their configuration has been made with regard to the setting used in (Kraska et al., 2018), except for the size of the embedding layer, reduced to 10, as done in (Ma and Liang, 2020), to comply with the reduced size of our dataset, and the smaller size of the overall filter as well. The embedding dimension and k affect the design of the Learned Filters. Therefore, we refer to them as key hyperparameters. We use the same terminology also for the other classifiers used in this study.

- *Naive Bayes (NB)*. A Naive Bayes Classifier (Duda and Hart, 1973) ranks instances based on Bayes' rule. It learns the conditional probabilities of having a certain label $Y = y_k$ given a specification of the attributes A_i:

$$Y \leftarrow \arg\max_{y_k} \frac{P(Y=y_k)P(A_1=x_1\ldots A_n=x_n|Y=y_k)}{\sum_j P(Y=y_j)P(A_1=x_1\ldots A_n=x_n|Y=y_j)}.$$

  The conditional probabilities on the right are parameters of the model, and are efficiently estimated from training data by assuming all attributes are conditionally independent given Y (naive). To deal with missing attribute configurations and to avoid zero estimates of some parameters, a regularization yielding smooth estimates is applied.

  Hyperparameters. The only hyperparameter is the one determining the strength of the smoothing for missing combinations of the attributes, which is not a key hyperparameter.

- *Logistic Regression (LR)*. The logistic regression (Cox, 1958) classifies an instance $x = (x_1,\ldots,x_n)$ as negative if

$$P(Y=-1|A=x) > P(Y=1|A=x),$$

  where

$$P(Y=-1|X=x) = \frac{\exp(w_0 + \sum_{i=1}^n w_i x_i)}{1+\exp(w_0 + \sum_{i=1}^n w_i x_i)},$$

  and $P(Y=1|X=x) = 1 - P(Y=-1|X=x)$. Here, $w = w_0,\ldots w_n$ are parameters of the classifiers, which are estimated from training data. A regularization is typically applied to impose the norm of w to be minimized along with the maximization of the conditional data likelihood.

  Key hyperparameters. The model has just one hyperparameter, the one determining the strength of the regularization, which is not key.

- *Linear Support Vector Machine (SVM)*. Linear SVMs (Cortes and Vapnik, 1995) classify a given instance $x \in \mathbb{R}^n$ as $f(x) = w^T \cdot x + b$, where $\cdot$ is the inner product operator, $w \in \mathbb{R}^n$, and $b \in \mathbb{R}$. The hyperplane (w,b) is learned to maximize the margin between positive and negative points:

$$\begin{aligned} \min_{w\in\mathbb{R}^n} & \frac{1}{2}\|w\|^2 + C\sum_{x\in D}\xi_x \\ \text{s.t.} \quad & y_x f(x) \geq 1-\xi_x \quad x \in D \\ & \xi_x \geq 0 \quad x \in D\,, \end{aligned}$$

  where ξ_x is the error in classifying instance x, and C is an hyperparameter regulating the tolerance to misclassifications. Non-linear SVMs, e.g. with Gaussian kernel, have been discarded due to their excessive size, in light of the fact that they need to store not just the hyperplane, but the also the kernel matrix.

  Hyperparameters. C is the unique hyperparameter for this classifier, and it is not key.

- *FFNN-h*. A Feed-Forward neural network (Zell, 1994) with the input layer of size n, one hidden layer with h units, and one output unit with sigmoid activation. The weights of connections and the neuron biases are the parameters to be estimated from training data.

  Hyperparameters. The learning rate, the activation function for the hidden layer, and the number of hidden units h are hyperparameters here, with the latter being also a key hyperparameter.

3.2 Data and Hardware

Following the literature, we concentrated on URL data. URLs are divided into benign and malicious addresses, with the latter typically stored in the filter. They are as follows.

- The first dataset comes from (UNIMAS, 2021), and contains 15k benign and 15k malicious URLs.
- The second dataset comes from (Machine Learning Lab, 2021), in particular as benign we have chosen 3637 entries in the category legitimate uncompromised, and as malicious 38419 entries in the category hidden fraudolent.

As for the training of the classifiers, we use both the mentioned datasets with their division in benign and malicious. Since the number of benign elements does not guarantee good training on the NN-based classifiers included in this research, we also consider the following dataset.

- All the 291753 URL available at (BOTW, 2021), that we take as being benign.

In analogy with (Ma and Liang, 2020), we processed the Web addresses as follows.

- *Standardization*. Common strings like `"www."` and `"http://"` have been removed from each URL, and URLs have been resized to a length of 150, by either padding shorter addresses with a marker symbol or by truncating excess characters, removing duplicated URLs. The resulting dataset contains 310329 benign and 43744 malicious URLs.

Finally, the datasets have been suitably coded for the input of the classifiers employed in this work.

- *Input coding for RNN.* In analogy with (Ma and Liang, 2020), URLs have been coded by mapping each character to consecutive integers starting from 1 to 128 in order of frequency in the training set, and to 0 all the remaining characters, eventually obtaining a 150-dimensional vector. For their nature, such classifiers need an embedding layer taking as input the whole sequence of characters in the address to execute the training.
- *Input coding for other classifiers.* Here a standard *bag of char* coding is adopted, based on the fact that the remaining classifiers are more flexible on their input format (no embedding needed). In this coding, each URL is represented with a vector containing for each distinct character in the training set its frequency in the URL–we tested both relative and absolute frequency, with no significant differences in the classifiers' performance. The extracted vectors are positional, in the sense each distinct character is assigned a fixed position in the vector. This way, addresses are coded with vector of dimension 79. We point out that other potential URL codings are applicable, which however are beyond the scope of this study, since URL coding is the same for all classifiers (except for RNN), and the aim is to evaluate the role of classifiers in this learned data structure.
- *Hardware.* The experiments were run on a Intel Core i5-8250U 1.60GHz Ubuntu machine with 16GB RAM.

3.3 Evaluation Framework

3.3.1 Classifiers Screening

Intuition suggests that the better a classifier is at discriminating benign from malicious URLs, the better the performance of the Learned Bloom Filter using it. Nevertheless, the space occupied by the classifier plays a central role, and the trade-off performance/space is to be taken into account.

In order to shed quantitative light on this aspect, we perform classification experiments involving the URLs and the classifiers, without the filter.

- *Classifier validation.* Classifiers' generalization capabilities have been evaluated through a 5-fold cross validation on the whole dataset, and the performance assessed in terms of Accuracy, $F1$ measures, and space occupied. The latter includes the space of both classifier structure and input encoding, stored as a standard in Python via the serialization method `dump` of library *Pickle* (Python Software Foundation, 2021). The results averaged across folds are calculated. In order to have a binary prediction necessary to compute Accuracy and $F1$, in this experiment the best threshold τ is estimated on the training set to maximize the $F1$.

Table 1: Cross validation performance of the compared classifiers. For each measure also the standard deviation across folds is shown.

Classifier	Accuracy	F1	Space (Kb)
NB	0.909 ± 0.009	0.522 ± 0.003	3.40
SVM	0.929 ± 0.001	0.673 ± 0.003	1.56
LR	0.930 ± 0.001	0.672 ± 0.003	1.61
RNN-16	0.935 ± 0.003	0.751 ± 0.010	9.45
RNN-8	0.925 ± 0.002	0.733 ± 0.005	6.39
RNN-4	0.916 ± 0.009	0.677 ± 0.035	5.51
FFNN-64	0.962 ± 0.001	0.846 ± 0.002	21.80
FFNN-16	0.952 ± 0.001	0.812 ± 0.002	5.63
FFNN-8	0.946 ± 0.001	0.789 ± 0.003	2.94

- *Classifier model selection.* The best performing hyperparameter configuration for each classifier (see Section 3.1) is selected through a 5-fold (inner) cross validation on the training set. In order to directly assess the trade-off Accuracy/Space, the key hyperparameters of the RNN and FFNN classifiers have been preset to $k = \{4, 8, 16\}$ (as done in (Ma and Liang, 2020)), and $h = \{8, 16, 64\}$.

3.3.2 Learned Bloom Filters

Following (Ma and Liang, 2020), both typologies of learned Bloom Filters have been validated in the following holdout setting. The training regards the construction of the Bloom Filter on the malicious URLs, which follows a standard approach, but it must also account for the training of the classifier embedded in the filter. Such a task is performed using half of the benign URL uniformly selected (in addition to the malicious ones), whereas the rest of benign URLs constitutes the holdout set used to compute the filter performance.

Evaluation. The filters have been also evaluated in terms of space, including both classifiers and auxiliary Bloom Filters space, and reject time. To this end, as baseline, the space and reject time of the classic Bloom Filter is also reported. As suggested in (Mitzenmacher, 2018), we tuned τ to achieve classifier false positive rates ϵ_τ that are within the bounds required, and the performance of the filter evaluated for different admissible values of ϵ_τ (see Section 2.1).

Table 2: Size (in Kb) and average reject time (in sec.) of Bloom Filters storing malicious URLs by varying the false positive rate ε.

ε	0.001	0.005	0.01	0.02
Space	76.8	58.9	51.2	43.5
Time	$1.22 \cdot 10^{-6}$	$1.03 \cdot 10^{-6}$	$1.02 \cdot 10^{-6}$	$1.00 \cdot 10^{-6}$

Table 3: Average reject time in seconds for one query of the different learned Bloom Filters.

	ε			
Classifier	0.001	0.005	0.010	0.020
		LBF		
NB	3.12e-6	3.24e-6	3.01e-6	3.37e-6
SVM	3.35e-6	3.23e-6	3.43e-6	3.58e-6
LR	3.77e-6	3.84e-6	3.95e-6	3.98e-6
RNN	1.64e-3	1.75e-3	1.76e-3	1.77e-3
FFNN	2.27e-4	2.73e-4	2.72e-4	2.73e-4
		SLBF		
Bayes	4.83e-6	4.90e-3	4.71e-6	5.03e-6
SVM	4.32e-6	4.58e-6	4.41e-6	4.59e-6
LR	4.47e-6	4.64e-6	4.35e-6	4.68e-6
RNN	1.57e-3	1.71e-3	1.71e-3	1.72e-3
FFNN	2.70e-4	2.64e-4	2.63e-4	2.65e-4

3.4 Results

3.4.1 Results of Classifiers Screening

In Table 1 we report performance comparison of the adopted classifiers when trained on the whole dataset. FFNNs and RNNs achieve Accuracy and $F1$ values higher that the other competitors, with the former being preferable w.r.t. RNNs from all the standpoints (Accuracy, $F1$ and space/performance trade-off). These two families of classifiers are able to unveil (unlike NB, SVM and LR) even non linear relationships between input and output, accordingly their superior results are not surprising. The remaining classifiers are competitive in terms of Accuracy, less in terms of $F1$. Their best feature here is the compactness, being SVM and LR around $1.88\times$ and $13.97\times$ smaller than the smallest ($FFNN$-8) and biggest ($FFNN$-64) NN-based models, respectively. This is a central issue in this setting, because the classifier size contributes to the total space of the learned Bloom Filter using it. To help this analysis, in Table 2 the size of the classical Bloom Filter with different desired false positive rates ε is shown. A large classifier, like $FFNN$-64, is practically not usable on this dataset, since it occupies, for example when $\varepsilon = 0.02$, till half of the space of the filter. For this reason, in the next experiment, evaluating the performance of learned Bloom Filters, the range of values tested for h is set to $\{10, 15, 20, 25, 30\}$.

3.4.2 Results of Bloom Filters Comparison

Figure 1 depicts overall filter size when employing different classifiers and when varying both ε and $\frac{\varepsilon_\tau}{\varepsilon}$. Having the classifiers FFNN and RNN different choices for the performance/space trade-off, depending on the configuration of their key hyperparameters, here the one leading to the smallest learned filter has been reported. Specifically, $k = 16$ and $h = 20$.

A first relevant emerging trend is that the choice of ε_τ is critical to determine the best gain in space. In the case of LBFs, for small ε values (first two columns), the best compression corresponds to low values of $\frac{\varepsilon_\tau}{\varepsilon}$, suggesting that the classifier precision (lower number of false positives) is more relevant when we aim at constructing LBFs with small false positive rates. Indeed, the filter size grows with $\frac{\varepsilon_\tau}{\varepsilon}$. On the other side, this means that when smaller ε values are required, the best choice is to rely more on the backup filter to guarantee a false positive rate ε (a classifier having a low ε_τ produces more false negatives). Only the FFNN-based LBF with $\varepsilon = 0.005$ already shows a trend observed also for larger ε values (last two columns), where for most LBFs the overall space decreases till a certain value of $\frac{\varepsilon_\tau}{\varepsilon}$ is attained, then it starts to increase. This means that the classifier plays a more crucial role in this cases, as confirmed by the higher space reduction with respect to the classic Bloom Filter (dashed line).

SLBFs work differently, and it is not intuitive to understand the reason of such a difference. We attempt to provide here some potential clues in this direction. A first and main difference, except for filters relying on NB and LR, for which the best ε_τ is always the lowest one (probably due to the poor performance of NB and LR), is that now for smaller ε values the classifiers are allowed to produce more false positives, and conversely for larger choices of ε. Exactly as opposite to what happens for the LBFs. A possible explanation resides in the role of the initial Bloom Filter of the SLBF. Filtering out a portion of negatives allows the classifier to produce less false positives, thus fostering a different tuning of τ. A second relevant difference with LBFs is the higher stability in terms of space with respect to the choice of τ, which confirms the analyses in (Mitzenmacher, 2018). Overall, SLBFs tend to achieve larger size reduction than LBFs, mainly for smaller values of ε, as a confirm of previous theoretical (Mitzenmacher, 2018) and empirical (Dai and Shrivastava, 2020) results.

Regarding the impact of individual classifiers on the overall size, in most cases the filter using a FFNN compresses the most with respect to the classic Bloom Filter, in accordance with the results shown in Table 1;

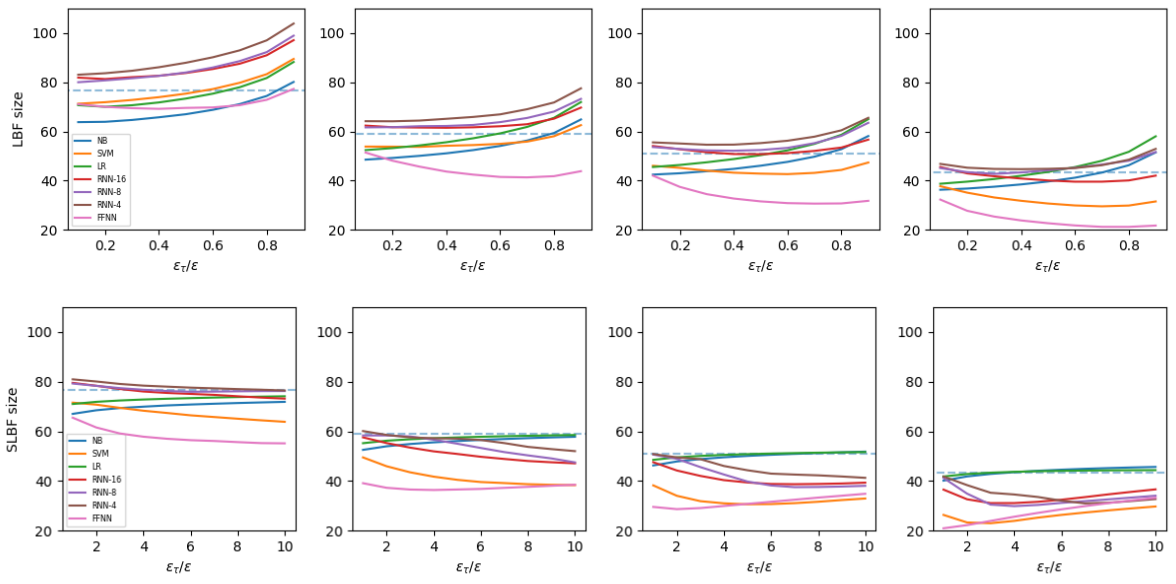

Figure 1: Size of LBF (first row) and SLBF (second row) for different classifier false positive rate. Dashed line is the size of the corresponding Bloom Filter. From the left, figures refer to a desired false positive rate ε of 0.001, 0.005, 0.01, 0.02.

however, the SVM is very competitive, mainly when embedded in the SLBF, being almost indistinguishable with FFNN in terms of space gain. Another surprising result is for the LBFs with $\varepsilon = 0.001$, where the NB allows the best space reduction. Such results are even more meaningful if we consider the reject times shown in Table 3: the filters built using SVMs and NB are two orders of magnitude faster than those using FFNNs. Thus, what is counterintuitive is that non-linear or complex is not necessarily better than linear and simple when we are declining the characteristics of a classifier to be employed in a learned Bloom Filter. It is likely such results are tied to the size of key set, being higher the potential gain of a learned version vs. its classic counterpart when the number of keys increases (Dai and Shrivastava, 2020; Ma and Liang, 2020). This is left to future investigations.

4 CONCLUSIONS AND FUTURE DEVELOPMENTS

We have preliminarily investigated the impact of general purpose classifiers within recently proposed extension of Bloom Filters, named learned Bloom Filters. Following (Mitzenmacher, 2018), which emphasized how these learned extensions are dependent on the data distribution, we have conducted an empirical evaluation in the context of malicious URLs detection, considering a wide range of general purpose classifiers. Previous works on learned Bloom Filters have not focused on the appropriate choice of the classifier, and simply relied on non-linear models. Our results have confirmed the suitability of such classifiers, but also have affirmed that simpler classifiers (e.g., linear) might be the best choice in some specific contexts, especially when the reject time is also a key factor. Future works in this directions would extend and further validate the learned Bloom Filters on different application domains, e.g. Genomics, Cyber security, Web networking, and test the obtained results against problems with different key set sizes and distributions, thus allowing to unveil further insights into the role played by the classifiers, possibly including even those discarded in this paper for their excessive space occupancy. Finally, the adoption of succinct representations of the trained classifiers, e.g. Neural Networks (Long et al., 2019; Marinò et al., 2021a; Marinò et al., 2021b), might sensibly contribute to reduce the size of a learned filter.

ACKNOWLEDGEMENTS

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 R.G. has been granted by Project INdAM - GNCS Project 2020 Algorithms, Methods and Software Tools for Knowledge Discovery in the Context of Precision Medicine.

REFERENCES

Amato, D., Giancarlo, R., and Bosco, G. L. (2021). Learned sorted table search and static indexes in small space: Methodological and practical insights via an experimental study. *CoRR*, abs/2107.09480.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Boffa, A., Ferragina, P., and Vinciguerra, G. (2021). A "learned" approach to quicken and compress rank/select dictionaries. In *Proceedings of the SIAM Symposium on Algorithm Engineering and Experiments (ALENEX)*.

BOTW (2021). Best of the Web – Free Business Listing. https://botw.org. Last checked on Oct. 18, 2021.

Broder, A. and Mitzenmacher, M. (2002). Network Applications of Bloom Filters: A Survey. In *Internet Mathematics*, volume 1, pages 636–646.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232.

Dai, Z. and Shrivastava, A. (2020). 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*, volume 33, pages 11700–11710. Curran Associates, Inc.

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk.

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification, 2nd Edition*. Wiley.

Ferragina, P., Lillo, F., and Vinciguerra, G. (2021). On the performance of learned data structures. *Theoretical Computer Science*, 871:107–120.

Ferragina, P. and Vinciguerra, G. (2020a). Learned Data Structures. In *Recent Trends in Learning From Data*, pages 5–41. Springer International Publishing.

Ferragina, P. and Vinciguerra, G. (2020b). The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *PVLDB*, 13(8):1162–1175.

Freedman, D. (2005). *Statistical Models : Theory and Practice*. Cambridge University Press.

Kipf, A., Marcus, R., van Renen, A., Stoian, M., Kemper, A., Kraska, T., and Neumann, T. (2020). Radixspline: A single-pass learned index. In *Proc. of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, aiDM '20, pages 1–5. Association for Computing Machinery.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). The case for learned index structures. In *Proc. of the 2018 Int. Conf. on Management of Data*, SIGMOD '18, pages 489–504, New York, NY, USA. Association for Computing Machinery.

Long, X., Ben, Z., and Liu, Y. (2019). A survey of related research on compression and acceleration of deep neural networks. *Journal of Physics: Conference Series*, 1213:052003.

Ma, J. and Liang, C. (2020). An empirical analysis of the learned bloom filter and its extensions. Unpublished. Paper and code no more available on line.

Machine Learning Lab (2021). Hidden fraudulent urls dataset. https://machinelearning.inginf.units.it/data-and-tools/hidden-fraudulent-urls-dataset. Last checked on Oct. 18, 2021.

Maltry, M. and Dittrich, J. (2021). A critical analysis of recursive model indexes. *CoRR*, abs/2106.16166.

Marcus, R., Kipf, A., van Renen, A., Stoian, M., Misra, S., Kemper, A., Neumann, T., and Kraska, T. (2020a). Benchmarking learned indexes. *arXiv preprint arXiv:2006.12804*, 14:1–13.

Marcus, R., Zhang, E., and Kraska, T. (2020b). CDFShop: Exploring and optimizing learned index structures. In *Proc. of the 2020 ACM SIGMOD Int. Conf. on Management of Data*, SIGMOD '20, pages 2789–2792.

Marinò, G. C., Ghidoli, G., Frasca, M., and Malchiodi, D. (2021a). Compression strategies and space-conscious representations for deep neural networks. In *Proceedings of the 25th International Conference on Pattern Recognition (ICPR)*, pages 9835–9842. doi:10.1109/ICPR48806.2021.9412209.

Marinò, G. C., Ghidoli, G., Frasca, M., and Malchiodi, D. (2021b). Reproducing the sparse huffman address map compression for deep neural networks. In *Reproducible Research in Pattern Recognition*, pages 161–166, Cham. Springer International Publishing. doi:10.1007/978-3-030-76423-4_12.

Mitzenmacher, M. (2018). A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Mitzenmacher, M. and Vassilvitskii, S. (2020). Algorithms with predictions. *CoRR*, abs/2006.09123.

Python Software Foundation (2021). pickle – python object serialization. https://docs.python.org/3/library/pickle.html. Last checked on Oct. 18, 2021.

UNIMAS (2021). Phishing dataset. https://www.fcsit.unimas.my/phishing-dataset. Last checked on Oct. 18, 2021.

Vaidya, K., Knorr, E., Kraska, T., and Mitzenmacher, M. (2021). Partitioned learned bloom filters. In *International Conference on Learning Representations*.

Zell, A. (1994). *Simulation neuronaler Netze*. habilitation, Uni Stuttgart.