Bloom Filter with Learned Hashes

SURA Project Report

Supervised by Prof. Srikanta Bedathur

Jain, Aditi

2019MT60739

Abstract

All existing learned Bloom filters [11, 9, 10, 12] employ a classifier-based approach, with a back-up Bloom filter to improve over the traditional Bloom filters in terms of false positives and memory footprint. However, there is still scope to use the patterns and similarities in data to utilise space more efficiently by increasing positive-positive and negative-negative key clashes. We propose an alternate Bloom filter with a projection-based approach to replace hash functions and thus assign spatially-closer data-points to the same bins. This approach shows significant improvement over the traditional Bloom filter in terms of false positive rate and memory. We also chose one such learned model from Kraska's work[11], to highlight our model's performance with that of Kraska's model.

1 Introduction

A Bloom filter is a space-efficient and randomized data structure that pertains to set membership queries with a small but constant probability of a false positive. While Bloom filters are highly space-efficient, they can still occupy a significant amount of memory. To address this issue, learned Bloom filters have been explored and built upon recently [11, 9, 10, 12]. These can yield the same false positive rates as traditional Bloom filters with a much lower memory footprint. However, current learned Bloom filters have primarily worked by training a classifier on the distribution of data, and added a back-up Bloom filter to it. Not much work has been done to train the hash functions of traditional Bloom filters themselves. A learned hash function has been made with the goal to maximize collisions among set elements (which we call keys in this report) and among non-set elements (called non-keys hereafter) while minimizing collisions of keys and non-keys. This can be done by training a classifier model and scaling the output into a bit array. Though this serves the purpose of making a hash function of our specifications, this cannot be repeated for multiple such hash functions, and hence a Bloom filter cannot be made with just this in its current state.

We aim to devise a way to make efficient hash functions which use the distribution of the data to maximize key and non-key collisions and minimize the collisions between keys and non-keys. This technique has to yield multiple hash functions in such a way that their outputs are independent of

each other as well to serve as replacement of randomized hash functions used in traditional Bloom filters. We plan to better the false positive rate(FPR) of this learned hash Bloom filter in comparison to a traditional Bloom filter by identifying and utilizing patterns and similarities in the distribution of data.

2 Methodology

On visualizing any given dataset as a plot of points in n-dimension space, depending on the structure of data there could exist some clusters in the data points. We aim to use such concentration of positives (keys) and negatives (non-keys) in $n-d$ space to increase collisions in the bit-array. In our proposed model, we replace the traditional set of hash functions with a set of vectors (say k) with the dimensionality tuned to that of the dataset (no of attributes of the dataset = dimension of vectors). After projecting every data point on each of the k vectors by taking their dot product, the vectors are divided into m segments that correspond to m bits in the bit array and these occupied bits are accordingly set to 1. The allocation of vectors to bins happens via a sigmoid/kernel function and the resultant is floored to get k hash values in the range $[0, m-1]$. We explored various methods to divide the vectors with the projections, such as using an RBF Kernel[13], a Polynomial Kernel[13], Min-Max Scaling[13], Sigmoid Scaling[13] etc. We elaborate more on these below.

2.1 Projection Hash Bloom Filter

Given a set X to be inserted, with integer parameters and a set of random vectors W, the hash functions trained using the set X are given by :

$$h_X^{(w)}(x) = \lceil f(x \cdot w) \times m \rceil \tag{1}$$

where $x \in X, w \in W$, m represents the bin-size i.e. total bits in the bit-array, and f is a scaling function such as sigmoid or max-min scaler. x and w are vectors from the same vector-space.

For projections using kernels, the hash function is as follows:

$$h_X^{(w)}(x) = \lceil f\big((Kernel(x,w)\big) \times m \rceil \tag{2}$$

Since we propose a projection-based method to substitute the hashes in a Bloom filter and better utilise the patterns and similarities in data points, we henceforth refer to our bloom filter as *Projection Based Bloom Filter* or PHBF.

We picked the vectors randomly and tested them on a small subset of the data and iterated multiple times to choose the best random set of vectors that would give the lowest FPR for the given dataset and parameters.

2.2 Kernels

We explored two kernels in our model: Polynomial and RBF kernel. With Polynomial Kernel we used Sigmoid Scaler. X represents a point of the dataset and Y is a vector from the vector-set.

$$\text{Polynomial}(X,Y) = (\gamma < X,Y > +c)^d \tag{3}$$

$$\text{RBF}(X,Y) = e^{-\gamma\|X-Y\|^2} \tag{4}$$

For polynomial kernel, c was taken to be 0 as it would not have an importance once sigmoid scaling was applied. d was used with values 1 and 2 i.e. in linear and quadratic terms. Gamma used was from the range $[0.01, 0.5]$. In the RBF Kernel, Gamma was taken from the range $[0.01, 0.5]$ with best results at 0.01.

2.3 Multiple Bit Arrays

We also explored an adjustment to the bit-array, to try to reduce overlap between different vector projections and reduce unwanted clashes between them, by dividing the m-bit array into k buckets and each section would correspond to one hash function i.e. one vector projection.

2.4 Scaling

For scaling, we took the lengths of projections and applied transformations to them to bring them to a desired range of lengths. We divided this range into m bins to get the hash value for each projection. One of the transformations we used was the sigmoid function, which brought the range from 0 to 1.

$$\text{Sigmoid}(x) = \frac{e^x}{1+e^x} \tag{5}$$

In another transformation, we found the difference between the maximum and minimum values of projections and divided the projection lengths by this. This yielded values between 0 and 1.

$$\text{Min-Max Scaler}(x) = \frac{x}{max - min} \tag{6}$$

2.5 Additional Parameters

We also experimented by using only unit/non-unit vectors, or positive/negative vectors. The projection model was also iterated over the dataset for a set no. of cycles and the best set of random vectors (i.e. those with the lowest FPR) were used.

3 Analysis

In our proposed model, PHBF, we aim to utilise similarities in data which show up as clusters in an $n-d$ space, and project these points to same or close by bins. In the analysis below, we show how this 'discriminativeness' of hash-functions improves the Bloom filter.

Definition 1.1 A function $h : X \to Y$, where Y is finite, is β discriminative for (A,B) (where $A,B \subset X$) if for any $x \in B$ picked uniformly at random $P(h(x) \in h(A)) \leq \beta$.

The precision of known classifiers corresponds to β discriminative power. The exact relation can be derived as follows:

(FP= False Positives, FN= False Negatives, TP= True Positives, TN= True Negatives)

$$precision(p) = \frac{TP}{TP+FP} \tag{7}$$

$$recall(r) = \frac{TP}{TP+FN} \tag{8}$$

$$\beta(fpr) = \frac{FP}{FP+TN} \tag{9}$$

$$|B| = FP+TN \tag{10}$$

$$|A| = FN+TP \tag{11}$$

Substituting 10 in 9

$$FP = \beta * |B| \tag{12}$$

Substituting 11 in 8

$$TP = r * |A| \tag{13}$$

Substituting 12 & 13 in 7 and assuming FN=0 (i.e. r=1) , we get the relation of FPR with p as

$$p = \frac{|A|}{|A| + \beta * |B|} \tag{14}$$

If h is β discriminative for (A,B), we use *Bayes* rule to comment about its discriminativeness for (B,A) as follows:

Let $Q = h^{-1}(h(A) \cap h(B))$. Using Bayes rule, h is β'-*discriminative* on (B,A), where $\beta' = \beta \frac{|Q \cap A|.|B|}{|Q \cap B|.|A|}$

Definition 1.2 A function $h : X \to Y$, where Y is finite, is β discriminative for (A,B) (where $A,B \subset X$) if for any $x \in B$ picked uniformly at random $P(h(x) \in h(A)) \leq \beta$.

On inserting a set A into a Bloom filter using k functions drawn from a hash family that is (α,β) discriminative for the pair (A,B), the false positive rate for the set B can be derived in terms of α and β as below:

Probability of false positives for one hash function h_i is: β with probability $1-\alpha$ and 1 with probability α. That is, $FPR_{h_i} = \alpha \cdot 1 + (1-\alpha)\beta$. This is the probability that $h_i(x) \in h_i(A)$ for $x \in B$. Therefore, assuming independence of h_i, $FPR_H = (\alpha + \beta - \alpha\beta)^k$. Assuming entire set A is inserted, FPR_H is independent of $|A|$ or $|B|$. Note that $|Y|$ is a property of H, that is, given H, $|Y|$ is fixed.

Extending this to a standard Bloom filter analysis, assume an array of size m. For $\alpha = 0$ and $\beta = 1 - \left(1 - \frac{1}{m}\right)^{|A|}$, $FPR_{m,k} = (\beta)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{|A|}\right)^k \approx \left(1 - e^{-|A|/m}\right)^k$

As expected on decreasing β (which is inversely proportional to discriminativeness of a hash function), the FPR reduces. Thus our aim is to find and minimize the β for our hash-functions.

4 Experiments

4.1 Experimental Setup

In our experiments we compared sizes and FPR against a Standard Ideal Bloom Filter(SBF), and with Kraska's Learned Hash Bloom Filter model (LHBF)[11]. We inserted only the positives from the train split in the Bloom filter. The test split contained only negatives (half of total negatives in the dataset) and the models were run on these to calculate corresponding FPR. The preprocessing performed was standard scaling of columns and relabelling/merging classes when there were more than 2 of them.

4.2 Results

4.2.1 Kraska's Learned Hash Bloom Filter (LHBF)

In LHBF[11], for smaller datasets [3, 2, 8, 1], 90+ accuracy was reached with layer size as less as 3 and 1000 epochs. It also had high accuracies with 10 layer sizes and 500 epochs. The neural network based classifiers were harder to train for the larger datasets [4, 5, 6, 7]. But due to the high sizes of the classifiers, their FPR were worse than that of SBF in almost all the datasets. Some dataset specific observations were :

- In normalised data consisting of 4 sets with mean 0,20,40 and 60, the model reached 90+ accuracy at 10 layer size and around 1000-2000 epochs. It worked poorly for layer size 5. The LHBF models performed better than a SBF.
- Iris: If 1st category was given a value of y and the rest 2 categories the other value of y, the model gained 0 FPR even with layer size=1 because the categories 0 and (1,2) are linearly separable. With 1^{st} 2 categories same and 3^{rd} category a different class, model was a little larger for good FPR.

4.2.2 Projection Model

For every m, we selected the ideal values of k and FPR for SBF. We compared the FPR of both SBF and PHBF using these values of m and k. Due to random vectors, different iterations gave different results. For 5 iterations in every dataset, at least one iteration gave better FPR than SBF. Using RBF Kernel and a polynomial Kernel with the PHBF did not have any improvements over the original PHBF.

Some datasets where it performed better or as well as original are: Heart disease, Diabetes (better), Cardio. Min Max scaler with kernel gave performance same as original PHBF in Diabetes.

Datasets where PHBF improved over LHBF and SBF : Breast Cancer, Wine, Iris, 2D Gaussian,SatLog ,Cardio (LHBF fared better though).

Datasets where PHBF performed only as well as SBF(but better than LHBF): Heart disease, Diabetes, Magic.

Datasets where it was worse than SBF : MNIST (better than LHBF though) , MiniBooNE.

Dataset	Size	Attributes	Classes
Iris	150	4	3 Positive keys: Iris Setosa, Iris Versicolour Negative: Iris Virginica (because first and third classes were linearly separable).
Breast Cancer	569	30	2 Positives: 1 Negatives: 0
Diabetes	768	8	2 Positives: Tested Positive Negatives: Tested Negative
Wine Recognition	178	13	3 Positives: 2 Negatives: 0 & 1
Heart Disease	178	13	2 Positives: 1 Negatives: 0
Cardiotocography	2126	35	3 Positive Keys: Class 1 Negative: Class 2 & 3
Magic	19020	11	2 Positive Keys: 'h' Negative:'g'
MiniBoone	130064	50	2 Positive: True class Negative: False class
Satellite Log	98528	100	2 Positive: '1' Negative:'-1'
MNIST	70000	784	10 Positive: '0','4','6','8','9' Negative: '1','2','3','5','7'
2-D Gaussian (Synthetic)	25000	2	2

Dataset	Size[bits]	SBF [FPR]	PHBF [FPR]	LHBF [FPR](Size)
Heart Disease	400	0.3383	0.2145	0.1642(17345)
2D-Gaussian	600	1	0.00352	0.3976(599)
Breast Cancer	500	0.5107	0.0906	0.0377(2289)
Diabetes	1850	0.0374	0.0504	0.272(6372)
Cardio	3000	0.4240	0.2144	0.0271(1338)
Iris	50	0.6358	0	0(459)
SatLog	130000	0.2823	0.3348	0.3427(71844)
Wine	100	0.3827	0.0338	0(564)
Magic	30785	0.1098	0.1015	0.1110(30785)
MNIST	107000	0.2362	0.2946	0.2564(110705)

Table 1: Average FPR of PHBF and baselines on various datasets

4.2.3 Other observations

In all datasets, SVM model performed better (or equal) than the Logistic Regression Model. Both the models performed particularly badly in the Breast Cancer dataset giving accuracy of around 65% each (therefore the large value of m was needed for a low FPR). Logistic Regression on 2-D gaussian dataset gave poor FPR at low values of m, compared to its SVM counterpart that gave FPR =0 at $m = 10$. Since the datasets we used are not very big, the LHBF models converged relatively fast and a minimum was attained very early. Also, due to most of the LHBF models being neural networks, the model sizes were in multiples of large sizes such as 25kB, which made them larger in size in comparison to the other 2 models, for most FPR values.

Dataset	Size[bits]	SBF [FPR]	PHBF [FPR]
Heart Disease	400	0.0011	0.1274
2D-Gaussian	600	0.0573	9.0262
Breast Cancer	500	0.0064	0.2679
Diabetes	1850	0.0055	0.4010
Cardio	3000	0.0236	0.6992
Iris	50	0.0003	0.0600
SatLog	130000	1.4760	22.4400
Wine	100	0.0004	0.0439
Magic	30785	0.0894	4.8881
MNIST	107000	3.4900	41.2628

Table 2: Average insertion time of PHBF and baselines on various datasets

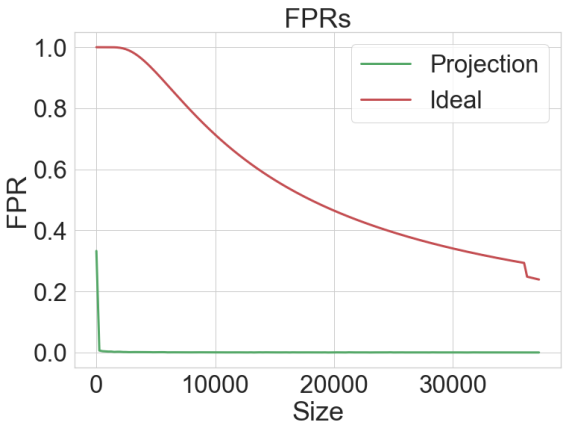

Figure 1: SBF vs PHBF for 2-D Gaussian Dataset

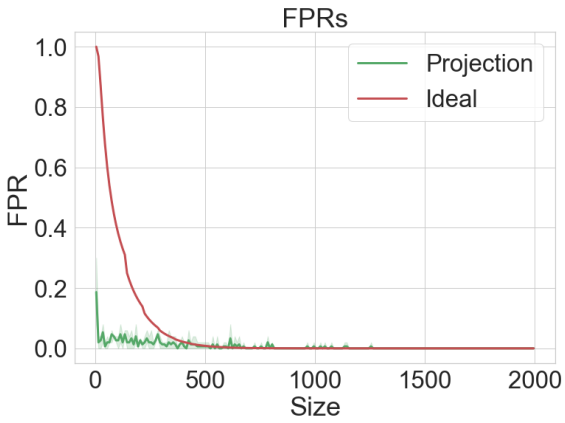

Figure 2: SBF vs PHBF for Iris Dataset

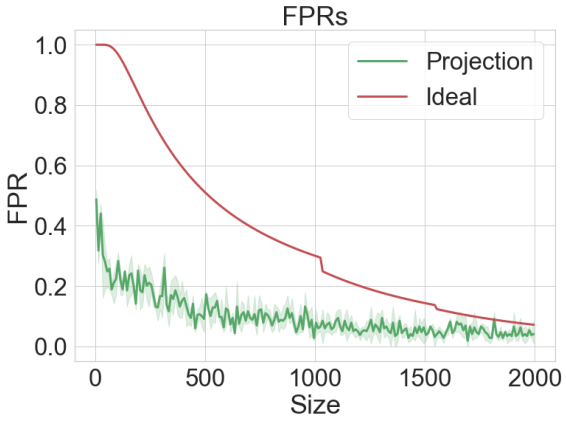

Figure 3: SBF vs PHBF for Breast Cancer Dataset

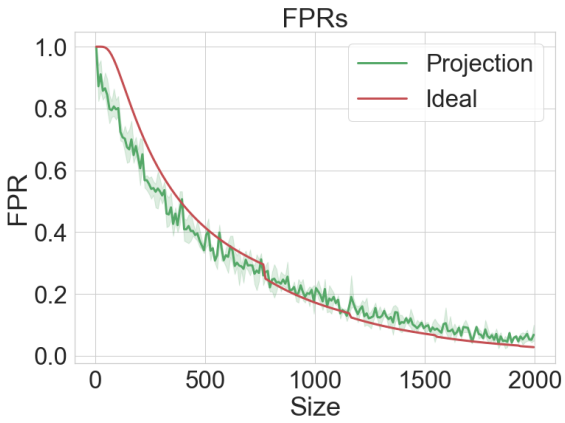

Figure 4: SBF vs PHBF for Diabetes Dataset

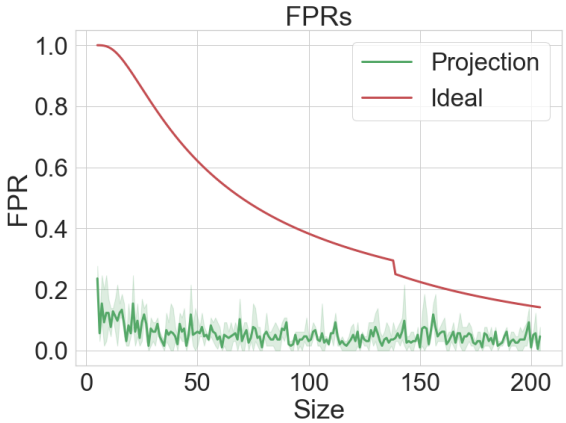

Figure 5: SBF vs PHBF for Wine Dataset

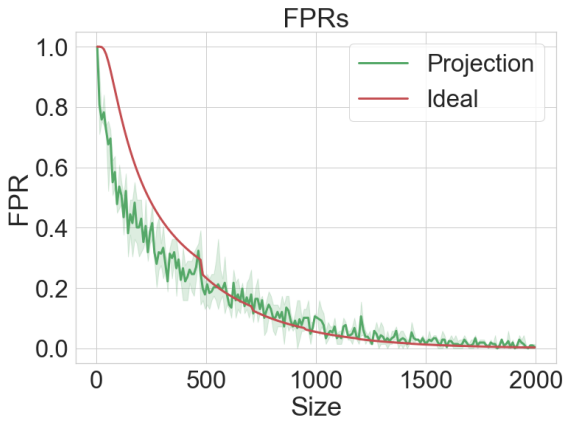

Figure 6: SBF vs PHBF for Heart Disease Dataset

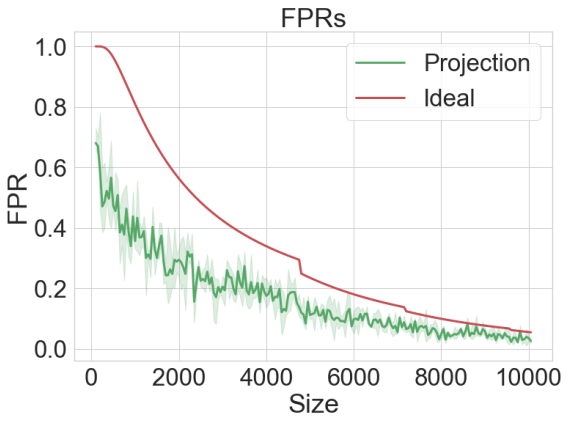

Figure 7: SBF vs PHBF for Cardio Dataset

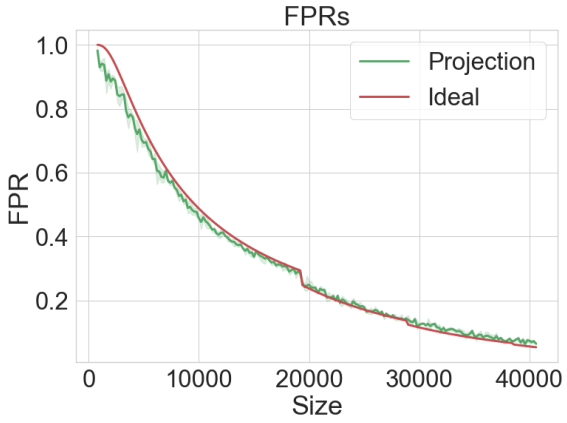

Figure 8: SBF vs PHBF for Magic Dataset

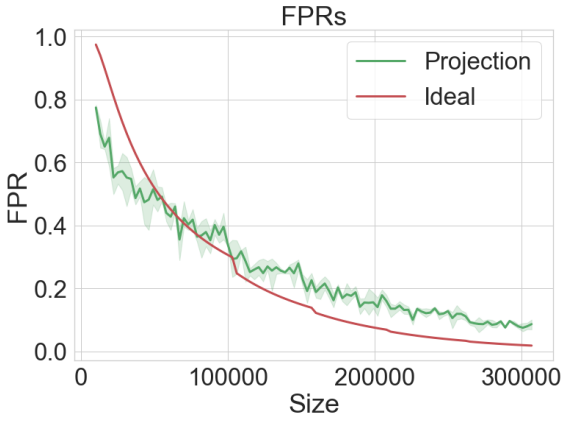

Figure 9: SBF vs PHBF for MiniBooNE Dataset

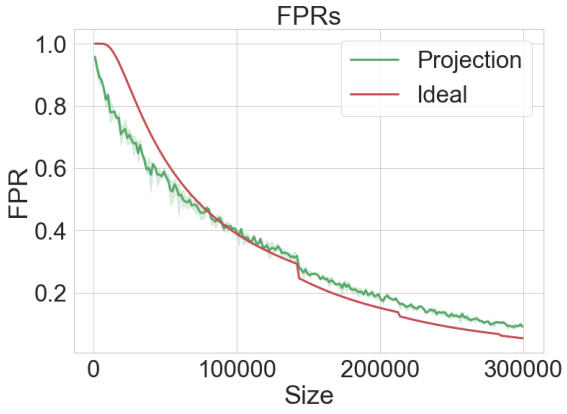

Figure 10: SBF vs PHBF for SatLog Dataset

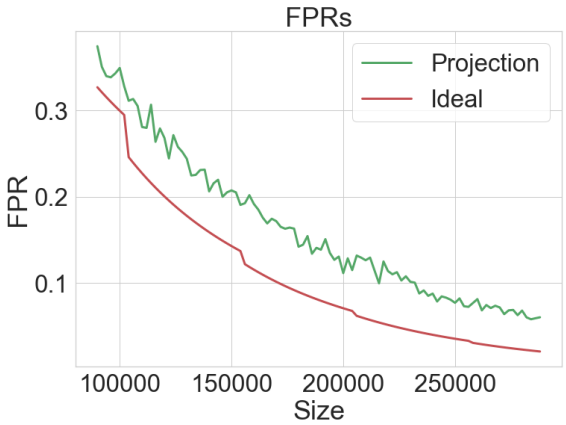

Figure 11: SBF vs PHBF for MNIST Dataset

References

[1] UCI Breast Cancer Dataset.
https://archive.ics.uci.edu/ml/datasets/breast+cancer.

[2] UCI Heart Disease Dataset.
https://archive.ics.uci.edu/ml/datasets/heart+disease.

[3] UCI Iris Dataset.
https://archive.ics.uci.edu/ml/datasets/iris.

[4] UCI Magic Gamma Telescope Dataset.
https://archive.ics.uci.edu/ml/datasets/magic+gamma+telescope.

[5] UCI MiniBooNE Dataset.
https://archive.ics.uci.edu/ml/datasets/MiniBooNE+part .

[6] UCI Optical Recognition of Handwritten Digits (MNIST) Dataset.
https://archive.ics.uci.edu/ml/datasets/optical+recognition+of
+handwritten+digits.

[7] UCI Statlog Dataset.
https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite).

[8] UCI Wine Dataset.
https://archive.ics.uci.edu/ml/datasets/Wine.

[9] Arindam Bhattacharya, Srikanta Bedathur, and Amitabha Bagchi. Adaptive learned bloom filters under incremental workloads. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, CoDS COMAD 2020, page 107115, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Adam Kirsch and Michael Mitzenmacher. Distance-sensitive bloom filters. In *Proceedings of the Meeting on Algorithm Engineering Expermiments*, page 4150, USA, 2006. Society for Industrial and Applied Mathematics.

[11] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures, 2017.

[12] Michael Mitzenmacher. A model for learned bloom filters and related structures, 2018.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.