Dynamic Learned Bloom Filters

Ștefan Manolache Alexandra Pește Dan Alistarh
IST Austria (Alistarh Group)

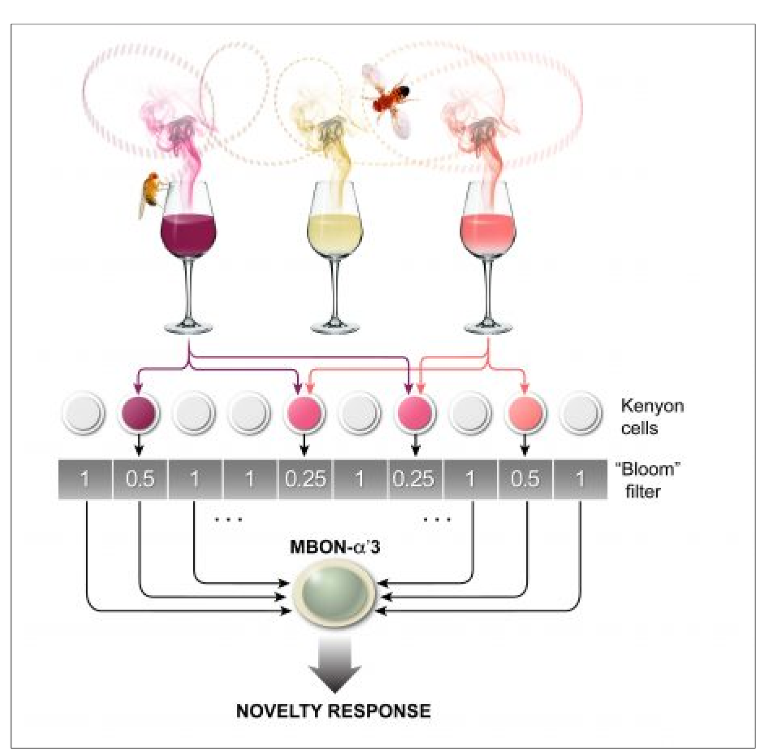
Abstract

- **Bloom Filters** are space-efficient probabilistic data structures used to approximately test the membership of an element in a set, trading off *accuracy* for *compactness*.
- Recently, it has been show that they can be further compressed through the use of machine learning, by training a **classifier** to memorise the set as well as possible and using its prediction to filter out some positive keys early.
- Building on this, we are exploring an approach that would allow for *changes* in the contents of the set after the model had already been trained, while keeping the accuracy high.
- Our method is to use a *single step update* on the model's parameters in order to perturb them just enough so that the model can classify the changed keys correctly.

Motivation

- Bloom filters are ubiquitous for applications that rely on *large databases*, heavily decreasing the memory footprint and latency.
- In cases such as web caching, malicious URL checking or weak password detection, the data may come in *streams*, thus the necessity to allow the storage of dynamic sets.

Figure 1:Fruit flies use a tactic similar to that of a Bloom Filter to detect novel odors, by having neurons called Kenyon cells process olfactory information and broadcast a "novelty alert" signal when a new odor is encountered.



Learned Bloom Filters

Figure 2:A standard Bloom filter uses k hash functions that map keys to indices in a bit array, and sets these bits to 1 for every key in the set. Hash collisions may cause false positives. Given a set of n elements and an upper bound on the FPR of ϵ, the space used by a BF is $\mathcal{O}(n \cdot log_2 \frac{1}{\epsilon})$.

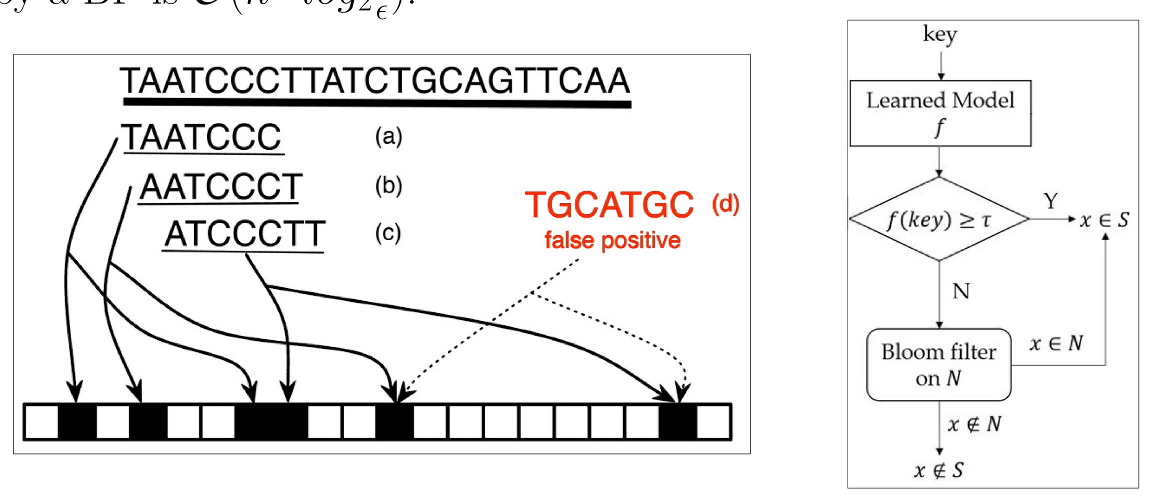
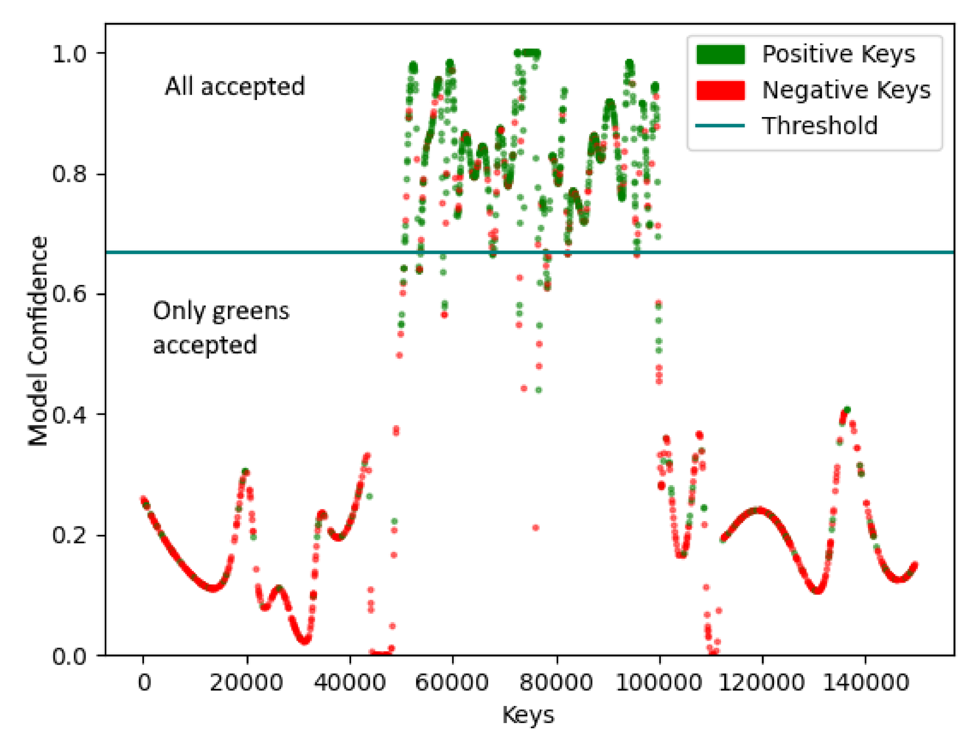


Figure 3:A Learned Bloom Filter uses a learned model f (such as a neural network) to prefilter some of the keys in the set, hence reducing the space used. Given some threshold τ, the LBF classifies all keys x with $f(x) \geq \tau$ as positive, while the rest of the keys are handled by a backup Bloom Filter. False positives can originate from both an overconfident model and hash collisions, so the threshold is chosen to find a balance.



Experiments

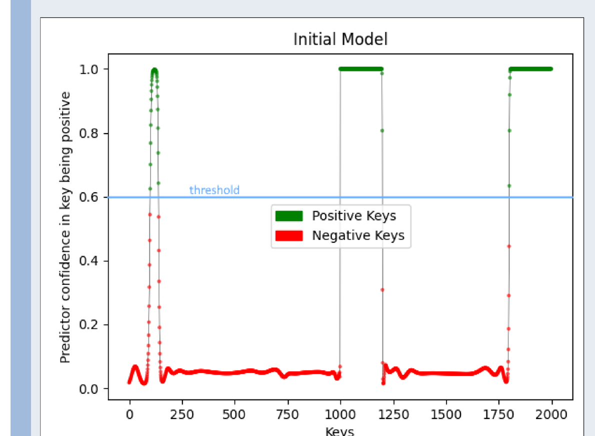


Figure 4:Comparison of key insertion via retraining from scratch vs. a single step full Hessian update. By adding noise to the labels of the negative keys, the model is able to easily adapt in this scenario.

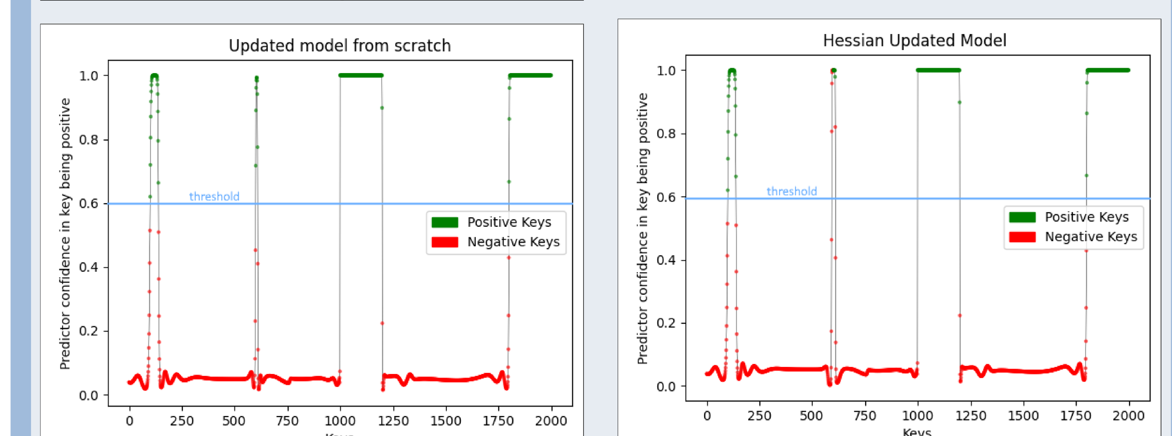
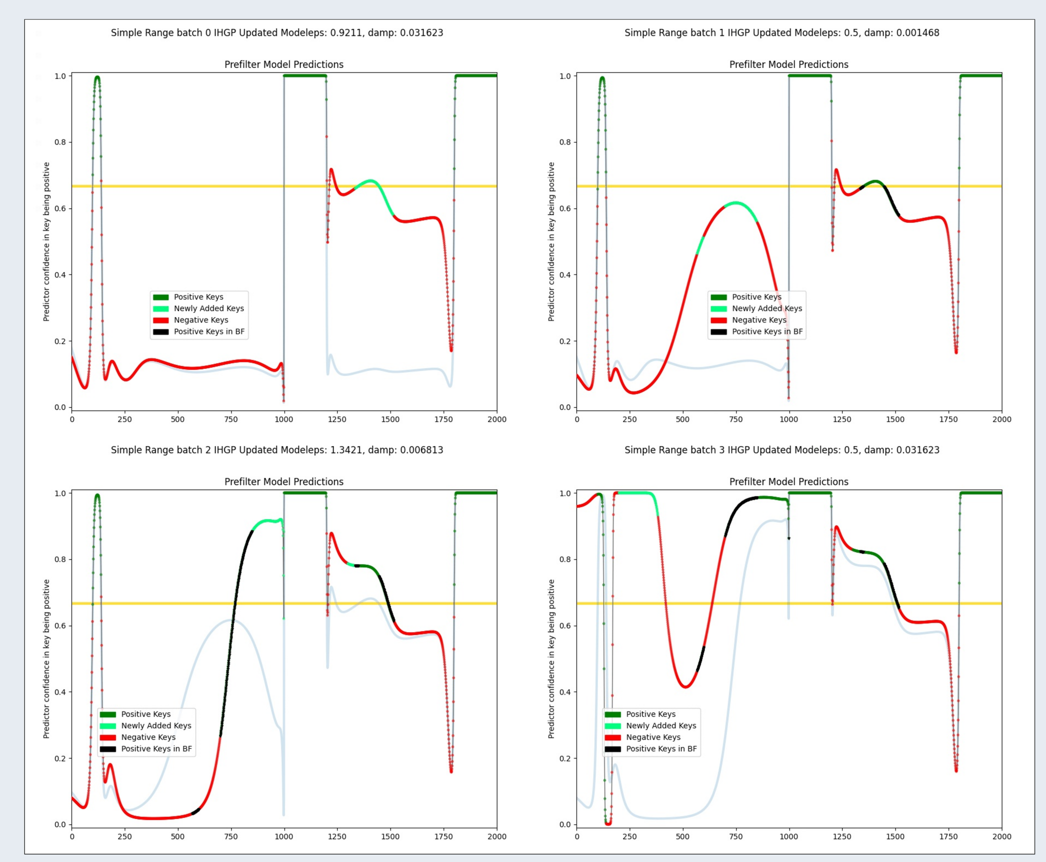


Figure 5:Behaviour of the model after several insertions of batches of keys. The update is sensitive to the choice of ϵ and λ, so a grid search is done to keep the FPR within bounds, while moving the newly inserted keys above the threshold.



(When using a neural network with one hidden layer)

Single Step Relabeling Update

Training a machine learning model f_θ to classify a universe $\mathcal{U}$ of keys into positive $\mathcal{K}$ and negative $\mathcal{N}$ comes down to calibrating its parameters θ to minimise the *loss function*: $\mathcal{L}(\theta) = \frac{1}{n} \cdot \sum_{i=1}^{n} l_i(\theta)$, where $l_i(\theta)$ is a measure of the error of the prediction on the ith data sample from the dataset $\mathcal{D} = (\mathcal{K} \times \{1\}) \cup (\mathcal{N} \times \{0\})$. Now, take $\mathcal{S}$ to be the keys that we want to insert into the set (i.e. change labels from 0 to 1). We need to minimise a new loss

$$\mathcal{L}_{new}(\theta) = \frac{1}{n} \cdot \Big(\sum_{(x_i,y_i)\in\mathcal{D}\setminus\mathcal{S}} l_i(\theta) + \sum_{(x_i,y_i)\in\mathcal{S}} l'_i(\theta) \Big)$$

(l'_i being the error for the updated labels).

Let $\hat{\theta} = argmin_\theta \mathcal{L}(\theta)$, $\hat{\theta}_{new} = argmin_\theta \mathcal{L}_{new}(\theta)$

A second-order Taylor approximation yields the following update:

$$\hat{\theta}_{new} \approx \hat{\theta} - \epsilon \nabla \mathcal{L}_{new}(\hat{\theta})$$

$$\hat{\theta}_{new} \approx \hat{\theta} - \epsilon \mathcal{H}^{-1}_{new}(\hat{\theta}) \cdot \nabla \mathcal{L}_{new}(\hat{\theta})$$

$$\hat{\theta}_{new} \approx \hat{\theta} - \epsilon \mathcal{F}^{-1}_{new}(\hat{\theta}) \cdot \nabla \mathcal{L}_{new}(\hat{\theta})$$

For models with many parameters, computing the *inverse Hessian gradient product* in real time is intractable. However, it can be approximated by the empirical Fisher Information Matrix:

$$\hat{\mathcal{F}}_{new}(\theta) = \frac{1}{n} \cdot \Big(\sum_{(x_i,y_i)\in\mathcal{D}\setminus\mathcal{S}} \nabla l_i(\theta) \otimes \nabla l_i(\theta) + \sum_{(x_i,y_i)\in\mathcal{S}} \nabla l'_i(\theta) \otimes \nabla l'_i(\theta) \Big)$$

Once we have computed it, we only require information about $\mathcal{S}$

Conclusion & Future Work

- The preliminary results seem to indicate that the model is able to adapt quite well to the changed dataset, at least with the Hessian update. However, when using a non-convex model like a neural network, there are (as of yet) no theoretical guarantees that all keys with $f_{old}(x) \geq \tau$ also satisfy $f_{new}(x) \geq \tau$., which means that we may have to relax the restriction that $FNR = 0$.
- Next to investigate are:
  - How well behaved the update is when using the empirical FIM in place of the Hessian.
  - Finding a good heuristic for choosing the hyperparameters of the update.
  - Using a Counting Bloom Filter to support both key insertions and removals.
  - Benchmarking this model against existing approaches for dynamic Bloom Filters.
  - Constraining the update to limit the number of false negatives.

References & Related Work

- T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The Case for Learned Index Structures. https://arxiv.org/abs/1712.01208, 2017.
- A neural data structure for novelty detection Sanjoy Dasgupta, Timothy C. Sheehan, Charles F. Stevens, Saket Navlakha Dec 2018, DOI: 10.1073/pnas.1814448115
- Koh, Pang Wei, and Percy Liang. "Understanding black-box predictions via influence functions." International Conference on Machine Learning. PMLR, 2017.
- Peste, Alexandra, Dan Alistarh, and Christoph H. Lampert. 'SSSE: Efficiently Erasing Samples from Trained Machine Learning Models.' arXiv preprint arXiv:2107.03860 (2021).
- D. Guo, X. Luo, H. Chen, Y. Yuan and J. Wu, 'The Dynamic Bloom Filters', 2010. DOI: 10.1109/TKDE.2009.57
- Arindam Bhattacharya, Srikanta Bedathur, and Amitabha Bagchi. 2020. Adaptive Learned Bloom Filters under Incremental Workloads. DOI: 10.1145/3371158.3371171
- Qiyu Liu, Libin Zheng, Yanyan Shen, and Lei Chen. 2020. Stable learned bloom filters for data streams. August 2020 DOI: 10.14778/3407790.3407830