Supervised Classification in Federated Learning via Locality-Sensitive Filters

Arthur Carvalho Walraven da Cuhna, Damien Rivet, Paulo Bruno Serafim, Chuan Xu, Emanuele Natale

Supervised Classification in Federated Learning via Locality-Sensitive Filters

Arthur C. W. da Cunha[1], Damien Rivet[1], Paulo Bruno Serafim[2],
Chuan Xu[1], Emanuele Natale[1]

[1]INRIA, Sophia-Antipolis, France
[2]GSSI, L'Aquila, Italy

{arthur.da-cunha, damien.rivet, chuan.xu, emanuele.natale}@inria.fr
paulo.serafim@gssi.it

Abstract

Federated Learning (FL) can struggle with high communication costs, a problem that is only exacerbated by the fact that, in FL, it is common to have heterogeneous data distributions among parties. Recently, the study of the brain of fruit flies inspired two novel ML ideas: a Locality-Sensitive Hashing (LSH) scheme, called FlyHash, and a data structure for novelty detection, called FlyBloomFilter. A recent study combined these two tools to provide a simple and efficient method for performing FL in a single shot, which is also agnostic to the heterogeneity of the data: FlyNN. Yet, despite their empirical success, theoretical understanding of both FlyHash and FlyBloomFilter is still limited. In this work, we distil the ideas underlying FlyHash into a variant of SimHash, one of the most famous LSH schemes. We provide a theoretical basis for the proposed algorithm and leverage the insights obtained to connect the novelty detection structure to classical Bayesian theory, yielding improvements also for FlyBloomFilter. Ultimately, we propose a simplification of FlyNN, for which we provide both a theoretical motivation and extensive experiments demonstrating its competitive performance.

1 Introduction

The present work aims to contribute to the recent progress in performing privacy-friendly classification in federated learning (FL) setups by leveraging new locality-sensitive hashing (LSH) schemes [1]. Concretely, consider the following basic FL setting. A dataset is split across multiple clients, with each data point having an associated class. New data points are then received by the clients, and the goal is to classify them by leveraging the information available in the whole dataset. However, for privacy reasons, the clients do not wish to reveal their data to other clients. The latter constraint prevents direct use of k-nearest neighbours (kNN) classification, one of the most popular supervised algorithms for this problem. While sophisticated cryptographic techniques could be employed to address the privacy constraint [2, 3], Ram and Sinha [1] recently introduced a simple LSH-based solution that can be readily combined with standard differential privacy (DP) techniques to achieve the desired privacy guarantees. More precisely, Ram and Sinha [1] empirically shows that FlyHash, an LSH scheme introduced in Dasgupta, Stevens and Navlakha [4], and an associated data structure for novelty detection, the FlyBloomFilter (FBF) introduced in Dasgupta et al. [5], can be combined to classify new data in a way that resembles kNN classification, and DP can be straightforwardly applied on top of the resulting algorithm to provide privacy guarantees. In that respect, they provide some theoretical evidence and empirical results showing that FlyHash performs competitively both against kNN and SimHash, the most popular LSH scheme.

In a nutshell, if we consider data points on a high-dimensional sphere, SimHash consists in considering many random hemispheres and producing a binary vector whose i-th coordinate indicates whether the input point is in the i-th hemisphere (see section 3.1). FlyHash, on the other hand, considers the sum of many small random subsets of entries of the input point and produces a binary vector whose 1s correspond to the ρ largest sums, where ρ is assumed to be small. In this way, the resulting binary vectors are *sparse* (see section 3.1). The trade-off for such sparsity is a much larger hash length: FlyHash's output binary vectors have a dimensionality which is greater than that of the input. This latter property crucially deviates from standard LSH schemes, that have traditionally been developed to reduce the dimensionality of the input space (see section 2).

While FlyHash's sparsity and high-dimensionality are in line with works in hyperdimensional computing [6], the advantage of using it in data mining applications and related field is still being explored (see section 2), with the introduction of FBF and their use in Ram and Sinha [1] being an important step in this direction. Their proposed algorithm, FlyNN, not only allows to perform FL in a single round of communication (one-shot FL), minimising both communication costs and privacy risks but is also completely robust to the non-homogeneity of the data distribution between the clients. Heterogeneous splitting of the data can be a strong limitation for other FL methods, especially in the one-shot setup, so FlyNN, despite its simplicity, can become a competitive method in this regime when the heterogeneity of the data distribution becomes high.

The high-level idea of Ram and Sinha [1] is to hash all data points belonging to a given class with some LSH scheme, and to combine the obtained hashes into a single vector, which we call *filter*; new points are then classified by comparing the projections of their hashes on the filter of each class. While Ram and Sinha [1] shows empirically that FlyHash outperforms other candidates in the above framework, it motivates the following natural questions, which are the focus of this work:

1. What makes a good filter?
2. Which properties of FlyHash make filters built with it perform better?

Our contributions. We make progress on the first question by introducing the *cross-entropy filter*, which we formally show to be equivalent to a naive Bayes estimation of the maximum likelihood classification of the input point under a general LSH function (see section 3.3). While the filter considered in [1] is motivated by previous work inspired by neuroscience insights, Dasgupta et al. [5], we provide the first filter with a theoretical statistical justification. As for the second question, we investigate a natural variant of the popular SimHash algorithm, in which SimHash's hemispheres (corresponding to the portion of the sphere cut by a hyperplane passing through the origin) are replaced by smaller *spherical caps* (corresponding to the portion of the sphere cut by a hyperplane slightly offset from the origin). Such variant, which we name *CapHash*, is an instantiation of Spherical LSH [7, 8], a well-established family of LSH methods that use spherical caps for partitioning. While the geometric principle underlying CapHash aligns with existing Spherical LSH techniques, our specific contribution lies in demonstrating how very sparse, large spherical LSH codes can achieve performance similar to FlyHash in the federated learning context. By considering large hash lengths, we obtain an LSH family similar to FlyHash, while still retaining from SimHash a more intuitive interpretation and being more amenable to theoretical analysis (see section 3.3). Empirically, the cross-entropy filter combined with CapHash performs competitively compared to FlyHash in our extensive experiments, corroborating our progress on the above questions regarding the theoretical principles underlying the filter framework introduced in Ram and Sinha [1].

2 Related Work

Locality Sensitive Hashing is a fundamental technique in data mining [9]. Among its many methods, we highlight the SimHash algorithm [10], which produces dense binary hashes by splitting the space with random hyperplanes. The idea of using affine hyperplanes instead is present in works such as Andoni et al. [8].

Spherical LSH and related methods. Our CapHash algorithm is an instantiation of Spherical LSH, a well-established family of LSH methods that partition the unit sphere using spherical caps [7, 8, 11, 12, 13]. The concept of partitioning data with spherical caps dates back to at least 1998 [12] and features heavily in theoretical analyses of the k-nearest neighbor problem [14]. Notably, several of these methods outperform SimHash, particularly through data-dependent approaches [7, 13]. While the underlying geometric principle of CapHash aligns with these established methods, our contribution lies in demonstrating how very sparse, large spherical LSH codes can achieve performance similar to FlyHash in the federated learning context, and in providing the first statistical justification for the filter-based classification framework.

Recently, Dasgupta, Stevens and Navlakha [4] proposed FlyHash, a hashing algorithm sensitive to locality that mimics the olfactory circuit of fruit flies, which introduced the counterintuitive use of hash vectors that are both high-dimensional and sparse. The relationship between FlyHash and Winner-Take-All (WTA) hashing has been explored in the literature [15, 16], with WTA methods providing alternative approaches to creating sparse high-dimensional representations. Dasgupta and Tosh [17] theoretically explored the properties of such high-dimensional sparse representations by showing a universal approximation property for arbitrary continuous functions and further showed that, assuming a general manifold structure in the input space, the winner-take-all mechanism limits the effectiveness of the representation. We then consider whether the representation is adaptive to the manifold structure in the input space. This is highly dependent on the specific sparsification method: we show that adaptivity is not obtained under the winner-take-all mechanism. Shen, Dasgupta and Navlakha [18] further investigated the olfactory system of the fruit fly and proposed expansive sparse encodings as a way to prevent neural networks from overwriting learned memories in a continual learning framework. Dasgupta, Hattori and Navlakha [19] used this type of encoding for sketch counting. Ma and Li [20] applied the Frank-Wolf algorithm to train the sparse binary matrix of FlyHash for natural language processing, and Ryali et al. [21] proposed a method to learn these matrices through a biologically plausible training process. Liang et al. [22] built on these works and proposed a learning projection matrix to produce high-quality word embeddings.

Soon after the introduction of the FlyHash algorithm, Dasgupta et al. [5] noticed that the olfactory circuit of fruit flies measures how much an odour differs from those previously experienced in a way similar to a data structure known as a *Bloom filter*. They proposed a data structure that mimics this biological device. The filters in this paper strongly resemble locality-sensitive Bloom filters [23, 24, 25], which combine locality-sensitive hashing with Bloom filter principles for approximate membership queries. While our approach uses these filters for k-NN classification rather than membership testing, the underlying geometric ideas share similarities with these established techniques. Shen, Dasgupta and Navlakha [26] showed that this structure can be effective for online discrimination tasks. Sinha and Ram [27] leveraged the ideas of Dasgupta et al. [5] and proposed the *FlyNN* algorithm as a way to efficiently approximate nearest neighbour classification. The authors soon followed up with Ram and Sinha [1], which shows that FlyNN is well suited for one-shot differentially private FL and is also robust to heterogeneous distributions between parties.

Group testing and nearest neighbor methods. Recent work has explored using group testing techniques for nearest neighbor search [28]. When the groups in such methods are chosen as the k classes in our setting, the resulting algorithm may share similarities with our CapNN approach, though with different

inference functions. While using filters for k-NN classification is novel, this potential connection highlights the broader landscape of techniques for efficient similarity search.

Other methods for one-shot FL commonly rely on distillation techniques, with works such as Guha, Talwalkar and Smith [29] and Li, He and Song [30] using models trained individually by the parties as teachers for central model training on a public dataset. Zhou et al. [31] proposed to distil the private datasets into synthetic data that can be securely communicated to the server and used to train a global model. Zhang et al. [32] introduced the idea of transmitting generative models trained on private data so that the server can train a central model on the data they generate.

3 The CapNN classifier

In this section, we describe our proposed algorithm, *CapNN*. It is composed of two elements: an LSH algorithm, *CapHash*, that maps samples from the dataset to binary strings; and a classifier, *CapFilter*, that assigns a class to hashed inputs.

Those techniques are strongly related to previous methods, so we first introduce them to more clearly frame our work relative to them.

3.1 Previous Methods

SimHash

One of the most widely used LSH algorithms, and also one of the simplest, is SimHash [10]. It consists in merely fixing a random hyperplane, thus dividing the input space $\mathbb{R}^d$ into two regions. Then, it assigns the value 1 to all points falling on one side of the hyperplane and 0 to those falling on the other side. Formally, we define the function $h_{\text{sim}} \colon \mathbb{R}^d \times \mathbb{R}^d \to \{0, 1\}$, given by

$$h_{\text{sim}}(\boldsymbol{v};\, \boldsymbol{x}) = \mathbf{1}_{\{\boldsymbol{v}^\top \boldsymbol{x} \geq 0\}}.$$

Larger hashes are obtained by concatenation. Specifically, given $m \in \mathbb{N}$, let $H_{\text{sim}} \colon (\mathbb{R}^d \times \cdots \times \mathbb{R}^d) \times \mathbb{R}^d \to \{0, 1\}^m$ be given by

$$H_{\text{sim}}(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m;\, \boldsymbol{x}) = \left(h_{\text{sim}}(\boldsymbol{v}_i;\, \boldsymbol{x})\right)_{1 \leq i \leq m}.$$

Since h_{sim} and, thus, H_{sim} are invariant under positive scaling, the similarity measure underlying SimHash is the cosine similarity, rather than the Euclidean distance. Moreover, as the hash functions discussed later are also invariant under positive scaling, from now on, w.l.o.g., we only consider input vectors in $\mathbb{S}^{d-1} \subset \mathbb{R}^d$.

The simplicity of SimHash makes it easy to implement efficiently and comprehend theoretically. Indeed, the algorithm amounts to a random projection followed by a binarization. Random projections are known to effectively preserve the geometry of the input data. More precisely, the Johnson-Lindenstrauss lemma [33] ensures that, with high probability, a random projection onto a space of dimension $O[(\ln n)/\varepsilon^2]$ suffices to preserve the pairwise distance of n input points up to a multiplicative error of $1 \pm \varepsilon$. Furthermore, it can be shown that such quasi-isometry is preserved after the binarization step when considering the Hamming distance in the hash space [34]. Given those properties, much like random projections, applications of SimHash tend to decrease dimensionality, i.e., $m \ll d$.

FlyHash

This made it quite surprising when the work of Dasgupta, Stevens and Navlakha [4], drawing inspiration from the working of the fruit fly's olfactory system, proposed an alternative to SimHash that uses expansive

sparse representations. That is, the proposed algorithm, called FlyHash, maps input vectors to much larger binary representations which contain mostly zeros. The authors empirically showed that FlyHash outperforms SimHash in different similarity search tasks.

FlyHash can be formalised as a function $H_{\text{fly}} \colon \mathbb{N} \times \{0,1\}^{m \times d} \times \mathbb{S}^{d-1} \to \{0,1\}^m$, given by

$$H_{\text{fly}}(\rho, \boldsymbol{M};\, \boldsymbol{x}) = \text{WTA}_\rho(\boldsymbol{M}\boldsymbol{x}),$$

where $\text{WTA}_\rho \colon \mathbb{R}^m \to \{0,1\}^m$ is the winner-take-all function: for $1 \leq i \leq m$, we have that $(\text{WTA}_\rho(\boldsymbol{x}))_i$ is 1 if x_i is among the ρ largest entries of $\boldsymbol{x}$, and 0 otherwise. Moreover, the lift matrix $\boldsymbol{M}$ is uniformly sampled from the subset of binary matrices with m rows and d columns, such that each row contains s ones, with $s \ll d$.

FlyBloomFilter

Hattori et al. [35] investigated how the brain of fruit flies leverages sparse expansive representations to detect the novelty of signals. Based on the model from that work, Dasgupta et al. [5] proposed a data structure that resembles a Bloom filter [36] and that can leverage similarity-preserving hashing schemes to efficiently estimate how much a new input differs from previous ones. Formally, given $\gamma \in [0, 1)$, a hash function $f \colon \mathbb{R}^d \to \{0,1\}^m$, and n input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, the proposed structure consists of a vector $\boldsymbol{w}_{f,\gamma} \in \mathbb{R}^m$, with

$$(w_{f,\gamma})_i = \prod_{j=1}^{n} \gamma^{(f(\boldsymbol{x}_j))_i},$$

where the parameter γ controls the rate at which the novelty of a signal decays.[1] A new input $\boldsymbol{y} \in \mathbb{R}^d$ can then be assigned a novelty score by simply taking the dot product ${\boldsymbol{w}_{f,\gamma}}^\top f(\boldsymbol{y})$. For convenience, we may omit the underlying function and the decay rate when those are clear from the context, writing simply $\boldsymbol{w}$.

While FlyBloomFilter can leverage any binary hash function, the high sparsity of FlyHash reduces the overlap of stimuli, which leads to better empirical performance [5]. Sinha and Ram [27] explore the use of such a combination of FlyHash and FlyBloomFilter for approximating nearest neighbours classification, calling it FlyNN, and the follow-up work of Ram and Sinha [1] proposes its use for federated learning. Overall, given a binary classification dataset, the method consists in building two FlyBloomFilters, $\boldsymbol{w}^{(0)}_{H_{\text{fly}}}$ and $\boldsymbol{w}^{(1)}_{H_{\text{fly}}}$ using the data of the respective class. Then, a new input $\boldsymbol{y} \in \mathbb{R}^d$ is predicted to belong to the class that minimizes its novelty score. That is, to class

$$\arg\min_{\ell \in \{0,1\}} \{(\boldsymbol{w}^{(\ell)}_{H_{\text{fly}}})^\top H_{\text{fly}}(\boldsymbol{y})\}.$$

In line with Dasgupta et al. [5], the work of Ram and Sinha [1] investigates the effect of the high hash sparsity on FlyNN by replacing H_{fly} by H_{sim}. Their experiments also show that the low sparsity of SimHash leads to a significant drop in performance.

3.2 CapHash

Motivated by those findings, in this work, we propose sparsifying SimHash in perhaps the most natural way: instead of deciding each bit of the hash by comparing $\boldsymbol{v}^\top \boldsymbol{x}$ with 0, we compare it with a new parameter $c > 0$. This approach is an instantiation of Spherical LSH [7, 8], where the unit sphere is partitioned using

[1] The original FlyBloomFilter in Hattori et al. [35] contain an additional parameter that models time. It is motivated by their application to novelty detection.

spherical caps rather than hemispheres. The concept of using spherical caps for locality-sensitive hashing is well-established in the literature [12, 11, 13], with our specific contribution being the demonstration of how very sparse, large spherical LSH codes can effectively replace FlyHash in federated learning applications.

More precisely, we consider $h_{\mathrm{cap}} \colon \mathbb{R}_{>0} \times \mathbb{R}^d \times \mathbb{S}^{d-1} \to \{0,1\}$ given by

$$h_{\mathrm{cap}}(c, \boldsymbol{v};\, \boldsymbol{x}) = \mathbf{1}_{\{\boldsymbol{v}^\top \boldsymbol{x} \geq c\}}.$$

Analogously to H_{sim}, we define $H_{\mathrm{cap}} \colon \mathbb{R}_{>0} \times (\mathbb{R}^d \times \cdots \times \mathbb{R}^d) \times \mathbb{S}^{d-1} \to \{0,1\}^m$ given by

$$H_{\mathrm{cap}}(c, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_m;\, \boldsymbol{x}) = \left(h_{\mathrm{cap}}(\boldsymbol{v}_i;\, \boldsymbol{x})\right)_{1 \leq i \leq m}.$$

The parameter c controls the expected sparsity of the hashes. Particularly, for $\mathbf{V} \sim \mathcal{N}(0, (1/d) \cdot \boldsymbol{I}_d)$, it holds that

$$\begin{aligned} \Pr\left[h_{\mathrm{cap}}(c, \mathbf{V};\, \boldsymbol{x}) = 1\right] &= \Pr\left[\mathbf{V}^\top \boldsymbol{x} \geq c\right] \\ &= \Pr\left[\mathcal{N}(0, 1/d) \geq c\right] \\ &= 1 - \Phi\left(c\sqrt{d}\right), \end{aligned}$$

where Φ denotes the standard normal cumulative distribution function. Hence, given $\rho \in \mathbb{R}_{>0}$, by setting $c = \Phi^{-1}(1 - \rho/m)/\sqrt{d}$ we obtain that, for random vectors $\mathbf{V}_1, \ldots, \mathbf{V}_m$ i.i.d. following $\mathcal{N}(0, (1/d) \cdot \boldsymbol{I}_d)$, the expected number of ones in $H_{\mathrm{cap}}(c, \mathbf{V}_1, \ldots, \mathbf{V}_m;\, \boldsymbol{x})$ is ρ.

3.3 Cross-entropy filter

Consider a binary classification task on $\mathbb{R}^d$ and let $\mathcal{X}_0$ and $\mathcal{X}_1$ denote the distributions generating the points of class 0 and class 1, respectively. Let $\mathbf{X}^{(0)} \sim \mathcal{X}_0$ and $\mathbf{X}^{(1)} \sim \mathcal{X}_1$. Given a generic hash function $f \colon \mathbb{R}^d \to \{0,1\}^m$, we can define the random variables $\mathbf{H}^{(0)} = f(\mathbf{X}^{(0)})$ and $\mathbf{H}^{(1)} = f(\mathbf{X}^{(1)})$, and consider the associated classification task under f. That is, $\mathbf{H}^{(0)}$ is random point belonging to class 0 and $\mathbf{H}^{(1)}$, to class 1. In this framework, given a test point $\boldsymbol{x}$, the maximum likelihood estimation for its class is

$$\operatorname*{arg\,max}_{\ell \in \{0,1\}} \left\{\Pr\left[\mathbf{H}^{(\ell)} = f(\boldsymbol{x})\right]\right\}.$$

We have that

$$\Pr\left[\mathbf{H}^{(\ell)} = f(\boldsymbol{x})\right] = \Pr\left[\bigwedge_{i=1}^{m} \mathrm{H}_i^{(\ell)} = f(\boldsymbol{x})_i\right].$$

Applying a naive Bayes heuristic, we estimate this probability as

$$\prod_{i=1}^{m} \left(\Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]\right)^{f(\boldsymbol{x})_i} \cdot \left(1 - \Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]\right)^{1 - f(\boldsymbol{x})_i}.$$

Maximizing such quantity is equivalent to maximizing its logarithm. This is the well-known fact that maximizing the likelihood is equivalent to minimizing the cross-entropy, motivating the name of the method.

Applying the logarithm, we obtain

$$\begin{aligned}
&\sum_{i=1}^{m} f(\boldsymbol{x})_i \cdot \log \Pr\left[\mathrm{H}_i^{(\ell)} = 1\right] \\
&\quad + \sum_{i=1}^{m} (1 - f(\boldsymbol{x})_i) \cdot \log\left(1 - \Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]\right) \\
&= \sum_{i=1}^{m} f(\boldsymbol{x})_i \cdot \log \frac{\Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]}{1 - \Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]} \\
&\quad + \sum_{i=1}^{m} \log\left(1 - \Pr\left[\mathrm{H}_i^{(\ell)} = 1\right]\right).
\end{aligned}$$

Our proposed method seeks to attribute to $\boldsymbol{x}$ the label that maximizes the above expression.

To this end, we estimate the probabilities $\Pr[\mathrm{H}_i^{(\ell)} = 1]$ from the training data. Thus, we consider a vector $\hat{\boldsymbol{p}}^{(\ell)} \in (0,1)^m$, where $\hat{p}_i^{(\ell)}$ is the empirical frequency associated to $\Pr[\mathrm{H}_i^{(\ell)} = 1]$ in the training set with the caveat that we employ the rule of succession to avoid frequencies equal to zero.

Finally, for each $\ell \in \{0,1\}$ we consider the vectors $\boldsymbol{z}_f^{(\ell)} \in \mathbb{R}^m$, and $b^{(\ell)} \in \mathbb{R}$, given by

$$(z_f^{(\ell)})_i = \log \frac{\hat{p}_i^{(\ell)}}{1 - \hat{p}_i^{(\ell)}} \quad \text{and} \quad b^{(\ell)} = \sum_{i=1}^{m} \log(1 - \hat{p}_i^{(\ell)}).$$

We can, thus, approximate a maximum likelihood estimation of the class of $\boldsymbol{x}$ as

$$\arg\max_{\ell \in \{0,1\}} \left\{ (\boldsymbol{z}_f^{(\ell)})^\top f(\boldsymbol{x}) + b^{(\ell)} \right\}.$$

CapNN is the above cross-entropy filter with CapHash as the hash function f. Algorithm 1 details the computation of the vectors $\boldsymbol{z}^{(i)}$ and biases $b^{(i)}$.

4 Theoretical results

In this section, we provide theoretical results that show some formal advantages of CapHash over SimHash and FlyHash. In particular, we show that there are input distributions for which the filters resulting from SimHash and FlyHash are the same in expectation for all classes. On the contrary, CapHash can correctly distinguish a broad class of distributions, under very general hypotheses.

For the sake of simplicity, we consider a variant of CapHash, which we call CapHash*, for which the rows of the projection matrix are normalized. Geometrically, this implies that all caps have the same size. Such restriction does not improve performance and it can be removed at the cost of a much more intricate analysis.

Definition 4.1. Two distributions $\mathcal{X}$ and $\mathcal{Y}$ on a set $D \subseteq \mathbb{R}^d$ have margin $\eta > 0$ if and only if for $\mathbf{X} \sim \mathcal{X}$ and $\mathbf{Y} \sim \mathcal{Y}$ we have that $\Pr[\|\mathbf{X} - \mathbf{Y}\|_2 \leq \eta] = 0$.

Our first claim shows that SimHash can fail to distinguish distributions that are very different margin-wise.

Claim 4.2. *There exist distributions $\mathcal{X}$ and $\mathcal{Y}$ over $\mathbb{S}^{d-1}$ with margin $\sqrt{2}$ such that for all $\boldsymbol{v} \in \mathbb{R}^d$, for $\mathbf{X} \sim \mathcal{X}$ and $\mathbf{Y} \sim \mathcal{Y}$, we have that* $\Pr[h_{\mathrm{sim}}(\boldsymbol{v}; \mathbf{X}) = 1] = \Pr[h_{\mathrm{sim}}(\boldsymbol{v}; \mathbf{Y}) = 1] = 1/2$.

Algorithm 1: The CapNN algorithm

input: Training set $D \subseteq \mathbb{R}^d \times [L]$, hash length $m \in \mathbb{N}$, target sparsity $\alpha \in [1/2, 1]$.

$c \leftarrow \frac{\Phi^{-1}(1-\alpha)}{\sqrt{d}}$;
$\boldsymbol{M} \in \mathbb{R}^{m \times d} \xleftarrow{\text{entrywise}} \mathcal{N}(0,1)$;
for $\ell \leftarrow 1$ **to** L **do**
  $\boldsymbol{z}^{(\ell)} \leftarrow \vec{\mathbf{0}}_m$;
end
foreach $(\boldsymbol{x}, y)$ *in the training data* D **do**
  `// CapHash`
  $\boldsymbol{x}' \leftarrow \boldsymbol{M}\boldsymbol{x}$; `// Random projection`
  for $i \leftarrow 1$ **to** m **do**
    $x'_i \leftarrow \mathbf{1}_{\{x'_i > c\}}$; `// Thresholding`
  end
  $\boldsymbol{z}^{(y)} \leftarrow \boldsymbol{z}^{(y)} + \boldsymbol{x}'$;
end
for $\ell \leftarrow 1$ **to** L **do**
  $n_\ell \leftarrow |\{(\boldsymbol{x}, y) \in D : y = \ell\}|$; `// # samples in class` ℓ
  for $i \leftarrow 1$ **to** m **do** `// Normalization`
    $z_i^{(\ell)} \leftarrow \frac{z_i^{(\ell)}+1}{n_\ell+2}$; `// Rule of succession`
  end
  $b^{(\ell)} \leftarrow \sum_{i=1}^m \log(1 - z_i^{(\ell)})$;
  for $i \leftarrow 1$ **to** m **do**
    $z_i^{(\ell)} \leftarrow \log \frac{z_i^{(\ell)}}{1-z_i^{(\ell)}}$
  end
end
return $((\boldsymbol{z}^{(1)}, b^{(1)}), \ldots, (\boldsymbol{z}^{(L)}, b^{(L)}))$

Proof. Let $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{S}^{d-1}$ be orthogonal, and and take $\mathcal{X} = \text{Unif}(\{\boldsymbol{p}, -\boldsymbol{p}\})$ and $\mathcal{Y} = \text{Unif}(\{\boldsymbol{q}, -\boldsymbol{q}\})$. For all $\boldsymbol{v} \in \mathbb{R}^d$ we have that either $\boldsymbol{v}^\top \boldsymbol{p}$ or $\boldsymbol{v}^\top(-\boldsymbol{p})$ is non-negative. Similarly, either $\boldsymbol{v}^\top \boldsymbol{q}$ or $\boldsymbol{v}^\top(-\boldsymbol{q})$ is non-negative. Thus, for both $\mathcal{X}$ and $\mathcal{Y}$ the probability of sampling a point in the hemisphere determined by $\boldsymbol{v}$ is $1/2$. □

The next result shows that FlyHash is *invariant* w.r.t. certain translations; as a consequence, it fails to distinguish some distributions regardless of the margin between them. For example, the distributions $\text{Unif}([0,\epsilon]^d)$ and $\alpha\vec{\mathbf{1}} + \text{Unif}([0,\epsilon]^d)$ are identical under H_{fly} for any $\alpha > 0$.

Claim 4.3. *Let $d, m, \rho, s \in \mathbb{N}$, with $\rho \leq m$ and $s \leq d$, and let $\boldsymbol{M} \in \{0,1\}^{m \times d}$ be a matrix with s ones per row. Let $\mathcal{X}$ be a distribution with bounded support $D \subset \mathbb{R}^d$. For all $\eta > 0$, there exists another distribution $\mathcal{Y}$ with margin η w.r.t. $\mathcal{X}$, such that the output of $H_{\text{fly}}(\rho, \boldsymbol{M}; \cdot)$ follows the same distribution for both $\mathcal{X}$ and $\mathcal{Y}$.*

Proof. Since D is bounded, there exists $r \in \mathbb{R}$ such that $\|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq r$ for any $\boldsymbol{x}, \boldsymbol{y} \in D$. Let $\mathbf{X} \sim \mathcal{X}$ and let $\mathbf{Y} = \alpha\vec{\mathbf{1}} + \mathbf{X}$, with $\alpha \geq r + \eta$, so that $\mathbf{Y}$'s distribution $\mathcal{Y}$ and $\mathcal{X}$ have margin η. We will show that, under the current coupling between $\mathcal{X}$ and $\mathcal{Y}$ (i.e. the fact that $\mathbf{Y} = \alpha\vec{\mathbf{1}} + \mathbf{X}$), we have that $H_{\text{fly}}(\rho, \boldsymbol{M}; \mathbf{X}) =$

$H_{\text{fly}}(\rho, \boldsymbol{M};\, \mathbf{Y})$. This implies that, even if we sample $\mathbf{Z} \sim \mathcal{Y}$ independently from $\mathbf{X}$, since $\mathbf{Z}$ follows the same distribution as $\mathbf{Y}$, then $H_{\text{fly}}(\rho, \boldsymbol{M};\, \mathbf{Z})$ also follows $\mathcal{X}$.

To obtain that $H_{\text{fly}}(\rho, \boldsymbol{M};\, \mathbf{X}) = H_{\text{fly}}(\rho, \boldsymbol{M};\, \mathbf{Y})$ it suffices to note that

$$\boldsymbol{M}\mathbf{Y} = \boldsymbol{M}\left(\alpha\vec{\mathbf{1}} + \mathbf{X}\right) = \alpha\boldsymbol{M}\vec{\mathbf{1}} + \boldsymbol{M}\mathbf{X} = \alpha s + \boldsymbol{M}\mathbf{X}.$$

Hence, for all $1 \leq i, j \leq m$, we have that $\boldsymbol{M}\mathbf{Y}_i \geq \boldsymbol{M}\mathbf{Y}_j$ if and only if $\boldsymbol{M}\mathbf{X}_i \geq \boldsymbol{M}\mathbf{X}_j$, and, thus, $\text{WTA}_\rho(\boldsymbol{M}\mathbf{Y}) = \text{WTA}_\rho(\boldsymbol{M}\mathbf{X})$. □

Definition 4.4. We define the *radius* of a cap $C \subseteq \mathbb{S}^{d-1}$ centred on $\boldsymbol{z} \in \mathbb{R}^d$ as the radius r of the ball $B_r(\boldsymbol{z})$ such that $B_r(\boldsymbol{z}) \bigcap \mathbb{S}^{d-1} = C$. Moreover, we denote by a_r the normalized area of a cap with radius r.

Next, we show that, for sufficient sparsity and hash lengths, CapHash* is able to *separate* two distributions with a margin by creating caps that contain only points from the correct distribution. This implies, in particular, that the resulting cross-entropy filters succeed in distinguishing the distributions. To do that, we provide the following definition in order to exclude *pathological* distributions that can generate samples which are too isolated.

Definition 4.5. A distribution $\mathcal{X}$ with support $D \subseteq \mathbb{S}^{d-1}$ is (η, δ)-dense if and only if for any $\boldsymbol{x} \in D$ it holds that $\Pr\left[B_\eta(\boldsymbol{x})\right] \geq \delta$.

Lemma 4.6. *Let*

$$m = \frac{\log \frac{\left(1+\frac{8}{\eta}\right)^d}{\delta}}{\log \frac{1}{1-a_{\frac{\eta}{4}}}}.$$

With probability $1-\delta$ any point $\boldsymbol{z} \in \mathbb{S}^{d-1}$ of the sphere is at distance at most $\eta/2$ from the centre of a cap of radius η.

Proof. By [37, Lemma5.2], there exists an $(\eta/4)$-net N_η of the d-dimensional sphere of size $(1+8/\eta)^d$, i.e. a set of points of the sphere for which all other points are at distance at most $\eta/4$ from it. Given $\boldsymbol{z} \in N_\eta$, let $\mathcal{E}_{\boldsymbol{z}}$ be the event "$\boldsymbol{z}$ is at distance at most $\eta/4$ from the center of a cap", then

$$\Pr\left[\neg\mathcal{E}_{\boldsymbol{z}}\right] = \left(1 - a_{\frac{\eta}{4}}\right)^m = \frac{\delta}{\left(1+\frac{8}{\eta}\right)^d}.$$

Let $\mathcal{E}$ be the event "all points in N_η are at distance at most $\eta/4$ from the center of a cap"; by a union bound, $\Pr\left[\neg\mathcal{E}\right] \leq \sum_{\boldsymbol{z}\in N_\eta} \Pr\left[\neg\mathcal{E}_{\boldsymbol{z}}\right] = \delta$. Suppose now that $\mathcal{E}$ holds. By definition of $(\eta/4)$-net, any point $\boldsymbol{x} \in \mathbb{S}^{d-1}$ of the sphere is at distance at most $\eta/4$ from a point $\boldsymbol{z}$ in N_η, and $\boldsymbol{z}$ is at distance at most $\eta/4$ from the center of a cap centred on $\boldsymbol{v}$, thus, by the triangle inequality,

$$\|\boldsymbol{x} - \boldsymbol{v}\|_2 \leq \|\boldsymbol{x} - \boldsymbol{z}\|_2 + \|\boldsymbol{z} - \boldsymbol{v}\|_2 \leq \frac{\eta}{4}.$$

□

Lemma 4.7. *Let $\mathcal{X}$ and $\mathcal{Y}$ be $(\eta/2, \delta)$-dense distributions with margin $\eta > 0$ and*

$$n = \frac{\log \frac{1}{\delta}}{\log \frac{1}{1-a_{\frac{\eta}{2}}}}.$$

The cross-entropy filter given by CapHash* with caps of radius η and output dimension

$$m = \frac{\log \frac{\left(1+\frac{8}{\eta}\right)^d}{\delta}}{\log \frac{1}{1-a_{\frac{\eta}{4}}}}$$

succeeds in correctly classifying a point sampled from either distribution with probability at least $1 - 2\delta$.

Proof. Without loss of generality, let $\boldsymbol{x}$ be a point sampled from $\mathcal{X}$. Let $\mathcal{E}_1$ be the event "$\boldsymbol{x}$ is at distance at most $\eta/2$ from the center of a cap $C_{\boldsymbol{x}}$", and $\mathcal{E}_2$ be the event "at least one among the n data points sampled from $\mathcal{X}$ is in $C_{\boldsymbol{x}}$". First observe that, since caps have diameter η which is smaller than the margin between the two distributions, any cap containing $\boldsymbol{x}$ will contain no point from $\mathcal{Y}$. Thus, for CapHash* to succeed, it remains to show that $\mathcal{E}_2$ holds.

By Lemma 4.6, $\mathcal{E}_1$ holds with probability $1-\delta$. When $\mathcal{E}_1$ holds, we also have that $B_{\frac{\eta}{2}}(\boldsymbol{x}) \bigcap \mathbb{S}^{d-1} \subseteq C_{\boldsymbol{x}}$. Since $\mathcal{X}$ is $(\eta/2, \delta)$-dense, we have that

$$\Pr[\neg\mathcal{E}_2 \mid \mathcal{E}_1] \leq \left(\Pr_{\mathbf{Y}\sim\mathcal{X}}\left[\mathbf{Y} \notin B_{\frac{\eta}{2}}(\boldsymbol{x})\right]\right)^n = \left(1 - a_{\frac{\eta}{2}}\right)^n = \delta.$$

Therefore, $\Pr[\mathcal{E}_2] \geq \Pr[\mathcal{E}_2 \mid \mathcal{E}_1]\Pr[\mathcal{E}_1] \geq (1-\delta)^2 \geq 1 - 2\delta$. □

5 Experiments

Following Ram and Sinha [1], we study the performance of the algorithms on 71 real-world datasets obtained from OpenML [38]. We provide a table with some details on the datasets in the supplementary material. To investigate the behaviour of the algorithms on high dimensional data, we also performed experiments on the classical vision datasets FashionMNIST [39] and CIFAR10 [40].

5.1 Results on OpenML Datasets

We experimented with all the possible combinations of the following hyperparameters:

- hash lengths of $2^i \cdot d$ for $0 \leq i \leq 6$, where d is the number of features of the dataset at hand;
- hash densities of 2^{-i} for $1 \leq i \leq 8$ (in the range of 50% to $\approx 0.4\%$).

FlyNN has two more hyperparameters: s (the number of ones per row of random projection matrix) and γ (the decay rate of the multiplicative bloom filter). Experiments in Ram and Sinha [1] suggest that FlyNN is robust to the choice of s and they recommend setting it to 5% of d. While our tests confirm this robustness, we saw slightly better results when using 10%, so we use this value instead. Similarly, our tests confirm the robustness to the choice of γ reported by previous works, and, in our experiments, we set $\gamma = 0.9$.

Since CapHash realizes sparsities in expectation, as discussed in section 3.2, it can take any value in $[0, 1/2]$ as target sparsity. FlyHash, on the other hand, deterministically sets the number of ones in the hash to be the discrete parameter ρ. We go around this limitation by setting $\rho = \lceil \text{hashdensity} \cdot m \rceil$.[2]

For each dataset, we randomly select 10% of the points to serve as the test set. Since standardization can be troublesome in some FL settings, we reserve 10% of the remaining data to serve as a sample for estimating the mean and variance of the data. This simulates public data that could serve such a purpose. For CapNN, we use those estimations of the mean and variance to standardize both the test set and the

[2]We use ceiling instead of rounding to ensure that any possible advantage would lean towards the competing method.

remaining points, which are used as the train set. FlyNN did not show improvements with standardization, so the sample set is simply discarded.

The performance of both methods increases monotonically with the hash length. To illustrate the scenarios of moderate and high expansion of dimensionality, in figure 1 we show results for hash lengths of $8d$ and $64d$.

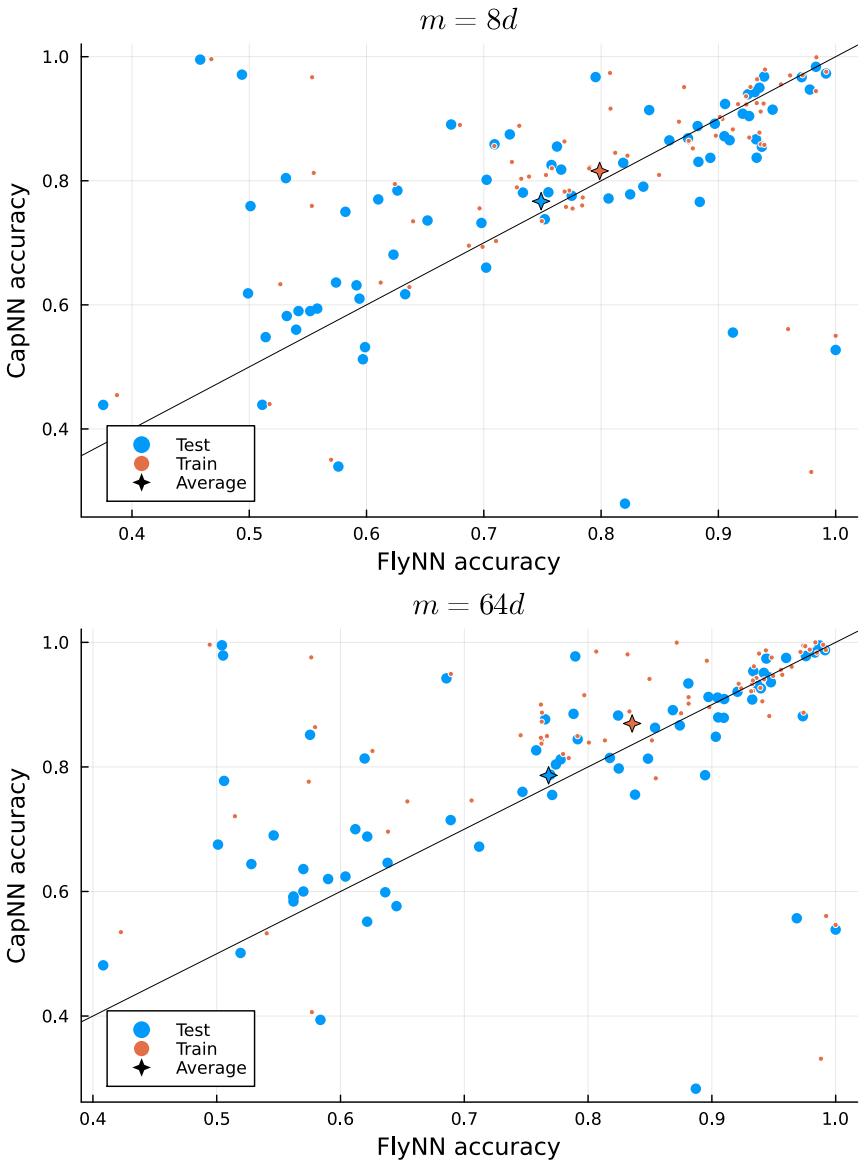

Figure 1: Best average sparsity among all tested hash densities for the 71 OpenML datasets. The stars indicate the centre of mass of the point clouds of the same colour. Averages are computed over 5 seeds.

We can see from the point cloud that CapNN has a slight advantage over FlyNN on average. Table 1 reports some other statistics that indicate the same. However, we only claim that the performance of the methods is comparable and figure 1 shows high variance across datasets. Thus, in table 1 we include the p-values of the paired 2-sided t-test whose null hypothesis states that the means of the two methods are equal. Assuming a significance level of 0.05, in none of the cases we can reject the null hypothesis. Thus,

m	Frac	W/T/L	Diff	TT
d	0.61	43/1/27	0.024	1.69e-01
$2d$	0.61	43/0/28	0.019	2.94e-01
$4d$	0.58	41/1/29	0.017	3.40e-01
$8d$	0.59	42/0/29	0.018	3.25e-01
$16d$	0.59	42/1/28	0.020	2.69e-01
$32d$	0.62	44/1/26	0.023	2.10e-01
$64d$	0.65	46/1/24	0.025	1.88e-01

Table 1: Evaluation of CapNN and FlyNN average test accuracies for the considered hash lengths (m) on the 71 OpenML datasets with fraction of datasets CapNN outperforms FlyNN (Frac), number of datasets CapNN has wins(W)/ties(T)/losses(L) over FlyNN, average test accuracy difference between CapNN and FlyNN (Diff), and p-values for the paired 2-sided t-test (TT). The relatively high p-values show that the mean results of both methods can be considered similar across all hash lengths tested.

we can assume the methods to be comparable across all hash lengths considered.

5.2 Results on Vision Datasets

Figure 2 shows the average accuracy on FashionMNIST and CIFAR10 for different hash densities.

The plateaus at very low hash densities in the lines associated with FlyNN are due to the fact that we compute ρ using ceiling, so it is equal to one at any sufficiently high sparsity. We see that CapNN outperforms FlyNN on FashionMNIST for both hash lengths considered and both test and train accuracy. On CIFAR10, however, FlyNN has the advantage, especially for test accuracy. We hypothesise that the randomness in the size of the caps of CapNN may harm its performance when the number of caps is very high (almost 200,000 in the largest experiment). Having many relatively large caps may overshadow the effect of the smaller ones. Moreover, while increasing hash sparsity makes it unlikely for large caps to be sampled, for high hash length this may require hash densities so low that the overfitting offsets any gains.

Figure 2 also shows that as sparsity increases (hash density decreases), the accuracy of both methods initially increases up to high sparsity values, and then decreases. Notably, CapNN shows fairly good accuracy even at 50% sparsity, when CapHash is reduced to SimHash. This indicates that the cross-entropy filter can leverage reasonably well dense hash schemes. This contrasts strongly with the behaviour of FlyBloomFilter reported in the previous works Ram and Sinha [1] and Dasgupta et al. [5], which indicated very poor performance when using SimHash as the hash function.

6 Conclusions

In this work, we introduced the CapHash hashing family (section 3.2) and the cross-entropy filter (section 3.3) and combined them into CapNN, a novel solution to the problem of supervised classification in the federated learning setting, recently considered in Ram and Sinha [1]. While CapHash is an instantiation of the well-established Spherical LSH family [7, 8], our specific contributions lie in: (1) demonstrating how very sparse, large spherical LSH codes can effectively replace FlyHash in federated learning applications, and (2) providing the first theoretical statistical justification for the filter-based classification framework through our cross-entropy filter. We show, through extensive experiments, that the resulting CapNN performs competitively to the state-of-the-art while retaining the same structure that can be readily combined with differential privacy techniques [1]. Thus, we believe that our work is an important step towards the development of a

principled theory and better algorithms for privacy-aware supervised classification in the federated learning setting.

On the theoretical side, we derived the cross-entropy filter from a direct application of Naive Bayes; we believe that the sparsity and high dimensionality leveraged by CapNN and FlyNN should allow us to prove, in a precise way, that the hashes behave in a *almost independent* way, and thus provide a stronger theoretical justification for the cross-entropy filter.

On the practical side, the simple use of the rule of succession in CapNN (algorithm 1) meaningfully improved its performance by mitigating the effect of variance around the mean. In future work, we plan to investigate more sophisticated variants of additive smoothing that could lead to substantial progress over the current state-of-the-art. Finally, it would be interesting to further explore the behaviour of CapNN for very large hash lengths (number of caps) and possible ways of improving its performance in this regime. Promising directions could be to normalise the vectors associated with the caps or to consider probabilities conditioned on their norms.

References

[1] Parikshit Ram and Kaushik Sinha. "Federated Nearest Neighbor Classification with a Colony of Fruit-Flies". In: *Thirty-Sixth AAAI Conference on Artificial Intelligence*. AAAI Press, 2022, pp. 8036–8044. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20775.

[2] Peter Kairouz, H. Brendan McMahan, Brendan Avent et al. "Advances and Open Problems in Federated Learning". In: *Found. Trends Mach. Learn.* 14.1-2 (2021), pp. 1–210. DOI: 10.1561/2200000083. URL: https://doi.org/10.1561/2200000083.

[3] Phillipp Schoppmann et al. "Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue". In: *Proceedings on Privacy Enhancing Technologies* 2020.2 (2020), pp. 209–229. DOI: 10.2478/popets-2020-0024. URL: https://doi.org/10.2478/popets-2020-0024.

[4] Sanjoy Dasgupta, Charles F. Stevens and Saket Navlakha. "A neural algorithm for a fundamental computing problem". In: *Science* 358.6364 (2017), pp. 793–796. URL: https://www.science.org/doi/abs/10.1126/science.aam9868.

[5] Sanjoy Dasgupta et al. "A neural data structure for novelty detection". In: *Proceedings of the National Academy of Sciences* 115.51 (2018), pp. 13093–13098. URL: https://doi.org/10.1073/pnas.1814448115.

[6] Anthony Thomas, Sanjoy Dasgupta and Tajana Rosing. "A Theoretical Perspective on Hyperdimensional Computing". In: *Journal of Artificial Intelligence Research* 72 (2021), pp. 215–249. URL: https://doi.org/10.1613/jair.1.12664.

[7] Alexandr Andoni et al. "Practical and Optimal LSH for Angular Distance". In: *Advances in Neural Information Processing Systems 28*. 2015, pp. 1225–1233. URL: https://proceedings.neurips.cc/paper/2015/file/2823f4797102ce1a1aec05359cc16dd9-Paper.pdf.

[8] Alexandr Andoni et al. "Beyond Locality-Sensitive Hashing". In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2014, pp. 1018–1028. URL: https://doi.org/10.1137/1.9781611973402.76.

[9] Jure Leskovec, Anand Rajaraman and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed*. Cambridge University Press, 2014. ISBN: 978-1107077232. URL: http://www.mmds.org/.

[10] Moses Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing*. ACM, 2002, pp. 380–388. URL: https://doi.org/10.1145/509907.509965.

[11] Anja Becker et al. “New Directions in Nearest Neighbor Searching with Applications to Lattice Sieving”. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*. ACM-SIAM, 2016, pp. 10–24. URL: https://doi.org/10.1137/1.9781611974331.ch2.

[12] David Karger, Rajeev Motwani and Madhu Sudan. “Approximate graph coloring by semidefinite programming”. In: *J. ACM* 45.2 (Mar. 1998), pp. 246–265. ISSN: 0004-5411. DOI: 10.1145/274787.274791. URL: https://doi.org/10.1145/274787.274791.

[13] Alexandr Andoni and Ilya P. Razenshteyn. “Optimal Data-Dependent Hashing for Approximate Near Neighbors”. In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM, 2015, pp. 793–801. URL: https://doi.org/10.1145/2746539.2746553.

[14] Liudmila Prokhorenkova and Aleksandr Shekhovtsov. “Graph-based Nearest Neighbor Search: From Practice to Theory”. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 7803–7813. URL: http://proceedings.mlr.press/v119/prokhorenkova20a.html.

[15] Jay Yagnik et al. “The power of comparative reasoning”. In: *Proceedings of the IEEE International Conference on Computer Vision, ICCV 2011*. IEEE, 2011, pp. 2431–2438. DOI: 10.1109/ICCV.2011.6126527. URL: https://doi.org/10.1109/ICCV.2011.6126527.

[16] Beidi Chen and Anshumali Shrivastava. “Densified Winner Take All (WTA) Hashing for Sparse Datasets”. In: *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018*. AUAI Press, 2018, pp. 1033–1042. URL: https://auai.org/uai2018/proceedings/papers/321.pdf.

[17] Sanjoy Dasgupta and Christopher Tosh. “Expressivity of expand-and-sparsify representations”. In: *CoRR* abs/2006.03741 (2020). arXiv: 2006.03741. URL: https://arxiv.org/abs/2006.03741.

[18] Yang Shen, Sanjoy Dasgupta and Saket Navlakha. “Algorithmic insights on continual learning from fruit flies”. In: *CoRR* abs/2107.07617 (2021). arXiv: 2107.07617. URL: https://arxiv.org/abs/2107.07617.

[19] Sanjoy Dasgupta, Daisuke Hattori and Saket Navlakha. “A neural theory for counting memories”. In: *Nature Communications* 13.1 (2022), p. 5961. URL: https://doi.org/10.1038/s41467-022-33577-2.

[20] Changyi Ma and Wenye Li. “Sparse Binary Optimization for Text Classification via Frank Wolfe Algorithm”. In: *ICMLC 2020: 2020 12th International Conference on Machine Learning and Computing*. ACM, 2020, pp. 171–176. URL: https://doi.org/10.1145/3383972.3383994.

[21] Chaitanya K. Ryali et al. “Bio-Inspired Hashing for Unsupervised Similarity Search”. In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, 2020, pp. 8295–8306. URL: http://proceedings.mlr.press/v119/ryali20a.html.

[22] Yuchen Liang et al. “Can a Fruit Fly Learn Word Embeddings?” In: *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=xfmSoxdxFCG.

[23] Adam Kirsch and Michael Mitzenmacher. “Distance-Sensitive Bloom Filters”. In: *Algorithm Engineering and Experiments, 8th International Workshop, ALENEX 2006*. Vol. 3919. Lecture Notes in Computer Science. Springer, 2006, pp. 41–50. URL: https://epubs.siam.org/doi/10.1137/1.9781611972863.4.

[24] Yu Hua et al. “Locality-Sensitive Bloom Filter for Approximate Membership Query”. In: *IEEE Trans. Computers* 61.6 (2012), pp. 817–830. DOI: 10.1109/TC.2011.108. URL: https://doi.org/10.1109/TC.2011.108.

[25] Jiangbo Qian, Qiang Zhu and Huahui Chen. "Multi-Granularity Locality-Sensitive Bloom Filter". In: *IEEE Trans. Computers* 64.12 (2015), pp. 3500–3514. DOI: 10.1109/TC.2015.2401011. URL: https://doi.org/10.1109/TC.2015.2401011.

[26] Yang Shen, Sanjoy Dasgupta and Saket Navlakha. "Habituation as a neural algorithm for online odor discrimination". In: *Proceedings of the National Academy of Sciences* 117.22 (2020), pp. 12402–12410. URL: https://doi.org/10.1073/pnas.1915252117.

[27] Kaushik Sinha and Parikshit Ram. "Fruit-fly Inspired Neighborhood Encoding for Classification". In: *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2021, pp. 1470–1480. URL: https://doi.org/10.1145/3447548.3467246.

[28] Joshua Engels, Benjamin Coleman and Anshumali Shrivastava. "Practical Near Neighbor Search via Group Testing". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*. 2021, pp. 27090–27102. URL: https://proceedings.neurips.cc/paper/2021/hash/5248e5118c84beea359b6ea385393661-Abstract.html.

[29] Neel Guha, Ameet Talwalkar and Virginia Smith. "One-shot federated learning". In: *arXiv preprint arXiv:1902.11175* (2019). URL: https://arxiv.org/abs/1902.11175.

[30] Qinbin Li, Bingsheng He and Dawn Song. "Practical One-Shot Federated Learning for Cross-Silo Setting". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 1484–1490. URL: https://doi.org/10.24963/ijcai.2021/205.

[31] Yanlin Zhou et al. "Distilled One-Shot Federated Learning". In: *ArXiv* abs/2009.07999 (2020). URL: https://arxiv.org/abs/2009.07999.

[32] Jie Zhang et al. "DENSE: Data-Free One-Shot Federated Learning". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=QFQoxCFYEkA.

[33] William B. Johnson. "Extensions of Lipschitz mappings into Hilbert space". In: *Contemporary mathematics* 26 (1984), pp. 189–206. URL: https://doi.org/10.1090/conm/026/737400.

[34] Yaniv Plan and Roman Vershynin. "Dimension Reduction by Random Hyperplane Tessellations". In: *Discrete & Computational Geometry* 51 (2013), pp. 438–461. URL: https://doi.org/10.1007/s00454-013-9561-6.

[35] Daisuke Hattori et al. "Representations of Novelty and Familiarity in a Mushroom Body Compartment". In: *Cell* 169 (May 2017), pp. 956–969. URL: https://doi.org/10.1016/j.cell.2017.04.028.

[36] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (1970), pp. 422–426. DOI: 10.1145/362686.362692. URL: https://doi.org/10.1145/362686.362692.

[37] Yonina C. Eldar and Gitta Kutyniok, eds. *Compressed Sensing*. Cambridge University Press, 2012. ISBN: 9780511794308. URL: https://doi.org/10.1017/cbo9780511794308.

[38] Joaquin Vanschoren et al. "OpenML: networked science in machine learning". In: *SIGKDD Explorations* 15.2 (2013), pp. 49–60. URL: http://doi.acm.org/10.1145/2641190.264119.

[39] Han Xiao, Kashif Rasul and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *ArXiv* (28th Aug. 2017). arXiv: cs.LG/1708.07747 [cs.LG]. URL: https://arxiv.org/abs/1708.07747.

[40] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

A Details on experiments

We performed the experiments on a cluster with a heterogeneous allocation of resources. The jobs were allocated to two machines with the following configurations:

- 2 Intel® Xeon® Gold 6240 CPU @ 2.60GHz (18 cores each) and 196 GB of RAM;
- 2 AMD® EPYC® 7542 32-Core Processor @ 2.60GHz (32 cores each) and 1024 GB of RAM.

A list of the OpenML datasets used in this work is given in Table 2 which also shows some of their properties.

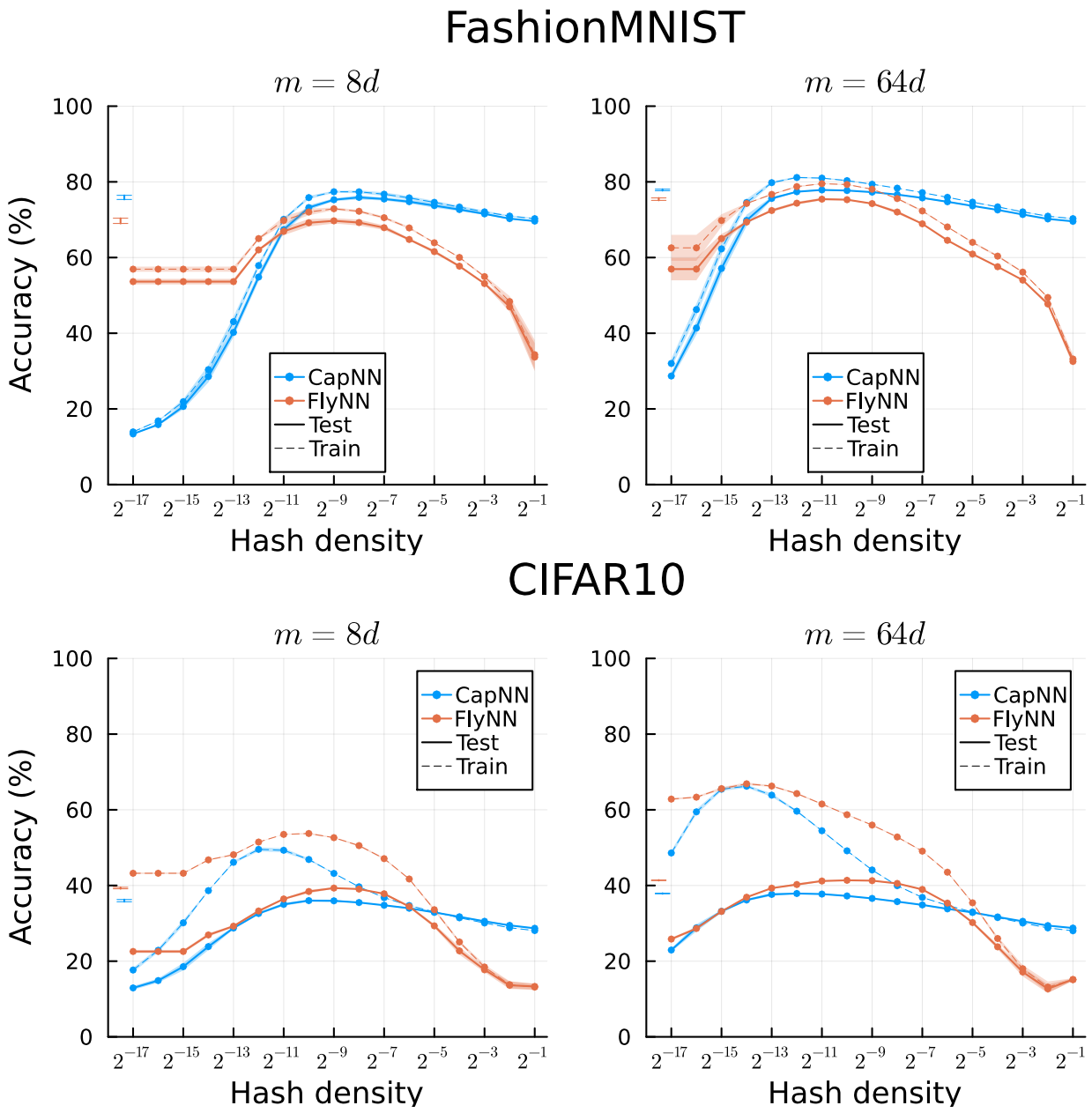

Figure 2: Average accuracy on FashionMNIST (first row, $d = 28 \times 28 \times 1 = 784$) and CIFAR10 (second row, $d = 32 \times 32 \times 3 = 3072$) datasets for hash densities 2^{-i} for $1 \leq i \leq 17$ (in the range of 50% to $\approx 0.00076\%$). Averages are computed over 3 seeds and shaded areas indicate the standard deviation. Bars on the left of each plot indicate the maximum accuracy obtained and its standard deviation.

Name	ID	L	n	$d+1$
mfeat-fourier	14	10	2000	77
mfeat-karhunen	16	10	2000	65
mfeat-zernike	22	10	2000	48
optdigits	28	10	5620	65
spambase	44	2	4601	58
waveform-5000	60	3	5000	41
satimage	182	6	6430	37
fri-c3-1000-25	715	2	1000	26
fri-c4-1000-100	718	2	1000	101
pol	722	2	15000	49
fri-c4-1000-25	723	2	1000	26
ailerons	734	2	13750	41
puma32H	752	2	8192	33
cpu-act	761	2	8192	22
fri-c4-1000-50	797	2	1000	51
fri-c3-1000-50	806	2	1000	51
bank32nh	833	2	8192	33
fri-c1-1000-50	837	2	1000	51
fri-c0-1000-25	849	2	1000	26
fri-c2-1000-50	866	2	1000	51
fri-c2-1000-25	903	2	1000	26
fri-c0-1000-50	904	2	1000	51
fri-c1-1000-25	917	2	1000	26
mfeat-fourier	971	2	2000	77
waveform-5000	979	2	5000	41
optdigits	980	2	5620	65
mfeat-zernike	995	2	2000	48
mfeat-karhunen	1020	2	2000	65
pc4	1049	2	1458	38
pc3	1050	2	1563	38
kc1	1067	2	2109	22
pc1	1068	2	1109	22
PizzaCutter3	1444	2	1043	38
PieChart3	1453	2	1077	38
cardiotocography	1466	10	2126	36
1st-order-theorem-proving	1475	6	6118	52
hill-valley	1479	2	1212	101
ozone-level-8hr	1487	2	2534	73
qsar-biodeg	1494	2	1055	42
⋮	⋮	⋮	⋮	⋮

Table 2: OpenML datasets used in this work and in Ram and Sinha [1]. For each dataset, we report the OpenML ID code (ID), number of classes (L), number of samples (n), and number of features ($d+1$ since the label dimension is counted in the OpenML platform).

Name	ID	L	n	$d+1$
⋮	⋮	⋮	⋮	⋮
ringnorm	1496	2	7400	21
wall-robot-navigation	1497	4	5456	25
steel-plates-fault	1504	2	1941	34
twonorm	1507	2	7400	21
autoUniv-au1-1000	1547	2	1000	21
cardiotocography	1560	3	2126	36
hill-valley	1566	2	1212	101
GesturePhaseSeg.	4538	5	9873	33
thyroid-ann	40497	3	3772	22
texture	40499	11	5500	41
Satellite	40900	2	5100	37
steel-plates-fault	40982	7	1941	28
sylvine	41146	2	5124	21
ada	41156	2	4147	49
microaggregation2	41671	5	20000	21
Sick-numeric	41946	2	3772	30
mfeat-factors	12	10	2000	217
isolet	300	26	7797	618
mfeat-factors	978	2	2000	217
gina-agnostic	1038	2	3468	971
gina-prior2	1041	10	3468	785
gina-prior	1042	2	3468	785
cnae-9	1468	9	1080	857
madelon	1485	2	2600	501
semeion	1501	10	1593	257
clean2	40666	2	6598	169
Speech	40910	2	3686	401
mfeat-pixel	40979	10	2000	241
USPS	41082	10	9298	257
madeline	41144	2	3140	260
philippine	41145	2	5832	309
gina	41158	2	3153	971

Table 2 (continued).