Fast One-class Classification using Class Boundary-preserving Random Projections

Arindam Bhattacharya
Dept. of Computer Science, IIT Delhi
Delhi, India
arindam@cse.iitd.ac.in

Sumanth Varambally
Dept. of Mathematics, IIT Delhi
Delhi, India
mt6170855@maths.iitd.ac.in

Amitabha Bagchi
Dept. of Computer Science, IIT Delhi
Delhi, India
bagchi@cse.iitd.ac.in

Srikanta Bedathur
Dept. of Computer Science, IIT Delhi
Delhi, India
srikanta@cse.iitd.ac.in

ABSTRACT

Several applications, like malicious URL detection and web spam detection, require classification on very high-dimensional data. In such cases anomalous data is hard to find but normal data is easily available. As such it is increasingly common to use a *one-class classifier* (OCC). Unfortunately, most OCC algorithms cannot scale to datasets with extremely high dimensions. In this paper, we present Fast Random projection-based One-Class Classification (FROCC), an extremely efficient, scalable and easily parallelizable method for one-class classification with provable theoretical guarantees. Our method is based on the simple idea of transforming the training data by projecting it onto a set of random unit vectors that are chosen uniformly and independently from the unit sphere, and bounding the regions based on separation of the data. FROCC can be naturally extended with kernels. We provide a new theoretical framework to prove that that FROCC generalizes well in the sense that it is stable and has low bias for some parameter settings. We then develop a fast scalable approximation of FROCC using vectorization, exploiting data sparsity and parallelism to develop a new implementation called ParDFROCC. ParDFROCC achieves up to 2 percent points better ROC than the next best baseline, with up to 12× speedup in training and test times over a range of state-of-the-art benchmarks for the OCC task.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning by classification**; *Anomaly detection*; *Ensemble methods*; • **Theory of computation** → *Sample complexity and generalization bounds*; Kernel methods.

KEYWORDS

one class classification; ensemble classifier; random projection; kernel based method

ACM Reference Format:
Arindam Bhattacharya, Sumanth Varambally, Amitabha Bagchi, and Srikanta Bedathur. 2021. Fast One-class Classification using Class Boundary-preserving Random Projections. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467440

1 INTRODUCTION

The One-class Classification (OCC) task finds wide applicability in real-world situations where it is difficult or expensive to obtain labeled anomalous data or the anomalous data points change over time, whereas obtaining accurate normal samples is relatively easy. Examples of such situations are network intrusion detection [11] and fraud detection [43]. Many of these situations involve extremely high-dimensional raw data, e.g., unusual consumer behavior detection where a very large number of commodities results in very high dimensions. In cases like spam filtering [41] and malicious website detection [25] characteristics like origin, route, text content etc. raise the dimensionality of the data to very high levels. In each of these scenarios a one class classifier is trained on the *normal* data and, when a test point is provided, the model decides if the point is normal or an *outlier*.

Primary approaches to OCC include one-class variants of support vector machines (SVM) [36], probabilistic density estimators describing the support of normal class [5, 34], and recently, deep-learning based methods such as Generative Adversarial Networks (GAN) and Variational Auto-encoders (VAE) which try to learn the distribution of normal data samples during training [4, 29, 30, 35]. One of the main challenges these extant state-of-the-art methods encounter is the question of how to scale with dimension for extremely high-dimensional data. As dimensionality increases training time goes up for these methods, and, more importantly, their test performance scales poorly with dimensionality, making them unsuitable for applications such as the ones discussed above.

In this paper, we present an alternative approach based on the idea of using a collection of random projections onto a number of randomly chosen 1-dimensional subspaces (vectors). Random projections down to a single dimension or to a small number of dimensions have been used in the past to speed up approximate nearest neighbor search [2, 15, 18] and other problems that rely on the distance-preserving property guaranteed by the Johnson-Lindenstrauss Lemma but our problem space is different. We use random projections in an entirely novel fashion based on the following key insight: for the one-class classification problem, the preservation of distances is *not* important, we only require the class boundary to be preserved. The preservation of distances is a *stronger* property which implies that the class boundary is preserved, but the converse need not hold. We don't need to preserve distances. All we need to do is "view" the training data from a "direction" that shows that the outlier is separated from the data.

Formalizing this intuition, we develop an algorithm for one-class classification called FROCC (Fast Random projection-based One-Class Classification) that essentially applies a simplified envelope method to random projections of training data to provide a very strong classifier. FROCC also comes with a theoretically provable generalization property: the method is stable and has low bias.

While the simplicity of FROCC allows it to be extremely fast, extremely high dimensional datasets, that of the order of 10M features, such as many natural language datasets, were still found to be significantly slow to train. To allow FROCC to scale to extreme dimensions, we developed ParDFROCC, an extremely fast, highly scalable (both in dimensions and number of training data points) parallelized approximation of FROCC. The approximation is achieved by discretizing the real space into bins, which allows for fast vectorized operations. ParDFROCC achieves nearly four-fold speed up over FROCC, with nearly no loss in quality of classification. Further, we beat state of the art deep learning and SVM baselines in scalability, speed and classification accuracy on real-world benchmarks.

1.1 Our contributions

In this paper, we propose a novel approach to one class classification: (1) We define a powerful and efficient random-projection based one class classifier named FROCC (Sec. 3). (2) We prove mathematically that for certain settings FROCC is a stable classification algorithm with low bias (Sec. 4). (3) We also propose fast approximations and parallelization to scale FROCC to extremely high-dimensional large datasets (Sec 5), namely: (a) Discretization and vectorization of FROCC resulting in DFROCC (Sec 5.1), (b) Use of sparse projection vectors, first introduced by Achlioptas [1], resulting in SparseDFROCC (Sec 5.2), and (c) A multi-core parallel implementation of SparseDFROCC that results in ParDFROCC (Sec 5.3). (4) We conduct an extensive empirical comparison with a range of state-of-the-art baselines, including the recent deep learning based models, and show that FROCC-family of methods offer significant advantages in terms of both one-class classification accuracy as well as computational cost.

2 RELATED WORK

One Class Classification. Khan and Madden [17] provide a detailed taxonomy of OCC tasks and learning algorithms prior to the advent of deep learning methods. These traditional methods are classified into OCSVM based approaches such as Schölkopf et al. [36] and Tax and Duin [40], and Non-OCSVM based approaches such as Liu et al. [24]. Pang et al. [28] provide a survey of recent advances in OCC using deep learning, with methods such as Ruff et al. [35], Burgess et al. [4], and Perera and Patel [30]. Unlike deep learning methods ParDFROCC does not rely on optimization for learning a representation of the data. This lends ParDFROCC an efficiency benefit over these methods.

Random Projection. Random projection is widely used for high-dimensional data due to the Johnson-Lindenstrauss property, i.e., random projections preserve distances. This makes it an attractive choice for nearest neighbor algorithms and locality sensitive hashing [2, 14]. Random projection methods are also employed for one class classification in Fowler and Du [10] and de Vries et al. [6]. LODA [31] also utilizes random projections on top a histogram based method such as Goldstein and Dengel [12] to provide an efficient algorithm for OCC. Unlike [31] which still has to rely on histogram calculation, ParDFROCC employs a simpler boundary generation technique which is not only faster, but allows for efficient parallel implementation.

Sparse Random Projection. The efficiency of random projections can be further enhanced by using Sparse Random Projections [1]. Unlike most methods that use random projection for representation alone, ParDFROCC can exploit the sparsity of the random projections to a greater degree owing to the fact that the only processing involved in computing ParDFROCC are vector multiplications and additions.

3 FAST RANDOM-PROJECTION BASED OCC

Definition 3.1 (FROCC (S, m, ε, K)). Assume that we are given a set of training points $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$. Then, given an integer parameter $m > 0$, a real parameter $\varepsilon \in (0, 1]$ and a kernel function $K(\cdot, \cdot)$, the *ε-separated Fast Random-projection based One-Class Classifier* FROCC (S, m, ε, K), comprises, for each i, $1 \leq i \leq m$,

(1) A *classifying direction* $\boldsymbol{w}_i$ that is a unit vector chosen uniformly at random from $\mathbf{1}_d$, the set of all unit vectors of $\mathbb{R}^d$, independent of all other classifying directions, and
(2) a set of intervals R_i defined as follows: Let $S_i' = \{K(\boldsymbol{w}_i, \boldsymbol{x}_j) : 1 \leq j \leq n\}$ and $S_i = \{y_1, \ldots, y_n\}$ is a shifted and scaled version of S_i' such that $y_i \in (0, 1), 1 \leq i \leq n$. Assume $y_1 \leq \cdots \leq y_n$. Then each interval of R_i has the property that it is of the form $[y_j, y_{j+k}]$ for some $j \geq 1, k \geq 0$ such that
   (a) for all t such that $0 \leq t < k$, $y_{j+t+1} - y_{j+t} < \varepsilon$,
   (b) $y_j - y_{j-1} > \varepsilon$ whenever $j - 1 > 0$, and
   (c) $y_{j+k+1} - y_{j+k} > \varepsilon$ whenever $j + k + 1 \leq n$.

Given a query point $\boldsymbol{y} \in \mathbb{R}^d$, FROCC (S, m, ε, K) returns YES if for every i, $1 \leq i \leq m$, $K(\boldsymbol{w}_i, \boldsymbol{y})$ lies within some interval of R_i.

In simple terms the intervals of R_i have the property that the points of S_i are densely scattered within each interval and there are wide gaps between the intervals that are empty of points of S_i. The density of the intervals is lower bounded by $1/\varepsilon$ and the width between successive intervals is lower bounded by ε. Setting $\varepsilon = 1$ gives us the special case where the entire normalized interval $[0, 1]$ is an inlier interval. This corresponds to the entire interval from $\max_{i=1}^{n} S_i$ to $\min_{i=1}^{n} S_i$ prior to normalization. In the simplest setting the kernel function $K(\cdot, \cdot)$ will be just the usual dot product associated with $\mathbb{R}^d$.

We show a small example in Figure 1. The light green points are training points. Two classifying directions $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are displayed. If we choose $\varepsilon < \varepsilon_d$, we have intervals $[a, b]$ and $[c, d]$ on $\boldsymbol{w}_1$. We demonstrate the decision process of FROCC using three test points

Algorithm 1: FROCC Training

```
Data: $S = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$: training set
input  : $m, \varepsilon$
output : $[R_i]_{i=1}^m$: list of $m$ interval vectors
1  Initialize $result \leftarrow$ empty list
2  for $i \leftarrow 1$ to $m$ do
3      $w_i \sim Uniform(1_d)$ // sample a d-dimensional unit vector
           uniformly
4      Initialize $projectionList \leftarrow$ empty list
5      for $j \leftarrow 1$ to $n$ do
6          $projection \leftarrow f(x_j, w_i)$
7          $projectionList.append(projection)$
8      end
9      $R_i \leftarrow buildIntervalVector(projectionList)$
10     result.append($R_i$)
11 end
```

$\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3$. $\boldsymbol{y}_1$'s projection on $\boldsymbol{w}_2$ falls within the interval $[e, f]$, but its projection on $\boldsymbol{w}_1$ falls outside of the two intervals. Thus it is correctly classified as an outlier. $\boldsymbol{y}_2$'s projection on $\boldsymbol{w}_1$ falls within the interval $[c, d]$ but the projection on $\boldsymbol{w}_2$ lies outside the interval $[e, f]$, correctly classifying it as an outlier. The projections of test point y_3 on both vectors lie within an interval, and so it is correctly classified as an inlier.

Computing FROCC. Algorithm 1 outlines the training procedure of FROCC. The function f in Line 6 is any valid kernel, which defaults to inner product $\langle \cdot, \cdot \rangle$. Once we create a *projectionList*, we generate an interval vector in Line 9, that separates the projections such that the inter-interval distance is at most ε and the distance between the intervals is at least ε. Unlike many optimization based learning methods, such as neural networks, our algorithm needs a single pass through the data.

The standard method for choosing uniformly random vectors in $\mathbb{R}^d$ is to sample from a spherical d-dimensional Gaussian centered at the origin and then normalizing it so that its length is 1. A quick calculation suggests that independently picking d random variables with distribution $N(0, 1)$ is good enough for this purpose. The spherical symmetry of the Gaussian distribution ensures that the random vector is uniformly picked from all possible directions.

Time complexity. The time taken to compute the boundaries of the *FROCC* is $O(mnd)$ for $\varepsilon = 1$ (assuming the kernel is dot product, which takes $O(d)$). For $\varepsilon < 1$ an additional factor of $\theta(mn \log n)$ is required to create ε-separated intervals along each classifying direction.

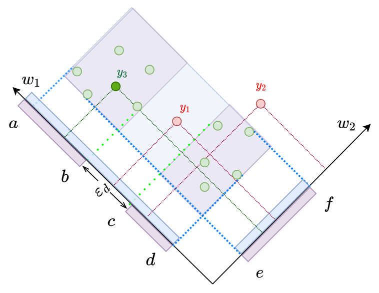


Figure 1: Decision process example: $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are identified as outliers by vectors $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ respectively, while y_3 is correctly identified as an inlier.

4 STABILITY OF FROCC

We now present our main theoretical result, Thm 4.6, which shows that under some mild assumptions the bias of FROCC with $\varepsilon = 1$ goes to 0 at an exponential rate as the size of the training set increases.

We follow the terminology defined in [3]. Let $\mathcal{S}$ denote the set of all finite labelled point sets of $\mathbb{R}^d$ where each point is labelled either 0 or 1. We assume we have a training set $S \in \mathcal{S}$ chosen randomly as follows: We are given an unknown distribution D over $\mathbb{R}^d \times \{0, 1\}$ and S comprises k samples with label 1 drawn i.i.d. from D. For a point $z = (x, y) \in \mathbb{R}^d \times \{0, 1\}$ we use $data(z)$ to denote x and $lab(z)$ to denote y. Let $\mathcal{R}$ be the set of all finite strings on some finite alphabet, and let us call the elements of $\mathcal{R}$ *decision strings*. Further, let $\mathcal{F}$ be the set of all *classification functions* from $\mathbb{R}^d$ to $\{0, 1\}$. Then we say a classification map $\Phi : \mathcal{S} \times \mathcal{R} \to \mathcal{F}$ maps a training set and a decision string to a classification algorithm.

With this setup we say that a *randomized classification algorithm* A takes a training $S \in \mathcal{S}$ as input, picks a random decision string $r \in \mathcal{R}$ and returns the classification function $\Phi(S, r)$ which we denote $A(S, r)$. $A(S, r)$ is a randomly chosen element of $\mathcal{F}$ which we call a classifier. Given an r, $A(S, r)$ is fixed but we will use A_S to denote the randomized classifier which has been given its training set S but is yet to pick a random decision string. Now, given a $z \in \mathbb{R}^d$, the *loss* function of a randomized classifier A_S is given by

$$V(A_S, z) = \mathrm{E}_r \left\{ 1_{A(S,r)(data(z)) \neq lab(z)} \right\},$$

where the expectation is over the randomness of the decision string r. In other words the loss is equal to $\Pr_r \{A(S, r)(data(z)) \neq lab(z)\}$. We define the *risk* of A_S as

$$R(A_S) = \mathrm{E}_{S,z} \{V(A_S, z)\},$$

where the expectation is over the random choice of S and of the point z. Note that $V(\cdot, \cdot)$ already contains an expectation on the randomness associated with the decision string which is inherent in A_S. The *empirical risk* of A_S is defined as

$$R_e(A_S) = \frac{1}{|S|} \sum_{z \in S} V(A_S, z).$$

Also, for $1 \leq i \leq |S|$ we say that $S^{\setminus i} = S \setminus \{z_i\}$ and $S^i = S^{\setminus i} \cup \{z_i'\}$ where z_i' is chosen from D independently of all previous choices. We now define a notion of stability that is a modification of the definition of stability for classification algorithms given by [3].

Definition 4.1 (0-1 uniform classification stability). Suppose we have a randomized classification algorithm A and a loss function $V(\cdot, \cdot)$ whose co-domain is $[0, 1]$. Then A is said to have *0-1 uniform classification stability* (β, η) if for all $S \subseteq \mathbb{R}^d$, for every i such that $1 \leq i \leq |S|$, and for every $z \in \mathbb{R}^d \times \{0, 1\}$

$$|V(A_S, z) - V(A_{S^{\setminus i}}, z)| \leq \beta$$

with probability at least $1 - \eta$.

If β is $O(1/|S|)$ and η is $O(e^{-c|S|})$ for some constant $c > 0$, we say that A is *0-1 uniform classification stable.*

Next we prove that a 0-1 uniform classification stable algorithm has low bias and converges exponentially fast in the size of S to its expected behavior. This is a general result that may be applicable to a large class of randomized classification algorithms.

Theorem 4.2. *Suppose we have a randomized classification algorithm A which has 0-1 uniform classification stability (β, η) with $0 < \beta, \eta < 1$, and suppose this algorithm is trained on a set S drawn i.i.d. from a hidden distribution in such a way that the random choices made by A are independent of S, then, for any $\alpha > 0$,*

$$Pr\{|R(A,S) - R_e(A,S)| \geq \alpha\} \leq 2\exp\left\{-\frac{2\alpha^2}{|S|\beta^2}\right\} + 2|S|\eta. \quad (1)$$

Moreover if A is 0-1 uniform classification stable then the RHS of (1) *tends to 0 at a rate exponential in $|S|$.*

Proof. As in [3] we will use McDiarmid's inequality [26] to bound the deviation of the empirical risk from the risk.

Lemma 4.3 (McDiarmid [26]). *Given a set of m independent random variables Y_i, $1 \leq i \leq m$ and a function f such that $|f(x_1, \dots, x_m) - f(y_1, \dots, y_m)| < c_i$ whenever $x_j = y_j$, $1 \leq j \leq m$, $j \neq i$, and $x_i \neq y_i$,*

$$Pr\{|E\{f(Y_1, \dots, Y_m)\} - f(Y_1, \dots, Y_m)| \geq \alpha\} \leq e^{\left(-\frac{2\alpha^2}{\sum_{i=1}^m c_i^2}\right)}.$$

To use Lemma 4.3 we have to show that $R_e(A,S)$ is a 2β-Lipschitz function. However, this property is only true with a certain probability in our case. In particular we will show that for all i, $1 \leq i \leq |S|$, $|R_e(A,S) - R_e(A,S^i)| \leq 2\beta$ with probability at least $1 - 2|S|\eta$. To see this note we note that for any i and any $z_j \in S$ (including $j = i$).

$$|V(A_S, z_j) - V(A_S^i, z_j)| = |V(A_S, z_j) - V(A_{S\setminus i}, z_j) + V(A_{S\setminus i}, z_j) - V(A_S^i, z_j)|.$$

Using the definition of (β, η) 0-1 uniform classification stability and the triangle inequality, we get that the RHS is bounded by 2β with probability at least $1 - 2\eta$. If A_j is the event that the RHS above is bounded by 2β then we want to bound the probability of the event that this is true for all j, $1 \leq j \leq |S|$, i.e., the event $F = \cap_{j=1}^{|S|} A_j$. Since $\Pr\left\{\overline{A_j}\right\} \leq 2\eta$, we can say that $\Pr\{F\}$ is at least $1 - 2|S|\eta$.

Now, let E be the event that $|R(A,S) - R_e(A,S)| > \varepsilon$ and F defined above be the event that $R_e(A,S)$, which is a function of the vector of $|S|$ elements chosen independently to form S is 2β-Lipschitz. We know that

$$\Pr\{E\} \leq \Pr\{E|F\}\Pr\{F\} + \Pr\left\{\overline{F}\right\}. \quad (2)$$

We have already argued that the second term on the RHS is upper bounded by $2|S|\eta$ so we turn to the first term. To apply Lemma 4.3 to the first term we note that E depends on the random selection of S which are selected independently of each other and, by assumption, independent of the random choices made by A. Therefore the random collection $\{V(A_S, z) : z \in S\}$ is independent even when conditioned on F which is determined purely by the random choices of A and is true for *every* choice of set S. Once this is noted then we can use McDiarmid's inequality to bound the first term of Equation (2). We ignore $\Pr\{F\}$ by upper bounding it by 1. □

We now prove that FROCC with $\varepsilon = 1$ is 0-1 uniform classification stable under a mild condition on the unknown distribution. We call this condition *spatial divisibility.*

Definition 4.4 (Spatial divisibility). Suppose that $\mu : \mathbb{R}^d \longrightarrow [0,1]$ is a probability density function. For any set T containing d points $x_i, \dots, x_d \in \mathbb{R}^d$ let H_{T+} and H_{T-} be the two half-spaces defined by the hyper-plane containing all the points of T. We say that μ is *spatially divisible* if for any set $d+1$ randomly chosen points $X = \{X_1, \dots, X_d\}$ and Y chosen independently according to density μ and any $A, B \subset \mathbb{R}^d$ such that $\mu(A), \mu(B) > 0$,

$$\Pr\{Y \in H_{X+} | X_i \in A, Y \in B\} > 0, \text{and}$$
$$\Pr\{Y \in H_{X-} | X_i \in A, Y \in B\} > 0.$$

Note that the set X defines a hyper-plane in d dimensions with probability 1 for any distribution defined on a non-empty volume of $\mathbb{R}^d$. Most standard distributions have the spatial divisibility property. For example, it is easy to see that a multi-variate Gaussian distribution has this property. If we pick a point uniformly at random from within a convex d-polytope then too the spatial divisibility property is satisfied. To see this we note that the only way it could not be satisfied is if the points X_i and Y are picked from the surface of the d-polytope which is an event of probability 0.

We now show that Theorem 4.2 applies to FROCC with $\varepsilon = 1$ if the unknown distribution D is spatially divisible.

Proposition 4.5. *If the unknown distribution D is spatially divisible then FROCC with $\varepsilon = 1$ is 0-1 uniform classification stable.*

Proof. For any i, $1 \leq i \leq n$, where $n = |S|$, we note that when $\varepsilon = 1$ the entire interval spanned by the projections in any direction is said to contain inliers, and so if x_i lies strictly within the convex hull of S then the removal of i from S does not affect the classifier. This is because the boundaries in any direction are determined by the points that are part of the convex hull. Let us denote this set conv(S). Therefore, since $V(\cdot,\cdot)$ takes value at most 1, we deduce the following upper bound

$$|V(A_S, z) - V(A_{S\setminus i}, z)| \leq \Pr\{x_i \in \text{conv}(S)\}.$$

To bound the probability that a point x_i lies in the convex hull of S we use an argument that Efron [8] attributes to Rényi and Sulanke [32, 33]. Given a region $B \subset \mathbb{R}^d$ let us denote by $D(B)$ the probability that a point drawn from the unknown distribution D places a point in B. Now given any d points, say $x_1, \dots, x_d \in S$, we divide $\mathbb{R}^d$ into two regions A_{X+} that lies on one side of the hyper-plane defined by $x_1, \dots, x_d$ and A_{X-} that lies on the other side. Then the probability that $x_1, \dots, x_d$ all lie in conv(S) is equal to the probability that all the remaining points of S are either in A_{X+} or in A_{X-}, i.e.,

$$\Pr\{x_1, \dots, x_d \in \text{conv}(S)\} = D(A_{X+})^{(n-d)} + D(A_{X-})^{(n-d)}.$$

From this, using the union bound we can say that

$$\Pr\{x_i \in \text{conv}(S)\} \leq \sum_{Y \in \mathcal{P}(S\setminus\{x_i\},d)} D(A_{Y+})^{(n-d)} + D(A_{Y-})^{(n-d)},$$

where $\mathcal{P}(A,d)$ is the set of all subsets of A of size d. This can further be bounded as

$$LHS \leq 2\binom{n-1}{d} \max_{Y \in \mathcal{P}(S\setminus\{x_i\},d)} \max\{D(A_{Y+})^{(n-d)}, D(A_{Y-})^{(n-d)}\}.$$

Since we have assumed that $D(A_{Y\pm}) < 1$ for all $Y \subset S$ such that $|Y| = d$ and $x_i \notin Y$, using the fact that $\binom{n}{k} \leq (en/k)^k$, we can say that there is an α such that $0 < \alpha < 1$ and a constant $C > 0$ such that

$$|V(A_S, z) - V(A_{S\setminus i}, z)| \leq C \cdot (n-1)^d \alpha^n$$

almost surely. Since the RHS goes to 0 exponentially fast in n with probability 1, we can say that FROCC is 0-1 uniform classification stable when $\varepsilon = 1$. □

From Theorem 4.2 and Proposition 4.5 we get

THEOREM 4.6. *For a training set S chosen i.i.d. from a spatially divisible unknown distribution, if ε is set to 1 then $R_e(FROCC, S)$ converges in probability to $R(FROCC, S)$ exponentially fast in $|S|$.*

We feel that it should be possible to use the framework provided by Theorem 4.2 to show that FROCC is stable even when ε lies in $(0, 1)$, however this problem remains open.

5 SCALING FROCC

In this section we show how to scale FROCC so that it can be trained on large datasets containing features in the order of 10M in minutes. We first describe a discretized version of the algorithm, DFROCC, (Sec. 5.1), then show how we can speed up DFROCC by introducing sparsity into our random vectors (Sec. 5.2), and finally describe ParDFROCC which is a parallelized version of DFROCC (Sec. 5.3).

5.1 DFROCC: Discretizing FROCC

Our key insight is this: *A collection of disjoint intervals can be represented by an array with 0s and 1s as entries.* To do this we discretize the real line into segments of a fixed length, placing a 1 for every segment that lies within an interval and 0 for a segment that lies outside all intervals. We call the discretized algorithm DFROCC.

Definition 5.1 (DFROCC $(S, m, \varepsilon, \gamma, K)$). Assume that we are given a set of training points $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$. Then, given an integer parameter $m > 0$, a real parameter $\varepsilon > 0$, an integer *discretization parameter* $\gamma > 1$ and a kernel function $K(\cdot, \cdot)$, the *ε-separated Discrete Fast Random-projection based One-Class Classifier* DFROCC $(S, m, \varepsilon, b, K)$, comprises, for each i, $1 \le i \le m$,

1. A *classifying direction* $\boldsymbol{w}_i$ which is a vector chosen randomly from a spherically symmetric distribution in $\mathbb{R}^d$ independent of all other classifying directions, and
2. a boolean array B_i (called *interval vector*) of size $N = \frac{\gamma}{\varepsilon}$ defined as follows: Let $S_i = \{K(\boldsymbol{w}_i, \boldsymbol{x}_j) : 1 \le j \le n\}$, where the projections are normalized to lie in $[0, 1]$. Name the elements of S_i as $y_1, \ldots, y_n$. Then (a) The cell $B_i[k]$ is set to 1, where $k = \lfloor N y_j \rfloor$ for $1 \le j \le n$, is the *normalized discrete projection* of y_j, and (b) if the $B_i[k_1]$ and $B_i[k_2]$ are marked as 1 such that $k_1 < k_2$ and $k_2 - k_1 \le \gamma$, then for all $\hat{k}$ such that $k_1 \le \hat{k} \le k_2$, $B_i[\hat{k}]$ is marked as 1.

Given a query point $\boldsymbol{y} \in \mathbb{R}^d$ the classifier returns a YES answer if the normalized discrete projection of $\boldsymbol{y}$ in each classifying direction's interval vector is 1. Otherwise it returns NO.

Note that in the above definition γ is our discretization parameter that determines how many bins the projection range is divided into. As $\gamma \to \infty$ the behavior of DFROCC approaches that of FROCC.

In Figure 2 the green points are training points and $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are two classifying directions. The bins shaded in pink and blue are considered inlier bins with respect to the directions $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ respectively. Even though the region δ_1 does not have any projections in it along $\boldsymbol{w}_1$, the bins have been marked as inliers

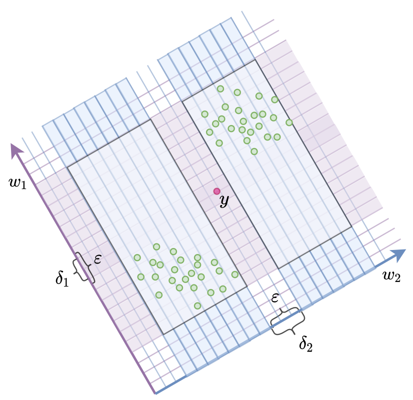


Figure 2: The vector w_1 classifies the test point y incorrectly since the bins in the region δ_1 is marked as positive. However, w_2 correctly classifies y as an outlier since the bins in δ_2 are not marked as inliers

since the two clusters have bins k and k' such that $\frac{|k-k'|}{N} < \varepsilon$. Consequently, the test point $\boldsymbol{q}$ is marked as an inlier by direction $\boldsymbol{w}_1$. However, since the region δ_2 has not been marked along $\boldsymbol{w}_2$, the test point $\boldsymbol{q}$ is correctly classified as an outlier by the classifying direction $\boldsymbol{w}_2$.

5.2 Sparse DFROCC

DFROCC needs to project several d-dimensional data points onto d-dimensional random vectors. This is an operation that can potentially take time $\Theta(d^2)$. To address this problem we introduce sparsity in our random vectors by applying an ingenious solution first presented by Achlioptas [1] in the context of Johnson-Lindenstrauss-style embeddings. We take a parameter $s \in (0, 1)$, the *sparsity* of our random vectors, and choose our vectors according to the following definition: $\boldsymbol{w}_i^{(j)} = +1$ and -1 each with probability $\frac{s}{2}$ and 0 with probability $1 - s$, where $1 \le i \le m$ and $1 \le j \le d$. All dimensions are chosen independent of each other.

Using Achlioptas's distribution we get sparse classification direction vectors that take $\Theta(sdm)$ memory. Calculating the projections takes $\Theta(sdmn)$ time. We get a reduction of both the time and storage complexity by a factor of $1 - s$. The tradeoff here is that when the sparsity is too high we may not learn anything from the data. So s has to be chosen carefully.

It must be noted here that the improvements over Achlioptas's method, such as Li et al. [23] and Kane and Nelson [16], provide better theoretical bounds on the sparsity of random vectors. We experimented with them but rejected them because their additional computational costs outweighed the benefits of extra sparsity.

5.3 ParDFROCC: Parallelizing DFROCC

In ParDFROCC each of the three phases of DFROCC has a parallel step done independently for each batch and an aggregation step which brings together the computations performed in parallel. Figure 3 presents a block diagram for ParDFROCC. We assume we have partitioned the training set S into ℓ batches $S^1, \ldots, S^\ell$.

5.3.1 Random vector generation Phase.

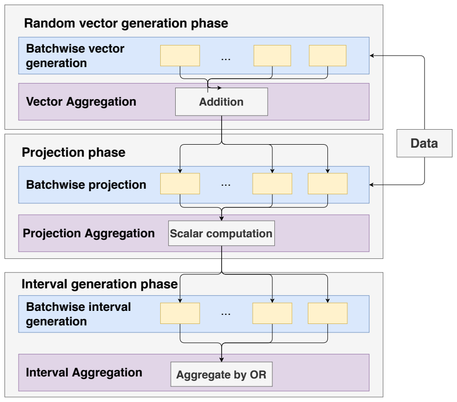


Figure 3: Architecture of ParDFROCC. Blue regions have the parallel steps and purple region have aggregation steps.

Parallel step. For each $b \in \{1, \ldots, \ell\}$ we create m classification direction vectors $\boldsymbol{w}_i^b$, $i = 1, \ldots, m$, independently in parallel. This involves determining which dimensions are non-zero for the training points in S^b, i.e., the j^{th} entry of the classification vectors is generated only if $\boldsymbol{x}[j] \neq 0$ for some $\boldsymbol{x} \in S^b$. Since the non-zero dimensions for one batch may be different from the non-zero dimensions of another we need to aggregate the vectors generated to arrive at the final set of classification directions.

Aggregation step. We aggregate the vectors from each thread to ensure all projections are non-zero on same set of classification directions. We aggregate in a surprisingly simple way: We just set $\boldsymbol{w}_i = \sum_b \boldsymbol{w}_i^b$ for all $1 \leq i \leq m$. Does this work? There are two considerations. Firstly, for correctness the resultant distribution should be spherically symmetrical. Secondly, the sparsity of the aggregated vectors should not decrease too much, i.e., when we add some non-zero entries they should not sum to zero. We now show that this solution works on both these fronts.

Proposition 5.2. *Given an $s \in (0, 1)$ and a collection of ℓ independent random variables $X_1, \ldots, X_\ell$ that take value 0 with probability $1 - s$ and values -1 or 1 with probability $s/2$, if $X = \sum_{i=1}^{\ell} X_i$ then if $s\ell < 4/e$.*

$$Pr\{X < 0\} = Pr\{X > 0\} \geq s\ell \cdot \frac{e}{4\pi}.$$

Proof. Firstly, we note that X is is exactly the position of a lazy random walk on $\mathbb{Z}$ that begins at the origin whose one step distribution stays at the same place with probability $1-s$ and moves with equal probability in either direction otherwise. If this position is non-zero then by symmetry it is equiprobable that it is positive or negative.

Now we upper bound the probability that X is 0. Consider two cases. *Case 1.* Each X_i is 0. This happens with probability $(1 - s)^\ell$. *Case 2.* Not every X_i is 0 but the sum is 0. We use the following lemma:

Lemma 5.3. *Given a collection $X_1, \ldots, X_t$ of random variables that take values uniformly in $\{-1, 1\}$ independently,*

$$P\left(\sum_{i=1}^{t} X_i = 0\right) \leq \frac{e}{\pi\sqrt{t}},$$

if t is even and 0 otherwise.

The proof follows from Eq. (2.19) of [22] and a short calculation. Since in Case 2 the number of non-zero entries is at least 1, we get an upper bound of $\frac{e}{\pi}$ on the probability that X is 0. Putting the cases together we have that

$$\Pr\{X \neq 0\} \geq 1 - (1 - s)^\ell - (1 - (1 - s)^\ell)\frac{e}{\pi}.$$

Since $s < 1$ and we have assumed $s\ell < 4/e$ we use the approximation $(1 - s)^\ell \leq 1 - s + \binom{\ell}{2}s^2$ and the approximation $\binom{\ell}{2} \leq (e\ell/2)^2$ to get the result. □

Now consider any dimension. For simplicity, let us consider coordinate 1 since the argument is similar for all coordinates. Let us say that $N = \{b : \exists \boldsymbol{x} \in S^b$ such that $\boldsymbol{x}[1] \neq 0\}$ is the set of batches for which the 1st coordinate has a non-zero entry for some data point, and hence the 1st coordinate is non-zero in each $\boldsymbol{w}_i^b$ with probability s where s is our sparsity parameter (Sec 5.2). Applying Proposition 5.2 with $\ell = |N|$ we see that the sparsity is maintained if $\ell > 2\pi/e$, i.e. if ℓ is 3 or more. So if three or more batches place a non-zero in the same coordinate, we maintain our target sparsity. The assumption that $s\ell < 4/e$ is very mild since typically we will use s of the order of 10^{-2} and ℓ will be in the range of 10-20, making $s\ell$ less than 1.

5.3.2 *Projection phase.*

Parallel step. In this step, we compute the projections $K(\boldsymbol{x}^b, \boldsymbol{w}^b)$ for each batch b in parallel.

Aggregation step. For each classifying direction we need to compute the maximum and minimum value of the projections of the data points in order to normalize the range for creating the interval vector. This cannot be done in parallel, so we aggregate the projections. We compute the minimum and maximum values and scale the projection to lie in the range $[0, 1]$.

5.3.3 *Interval creation phase.*

Parallel step In this step, we create the interval array B_i^b using the scaled projections. This procedure is performed concurrently on all the batches.

Aggregation step Next we efficiently aggregate the intervals of individual batches. Here again, we find that: $B_i = \bigvee_b B_i^b$. This means that if any one of B_i^b considers an interval $[p, q]$ to be a positive region, B_i also marks the interval $[p, q]$ as positive. This is done efficiently with a vectorized operation.

6 EXPERIMENTS AND RESULTS

We evaluated the performance of the FROCC family of methods on a wide range of real world benchmark datasets and compared them with existing state of the art baselines. We compare the performance using area under the ROC curve — a recommended metric for OCC evaluation [39], denoted as AUC ROC. We also compare the wall-clock times of the training and testing stages for each method. Further, we perform experiments to compare the scalability of the methods, both in terms of number of dimensions and number of training samples. In this section we first present the details of the datasets and baselines used, and then present the results of the empirical evaluation.

Table 1: Dataset statistics. Group A: <100 features; Group B 100 to 1M features; Group C > 1M features. †: $> 10,000$ training samples.

ID	Name	Features	Training	Validation (Equal +/− Split)	Test (Equal +/− Split)
A1	Diabetes [7]	8	268	460	508
A2†	MAGIC Telescope [7]	10	12332	18980	20870
A3†	MiniBooNE [7]	50	18250	24650	30650
A4	Cardiotocography [7]	35	147	294	294
B1†	SATLog-Vehicle [37]	100	24623	30976	36864
B2†	Kitsune Network Attack [27]	115	3018972	1806264	3006490
B3	MNIST [21]	784	3457	4880	5390
B4	CIFAR-10 [19]	3072	3000	3896	5896
B5	CIFAR-100 [19]	3072	300	594	594
B6	GTSRB [38]	3072	284	562	562
B7	Omniglot [20]	33075	250	400	400
C1†	Malicious URLs detection [25]	3.2m	795490	454566	297744
C2†	Webb Spam corpus [41]	16.6m	175000	100000	78410

6.1 Datasets

We evaluate FROCC and the baseline methods on a set of well-known benchmark datasets whose number of features, training / validation / test split statistics are summarized in Table 1. The set of benchmarks we have chosen covers a large spectrum of data characteristics such as the number of samples (from a few hundreds to millions) as well as the number of features (from a handful to tens of millions). Thus it broadly covers the range of scenarios where the OCC task appears.

To adapt multi-class datasets for evaluation in an OCC setting, we perform processing similar to Ruff et al. [35]: Given a dataset with multiple labels, we assign one class the *positive* label and all others the *negative* label. Specifically, we select the positive classes for (i) MNIST: randomly from 0, 1 and 4, (ii) CIFAR 10: airplane, automobile and deer and (iii) CIFAR 100: beaver, motorcycle and fox. We then split the *positive* class instances into *train* and *test* sets. Using labelled data allows us to evaluate the performance using true labels. We *do not* use any labels for training. We double the test set size by adding equal number of instances from the *negative*ly labeled instances. We train the OCC on the *train* set which consists of only *positive* examples. Metrics are calculated on *test* set which consists of equal number of *positive* and *negative* examples.

6.2 Baselines

We compare our methods with the following state of the art algorithms for OCC:

- SVM, tree and random projection based methods: (1) **OCSVM:** One class SVM (Schölkopf et al. [36] version). (2) **Isolation Forest:** Isolation forest algorithm [24] (3) **HBOS:** Histogram-based outlier score [12] (4) **LODA:** Lightweight Online Detector of Anomaly [31]
- Deep neural network based methods: (1) β**-VAE:** VAE based supervised outlier detection (Burgess et al. [4]) (2) **OCGAN:** One Class GAN [30] (3) **DROCC:** Deep Robust One Class Classifier [13]

We implemented the FROCC-family of algorithms using Python 3.8 with the `numpy` library. We used `scikit-learn` implementations of OCSVM and Isolation Forest. The PyOD toolkit [42] was used for HBOS, LODA, β-VAE and OCGAN. The deep neural network based baselines use Tensorflow [9] for their backend. We used the author's implementation of DROCC, released as part of EdgeML library at https://github.com/microsoft/EdgeML. All experiments were conducted on a server with Intel Xeon Gold 6142, and NVIDIA GeForce GTX 1080 Ti GPU, with 400GB of RAM. Only the deep learning based methods −*viz.*, OCGAN, β-VAE and DROCC − require GPU both during training and testing. None of the other methods, including all FROCC methods require GPU.

For each method, hyper-parameters were set using a grid-search over the validation set. The source code of FROCC-family of methods, the hyper-parameters used, and the scripts used in this evaluation are freely available on github[1].

6.3 Results and Discussion

Table 2 shows the `AUC ROC` values of the best performing ParDFROCC along with the baselines for all datasets. As we show in Sec. 6.4, ParDFROCC is not only fast to train and test, but also offered consistently high `AUC ROC` in all our experiments. Specifically, ParDFROCC had only ≈ 0.01 *worse* `AUC ROC` than the best performing FROCC variant across all datasets. Therefore, we present empirical results with ParDFROCC. For low dimensional datasets (marked with ‡ in Table 2) best results are obtained when ParDFROCC uses dense projection vectors ($s = 1$). Higher dimensional data requires more projection vectors (high m) that are sparse ($s = 0.01$). Details of hyper-parameters used are available in the code repository.

From Table 2 we note that ParDFROCC outperforms the baselines in nine of the thirteen datasets we experimented with. Further, we note that for high dimensional datasets −i.e., datasets with $\geq$ *100 features*, it outperforms the baselines in **eight out of nine datasets while ranking second in the remaining one**. This demonstrates the superiority of ParDFROCC for high dimensional data which are known to be challenging for most OCC methods.

It is to be noted that the available implementations of some baselines (marked in Table 2 using ⋆) do not have the ability to handle sparse data directly. They have to convert the dataset into a dense matrix representation, resulting in memory failure while processing the entire data. In Section 6.4 we present the details of where each of these methods runs out of memory. While it may be possible that these implementations can be engineered to be more memory efficient, we chose to operate with the available implementations provided either by the authors or by (optimized) libraries.

Turning our attention to computational performance of all the methods −summarized in Table 3, we observe that the methods using the sparse data directly, *viz.,* Isolation Forest, OCSVM and ParDFROCC, are considerably faster. Among these methods, we see that ParDFROCC is superior in training and test times (as well as `AUC ROC` as we discussed above) for most datasets, although for B2 −a low dimensional data set− it is only third behind Isolation Forest and OCSVM. It is worth noting that though Isolation Forest trains comparably fast in larger dimensional data its query time suffers due to the large number of trees that need to be queried.

6.4 Scalability

In this section, we underscore the scalability of ParDFROCC by studying how increasing the number of features and samples affects its training and test times in comparison with the baselines.

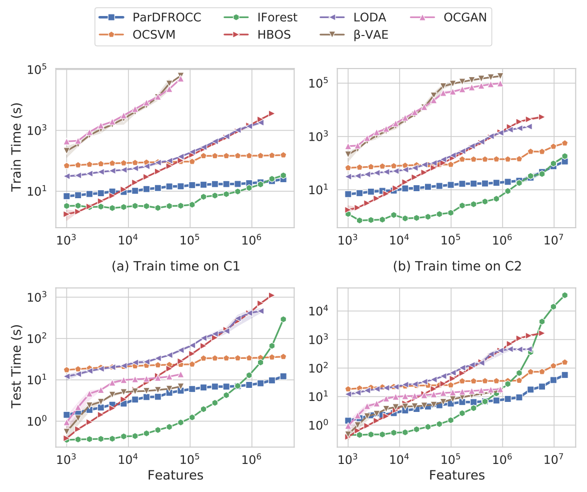
[1]https://github.com/data-iitd/frocc

Table 2: Area under the ROC curve. ID corresponds to the dataset ID in Table 1. Bold values represent the best scores and *italics* represent the second best scores. * indicates best values for methods that ran out of memory.

ID	OC-SVM	IsoForest	HBOS	LODA	β-VAE	OCGAN	DROCC	ParDFROCC
A1	53.35 ± 0.95	63.48 ± 0.46	64.32 ± 0.35	62.43 ± 0.26	69.32 ±0.49	69.64± 0.31	*72.32 ±0.25*	**73.18 ± 0.26**
A2	68.12 ± 0.85	*77.09 ± 0.84*	62.39 ± 0.37	75.96 ± 0.44	66.43 ±0.16	65.63± 0.38	**78.96 ±0.31**	72.30 ± 0.33
A3	54.22 ± 0.45	**81.21 ± 0.14**	77.93 ± 0.21	56.27 ± 0.22	68.41 ±0.08	69.19± 0.34	*79.65 ±0.55*	69.32 ± 0.30
A4	57.93 ± 0.73	85.10 ± 0.33	59.64 ± 0.43	59.43 ± 0.60	84.25 ±0.13	*85.14± 0.48*	**85.21 ±0.48**	83.57 ± 0.44
B1	73.51 ± 0.41	83.19 ± 0.44	71.16 ± 0.44	76.42 ± 0.76	81.26± 0.31	81.26± 0.45	*83.27± 0.21*	**85.14 ± 0.74**
B2	*86.01 ± 0.60*	80.05 ± 0.43	79.73 ± 0.37	79.23 ± 0.34	75.21± 0.09	73.79± 0.19	75.17± 0.20	**86.64 ± 0.51**
B3	98.99 ± 0.11	98.86 ± 0.31	96.14 ± 0.18	*99.38 ± 0.13*	98.60± 0.42	97.29± 0.16	98.56± 0.56	**99.61 ± 0.35**
B4	52.21 ± 0.39	51.47 ± 0.59	59.32 ± 0.62	56.73 ± 0.47	59.23± 0.08	59.99± 0.37	*60.20± 0.20*	**61.93 ± 0.64**
B5	49.62 ± 0.12	54.13 ± 0.17	67.81 ± 0.39	55.73 ± 0.53	66.32± 0.00	67.66± 0.31	*70.36± 0.30*	**71.04 ± 0.37**
B6	63.55 ± 0.12	61.23 ± 0.35	61.88 ± 0.41	66.73 ± 0.44	62.54± 0.47	63.39± 0.35	**67.55± 0.55**	*67.12 ± 0.60*
B7	*90.21 ± 0.63*	70.83 ± 0.35	69.82 ± 0.21	62.38 ± 0.12	72.51± 0.36	71.66± 0.29	84.54± 0.49	**92.46 ± 0.21**
C1	78.12 ± 0.48	*80.43 ± 0.32*	61.03 ± 0.00 *	51.32 ± 0.54 *	66.24 ± 0.79 *	61.04 ± 0.83 *	72.49 ± 0.43 *	**88.02 ± 0.78**
C2	75.42 ± 00.11	*78.62 ± 00.61*	69.83 ± 00.00 *	59.97 ± 00.22 *	72.54 ± 0.17 *	69.19 ± 00.44 *	75.35 ± 0.27 *	**79.01 ± 00.13**

Table 3: Training and test times (m). Bold values represent the best scores and *italics* represent the second best scores. * indicates best values for methods that ran out of memory.

Methods	C1		C2		B7		B2	
	Train	Test	Train	Test	Train	Test	Train	Test
ParDFROCC	**29.75**	**24.02**	**109.43**	**47.76**	**0.31**	**0.32**	*0.18*	0.16
OCSVM	154.95	36.32	286.07	60.55	1.66	0.46	0.58	*0.13*
IsoForest	*31.54*	71.90	*117.99*	92.55	*0.33*	0.77	**0.16**	**0.08**
HBOS	181.64 *	45.21	248.05 *	73.98	1.95	0.47	0.67	0.17
LODA	212.56 *	*35.23*	303.15 *	*50.12*	2.24	*0.37*	0.67	0.16
β-VAE	326.14 *	62.93	687.97 *	74.44	3.44	0.68	1.06	0.20
OCGAN	298.23 *	56.76	671.58 *	66.37	3.11	0.63	1.00	0.25



(a) Train time on C1

(b) Train time on C2

(c) Test time on C1

(d) Test time on C2

Figure 4: Feature Scalability: comparison of training and test times with increasing number of features for 100000 training samples. Both axes are log-scaled.

Figure 4 shows how ParDFROCC and the baselines scale with number of features, both on Malicious URL and Webb Spam dataset. Notice that both axes are logarithmic, to accommodate the wide range of scalability and the high number of dimensions. In these experiments, ParDFROCC utilizes 8 CPU cores to parallelize training and testing (Sec. 5.3).

From Figure 4(a), which shows the scaling of train time with number of features on high-dimensional sparse Malicious URL data, we notice while Isolation Forest outperforms ParDFROCC at lower dimensions, ParDFROCC’s running time increases very slowly and outperforms Isolation Forest at very high dimensions. OCSVM scales at a slow rate with features for sparse data, but is consistently outperformed by ParDFROCC. HBOS outperforms ParDFROCC at very low dimensions but does not scale well. LODA is slightly faster than OCSVM at lower dimensions, but rapidly deteriorates in performance and is consistently beaten by ParDFROCC. The deep learning algorithms, both OCGAN and β-VAE based, scale very poorly with features, as larger features require larger GANs and VAEs respectively. This trend is observed on the Webb Spam dataset as well, as seen in Figure 4(c).

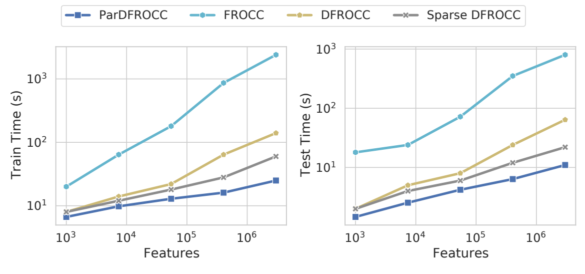
In most application settings, the scalability of test time is of greater importance than train time. Figure 4(b) shows how the test time for the methods scale with number of features. Increase in features considerably increases the test time for Isolation Forest, because of deeper trees. OCSVM displays the same trend as train time. The test times of deep learning methods are much more competitive compared to the train times, but is still outperformed by ParDFROCC at high dimensions. Test times of HBOS and LODA scales poorly with increasing features. A similar observation follows on the Webb Spam dataset also, shown in Figure 4(d).

6.5 Effect of Optimizations on FROCC

In this section, we explore the performance gains obtained by each of our three optimizations over FROCC– that is, DFROCC, SparseDFROCC and ParDFROCC.

Figures 5 (a) and (b) summarize train and test time of all the four variants in the FROCC-family of methods as we increase the number of features on a subset of Malicious URL dataset. Clearly, by using discrete intervals, the training times of DFROCC improve by almost a factor of 2 at low dimensions and by *an order of magnitude* at high dimensions over FROCC. SparseDFROCC, by using sparse projections, is *twice* as fast as DFROCC at high dimensions while retaining its efficiency at low dimensions. Finally, parallelization in ParDFROCC further enhances the performance by using more CPU cores to result in a 2.5× speedup over SparseDFROCC in high dimensional. The test times follow a similar trend.

In summary, for training on high dimensional data, ParDFROCC is 2.5× faster than SparseDFROCC, which in turn is 2× faster than DFROCC, which is 10× faster than FROCC.



(a) Train Scaling on C1 (b) Test Scaling on C1

Figure 5: Comparison of effects of optimizations on FROCC

7 CONCLUSION

In this paper, we presented a surprisingly simple and highly effective one-class classification strategy based on projecting data onto randomly selected vectors from the unit sphere. By representing these projections as a collections of intervals we developed FROCC. We prove that FROCC is stable and has low bias for the special case $\varepsilon = 1$. We then modify FROCC into a discrete version called DFROCC, a sparse version of DFROCC, and finally, a parallel version of sparse DFROCC called ParDFROCC, that can scale to extremely large datasets.

We demonstrated the challenge that the current state of the art algorithms face when it comes to dealing with large number of features and establish by extensive experiments that ParDFROCC can handle these challenges.

In the future, we plan to extend the idea behind ParDFROCC to application settings that deal with high dimensional data where a small fraction of negative samples are available, i.e. highly imbalanced number of positive and negative samples, as well as dealing with unlabeled data, i.e. PU learning. On the theoretical side, we plan to address the question of proving stability for FROCC when $\varepsilon \in (0, 1)$.

REFERENCES

[1] Dimitris Achlioptas. 2001. Database-friendly random projections. In *PoDS*. 274–281.
[2] Nir Ailon and Bernard Chazelle. 2006. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *SoTC*. 557–563.
[3] Olivier Bousquet and André Elisseeff. 2002. Stability and generalization. *J. Mach. Learn. Res* 2 (2002), 499–526.
[4] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. 2018. Understanding disentangling in β-VAE. *arXiv preprint arXiv:1804.03599* (2018).
[5] Gilles Cohen, Hugo Sax, Antoine Geissbuhler, et al. 2008. Novelty detection using one-class Parzen density estimator. An application to surveillance of nosocomial infections.. In *Stud Health Technol*, Vol. 136. 21–26.
[6] Timothy de Vries, Sanjay Chawla, and Michael E Houle. 2012. Density-preserving projections for large-scale local anomaly detection. *Knowl Inf Syst* 32 (2012), 25–52.
[7] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml
[8] Bradley Efron. 1965. The convex hull of a random set of points. *Biometrika* 52, 3-4 (1965), 331–343.
[9] Martín Abadi et. al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[10] James E Fowler and Qian Du. 2011. Anomaly detection and reconstruction from random projections. *IEEE Trans. Image Process.* 21, 1 (2011), 184–195.
[11] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* 28, 1 (2009), 18–28.
[12] Markus Goldstein and Andreas Dengel. 2012. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI* (2012), 59–63.
[13] Sachin Goyal, Aditi Raghunathan, Moksh Jain, Harsha Vardhan Simhadri, and Prateek Jain. 2020. DROCC: Deep robust one-class classification. In *ICML*. PMLR, 3711–3721.
[14] P. Indyk and R. Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *SToC*. 604–613.
[15] Piotr Indyk and Assaf Naor. 2007. Nearest-neighbor-preserving embeddings. *ACM Trans. Algorithms* 3, 3 (2007), 31–es.
[16] Daniel M Kane and Jelani Nelson. 2014. Sparser johnson-lindenstrauss transforms. *J. ACM* 61, 1 (2014), 1–23.
[17] Shehroz S Khan and Michael G Madden. 2009. A survey of recent trends in one class classification. In *AICS*. Springer, 188–197.
[18] J. M. Kleinberg. 1997. Two algorithms for nearest-neighbor search in high dimensions. In *SToC*. 599–608.
[19] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images.* Technical Report. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
[20] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338.
[21] Yann LeCun. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/
[22] David A. Levin and Yuval Peres. 2017. *Markov Chains and Mixing Times* (2nd. revised edition ed.). AMS.
[23] Ping Li, Trevor J Hastie, and Kenneth W Church. 2006. Very sparse random projections. In *KDD*. 287–296.
[24] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *ICDM*. IEEE, 413–422.
[25] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *ICML*. 681–688.
[26] Colin McDiarmid. 1989. *On the method of bounded differences.* Cambridge University Press, 148–188.
[27] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *Network and Distributed System Security Symposium (NDSS)* (2018).
[28] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton van den Hengel. 2020. Deep learning for anomaly detection: A review. *arXiv preprint arXiv:2007.02500* (2020).
[29] Pramuditha Perera and Vishal M. Patel. 2018. Learning Deep Features for One-Class Classification. *IEEE Trans. Image Process.* 28 (2018), 5450–5463.
[30] Pramuditha Perera and Vishal M. Patel. 2019. Learning Deep Features for One-Class Classification. *IEEE Trans. Image Process.* 28, 11 (2019), 5450–5463.
[31] Tomáš Pevnỳ. 2016. Loda: Lightweight on-line detector of anomalies. *Mach Learn* 102, 2 (2016), 275–304.
[32] A. Rényi and R. Sulanke. 1963. Uber die convexe hulle von is zufallig gewahlten punkten I. *Z. Wahr. Verw. Geb.* 2 (1963), 75–84.
[33] A. Rényi and R. Sulanke. 1964. Uber die convexe hulle von is zufallig gewahlten punkten II. *Z. Wahr. Verw. Geb.* 3 (1964), 138–147.
[34] Peter J. Rousseeuw and Katrien Van Driessen. 1999. A Fast Algorithm for the Minimum Covariance Determinant Estimator. *Technometrics* 41, 3 (Aug 1999), 212–223. https://doi.org/10.1080/00401706.1999.10485670
[35] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep one-class classification. In *ICML*. 4393–4402.
[36] Bernhard Schölkopf, Robert Williamson, Alex Smola, John Shawe-Taylor, and John Piatt. 2000. Support vector method for novelty detection. In *NeurIPS*. 582–588.
[37] J Paul Siebert. 1987. *Vehicle Recognition Using Rule Based Methods.* Project Report. Turing Institute, Glasgow. http://eprints.gla.ac.uk/91397/
[38] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* (2012).
[39] Lorne Swersky, Henrique O Marques, Jöerg Sander, Ricardo JGB Campello, and Arthur Zimek. 2016. On the evaluation of outlier detection and one-class classification methods. In *DSAA*. IEEE, 1–10.
[40] David M.J. Tax and Robert P.W. Duin. 2004. Support Vector Data Description. *Mach Learn* 54 (2004), 45–66.
[41] Steve Webb, James Caverlee, and Calton Pu. 2006. Introducing the Webb Spam Corpus: Using Email Spam to Identify Web Spam Automatically. In *CEAS*.
[42] Yue Zhao, Zain Nasrullah, and Zheng Li. 2019. PyOD: A Python Toolbox for Scalable Outlier Detection. *J Mach Learn Res* 20, 96 (2019), 1–7.
[43] Panpan Zheng, Shuhan Yuan, Xintao Wu, Jun Yu Li, and Aidong Lu. 2018. One-Class Adversarial Nets for Fraud Detection. In *AAAI*, Vol. 33. 1286–1293.