Super-Sampling with a Reservoir

Brooks Paige
Department of Engineering Science
University of Oxford
brooks@robots.ox.ac.uk

Dino Sejdinovic
Department of Statistics
University of Oxford
dino.sejdinovic@stats.ox.ac.uk

Frank Wood
Department of Engineering Science
University of Oxford
fwood@robots.ox.ac.uk

Abstract

We introduce an alternative to reservoir sampling, a classic and popular algorithm for drawing a fixed-size subsample from streaming data in a single pass. Rather than draw a random sample, our approach performs an online optimization which aims to select the subset that provides the best overall approximation to the full data set, as judged using a kernel two-sample test. This produces subsets which minimize the worst-case relative error when computing expectations of functions in a specified function class, using just the samples from the subset. Kernel functions are approximated using random Fourier features, and the subset of samples itself is stored in a random projection tree. The resulting algorithm runs in a single pass through the whole data set, and has a per-iteration computational complexity logarithmic in the size of the subset. These "super-samples" subsampled from the full data provide a concise summary, as demonstrated empirically on mixture models and the MNIST dataset.

1 INTRODUCTION

We receive a stream of samples $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, distributed according to an unknown distribution $p(\mathbf{x})$, where N is large and possibly not known ahead of time. Rather than store all the samples for later processing, we would like an online method for selecting a subsample of size M, typically with $M \ll N$, in a single pass through the data N.

Reservoir sampling algorithms [Vitter, 1985] solve exactly this problem: they produce a running subsample, such that for any i from $M, \dots, N$, the *reservoir* contains a set of M points which themselves are subsampled without replacement from the full stream up through point i. As each new $\mathbf{x}_i$ arrives, the subsample is updated. This update involves swapping the new $\mathbf{x}_i$ with one of the existing M points at random, with appropriate probability. After sweeping through N points, we have a random sample of size M, produced in a single pass through the data, and requiring only $\mathcal{O}(M)$ storage.

In this paper we ask whether instead of subsampling at random, we can change this into a decision problem which at each new $\mathbf{x}_i$ inspects the actual values of our current subset of M points, and aims to ultimately construct a "best" possible subset of a given fixed size.

We take the "best" subset $Y_M = \{\mathbf{y}_j\}_{j=1}^M$, with $Y_M \subset X_N = \{\mathbf{x}_i\}_{i=1}^N$, to be the subset which minimizes the worst-case error when using the small sample Y_M to estimate expectations, instead of the full sample X_N, across all functions f in some function class $\mathcal{F}$. This loss function is known as the *maximum mean discrepancy* (MMD) [Smola et al., 2007] and takes the form

$$\mathcal{L}_{MMD} := \sup_{f \in \mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) - \frac{1}{M} \sum_{j=1}^{M} f(\mathbf{y}_j) \right). \quad (1)$$

By minimizing the MMD, we construct a subset whose empirical distribution mimics the full data distribution as closely as possible and allows the most accurate estimation of expectations of functions in $\mathcal{F}$. We particularly focus on the case when the function class is the unit ball in a reproducing kernel Hilbert space (RKHS).

As motivation, we note that the empirical estimate of the maximum mean discrepancy between two distributions is used as a statistic for a kernel two-sample test [Gretton et al., 2012], a state-of-the-art nonparametric approach for testing the hypothesis that two sets of samples are drawn from the same underlying distribution (with large values of MMD being evidence against this hypothesis). We will construct our Y_M to be a subset of the full data X_N which minimizes the value of this statistic.

Related work constructing point sets which minimize MMD to a target data distribution includes kernel herding [Chen et al., 2010], which provides a deterministic alternative for sampling from a specified target density, and sequential

Algorithm 1 "Algorithm R" [Vitter, 1985]

```
Input: Stream of samples $\mathbf{x}_1, \dots, \mathbf{x}_N$
Output: Subsample $\mathbf{y}_1, \dots, \mathbf{y}_M$, of size $M \ll N$
  Initialize $\mathbf{y}_1 = \mathbf{x}_1, \dots, \mathbf{y}_M = \mathbf{x}_M$
  for $n = M+1, \dots, N$ do
    $j \sim \text{Uniform}\{1, \dots, n\}$
    if $j \leq M$ then
      $\mathbf{y}_j = \mathbf{x}_n$
    end if
  end for
```

Bayesian quadrature [Huszár and Duvenaud, 2012], which selects weighted point sets to minimize the MMD. Both of these methods differ from our approach in that they sequentially generate new points, providing an alternative to drawing random samples from a target density function $p(\mathbf{x})$. Our algorithm provides instead an alternative to random subsampling without replacement, and is designed to be appropriate for processing streaming data online.

We name our approach "super-sampling with a reservoir", after the original paper of Vitter [1985], replacing the random sampling with the "super-sampling" moniker given to the output of the kernel herding Chen et al. [2010]. We provide some background material in Section 2, and then introduce our algorithm in Section 3. Theoretical results are provided in Section 4, with experimental validation in Section 5.

2 BACKGROUND

The simplest reservoir sampling algorithm for unweighted data, introduced as "Algorithm R" by Vitter [1985], is reproduced here in Algorithm 1. After initializing the reservoir set Y_M to the first M values in the stream, the algorithm proceeds by inserting each subsequent $\mathbf{x}_n$, for $n = M+1, \dots, N$, into the reservoir with probability M/n. During the whole duration of the algorithm, the reservoir always contains a random sample (without replacement) from the n data points observed thus far.

An alternative approach is to imagine drawing a random subsample of size M by assigning a random uniform priority to each of the N items, and then selecting the M with the lowest priorities. This is equivalent to shuffling the N items by sorting them based on a random key, and selecting the first M values; it can be implemented as an online algorithm by storing the M samples in a priority queue, in which all the priorities are assigned at random.

Although such an algorithm has runtime $\mathcal{O}(\log M)$ at each new candidate point $\mathbf{x}_n$, it can be generalized to allow performing reservoir sampling on streams of arbitrarily weighted values [Efraimidis and Spirakis, 2006].

2.1 Kernel embeddings of distributions

Our algorithm will replace the random selection used in Algorithm R with an active selection aiming to minimize Equation (1). This is possible thanks to the properties of reproducing kernel Hilbert spaces and kernel mean embeddings of distributions [Smola et al., 2007; Song, 2008], which we review briefly here. A reproducing kernel Hilbert space $\mathcal{H}$ is a function space equipped with an inner product $\langle \cdot, \cdot \rangle$, and has a symmetric positive-definite reproducing kernel $k(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Elements of $\mathcal{H}$ are functions defined on $\mathcal{X}$. The *reproducing* property of the kernel means that we can write function evaluation as an inner product, where for any function $f \in \mathcal{H}$, we have

$$f(\mathbf{x}) = \langle k(\mathbf{x}, \cdot), f(\cdot) \rangle. \tag{2}$$

This kernel can equivalently be defined as an inner product over an explicit "feature space" mapping $\phi : \mathcal{X} \to \mathcal{H}$, with

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle. \tag{3}$$

The "canonical" feature map is defined as $\phi(\mathbf{x}) = k(\mathbf{x}, \cdot)$, where we see $\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle = k(\mathbf{x}, \mathbf{x}')$; we will use the notation $\phi(\mathbf{x})$ and $k(\mathbf{x}, \cdot)$ interchangeably.

Similarly to how the reproducing kernel acts as an evaluation functional on f, computing $f(\mathbf{x})$ as an inner product, mean embeddings of distributions act as an expectation functional, computing $\mathbb{E}[f]$ as an inner product. For some distribution with density $p(\mathbf{x})$, the kernel mean embedding of a distribution [Smola et al., 2007] is defined as

$$\mu(\cdot) = \int k(\mathbf{x}, \cdot) p(\mathbf{x}) d\mathbf{x}. \tag{4}$$

If $k(\cdot, \cdot)$ is measurable, and $\mathbb{E}[k(\mathbf{x}, \mathbf{x})^{1/2}] < \infty$, then μ exists and $\mu \in \mathcal{H}$ [Gretton et al., 2012]; the mean embedding can then be used to compute expectations of functions $f \in \mathcal{H}$ as

$$\mathbb{E}[f] = \langle \mu, f \rangle. \tag{5}$$

There are two sources of intractability standing in between us and the application of Equation (5): neither evaluating the inner product $\langle \mu, f \rangle$ nor computing the mean embedding μ in Equation (4) are necessarily any simpler than the original integration with respect to $p(\mathbf{x})$. Two approximations will be useful in practice.

First, using a finite set of sample points, we can define an empirical estimate of the mean embedding in Equation (4)

$$\mu_N = \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i), \qquad \mathbf{x}_i \sim p(\mathbf{x}). \tag{6}$$

Note that μ_N itself is still a function in $\mathcal{H}$, and inner products of this estimator correspond to computing empirical finite-sample estimates of expectations, since via the reproducing property,

$$\langle \mu_N, f \rangle = \frac{1}{N}\sum_{i=1}^{N} \langle \phi(\mathbf{x}_i), f \rangle = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i). \tag{7}$$

The second problem is for many common and interesting kernels, the feature map $\phi(\mathbf{x})$ is infinite dimensional (as in e.g. squared exponential kernel and the Laplacian kernel). By considering only the kernel function $k(\mathbf{x}, \mathbf{x}')$, one can avoid needing to explicitly instantiate these features, a benefit known as the "kernel trick". However, for computational purposes, and to make it possible to construct an online algorithm, we will find it advantageous to explicitly instantiate an approximate feature space representation $\hat{\phi}(\mathbf{x}) \in \mathbb{R}^D$. In particular, this can be accomplished with a finite vector of D random Fourier projections [Rahimi and Recht, 2007], where each feature has the form

$$\hat{\phi}(\mathbf{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos(\boldsymbol{\omega}_1^\top \mathbf{x} + b_1) \\ \vdots \\ \cos(\boldsymbol{\omega}_D^\top \mathbf{x} + b_D) \end{bmatrix}. \tag{8}$$

Each $\boldsymbol{\omega}_d$ is drawn from the distribution $p(\boldsymbol{\omega})$ which arises by taking the Fourier transform of the kernel, and each b_d is uniform on $[0, 2\pi]$; Bochner's theorem [Bochner, 1959] guarantees that for any shift invariant positive-definite kernel $k(\mathbf{x}, \mathbf{x}')$, its Fourier transform is a finite and nonnegative measure, so $p(\boldsymbol{\omega})$ can be assumed to be a probability distribution. The random Fourier features $\hat{\phi}(\mathbf{x}) \in \mathbb{R}^D$ approximate the true (possibly infinite-dimensional) feature map $\phi(\mathbf{x}) \in \mathcal{H}$. An approximating kernel defined by taking the inner product of the approximate feature maps, i.e. $k(\mathbf{x}, \mathbf{x}') \approx \hat{\phi}(\mathbf{x})^\top \hat{\phi}(\mathbf{x}')$, provides an unbiased estimate of evaluations of the kernel function [Rahimi and Recht, 2007], with

$$\mathbb{E}_{\boldsymbol{\omega},\mathbf{b}}[\hat{\phi}(\mathbf{x})^\top \hat{\phi}(\mathbf{x}')] = k(\mathbf{x}, \mathbf{x}'). \tag{9}$$

Taken together with Equation (6), we can thus approximate the mean embedding using random Fourier features evaluated at finite sample points as

$$\hat{\mu}_N = \frac{1}{N}\sum_{i=1}^{N} \hat{\phi}(\mathbf{x}_i), \tag{10}$$

yielding an explicit representation of the distribution $p(\mathbf{x})$ as a vector in $\mathbb{R}^D$.

Now, consider how we can use this representation to approximate $\mathcal{L}_{MMD}$ in Equation (1). Following Gretton et al. [2012], we take our test function space to be the unit ball in $\mathcal{H}$, i.e. with

$$\mathcal{F} = \{f : f \in \mathcal{H}, ||f||_{\mathcal{H}} \leq 1\}, \tag{11}$$

where the Hilbert space norm is defined as

$$||f||_{\mathcal{H}} = \langle f, f \rangle^{1/2}. \tag{12}$$

Define a second empirical estimate of the mean embedding

$$\nu_M = \frac{1}{M}\sum_{j=1}^{M} \phi(\mathbf{y}_j), \qquad \mathbf{y}_j \in Y_M \subset X, \tag{13}$$

on the small subset of the full points in the set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. We then rewrite the maximum mean discrepancy as an RKHS norm [Borgwardt et al., 2006; Gretton et al., 2012, Lemma 4], with

$$\begin{aligned} \mathcal{L}_{MMD} &= \sup_{f \in \mathcal{F}} \Big(\langle \mu_N, f \rangle - \langle \nu_M, f \rangle \Big) \\ &= \sup_{f \in \mathcal{F}} \langle \mu_N - \nu_M, f \rangle \\ &= ||\mu_N - \nu_M||_{\mathcal{H}}. \end{aligned} \tag{14}$$

Thus, to minimize the MMD we just need to select our points in Y_M such that this RKHS norm is as small as possible. Using random Fourier features approximations for $\hat{\mu}_N$ and $\hat{\nu}_M$ allows us to define a computationally efficient estimator for $\mathcal{L}_{MMD}$ which we can update online while processing the sample points in X. If our set of subsampled points Y_M has $\hat{\nu}_M \approx \hat{\mu}_N$, then we can use those points to evaluate expectations in a way that approximates expectations w.r.t. the full sample set.

3 STREAMING SUBSET SELECTION

We now introduce a sequential optimization algorithm for minimizing the MMD between the streaming data and our local subset. Analogous to reservoir sampling, at any stage of the algorithm we have a reservoir Y_M which contains points drawn without replacement from X_N, representing our current approximation to the distribution of the first n data points. Globally, this procedure takes the form of a greedy optimization algorithm in which a new candidate point $\mathbf{x}_i$ is inserted into our set Y_M, replacing an existing element when the substitution reduces the MMD. Locally, at each candidate point, we must solve an inner optimization problem to decide whether to keep $\mathbf{x}_i$, and if so, which of the $\mathbf{y}_j$ it should replace.

Samples $\mathbf{x}_i$ arrive sequentially; let X_n denote the subset of X comprising the first n points, and let Y_M^n denote the subset of M points selected after processing the points in X_n, with $Y_M \equiv Y_M^N$ the subset selected after all points in X have been processed. As in a standard reservoir sampling algorithm we initialize Y_M^M to be $\mathbf{x}_1, \dots, \mathbf{x}_M$, i.e. the first M points. We will keep running estimates of the kernel mean embeddings

$$\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} \hat{\phi}(\mathbf{x}_i), \qquad \mathbf{x}_i \in X_n \tag{15}$$

$$\hat{\nu}_M^n = \frac{1}{M}\sum_{j=1}^{M} \hat{\phi}(\mathbf{y}_j), \qquad \mathbf{y}_j \in Y_M^n. \tag{16}$$

These are (respectively) the Monte Carlo estimate of the true mean element μ from the first n samples $\mathbf{x}_1, \dots, \mathbf{x}_n$, and the estimate of μ recovered from the M values selected from the first n, with random Fourier features $\hat{\phi} : \mathcal{X} \to \mathbb{R}^D$ defining an explicit feature space.

From the expression in Equation (14), we define an estimator of the MMD based on these random feature expansions after seeing n data points as

$$\widehat{\mathcal{L}}_{MMD} = ||\hat{\mu}_n - \hat{\nu}_M^n||_2. \tag{17}$$

Suppose we have seen $n-1$ candidates thus far and are now considering some $\mathbf{x}_n$. Note we can do a sequential update

$$\hat{\mu}_n = \frac{n-1}{n}\hat{\mu}_{n-1} + \frac{1}{n}\hat{\phi}(\mathbf{x}_n). \tag{18}$$

A similarly simple incremental update is possible when the sample stream X is comprised of weighted samples. If we suppose every point x_n has an associated nonnegative weight w_n, we must additionally track the running sum of all weights $\bar{w}_n = \sum_{i=1}^n w_i$, and perform updates

$$\hat{\mu}_n = \frac{\bar{w}_{n-1}}{\bar{w}_{n-1} + w_n}\hat{\mu}_{n-1} + \frac{w_n}{\bar{w}_{n-1} + w_n}\hat{\phi}(\mathbf{x}_n), \tag{19}$$

with $\bar{w}_n = \bar{w}_{n-1} + w_n$. We recover the update in Equation (18) for unweighted sample sets if all weights are identically $w_i = 1$.

We need to decide whether or not the new $\mathbf{x}_n$ should replace an existing $\mathbf{y}_j \in Y_M^{n-1}$, or if it should be ignored. This means there are $M+1$ possible candidates for $\hat{\nu}_M^n$; we want to determine which substitution minimizes $\widehat{\mathcal{L}}_{MMD}$. A naïve approach is to compute the $\hat{\nu}_M^n$ for all $M+1$ options, and choose that which yields the smallest $\widehat{\mathcal{L}}_{MMD}$. This gives an overall algorithm in which we perform an $\mathcal{O}(MD)$ computation for each new candidate point, though this approach can be made reasonably efficient through incremental computation of the possible values of $\widehat{\mathcal{L}}_{MMD}$. We instead focus on providing an approximate optimization here with runtime logarithmic in M.

3.1 Formulation as nearest neighbor search

An alternate way of formulating the inner per-datapoint problem of selecting whether and where to swap in a candidate point $\mathbf{x}_n$, instead of as an optimization problem where we minimize $\widehat{\mathcal{L}}_{MMD}$, is as a nearest-neighbor search. As each new candidate point $\mathbf{x}_n$ arrives, we have an existing subsample estimate $\hat{\nu}_M^{n-1}$, and compute an updated running estimate $\hat{\mu}_n$. Consider the "expanded" estimator for the mean embedding defined as

$$\hat{\nu}_{M+1}^n \triangleq \frac{M}{M+1}\hat{\nu}_M^{n-1} + \frac{1}{M+1}\hat{\phi}(\mathbf{x}_n) \tag{20}$$

which incorporates the new point $\mathbf{x}_n$ alongside the existing M point estimate, by averaging the feature maps of all

Algorithm 2 Streaming MMD Minimization

```
Input: Stream of samples $\mathbf{x}_1, \dots, \mathbf{x}_N$;
  explicit feature map $\phi : \mathcal{X} \to \mathbb{R}^D$
Output: Subset $Y_M = \{\mathbf{y}_1, \dots, \mathbf{y}_M\}$
  Initialize $\mathbf{y}_1 = \mathbf{x}_1, \dots, \mathbf{y}_M = \mathbf{x}_M$
  Compute initial mean estimates $\hat{\nu}_M^M$ and $\hat{\mu}_M$        Eq. (6)
  for $n = M+1, \dots, N$ do
      Update $\hat{\mu}_n$ to include $\phi(\mathbf{x}_n)$        Eq. (18) or (19)
      Compute target $\phi^\star$        Eq. (23)
      if NearestNeighbor$_{\phi^\star}(\{\mathbf{x}_n\} \cup Y_M)$ in $Y_M$ then
          $\hat{\nu}_M^n \leftarrow \hat{\nu}_M^{n-1} + \frac{1}{M}\left(\phi(\mathbf{x}_n) - \phi(\mathbf{y}_j)\right)$
          $\mathbf{y}_j = \mathbf{x}_n$
      else
          $\hat{\nu}_M^n \leftarrow \hat{\nu}_M^{n-1}$
      end if
  end for
```

points in $Y_{M+1}^{n-1} := \{\mathbf{x}_n\} \cup Y_M^{n-1}$. Equation (20) is also an approximation to $\hat{\mu}_n$, but using $M+1$ total points. Any next estimator $\hat{\nu}_M^n$ for $\hat{\mu}_n$ using only M points can be found by discarding a single one of the points in Y_{M+1}^{n-1}. For whichever point $\mathbf{y}^{drop}$ we choose to discard, we can then express $\widehat{\mathcal{L}}_{MMD}$ using the expanded estimator $\hat{\nu}_{M+1}^n$, with

$$\hat{\nu}_M^n = \frac{M+1}{M}\hat{\nu}_{M+1}^n - \frac{1}{M}\hat{\phi}(\mathbf{y}^{drop}). \tag{21}$$

Selecting the best $\hat{\nu}_M^n$ to minimize $\widehat{\mathcal{L}}_{MMD} = ||\hat{\mu}_n - \hat{\nu}_M^n||_2$ then corresponds to solving an optimization problem

$$\mathbf{y}^{drop} = \operatorname*{argmin}_{\mathbf{y} \in Y_{M+1}^{n-1}} \left|\left|\hat{\mu}_n - \frac{M+1}{M}\hat{\nu}_{M+1}^n + \frac{1}{M}\hat{\phi}(\mathbf{y})\right|\right|_2 \tag{22}$$

in which we select the $\mathbf{y}_j$ to remove, such that the estimate $\widehat{\mathcal{L}}_{MMD}$ from the resulting Y_M^n is minimized. If there were a somehow "perfect" choice of $\mathbf{y}_j$ to remove, then this would bring the quantity on the right of Equation (22) to zero. By setting Equation (22) to zero, solving for ideal feature vector $\hat{\phi}(\mathbf{y})$, and then using Equation (20) to expand out $\hat{\nu}_{M+1}^N$ we find the optimal choice of feature vector to remove would be

$$\phi^\star = \hat{\phi}(\mathbf{x}_n) + M(\hat{\nu}_M^{n-1} - \hat{\mu}_n). \tag{23}$$

In general, none of our current $\phi(\mathbf{y}_j) = \phi^\star$ exactly, but we can still try to get as close as possible. Since we have explicit feature vectors $\hat{\phi}(\mathbf{y}_j) \in \mathbb{R}^D$ and $\phi^\star \in \mathbb{R}^D$, we can thus find the best choice $\mathbf{y}^{drop}$ by considering

$$\begin{aligned}\mathbf{y}^{drop} &= \operatorname*{argmin}_{\mathbf{y} \in Y_M^{n-1}} \left|\left|\hat{\phi}(\mathbf{y}) - \phi^\star\right|\right|_2^2 \\ &= \operatorname*{argmin}_{\mathbf{y} \in Y_M^{n-1}} \sum_{d=1}^{D} \left(\hat{\phi}(\mathbf{y})_d - \phi_d^\star\right)^2 .\end{aligned} \tag{24}$$

This minimum can by found by performing a D-dimensional nearest-neighbor search. The easiest option for this is still to scan through all $M+1$ candidates and compute each distance in Equation (24). However for large M we can use an approximate nearest neighbor search to reduce the overall runtime of each iteration. The overall streaming MMD minimization algorithm is given in Algorithm 2, where the function $\text{NearestNeighbor}_{\phi^\star}(Y)$ is an exact or approximate procedure for selecting the nearest neighbor (in Euclidean distance) to $\phi^\star$ in the set Y.

3.2 Approximate nearest neighbor search

Exact nearest neighbor search in more than a small number of dimensions remains a challenging problem; we are likely to use order a few hundred random features D. Classic approaches such as KD-trees [Bentley, 1975] which are based on coordinate-aligned splits tend to break down in such settings, requiring very deep trees in order to partition the data. However, there is some hope for *approximate* nearest neighbor search, where we are willing to not necessarily find the best choice, but merely a "sufficiently close" neighbor, based on theory of intrinsic dimensionality [Johnson and Lindenstrauss, 1984].

Random projection trees are introduced in Freund et al. [2007], based on the observation that for high dimensional datasets, partitioning data into subsets based on a completely random projection direction is nearly as good as the optimal partition direction. With this in mind, to implement a fast approximate nearest neighbor search, define hash functions

$$h(\phi) = \operatorname{sign}(\mathbf{r}^\top \phi - s) \tag{25}$$

where $\mathbf{r} \in \mathbb{R}^D$ is a random unit vector and s is a random split point. We place these hash functions at each nodes in a binary search tree; values ϕ which hash to a positive value are sent to the right subtree, those which hash to a negative value are sent left. This tree resembles a KD-tree, except with random projections splits instead of axis-aligned splits.

If we have L hash functions, arranged in a binary tree, we need to evaluate $\log_2 L$ when considering each new candidate point, and again when accepting a new point $\mathbf{x}_n$ into the set Y_M^n. For each point $\mathbf{y}_j$ in our subset, we cache which leaf node of the binary tree it falls in. In practice there may be several points in each leaf node, but instead of evaluating all M points in Y_M to check whether it is closest to $\phi^\star$, we only check however many are in the leaf.

Our particular random projection tree variant we implement is as described in Dasgupta and Sinha [2015]. We construct an initial tree from the first M points by sampling random vectors $\mathbf{r}$, and setting each split point s to be a uniform random quantile in $[0.25, 0.75]$ of the projected values at that node. This tree will have points evenly distributed among the leaves; however, as we swap out points it may become less balanced. To deal with this we periodically recompute the split points, rebalancing the tree; this operation is performed sufficiently rarely as to maintain amortized logarithmic cost. A free parameter in the search tree is the search tree depth (or equivalently, how many values ϕ to keep in each leaf node). In all our later experiments this is set to target approximately $2\log_2 M$ nodes per leaf.

Theoretical results in Dasgupta and Sinha [2015] characterize the loss relative to an exact search; we compare the exhaustive nearest neighbor search and approximate results empirically in our particular setting in Section 5.

4 BOUNDS ON SUBSET ERROR

At any point in the algorithm, it is straightforward to use our current distribution embedding approximations to compute $\widehat{\mathcal{L}}_{MMD} = ||\hat{\mu}_n - \hat{\nu}_M^n||_2$. It would be nice to characterize how this estimate compares to the true maximum mean discrepancy. In this section, we derive bounds for the estimation error which occurs due to using our size M subset to compute expectations, instead of using the full size N set. Above and beyond the implicit Monte Carlo error in using the points in X to approximate expectations with respect to $p(\mathbf{x})$, our procedure introduces additional error by restricting to only M of N total points, and using random Fourier features to approximate the kernel.

The online algorithm we use to sample from a stream is only possible when we have an explicit feature space representation. Following Sutherland and Schneider [2015], we can use bounds on the error introduced by approximating the kernel function with random Fourier features to provide bounds on the estimation error introduced by using only the subset $Y_M \subset X_N$. Being careful of the difference between the empirical mean embeddings $\mu_N, \nu_M \in \mathcal{H}$, and their random Fourier feature approximations $\hat{\mu}_N, \hat{\nu}_M \in \mathbb{R}^D$, we can compare two empirical estimates of the squared MMD, one using the (intractable) features $\phi(\mathbf{x})$, the other using the random Fourier features $\hat{\phi}(\mathbf{x})$:

$$\mathcal{L}^2_{MMD} = ||\mu_N - \nu_M||^2_{\mathcal{H}} \tag{26}$$

$$\widehat{\mathcal{L}^2}_{MMD} = ||\hat{\mu}_N - \hat{\nu}_M||_2^2 = \sum_{d=1}^{D} \left(\hat{\mu}_d^N - \hat{\nu}_d^M\right)^2. \tag{27}$$

The squared MMD $\mathcal{L}^2_{MMD}$ can be decomposed as [Gretton et al., 2012]

$$\mathcal{L}^2_{MMD} = \frac{1}{N^2}\sum_{i,i'} k(\mathbf{x}_i, \mathbf{x}_{i'}) + \frac{1}{M^2}\sum_{j,j'} k(\mathbf{y}_j, \mathbf{y}_{j'}) - \frac{2}{NM}\sum_{i,j} k(\mathbf{x}_i, \mathbf{y}_j),$$

with a matching expansion for $\widehat{\mathcal{L}^2}_{MMD}$, and since the random Fourier features [Rahimi and Recht, 2007] provide an unbiased estimate of $k(\mathbf{x}, \mathbf{x}')$ as in Equation (9), we have

$$\mathbb{E}\left[\widehat{\mathcal{L}^2}_{MMD}\right] = \mathcal{L}^2_{MMD}. \tag{28}$$

Now, directly following Sutherland and Schneider [2015, section 3.3], we view $\widehat{\mathcal{L}^2}_{MMD}$ as a function of the random variables $\{\boldsymbol{\omega}_d, b_d\}$ and consider how changing any one of these random variables modifies $\widehat{\mathcal{L}^2}_{MMD}$. The random Fourier feature approximation to the kernel decomposes into a sum across all D features as

$$k(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^{D} \mathbb{E}_{\boldsymbol{\omega}_d, b_d}[\hat{\phi}_d(\mathbf{x})\hat{\phi}_d(\mathbf{x}')] \tag{29}$$

$$= \frac{2}{D}\sum_{d=1}^{D} \mathbb{E}_{\boldsymbol{\omega}_d, b_d}[\cos(\boldsymbol{\omega}_d^\top \mathbf{x} + b_d)\cos(\boldsymbol{\omega}_d^\top \mathbf{x}' + b_d)].$$

Since $\cos(\cdot)$ is bounded by $\{-1, 1\}$, modifying either $\boldsymbol{\omega}_d$ or b_d changes a single term in the sum at the right of Equation (29) by at most 2, and thus changes the overall random feature approximation to the kernel by at most $4/D$. This in turn bounds the change in $\widehat{\mathcal{L}^2}_{MMD}$ by at most $16/D$. Then from the bounded differences inequality of McDiarmid [1989] we have

$$\Pr\left(\mathcal{L}^2_{MMD} - \widehat{\mathcal{L}^2}_{MMD} \geq \epsilon\right) \leq e^{-\frac{D\epsilon^2}{128}}. \tag{30}$$

An algebraic rearrangement of Equation (30) can be used to provide an upper bound in probability for $\mathcal{L}^2_{MMD}$. We have, with probability at least $1 - \delta$,

$$\mathcal{L}^2_{MMD} - \widehat{\mathcal{L}^2}_{MMD} \leq \sqrt{\frac{128\log(1/\delta)}{D}}$$

which combined with our estimator for $\widehat{\mathcal{L}^2}_{MMD}$ yields

$$\mathcal{L}^2_{MMD} \leq \sqrt{\frac{128\log(1/\delta)}{D}} + \sum_{d=1}^{D}\left(\hat{\mu}_d^N - \hat{\nu}_d^M\right)^2.$$

This directly translates into bounds on the error, due to subsampling, in estimating the average $\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i)$ of any function $f \in \mathcal{H}$ over the full dataset, since we have

$$\left|\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) - \frac{1}{M}\sum_{j=1}^{M} f(\mathbf{y}_j)\right|^2 = \left|\langle \mu_N - \nu_M, f\rangle\right|^2$$
$$\leq ||f||^2_{\mathcal{H}}\, ||\mu_N - \nu_M||^2_{\mathcal{H}}$$
$$= ||f||^2_{\mathcal{H}}\, \mathcal{L}^2_{MMD}.$$

Thus the worst-case squared error introduced by using the subsampled points for any $f \in \mathcal{F}$ can be bounded by the empirical squared error in the estimated mean embedding vectors, plus an error term due to the random Fourier feature approximation. We summarize this result as the following Theorem.

Theorem 1 *Let $\hat{\mu}_N, \hat{\nu}_M \in \mathbb{R}^D$ be estimates of the mean embedding from D random Fourier features, defined on sets of points X_N and $Y_M \subset X_N$. Then, with probability at least $1 - \delta$,*

$$\mathcal{L}^2_{MMD} = \sup_{f\in\mathcal{F}}\left|\frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) - \frac{1}{M}\sum_{j=1}^{M} f(\mathbf{y}_j)\right|^2$$
$$\leq \sqrt{\frac{128\log(1/\delta)}{D}} + \sum_{d=1}^{D}\left(\hat{\mu}_d^N - \hat{\nu}_d^M\right)^2.$$

We note that this result can be combined with existing bounds from e.g. Song [2008] on $||\mu_N - \mu||_{\mathcal{H}}$ to characterize overall error in estimating of expectations from the points in Y_M relative to true expected values $\mathbb{E}[f]$ over the population distribution of $\{\mathbf{x}_i\}$.

5 EXPERIMENTS

We run a variety of empirical tests to quantify the performance and characteristics of this algorithm. In addition to benchmarking against to random subsampling, we also compare to a benchmark of using a random Fourier features implementation of kernel herding [Chen et al., 2010]. The kernel herding algorithm is a method for sequentially generating M points, which performs a greedy optimization on the same approximation to the MMD targeted by our online algorithm; however, it is not an algorithm for processing streaming data as it requires the estimate $\hat{\mu}_N$ as computed from the full sample set as input, and furthermore it requires a potentially expensive optimization operation for generating each new point.

We also confirm experimentally that the approximation error due to the inexact nearest neighbor search does not significantly impact overall performance.

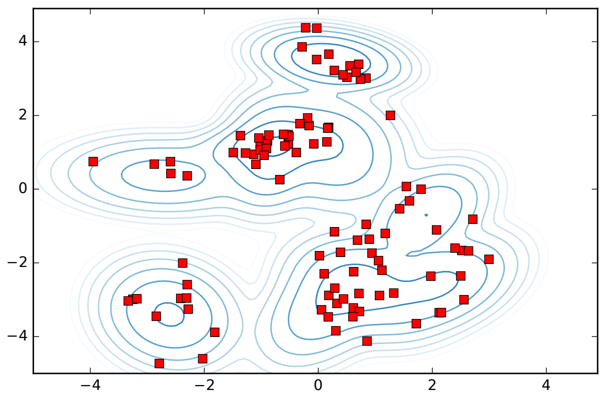
5.1 Mixtures of Gaussians

Our initial test model is a multivariate mixture of Gaussians, as considered in both Chen et al. [2010] and Huszár and Duvenaud [2012]. We experiment with downsampling a set of $N = 100,000$ points drawn from a 2-dimensional mixture of 10 Gaussians to a target set of size $M = 100$. We use a squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{||\mathbf{x} - \mathbf{x}'||_2^2}{2\gamma^2}\right\} \tag{31}$$

where the lengthscale γ is set to the median pointwise distance between the first M points; this is known as the median heuristic [Gretton et al., 2012]. We approximate the basis functions with $D = 200$ random Fourier features of the form in Equation (8), where each ω element is drawn from a normal distribution with zero mean and standard deviation γ^{-1}.

An example mixture of Gaussians target density and the selected points are shown in Figure 1, alongside a plot of the



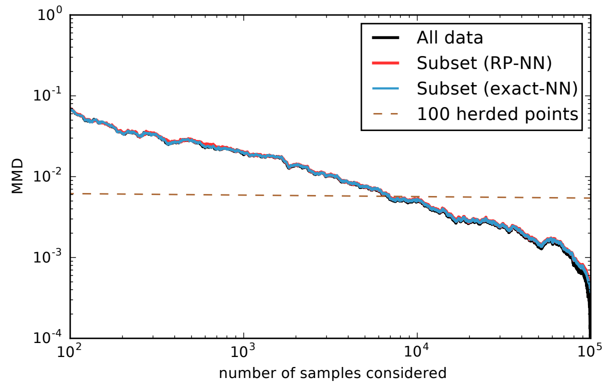


Figure 1: **(Left)** The final 100 selected points in the 2d Gaussian mixture, using the approximate nearest neighbor search. **(Right)** The empirical MMD estimate between $\hat{\mu}_n$ and $\hat{\mu}_N$, as well as between $\hat{\nu}_M^n$ and $\hat{\mu}_N$, as $n \to N$, when selecting a subset of size $M = 100$. The red line uses the random projection search tree; the blue line uses a linear scan of all options to perform an exact nearest-neighbor search. The subset estimates track the all-data MMD $\hat{\mu}_n$ very closely, despite subsampling, and even despite the approximate search. Both methods are initialized from the same set of randomly sampled points. The herding benchmark consists of M points to approximate $\hat{\mu}_N$. All estimates are averaged over 10 different synthetic datasets, drawn from mixtures of Gaussians with different parameters.

convergence of the maximum mean discrepancy estimates $||\hat{\mu}_N - \hat{\mu}_n||_2$ and $||\hat{\mu}_N - \hat{\nu}_M^n||_2$. As we view more data points, the running estimate $\hat{\mu}_n$ gradually approaches the full-data estimate $\hat{\mu}_N$. We compare to two implementations of the subsampling algorithm for selecting Y_M^n and computing $\hat{\nu}_M^n$ — one using the approximate nearest neighbor search, and one using a linear scan for an exact nearest neighbor search — and see that in both cases the running estimate based on the subsample very closely tracks the full data estimate. The overall difference in performance between the two methods is negligible despite making far fewer comparisons per iteration of the algorithm.

The herding benchmark consists of the first $M = 100$ points selected by the kernel herding [Chen et al., 2010], targeting $\hat{\mu}_N$. Theoretical results for herding suggest that the sample efficiency of the first 100 herded points should approximately match the first 10,000 random samples.

Figure 2 shows the empirical distribution over the number of comparisons actually made while searching for $\mathbf{y}^{drop}$ at each iteration when using the random projection tree for nearest neighbor search.

In Figure 3 we compare the error in computing expectations using our subsampled points, testing on the same set of functions used as a benchmark in Chen et al. [2010]. We find that the mean squared error on these test functions closely tracks the error from the full set of points. This is a remarkably promising result: expectations with respect to the full data converge at the Monte Carlo rate, and the subset of $M = 100$ points continues to perform comparably even at $N = 100,000$, for three of four test functions, and reliably outperforms both the random sampling and herding benchmarks.

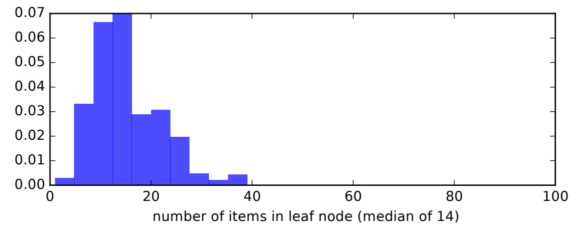


Figure 2: The actual number of comparisons made by the algorithm for processing each new data point is the same as the number of items encountered in each leaf of the random projection search tree; a full scan would require $M = 100$ comparisons, while here the median was 14. In the 2d Gaussian mixture example, we have a tree with depth 3, with an expected 12.5 items in each leaf. Compared to running a full scan, the tree-search version made the best overall decision 97.8% of the time.

5.2 Data summarization

This procedure can also be used to efficiently summarize high dimensional data, through a small handful of exemplars. We demonstrate this on $10,000$ digits taken from the MNIST dataset, reduced to be represented by a subset of size $M = 30$ in Figure 4. Each element in the MNIST image dataset is a 28×28 image of a single hand-written numeric digit, with $\mathbf{x}_i \in [0,1]^{768}$. Again, we simply use a squared exponential kernel, with lengthscale set to the median of the first M points, and $D = 200$ random features.

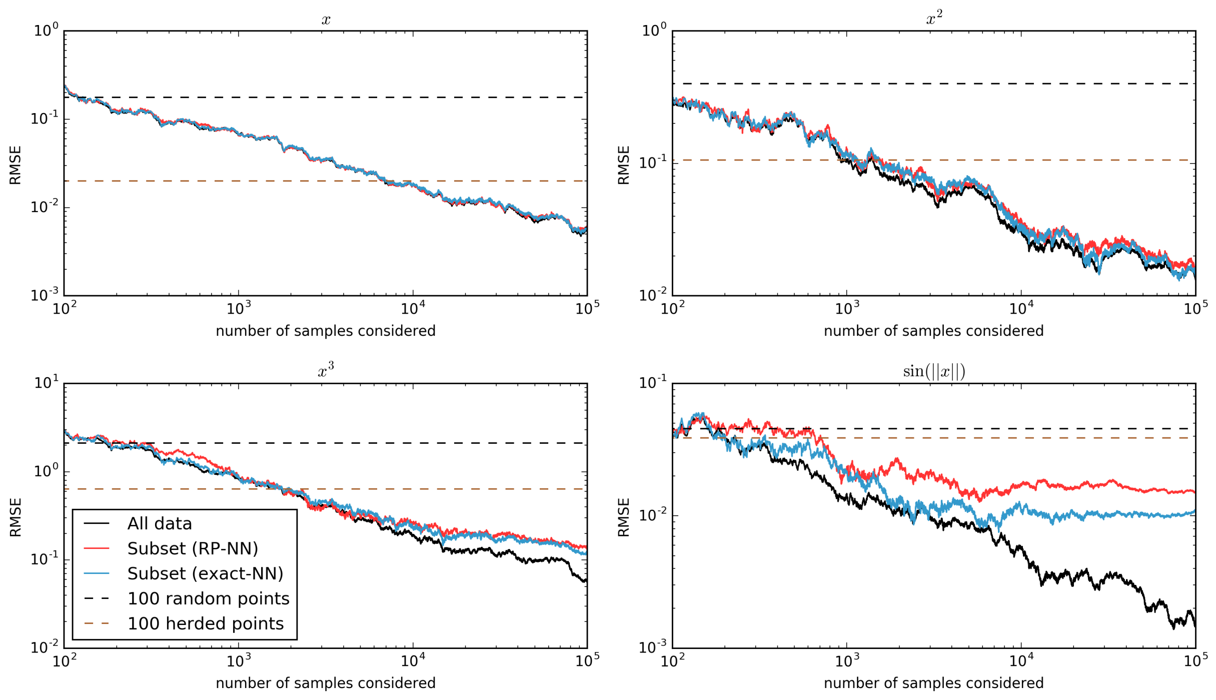


Figure 3: Error plots for expectations of four test functions: $\mathbf{x}$, $\mathbf{x}^2$, $\mathbf{x}^3$, and $\sin(||\mathbf{x}||_2)$. Legend is shared across subplots. The error is root mean squared error (RMSE) across dimensions of the test function, relative to a ground truth value computed on a separate sample of 2×10^7 points. The "random" benchmark is the median RMSE across 100 different random subsets of size $M = 100$; the "herding" benchmark is the RMSE from the first 100 herded points, targeting $\hat{\mu}_N$. We can also judge the loss of accuracy in using the approximate nearest neighbor search: there is a qualitative difference only in the $\sin(||\mathbf{x}||_2)$ example, where there is plateau in convergence for both methods. All estimates are averaged over 10 different synthetic datasets, drawn from mixtures of Gaussians with different parameters.

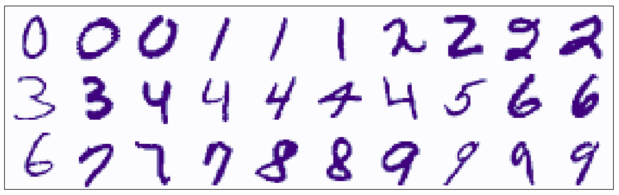


Figure 4: We summarize the MNIST digits dataset with a small number of exemplars. The subsampled set of points provides good coverage for the different digits, and also shows variation in style and form within each digit. The subsampling algorithm was given only the unlabeled digits.

The result summarizes the MNIST digits far better than a random sample would, and enormously better than other simple techniques for visualizing high-dimensional data, such as computing marginal means or quantiles across each dimension. We see a small number of exemplars from each of the 10 digits, with a good variety of handwriting styles expressed.

5.3 Resampling output of an importance sampler

One advantage of this algorithm relative to standard reservoir sampling is that it is trivially modified to run on weighted streaming data, requiring only changing the way in which the running average $\hat{\mu}_n$ is computed from Equation (18) to Equation (19). A promising use of this algorithm is to resample the output from sampling schemes such as importance sampling and sequential Monte Carlo methods, which return large numbers of weighted sets of samples. We may wish to resample from these weighted samples to obtain a new unweighted set of particles; such a resampling step is also used within sequential Monte Carlo as a means of reducing particle degeneracy [Douc et al., 2005].

As an illustrative example, we run this algorithm on the output of an importance sampler targeting a simple one-dimensional mixture distribution, proposing from a broad Gaussian prior. The results are shown in Figure 5, showing the incremental progress as the algorithm processes more points (i.e., as n increases), for a variety of small values of M. Due to the small number of points M being selected, we can see that as n becomes large, the selected points roughly approximate the quantiles of the bimodal mixture model.

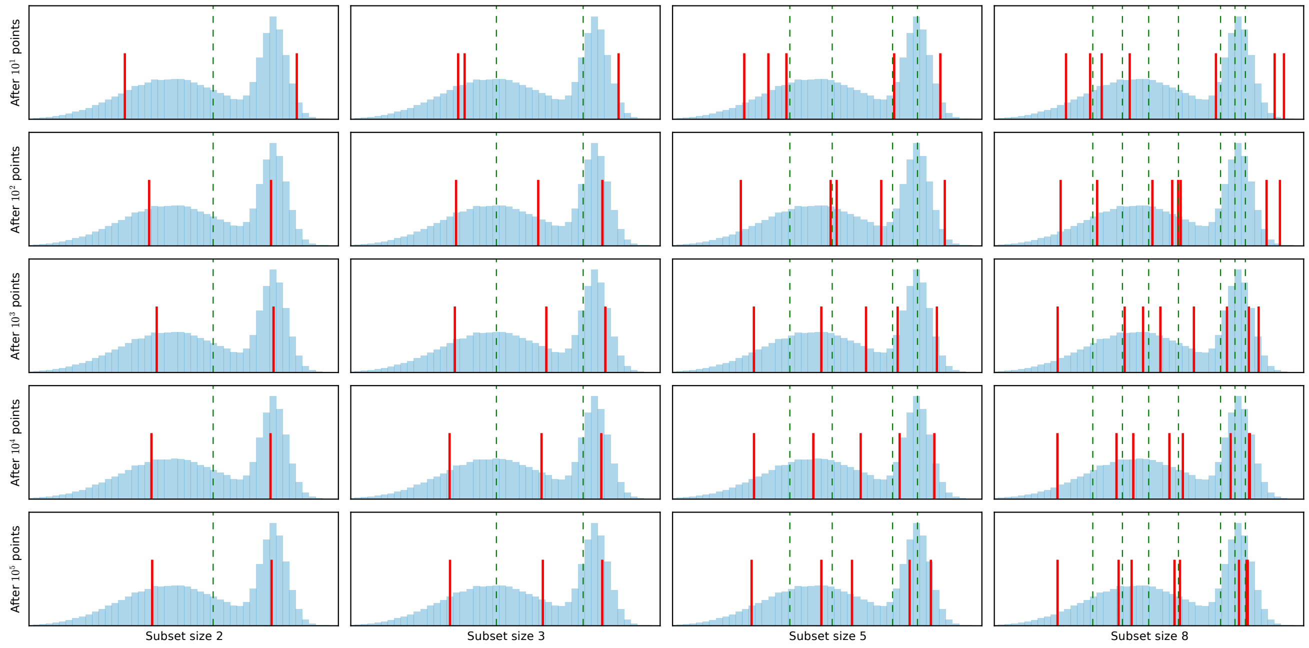


Figure 5: Downsampling points from an importance sampler. Selected points shown in red, with separate runs in each column for $M = 2, 3, 5, 8$, showing the results after observing each of $N = 10, 10^2, 10^3, 10^4, 10^5$. The green dashed lines split the probability density function of the blue target density into regions of equal probability. As N increases, the selected points approximate an even weighting across the probability density.

That is, in this example we see empirically that the points are selected such that they partition the target density $p(\mathbf{x})$ into regions of equal probability mass.

6 DISCUSSION

Reservoir sampling is a popular algorithm for drawing "manageable" subsets from large data, for both reasons of computer memory limitations, and for human visualization. For either of these purposes, our reservoir super-sampling algorithm can be used as a drop-in replacement which provides improved performance when subsampling from any data for which a kernel function can be defined.

When sampling from a weighted set of points, then the computational complexity of this algorithm is of the same order in M and N as the random sampling algorithm of [Efraimidis and Spirakis, 2006], while providing performance in computing expectations which far closer emulates computing expectations from the full data stream. This subsampling method, which explicitly use the values of the different points, could perhaps be used as a replacement for traditional resampling in sequential Monte Carlo methods, which only consider the particle weights and not the actually values at each point. Such an approach may be advantageous in settings such as considered in Jun and Bouchard-Côté [2014], where the memory usage for storing the particle set is the primary bottleneck.

Although the algorithm aims only to minimize the maximum mean discrepancy, there is evidence from the MNIST example and the point locations in the importance sampling reweighting that the selected points also have a promising use in general situations where one might want to summarize data with a small number of representative points.

As future work, we also hope to investigate the relationship between these selected locations and other methods for constructing low-discrepancy point sets.

Acknowledgments

BP would like to thank Guillaume Alain for first introducing him to reservoir sampling. FW is supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement number FA8750-14-2-0006, Sub Award number 61160290-111668.

References

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bochner, S. (1959). *Lectures on Fourier integrals*. Number 42. Princeton University Press.

Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schölkopf, B., and Smola, A. J. (2006). Integrating structured biological data by kernel maximum mean discrepancy. *Bioinformatics*, 22(14):e49–e57.

Chen, Y., Welling, M., and Smola, A. (2010). Super-samples from kernel herding. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence (UAI2010)*.

Dasgupta, S. and Sinha, K. (2015). Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263.

Douc, R., Cappé, O., and Moulines, E. (2005). Comparison of resampling schemes for particle filtering. In *In 4th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 64–69.

Efraimidis, P. S. and Spirakis, P. G. (2006). Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185.

Freund, Y., Dasgupta, S., Kabra, M., and Verma, N. (2007). Learning the structure of manifolds using random projections. In *Advances in Neural Information Processing Systems*, pages 473–480.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773.

Huszár, F. and Duvenaud, D. (2012). Optimally-weighted herding is Bayesian quadrature. *Uncertainty in Artificial Intelligence (UAI)*.

Johnson, W. B. and Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics*, 26(189-206):1.

Jun, S.-H. and Bouchard-Côté, A. (2014). Memory (and time) efficient sequential Monte Carlo. In *Proceedings of the 31st international conference on Machine learning*, pages 514–522.

McDiarmid, C. (1989). On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188.

Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184.

Smola, A., Gretton, A., Song, L., and Schölkopf, B. (2007). A hilbert space embedding for distributions. In *Algorithmic learning theory*, pages 13–31. Springer.

Song, L. (2008). Learning via Hilbert space embedding of distributions.

Sutherland, D. J. and Schneider, J. (2015). On the error of random Fourier features. In *Uncertainty in Artificial Intelligence (UAI)*.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57.