# DROP: Optimizing Stochastic Dimensionality Reduction via Workload-Aware Progressive Sampling

Sahaana Suri, Peter Bailis
Stanford University

## ABSTRACT

Dimensionality reduction is a critical step in machine learning pipelines. Principal Component Analysis (PCA) is frequently the method of choice, yet is often prohibitively expensive. Theoretical means of accelerating PCA via sampling have been proposed, but typically treat PCA as a reusable statistical operator, independent of downstream analytics workflows. We show how accounting for downstream analytics operations during dimensionality reduction via PCA allows stochastic methods to efficiently operate over small (e.g., 1%) subsamples of input data, reducing computational overhead and end-to-end runtime. We propose a dimensionality reduction optimizer that enables speedups of up to 5× over Singular-Value-Decomposition-based PCA techniques, and achieves parity with or exceeds conventional approaches like FFT and PAA by up to 16× in end-to-end workloads.

## 1 INTRODUCTION

Rapid growth in high-dimensional data from automated data sources [11, 38] poses a scalability challenge for machine learning (ML) pipelines. Dimensionality reduction (DR) techniques can alleviate this scalability challenge [15, 25, 39, 41]. In exchange for a runtime cost ($R$), DR methods transform an $n$-dimensional dataset to a lower $k$-dimensional representation while preserving salient dataset features, allowing downstream analytics routines to run in time proportional to $k$, while preserving downstream task accuracy (see Figure 1).

Principal Component Analysis (PCA) is often practioners' DR method of choice with respect to transformation quality ($k$) for a target accuracy [37]. However, naïve, task-independent PCA implementations scale poorly with dimension, resulting in runtimes ($R$) that outweigh the downstream runtime benefit of DR. Thus, practitioners may sacrifice quality for end-to-end runtime, and use PCA alternatives [22].

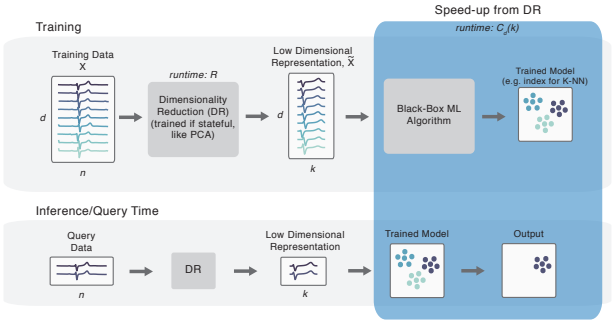


**Figure 1: Sample machine learning pipeline with dimensionality reduction. Spending time on DR provides downstream runtime speed-ups.**

Sample-based, stochastic PCA algorithms [19, 57] are a scalable alternative to classical PCA. However, the amount of sampling required is data-dependent. If we sample too many data points, the runtime overhead of PCA could outweigh the transformation quality. If sample too few data points, PCA could fail to deliver a sufficiently high-quality reduction and compromise the runtime and/or accuracy of downstream analytics. As a result, we ask: can we develop a workload-dependent means of efficiently and accurately determining the sampling rate for stochastic PCA, so we can obtain PCA's transformation quality *and* minimize workload runtime?

To this end, we develop DROP[1], a system that performs whole-workload runtime optimization by dynamically identifying the amount of sampling required for stochastic PCA. DROP takes a high-dimensional dataset,[2] property to preserve (e.g., pairwise Euclidean distance to 5%), and optional runtime model expressing downstream workload performance as a function of dimensionality (e.g., for k-Nearest Neighbors [k-NN], runtime is linear in dimensionality). DROP returns a low-dimensional transformation for the input using as few samples as needed to minimize the projected overall workload runtime while satisfying quality constraints.

DROP addresses the question of how much to sample the input dataset via data-dependent progressive sampling and online progress estimation at runtime. DROP performs


*DEEM'19, June 2019, Amsterdam, The Netherlands*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://github.com/stanford-futuredata/DROP
[2] Our focus is on time series similarity search, given the amount of study in the database community [22] and resurging interest in time series analytics systems [10, 11, 53]. We provide preliminary generalizability analyses in §5.

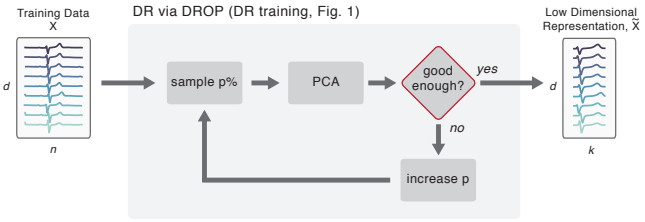


**Figure 2: DROP is a workload-aware DR operator compatible with standard ML pipelines. DROP solves the challenge of when to stop sampling ("good enough?")**

PCA on a small sample to obtain a candidate transformation, then increases the number of samples until termination (see Figure 2). To identify the termination point that minimizes runtime, DROP must overcome three challenges:

First, given the results of PCA on a data sample, DROP must *evaluate the quality* of the current candidate transformation. Popular analytics and data mining tasks often require approximate preservation of metrics such as average pairwise distances between data points [24, 33], which are costly to compute. Thus, DROP adapts confidence intervals for fast estimation of the input metric to preserve.

Second, DROP must *estimate the marginal benefit of sampling additional datapoints*. When running PCA on a series of larger samples, later samples will increase $R$, but may return lower $k$ (lower downstream runtime). To navigate this trade-off between end-to-end runtime and transformation quality, DROP uses the results obtained from previous iterations to build a performance model for future iterations.

Finally, given the expected marginal benefit of the next iteration, DROP must *optimize end-to-end runtime*. While an application-agnostic approach would iterate until successive iterations yield no quality benefit, a user-provided cost model may reveal that trading a higher $k$ for a lower $R$ may decrease end-to-end runtime. DROP therefore evaluates this cost at each iteration to minimize the expected workload runtime.

DROP is a combination of recent theoretical advances in DR and classic techniques from approximate query processing, and useful system for end-to-end workflow optimization. In this work, we make the following contributions:

- We show the data sample required to perform accuracy-achieving PCA is often small (as little as 1%), and sampling can enable up to 91× speedup over baseline PCA.
- We propose DROP, an online optimizer for DR that uses information about downstream analytics tasks to perform efficient stochastic PCA.
- We present techniques based on progressive sampling, approximate query processing, online progress estimation, and cost based optimization to enable up to 5× faster end-to-end execution over PCA via SVD.

## 2 BACKGROUND AND PROBLEM

We provide background on dimensionality reduction (DR), and define our problem of workload-aware DR.

### 2.1 Dimensionality Reduction

DR refers to finding a low-dimensional representation of a dataset that preserves properties of interest, such as data point similarity [18, 27]. Formally, consider a data matrix $X \in \mathbb{R}^{d \times n}$, where each row $i$ corresponds to data point $x_i \in \mathbb{R}^n$, with $d > n$. DR computes a transformation function ($T : \mathbb{R}^n \rightarrow \mathbb{R}^k$) that maps each $x_i$ to a new basis as $\tilde{x}_i \in \mathbb{R}^k$ where $k \leq n$, resulting in a new data matrix $T(X) = \tilde{X} \in \mathbb{R}^{d \times k}$.

*Principal Component Analysis (PCA).* PCA is a linear DR technique that identifies a new orthogonal basis for a dataset that captures its directions of highest variance. Of all linear transformations, this basis minimizes reconstruction error in a mean square sense. Classically implemented PCA uses a Singular Value Decomposition (SVD) routine [59].

### 2.2 DR for Repeated-Query Workloads

In workloads such as similarity search, clustering, or classification, ML models are periodically trained over historical data, and are *repeatedly queried* as incoming data arrives or new query needs arise (see Fig 1). Indexes built over this data can improve the efficiency of this repeated query workload in exchange for a preprocessing overhead. DR with a multi-dimensional index structure in the reduced space is a classic way of achieving this, and is the basis for popular similarity search procedures and extensions in the data mining and machine learning communities [15, 33, 39, 45, 54, 66].

*DR in Similarity Search.* Similarity search is a repeated-query workload performed over data types including images, documents and time series [22, 28]. We use similarity search as a running case study given its popularity and the large amount of research in the space. The Tightness of Lower Bounds ($TLB$) is typically the metric preserved by DR in this setting [22], as it measures how well a *contractive* transformation (i.e. distances in the transformed space are less than or equal to those in the original) preserves pairwise distances. It can estimate the accuracy of a low dimensional transformation without performing the similarity search task:

$$TLB = \frac{2}{d(d-1)} \sum_{i<j} \frac{\|\tilde{x}_i - \tilde{x}_j\|_2}{\|x_i - x_j\|_2}. \quad (1)$$

### 2.3 Problem: Workload-Aware DR

In workload-aware DR, we perform DR to minimize whole workload runtime subject to downstream accuracy constraints. DR is a fixed cost (i.e., index construction for similarity

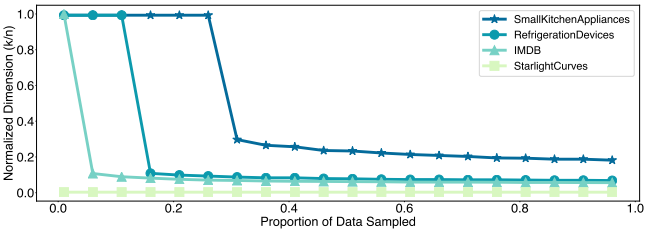


**Figure 3: Improvement in representation size for $TLB = 0.80$ across three datasets. Higher sampling rates improve quality until reaching a state equivalent to running PCA over the full dataset ("convergence")**

search), while each query incurs a marginal cost that is dependent on DR quality (i.e., nearest neighbor query); lower-dimensional data points result in faster queries.

As input, consider an input dataset ($X$), target metric preservation ($B$; e.g., $TLB \geq .99$), and optional downstream runtime as a function of dimensionality ($C_d(n)$ for an $d \times n$ matrix). Denoting DR runtime as $R$, we define the problem:

**Problem 2.1.** *Given $X \in \mathbb{R}^{d \times n}$, TLB constraint $B \in (0, 1]$, confidence $c$, and workload runtime function $C_d : \mathbb{Z}_+ \rightarrow \mathbb{R}_+$, find $k$ and transformation matrix $T_k \in \mathbb{R}^{n \times k}$ that minimizes $R + C_d(k)$ such that $TLB(XT_k) \geq B$ with confidence $c$.*

We assume the downstream runtime model $C_d(n)$ is monotonically increasing in $n$. Absent the cost model, we default to execution until convergence (i.e, until $k$ plateaus) as described in §3, and demonstrate the cost of doing so in §5.

The more time spent on DR ($R$), the smaller the transformation ($k$), thus the lower the workload runtime. To minimize $R + C_d(k)$, we must determine how much time to spend on DR to minimize end-to-end runtime.

## 3 SIMILARITY SEARCH CASE STUDY

To tackle workload-aware DR, we first perform a case study on time series similarity search by revisiting a comparison of DR techniques from VLDB 2008 [22]. We show that PCA can outperform classic techniques (low $k$), but at a high cost ($R$). We then demonstrate how sample-based PCA can bridge this gap, but that the number of samples required varies per dataset. Finally, we show how dynamically increasing the sampling rate can help identify how much to sample a given dataset, providing a foundation for workload-aware DR.

### 3.1 PCA Speed vs. Quality

While improved quality provides faster repeated query execution, the cost of DR via PCA dominates this speedup, encouraging the use of faster, lower-quality alternatives [22].

To briefly quantify this trade-off, we augment a widely-cited time series similarity search DR study from VLDB 2008 [22] by evaluating PCA—which the authors did not benchmark due to it being "untenable for large data sets." We compare PCA via SVD to baseline techniques based on runtime and DR performance with respect to $TLB$ over the largest datasets from [22]. We use their two fastest methods as our baselines as they show the remainder exhibited "very little difference": Fast Fourier Transform (FFT) and Piecewise Aggregate Approximation (PAA).

**TLB Performance Comparison** We compute the minimum dimensionality achieved by each technique subject to a $TLB$ constraint. On average, PCA provides bases that are 2.3× (up to 3.9×) and 3.7× (up to 26×) smaller than PAA and FFT for $TLB = 0.75$, and 2.9× (up to 8.3×) and 1.8× (up to 5.1×) smaller for $TLB = 0.99$. While the margin between PCA and alternatives is dataset-dependent, PCA almost always preserves $TLB$ with a lower dimensional representation.

**Runtime Performance Comparison** PCA implemented via out-of-the-box SVD is on average over 26× (up to 56×) slower than PAA and over 4.6× (up to 9.7×) times slower than FFT when computing the smallest $TLB$-preserving basis.

### 3.2 Incremental, Progressive Sampling

To bridge this performance-runtime gap, we turn to data sampling. Many real-world datasets are intrinsically low-dimensional, as evidenced by their rapid falloff in their eigenvalue spectrum. A data sample thus captures much of the dataset's "interesting" behavior, so fitting models over data samples generalize well. We verify this by varying the target $TLB$ and examining the minimum number of uniformly selected samples required to obtain a $TLB$-preserving transform with output dimension $k$ equal to input dimension $n$.

On average, a sample of under 0.64% (up to 5.5%) of the input is sufficient for $TLB = 0.75$, and under 4.2% (up to 38.6%) is sufficient for $TLB = 0.99$. If this sample rate is known, we obtain up to 91× speedup compared to a naïve implementation of PCA via SVD—with no algorithmic improvement.

However, this benefit is dataset-dependent, and unknown a priori. We thus turn to progressive sampling (gradually increasing the sample size) to identify how large a sample suffices. Figure 3 shows how the dimensionality required to attain a given $TLB$ changes when we vary dataset and proportion of data sampled. Increasing the number of samples provides lower dimensional transformations for the same quality. However, this decrease in dimension plateaus as the number of samples increases. Thus, while progressive sampling would allow us to tune the amount of time spent on DR, we must determine when the downstream value of decreased dimension is overpowered by the cost of additional DR—that is, whether to sample to convergence (evaluated in §5.3) or terminate early (e.g., at 0.3 proportion of data sampled for SmallKitchenAppliances).

**Algorithm 1** DROP Algorithm

**Input:**
$X$: data matrix
$B$: target metric preservation level
$C_d$: cost of downstream operations; default tuned to k-NN

**Output:**
$T_k$: $k$-dimensional transformation matrix

1: **function** DROP($X, B, C_d$):
2: Initialize: $i = 0; k_0 = \infty$ ▷ iteration and current basis size
3: **do**
4: i++, CLOCK.RESTART
5: $X_i$ = SAMPLE($X$, SAMPLE-SCHEDULE($i$)) ▷ § 4.2
6: $T_{k_i}$ = COMPUTE-TRANSFORM($X, X_i, B$) ▷ § 4.3
7: $r_i$ = CLOCK.ELAPSED ▷ $R = \sum_i r_i$
8: $\hat{k}_{i+1}, \hat{r}_{i+1}$ = ESTIMATE($k_i, r_i$) ▷ § 4.4
9: **while** OPTIMIZE($C_d, k_i, r_i, \hat{k}_{i+1}, \hat{r}_{i+1}$) ▷ § 4.5
10: **return** $T_{k_i}$

# 4 DROP: WORKLOAD OPTIMIZATION

DROP answers a crucial question that stochastic PCA techniques have traditionally ignored: how long should these methods run?

## 4.1 DROP Architecture

DROP operates over a series of data samples, and determines when to terminate via a four-step procedure that is repeated for each iteration:

**Step 1: Progressive Sampling (§4.2, Alg 1 L5, Fig 2A)** DROP draws a data sample, performs PCA over it, and uses of a novel reuse mechanism across iterations (§4.7).

**Step 2: Transform Evaluation (§4.3, Alg 1 L6, Fig 2B)** DROP then evaluates the quality of the above by identifying the size of the smallest metric-preserving transformation that can be extracted.

**Step 3: Progress Estimation (§4.4, Alg 1 L8, Fig 2C)** Given the size of the metric-preserving transform and the time required to obtain this transform, DROP estimates the size and computation time of continued iteration.

**Step 4: Cost-Based Optimization (§4.5, Alg 1 L9, Fig 2D)** DROP optimizes over DR and downstream task runtime to determine if it should terminate.

## 4.2 Progressive Sampling

DROP repeatedly chooses a subset of data and computes PCA on the subsample (via one of the methods in § 4.6). We consider a simple uniform sampling strategy: each iteration, DROP samples a fixed percentage of the data.

## 4.3 Evaluating Transformations

Given a transformation obtained by running PCA over a sample, DROP must accurately and efficiently evaluate the performance of this transformation with respect to a metric of interest over the entire dataset, not just the data sample.

We define the performance of a transformation computed over a sample as the size of the lowest dimensional *TLB*-preserving transform ($k$) that can be extracted. There are two challenges in evaluating this performance. First, the $k$ that achieves the *TLB* constraint is rarely known a priori. Second, brute-force *TLB* computation would dominate the runtime of computing PCA over a sample.

*4.3.1 Computing the Lowest Dimensional Transformation.* DROP first computes a full, $n$-dimensional basis via PCA over the data sample. To reduce dimensionality, DROP must determine if a smaller dimensional *TLB*-preserving transformation can be computed over this sample, and return the smallest such transform. Ideally, the smallest $k$ would be known a priori, but in practice, this is not true. Thus, DROP uses the *TLB* constraint and two properties of PCA to automatically identify the size of the returned transformation.

First, PCA via SVD produces an orthogonal linear transformation where the principal components are returned in order of decreasing dataset variance explained. Once DROP has computed the transformation matrix for dimension $n$, DROP obtains the transformations for all dimensions $k$ less than $n$ truncating the matrix to $n \times k$, as .

Second, with respect to *TLB* preservation, the more principal components that are retained, the better the lower-dimensional representation in terms of *TLB*. This is because orthogonal transformations such as PCA preserve inner products. Therefore, a $n$-dimensional PCA perfectly preserves $\mathcal{L}_2$-distance between data points. As the $\mathcal{L}_2$-distance is a sum of squared (positive) terms, the more principal components that are retained, the better the representation preserves $\mathcal{L}_2$-distance.

Using the first property, DROP obtains all low-dimensional transformations for the sample from the $n$-dimensional basis. Using the second property, DROP runs binary search over these transformations to return the lowest-dimensional basis that attains $B$ (i.e., COMPUTE-TRANSFORM, line 1 of Algorithm 2). If a target $B$ cannot be realized with this sample, DROP omits all further optimization steps in this iteration and continues the next iteration by drawing a larger sample.

Computing the full $n$-dimensional basis at every step may be wasteful. Thus, if DROP has previously found a candidate *TLB*-preserving basis of size $n' < n$ in prior iterations, then DROP only computes $n'$ components at the start of the next iteration. This allows for more efficient PCA computation for future iterations, as advanced PCA routines can exploit the $n'$-th eigengap to converge faster (§6).

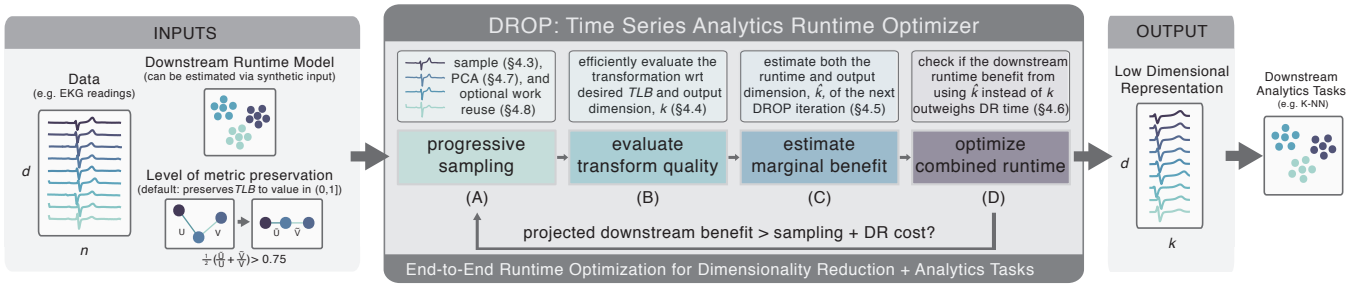


**Figure 4: High-level DROP architecture depicting DROP's inputs, outputs, and core components.**

*4.3.2 TLB Computation.* Given a transformation, DROP must efficiently determine if the basis preserves the desired *TLB*. Computing pairwise *TLB* for all data points requires $O(d^2n)$ time, which dominates the runtime of computing PCA on a sample. However, as the *TLB* is an average of random variables bounded from 0 to 1, DROP can use statistical sampling and confidence intervals to compute the *TLB* to arbitrary confidences.

Given a transformation, DROP iteratively refines an estimate of its *TLB* (function EVALUATE-TLB in Algorithm 2, line 11) by incrementally sampling an increasing number of pairs from the input data (Algorithm 2, line 15), transforming each pair into the new basis, then measuring the distortion of $\mathcal{L}_2$ distance between the pairs, providing a *TLB* estimate to confidence level $c$ (Algorithm 2, line 19). If the confidence interval's lower bound is greater than the target *TLB*, the basis is a sufficiently good fit; if its the upper bound is less than the target *TLB*, the basis is not a sufficiently good fit. If the confidence interval contains the target *TLB*, DROP cannot determine if the target *TLB* is achieved. Thus, DROP automatically samples additional pairs to refine its estimate.

To estimate the *TLB* to confidence $c$, DROP uses the Central Limit Theorem (similar to online aggregation [34]): computing the standard deviation of a set of sampled pairs' *TLB* measures and applying a confidence interval to the sample according to the $c$. For data with low variance, DROP evaluates a candidate basis with few samples from the dataset as the confidence intervals shrink rapidly.

The techniques in this section are presented in the context of *TLB*, but can be applied to any downstream task and metric for which we can compute confidence intervals and are monotonic in number of principal components retained.

## 4.4 Progress Estimation

Given a low dimensional *TLB*-achieving transformation from the evaluation step, DROP must identify the quality (dimensionality) and cost (runtime) of the transformation that would be obtained from an additional DROP iteration.

**Algorithm 2** Basis Evaluation and Search

**Input:**
$X$: sampled data matrix
$B$: target metric preservation level; default $TLB = 0.98$

1: **function** COMPUTE-TRANSFORM($X, X_i B$):
2: PCA.FIT($X_i$) ▷ fit PCA on the sample
3: Initialize: high = $k_{i-1}$; low = 0; $k_i = \frac{1}{2}$(low + high); $B_i = 0$
4: **while** (low ! = high) **do**
5: $T_{k_i}, B_i$ = EVALUATE-TLB($X, B, k_i$)
6: **if** $B_i \leq B$ **then** low = $k_i + 1$
7: **else** high = $k_i$
8: $k_i = \frac{1}{2}$(low + high)
9: $T_{k_i}$ = cached $k_i$-dimensional PCA transform
10: **return** $T_{k_i}$

11: **function** EVALUATE-TLB($X, B, k$):
12: numPairs = $\frac{1}{2}d(d-1)$
13: $p = 100$ ▷ number of pairs to check metric preservation
14: **while** ($p <$ numPairs) **do**
15: $B_i, B_{lo}, B_{hi}$ = TLB($X, p, k$)
16: **if** ($B_{lo} > B$ or $B_{hi} < B$) **then break**
17: **else** pairs ×= 2
18: **return** $B_i$

19: **function** TLB($X, p, k$):
20: **return** mean and 95%-CI of the *TLB* after transforming $p$ $d$-dimensional pairs of points from $X$ to dimension $k$. The highest transformation computed thus far is cached to avoid recomputation of the transformation matrix.

Recall that DROP seeks to minimize objective function $R + C_d(k)$ such that $TLB(XT_k) \geq B$, with $R$ denoting DROP's total runtime, $T_k$ the $k$-dimensional *TLB*-preserving transformation of data $X$ returned by DROP, and $C_d(k)$ the workload cost function. Therefore, given a $k_i$-dimensional transformation $T_{k_i}$ returned by the evaluation step of DROP's $i^{\text{th}}$ iteration, DROP can compute the value of this objective function by substituting its elapsed runtime for $R$ and $T_{k_i}$ for $T_k$. We denote the value of the objective at the end of iteration $i$ as $obj_i$. To decide whether to continue iterating to find an

improved transformation, DROP must be able to estimate the objective function value of future iterations.

In §4.5 we show that DROP requires $obj_{i+1}$ to minimize this objective function. To estimate $obj_{i+1}$, DROP must estimate the runtime required for iteration $i+1$ (which we denote as $r_{i+1}$, where $R = \sum_i r_i$ after $i$ iterations) and the dimensionality of the $TLB$-preserving transformation produced by iteration $i+1$, $k_{i+1}$. Because DROP cannot directly measure $r_{i+1}$ or $k_{i+1}$ without performing iteration $i+1$, DROP performs online progress estimation to estimate these quantities. Specifically, DROP performs online parametric fitting to compute future values based on prior values for $r_i$ and $k_i$ in line 8 of Algorithm 1. By default, given a sample of size $m_i$ in iteration $i$, DROP performs linear extrapolation to estimate $k_{i+1}$ and $r_{i+1}$. The estimate of $r_{i+1}$, for instance, is:

$$\hat{r}_{i+1} = r_i + \frac{r_i - r_{i-1}}{m_i - m_{i-1}}(m_{i+1} - m_i)$$

## 4.5 Cost-Based Optimization

Given the results of the progress estimation step, DROP must determine if continued PCA on additional samples will be beneficial to overall runtime, or if it is better to terminate.

Given predictions of the next iteration's runtime ($\hat{r}_{i+1}$) and dimensionality ($\hat{k}_{i+1}$), DROP uses a greedy heuristic in estimating the optimal objective-minimizing stopping point. Concretely, if the objective function estimate for the next iteration is greater than its current objective function value ($obj_i < \widehat{obj}_{i+1}$), then DROP will terminate. If DROP's runtime is convex in the number of iterations, it is straightforward to prove that this condition is in fact the optimal stopping criterion (i.e., via convexity of composition of convex functions). This stopping criterion leads to a simple check at each DROP iteration that is used by OPTIMIZE in Algorithm 1 line 9:

$$obj_i < \widehat{obj}_{i+1}$$

$$C_d(k_i) + \sum_{j=0}^{i} r_j < C_d(\hat{k}_{i+1}) + \sum_{j=0}^{i} r_j + \hat{r}_{i+1}$$

$$C_d(k_i) - C_d(\hat{k}_{i+1}) < \hat{r}_{i+1} \quad (2)$$

DROP terminates when the projected time of the next iteration exceeds the estimated downstream runtime benefit.

## 4.6 Choice of PCA Subroutine

The most straightforward means of implementing PCA via SVD in DROP is computationally inefficient compared to DR alternatives (§2). In DROP, we compute PCA via a randomized SVD algorithm from [31] (SVD-Halko). While additional advanced methods for efficient PCA exist (§6), we found that not only is SVD-Halko asymptotically of the same running time as techniques used in practice, it is straightforward to implement, can take advantage of optimized linear algebra libraries, and does not require tuning for hyperparameters such as batch size, learning rate, or convergence criteria. While SVD-Halko is not as efficient as other techniques with respect to communication complexity as in [23], or convergence rate as in [19], these techniques can be easily substituted for SVD-Halko in DROP's architecture.

## 4.7 Work Reuse

A natural question arises due to DROP's iterative architecture: can we combine the information from each sample's transformation without computing PCA over the union of the sampled data points? Stochastic methods for PCA enable such work reuse across samples as they iteratively refine a single transformation matrix, but other methods do not. DROP uses two key insights in order to enable work reuse when utilizing arbitrary PCA routines.

First, given two PCA transformation matrices, $T_1$ and $T_2$, their horizontal concatenation $H = [T_1|T_2]$ is a transformation into the union of their range spaces. Second, the principal components returned from running PCA on repeated data samples will generally concentrate to the true top principal components for datasets with rapid spectrum drop off. Work reuse proceeds via two-step concatenate-distill approach: DROP maintains a transformation history consisting of the horizontal concatenation of all PCA transformations to this point, and then computes the SVD of this matrix and returns the first $k$ columns as the transformation matrix.

Although this routine requires an SVD computation, computational overhead is dependent size of the history matrix, not the dataset size. This size is proportional to the original dimensionality $n$ and size of lower dimensional transformations, which are in turn proportional to the data's intrinsic dimensionality and the $TLB$ constraint. As preserving *all history* can be expensive in practice, DROP periodically shrinks the history matrix using DR via PCA. We validate the benefit of using work reuse—up to 15% on real-world data—in §5.

# 5 EXPERIMENTAL EVALUATION

We evaluate DROP's efficiency along three dimensions: runtime, accuracy, and extensibility. We demonstrate that (1) DROP outperforms PAA and FFT in end-to-end, repetitive-query workloads, (2) DROP's optimizations each contribute to performance, and (3) DROP extends beyond time series.

## 5.1 Experimental Setup

**Implementation** We implement DROP as an in-memory, batch-oriented feature transformation dataflow operator in Java using the multi-threaded Matrix-Toolkits-Java (MTJ) library [32] and netlib-java [4] linked against Intel MKL [5] for

compute-intensive linear algebra operations. We use multi-threaded JTransforms [2] for FFT, and implement multi-threaded PAA from scratch. We use the Statistical Machine Intelligence and Learning Engine (SMILE) library [1] for k-NN with different index structures, and k-means.

**Datasets** We first consider the UCR Time Series Classification Archive [17] for indexing experiments and lesion studies. We exclude datasets with fewer than 1 million entries, and fewer datapoints than dimensionality, leaving 14 datasets.

Due to the relatively small size of these time series datasets, we consider three additional datasets to showcase tangible wall-clock runtime improvements with DROP. We use the standard MNIST hand-written digits dataset [46], the FMA featurized music dataset [21], and a labeled sentiment analysis IMBb dataset [8], which also demonstrates extensibility beyond time series data.

**DROP Configuration** We use a runtime cost function for k-NN obtained via linear interpolation on data of varying dimension (implemented via cover trees [12], K-D trees [56],or brute force search in SMILE). To evaluate the sensitivity to cost model, we also report on the effect of operating without a cost model (i.e., sample until convergence) in §5.3. We set *TLB* constraints such that the accuracy of K-NN tasks remain unchanged before and after indexing via DR, corresponding to $B = 0.99$ for the UCR datasets. Unless otherwise specified, we use a default sampling schedule that begins with and increases by 1% of the input. It is possible to optimize (and possibly overfit) this schedule for our target time series, but we provide a conservative, more general schedule.

**Baselines** We report runtime, accuracy, and reduced dimension compared to FFT, PAA, PCA via SVD-Halko, and PCA via SVD. Each computes a transformation over the entire data, then performs binary search to identify the smallest dimensional basis that satisfies the target *TLB*.

**Similarity Search/k-NN Setup** While many methods for similarity search exist, as in [22], we consider k-NN in our evaluation as it is classically used and interoperates with new use cases including one-shot learning and deep metric learning [49, 55, 62]. To evaluate DR performance when used with downstream indexes, we vary k-NN's multidimensional index structure: cover trees, K-D trees, or no index.

End-to-end performance depends on the number of queries in the workload, and DROP is optimized for the repeated-query use case. Due to the small size of the UCR datasets, we choose a 1:50 ratio of data indexed to number of query points, and vary this index-query ratio in later microbenchmarks and experiments. We also provide a cost model for assessing the break-even point that balances the cost of a given DR technique against it's indexing benefits to each query point.

## 5.2 DROP Performance

We evaluate DROP's performance compared to PAA and FFT using the time series case study.

**k-NN Performance** We summarize DROP's results on an end-to-end 1-Nearest Neighbor classification in Figure 5. We display the end-to-end runtime of DROP, PAA, and FFT for each of the considered index structures: no index, K-D trees, cover trees. We display the size of the returned dimension for the no indexing scenario, as the other two scenarios return near identical values. This occurs as many of the datasets used in this experiment are small and possess low intrinsic dimensionality; DROP's cost model thus determines to quickly identify this dimensionality prior to termination. We do not display k-NN accuracy as all techniques meet the *TLB* constraint, and achieve the same accuracy within 1%.

On average, DROP returns transformations that are 2.3× and 1.4× smaller than PAA and FFT, translating to significantly smaller k-NN query time. End-to-end runtime with DROP is on average 2.2× and 1.4× (up to 10× and 3.9×) faster than PAA and FFT, respectively, when using brute force linear search, 2.3× and 1.2× (up to 16× and 3.6×) faster when using K-D trees, and 1.9× and 1.2× (up to 5.8× and 2.6×) faster when using cover trees. When evaluating Figure 5, it becomes clear that DROP's runtime improvement is data dependent for both smaller datasets, and for datasets that do not possess a low intrinsic dimension (such as Phoneme, elaborated on in §5.3) Determining if DROP is a good fit for a dataset is an exciting area for future work (§7).

**Varying Index-Query Ratio**

DROP is optimized for a low index-query ratio, as in many streaming and/or high-volume data use cases. If there are many more data points queried than used for constructing an index, DROP will outperform alternatives. A natural question that arises is in which scenarios is it beneficial to use DROP. While domain experts are typically aware of the scale of their query workloads, we also provide a heuristic to answer this question given rough runtime and cardinality estimates of the downstream task and the alternative DR technique.

Let $x_d$ and $x_a$ be the per-query runtime of running a downstream task with the output of DROP and a given alternative method, respectively. Let $r_d$ and $r_a$ denote the amortized per-datapoint runtime of DROP and the alternative method, respectively. Let $n_i$ and $n_q$ the number of indexed and queried points. DROP is faster when $n_q x_d + n_i r_d < n_q x_a + n_i r_a$.

To verify, we obtained estimates of the above and compared DROP against FFT and PAA in lower-query-volume scenarios when running k-NN using cover trees, and display the results in Figure 6. We first found that in the 1:1 index-query ratio setting, DROP should be slower than PAA and FFT, as observed. However, as we decrease the ratio, DROP

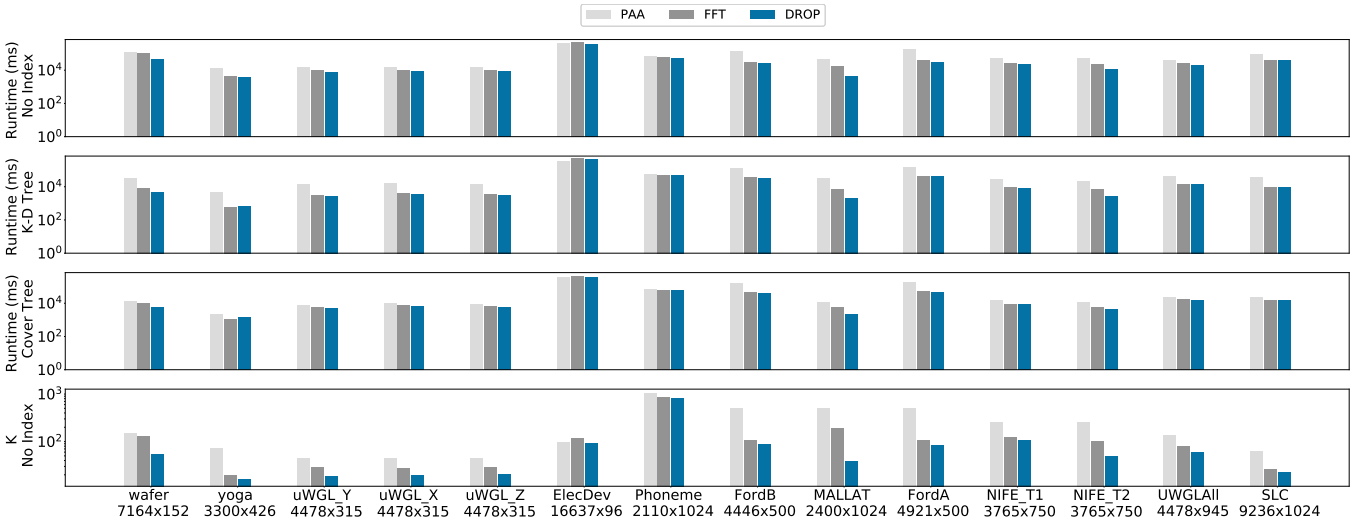


**Figure 5: End-to-End DR and k-NN runtime (top three) and returned lower dimension (bottom) over the largest UCR datasets for three different indexing routines. DROP consistently returns lower dimensional bases than conventional alternatives (FFT, PAA), and is on average faster than PAA and FFT.**

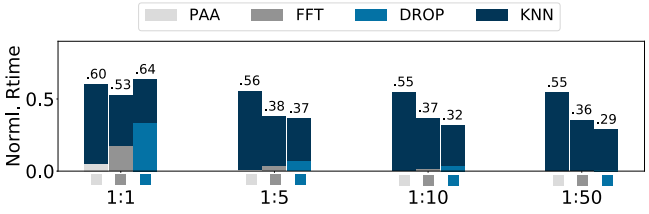


**Figure 6: Effect of decreasing the index-query ratio. As an index is queried more frequently, DROP's relative runtime benefit increases.**

becomes faster, with a break-even point of slightly lower than 1:3. We show that DROP does indeed outperform PAA and FFT in the 1:5 index-query ratio case, where it is is on average 1.51× faster than PAA and 1.03× faster than FFT. As the ratio decreases to 1:50, DROP is 1.24× faster than FFT and 1.9× faster than PAA.

**Time Series Similarity Search Extensions** Given the breadth of research in time series indexing, we evaluate how DROP, a general operator for PCA, compares to time series indexes. As a preliminary evaluation, we consider iSAX2+ [14], a state-of-the-art indexing tool, in a 1:1 index-query ratio setting, using a publicly available Java implementation [3]. While these indexing techniques also optimize for the low index-query ratio setting, we find index construction to be a large bottleneck in these workloads. For iSax2+, index construction is on average 143× (up to 389×) slower than DR via DROP, but is on average only 11.3× faster than k-NN on the reduced space. However, given high enough query workload, these specialized techniques will surpass DROP.

We also verify that DROP is able to perform well when using downstream similarity search tasks relying on alternative distance metrics, namely, Dynamic Time Warping (DTW)—a commonly used distance measure in the literature [58]. As proof-of-concept, we implement a 1-NN task using DTW with a 1:1 index-query ratio, and find that even with this high ratio, DROP provides on average 1.2× and 1.3× runtime improvement over PAA and FFT, respectively.

## 5.3 Factor Analysis

We perform a factor analysis of the incremental runtime contributions of each of DROP's components compared to baseline SVD methods. We only display the results of k-NN with cover trees; the results hold for the other indexes. We use a 1:1 index-query ratio with data inflated by 5× to better highlight the effects of each contribution to the DR routine.

Figure 7 first demonstrates the boost from using SVD-Halko over a naïve implementation of PCA via SVD, which comes from not computing the full transformation a priori, incrementally binary searching as needed. It then shows the runtime boost obtained from running on samples until convergence, where DROP samples and terminates after the returned lower dimension from each iteration plateaus. This represents the naïve sampling-until-convergence approach that DROP defaults to sans user-specified cost model. We finally introduce cost based optimization and work reuse. Each of these optimizations improves runtime, with the exception

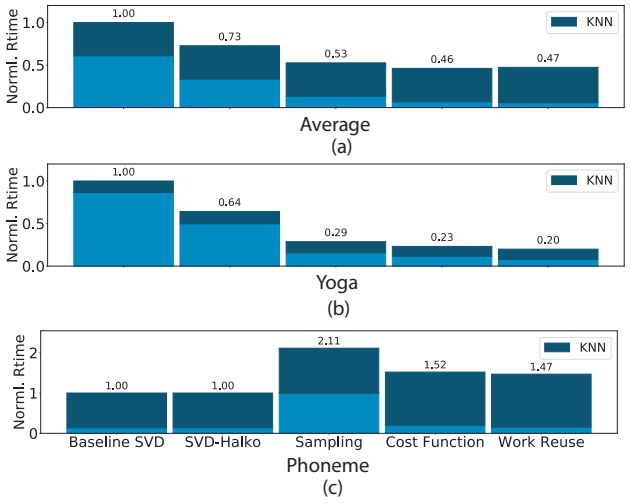


**Figure 7: Lesion studies over the UCR datasets.**

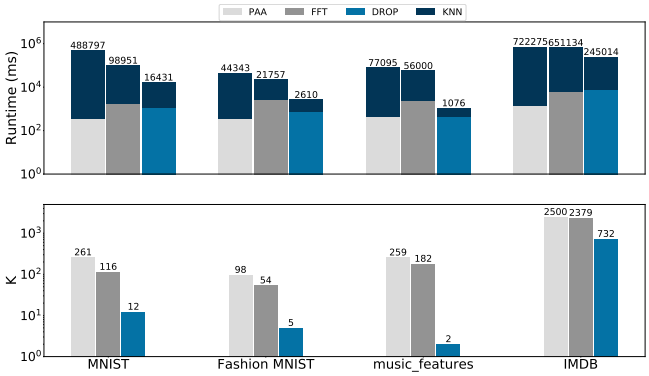


**Figure 8: End-to-End k-NN runtime (top) and returned dimension $k$ (bottom) over the entire MNIST dataset and the FMA featurized music dataset.**

of work reuse, which has a negligible impact on average but disproportionately impacts certain datasets.

Work reuse typically only slightly affects end-to-end runtime as it is useful primarily when a large number of DROP iterations are required. We also observe this behavior on certain small datasets with moderate intrinsic dimensionality, such as the yoga dataset in Figure 7b. Work reuse provides a 15% improvement in addition to cost based optimization.

DROP's sampling operates on the premise that the dataset has data-point-level redundancy. However, datasets without this structure are more difficult to reduce the dimensionality of. Phoneme is an example of one such dataset (Figure 7c). In this setting, DROP incrementally examines a large proportion of data before enabling cost-based optimization, resulting in a performance penalty.

## 5.4 Beyond the Time Series Case Study

We consider generalizability beyond our initial case study along two axes: data domain and downstream workload. These preliminary results show promise in extension to additional domains and target tasks.

*Data Domain.* We examine classification/similarity search workloads across image classification, music analysis, and natural language processing. We repeat the k-NN retrieval experiments with a 1:1 index-query ratio. We use the MNIST hand-written digit image dataset of 70,000 images of dimension 784 (obtained by flattening each $28 \times 28$-dimensional image into a single vector [46], combining both the training and testing datasets); FMA's featurized music dataset, providing 518 features across 106,574 music tracks; a bag-of-words representation of an IMDb sentiment analysis dataset across 25,000 movies with 5000 features [8]; Fashion MNIST's 70,000 images of dimension 784 [64]. We present our results in Figure 8. As these datasets are larger than those in [17], DROP's ability to find a *TLB*-preserving low dimensional basis is more valuable as this more directly translates to significant reduction in end-to-end runtime—up to a 7.6 minute wall-clock improvement in MNIST, 42 second improvement in Fashion MNIST, 1.2 minute improvement in music features, and 8 minute improvement in IMDB compared to PAA. These runtime effects will only be amplified as the index-query ratio decreases, to be more typical of the repeated-query setting. For instance, when we decrease the ratio to 1:5 on the music features dataset, DROP provides a 6.1 and 4.5 minute improvement compared to PAA and FFT, respectively.

*Downstream Workload.* To demonstrate the generalizability of DROP's pipeline as well as black-box runtime cost-model estimation routines, we extend our pipeline to perform a k-means task over the MNIST digits dataset. We fit a downstream workload runtime model as we did with k-NN, and operate under a 1:1 index-query ratio. DROP terminates in 1488ms, which is 16.5× and 6.5× faster than PAA and FFT.

# 6 RELATED WORK

**Dimensionality Reduction** DR is a classic operation [18, 27, 47, 59] that is well studied in the database [7, 15, 41, 54], data mining [40, 48], statistics and machine learning [20, 57], and theoretical CS [29, 35] communities.

Recent breakthroughs in the theoretical statistics community provided new algorithms for PCA that promise substantial scalability improvements without compromising result quality [19, 20, 31, 31]. To the best of our knowledge, advanced methods for PCA have not been empirically compared head-to-head with conventional DR approaches such as Piecewise Approximate Averaging [40].In addition, DROP

*combines* these methods with row-level sampling to provide benefits similar to using stochastic methods for PCA.

**Approximate Query Processing** Inspired by approximate query processing engines [52] as in online aggregation [34], DROP performs progressive sampling. In contrast with more general data dimensionality estimation methods [13], DROP optimizes for *TLB*. As we illustrated in §5, this strategy confers substantial runtime improvements. While DROP performs simple uniform sampling, the literature contains a wealth of techniques for various biased sampling techniques [9, 16]. Finally, DROP performs online progress estimation to minimize the end-to-end analytics cost function. This is analogous to query progress estimation [50] and performance prediction [51] in database and data warehouse settings and has been exploited in approximate query processing engines such as BlinkDB [6].

**Scalable Workload-Aware, Complex Analytics** DROP is an operator for analytics dataflow pipelines. Thus, DROP is as an extension of recent results on integrating complex analytics function including model training [26, 36, 44] and data exploration [61, 63, 65] operators into analytics engines.

## 7 FUTURE WORK AND CONCLUSIONS

DROP provides a first step in bridging the gap between quality and efficiency in DR for downstream analytics. However, there are several avenues to explore for future work, such as sophisticated sampling methods and streaming execution.

### 7.1 Data-Aware Sampling

DROP's efficiency is determined by the dataset's spectrum; MALLAT, with the sharpest drop-off, performs extremely well, and Phoneme, with a near uniform distribution, does not. Datasets such as Phoneme perform poorly under the default configuration as we enable cost-based optimization after reaching a feasible point. Thus, DROP spends a disproportionate time sampling (Fig. 7c). Extending DROP to determine if a dataset is amenable to aggressive sampling is an exciting area of future work. For instance, recent theoretical results that use sampling to estimate spectrum, even when the number of samples is small in comparison to the input dimensionality [43], can be run alongside DROP.

### 7.2 Streaming Execution

Given a stationary input distribution, users can extract fixed-length sliding windows from the source and apply DROP's transformation over these segments as they arrive. Should the data distribution not be stationary over time, DROP can be be periodically retrained in one of two ways. First, DROP can use of the wide body of work in changepoint or feature drift detection [30, 42] to determine when to retrain. Alternatively, DROP can maintain a reservoir sample of incoming data [60], tuned to the specific application, and retrain if the metric of interest no longer satisfies user-specified constraints. Due to DROP's default termination condition, cost-based optimization must be disabled until the metric constraint is achieved to prevent early termination.

## REFERENCES


[1] 2008. SMILE. (2008). http://haifengl.github.io/smile/.
[2] 2015. JTransforms. (2015). https://sites.google.com/site/piotrwendykier/software/jtransforms.
[3] 2017. DPiSAX. (2017). http://djameledine-yagoubi.info/projects/DPiSAX/.
[4] 2017. netlib-java. (2017). https://github.com/fommil/netlib-java.
[5] 2018. Intel MKL. (2018). https://software.intel.com/en-us/mkl.
[6] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 29–42.
[7] Charu C Aggarwal. 2001. On the effects of dimensionality reduction on high dimensional similarity search. In *PODS*.
[8] Peter T. Pham Dan Huang Andrew Y. Ng Andrew L. Maas, Raymond E. Daly and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *ACL 2011*.
[9] Brian Babcock, Surajit Chaudhuri, and Gautam Das. 2003. Dynamic sample selection for approximate query processing. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 539–550.
[10] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. 2017. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*. ACM.
[11] Peter Bailis, Edward Gan, Kexin Rong, and Sahaana Suri. 2017. Prioritizing Attention in Fast Data: Principles and Promise.
[12] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 97–104.
[13] Francesco Camastra. 2003. Data dimensionality estimation methods: a survey. *Pattern recognition* 36, 12 (2003), 2945–2954.
[14] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.
[15] Kaushik Chakrabarti and Sharad Mehrotra. 2000. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*.
[16] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)* 32, 2 (2007), 9.
[17] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. (July 2015). www.cs.ucr.edu/~eamonn/time_series_data/.
[18] John P Cunningham and Zoubin Ghahramani. 2015. Linear dimensionality reduction: survey, insights, and generalizations. *Journal of Machine Learning Research* 16, 1 (2015), 2859–2900.
[19] Christopher De Sa, Bryan He, Ioannis Mitliagkas, Chris Re, and Peng Xu. 2018. Accelerated Stochastic Power Iteration. In *Artificial Intelligence and Statistics*.

[20] Christopher De Sa, Kunle Olukotun, and Christopher Ré. 2015. Global Convergence of Stochastic Gradient Descent for Some Non-convex Matrix Problems. In *ICML*.
[21] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. 2017. FMA: A Dataset For Music Analysis. In *18th International Society for Music Information Retrieval Conference*.
[22] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. In *VLDB*.
[23] Tarek Elgamal, Maysam Yabandeh, Ashraf Aboulnaga, Waleed Mustafa, and Mohamed Hefeeda. 2015. sPCA: Scalable Principal Component Analysis for Big Data on Distributed Platforms. In *SIGMOD*.
[24] Philippe Esling and Carlos Agon. 2012. Time-series data mining. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 12.
[25] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. *Fast subsequence matching in time-series databases*.
[26] Xixuan Feng, Arun Kumar, Benjamin Recht, and Christopher Ré. 2012. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*.
[27] Imola K Fodor. 2002. *A survey of dimension reduction techniques*. Technical Report. Lawrence Livermore National Lab., CA (US).
[28] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *VLDB*, Vol. 99. 518–529.
[29] Navin Goyal, Santosh Vempala, and Ying Xiao. 2014. Fourier PCA and Robust Tensor Decomposition. In *STOC*.
[30] Valery Guralnik and Jaideep Srivastava. 1999. Event detection from time series data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 33–42.
[31] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.
[32] Sam Halliday and BjÃ¿rn-Ove Heimsund. 2008. matrix-toolkits-java. (2008). https://github.com/fommil/matrix-toolkits-java.
[33] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.
[34] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. In *SIGMOD*.
[35] Prateek Jain, Chi Jin, Sham M Kakade, Praneeth Netrapalli, and Aaron Sidford. 2016. Streaming PCA: Matching Matrix Bernstein and Near-Optimal Finite Sample Guarantees for Oja's Algorithm. In *COLT*.
[36] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. 2008. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*.
[37] Ian T Jolliffe. 1986. Principal component analysis and factor analysis. In *Principal component analysis*. Springer, 115–128.
[38] Yannis Katsis, Yoav Freund, and Yannis Papakonstantinou. 2015. Combining Databases and Signal Processing in Plato.. In *CIDR*.
[39] Eamonn Keogh. 2006. A decade of progress in indexing and mining large time series databases. In *VLDB*.
[40] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems* 3, 3 (2001), 263–286.
[41] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Sigmod Record* 30, 2 (2001), 151–162.
[42] Rebecca Killick, Paul Fearnhead, and Idris A Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107, 500 (2012), 1590–1598.
[43] W. Kong and G. Valiant. 2017. Spectrum Estimation from Samples. *Annals of Statistics* 45, 5 (2017), 2218–2247.
[44] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. 2013. MLbase: A Distributed Machine-learning System.. In *CIDR*.
[45] Hans-Peter Kriegel, Peer Kröger, and Matthias Renz. 2010. Techniques for efficiently searching in spatial, temporal, spatio-temporal, and multimedia databases. In *ICDE*. IEEE.
[46] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun.com/exdb/mnist/* (1998).
[47] John A Lee and Michel Verleysen. 2007. *Nonlinear dimensionality reduction*. Springer Science & Business Media.
[48] Jessica Lin, Eamonn Keogh, Wei Li, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery* 15, 2 (2007), 107.
[49] Renqiang Min, David A Stanley, Zineng Yuan, Anthony Bonner, and Zhaolei Zhang. 2009. A deep non-linear feature mapping for large-margin knn classification. In *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*. IEEE, 357–366.
[50] Chaitanya Mishra and Nick Koudas. 2007. A lightweight online framework for query progress indicators. In *ICDE*.
[51] Kristi Morton, Abram Friesen, Magdalena Balazinska, and Dan Grossman. 2010. Estimating the progress of MapReduce pipelines. In *ICDE*.
[52] Barzan Mozafari. 2017. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*.
[53] Milos Nikolic, Badrish Chandramouli, and Jonathan Goldstein. 2017. Enabling Signal Processing over Data Streams. In *SIGMOD*. ACM.
[54] KV Ravi Kanth, Divyakant Agrawal, and Ambuj Singh. 1998. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*.
[55] Ruslan Salakhutdinov and Geoff Hinton. 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*. 412–419.
[56] Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[57] Ohad Shamir. 2015. A stochastic PCA and SVD algorithm with an exponential convergence rate. In *ICML*. 144–152.
[58] Jin Shieh and Eamonn Keogh. 2008. i SAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 623–631.
[59] Lloyd N Trefethen and David Bau III. 1997. *Numerical linear algebra*. Vol. 50. Siam.
[60] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)* 11, 1 (1985), 37–57.
[61] Abdul Wasay, Xinding Wei, Niv Dayan, and Stratos Idreos. 2017. Data Canopy: Accelerating Exploratory Statistical Analysis. In *SIGMOD*.
[62] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*. 1473–1480.
[63] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*.
[64] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).
[65] E. Zgraggen, A. Galakatos, A. Crotty, J. D. Fekete, and T. Kraska. 2016. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Transactions on Visualization and Computer Graphics* (2016).
[66] Yunyue Zhu and Dennis Shasha. 2003. Warping indexes with envelope transforms for query by humming. In *SIGMOD*.