Random Projection Clustering on Streaming Data

Lee A. Carraher, Philip A. Wilsey, Anindya Moitra, and Sayantan Dey
Dept of EECS, University of Cincinnati Cincinnati, USA 45221-0030
Email: carrahle@email.uc.edu, wilseypa@gmail.com, moitraaa@mail.uc.edu, and deysn@mail.uc.edu

***Abstract*—Clustering streaming data has gained importance in recent years due to an expanding opportunity to discover knowledge in widely available data streams. As streams are potentially evolving and unbounded sequence of data objects, clustering algorithms capable of performing fast and incremental processing of data points are necessary. This paper presents a method of clustering high-dimensional data streams using approximate methods called streamingRPHash. streamingRPHash combines random projections with locality-sensitivity hashing to construct a high-performance clustering method. streamingRPHash is amenable to distributed processing frameworks such as Map-Reduce, and also has the benefits of constrained overall complexity growth. This paper describes streamingRPHash algorithm and its various configurations. The clustering performance of streamingRPHash is compared to several alternatives. Experimental results show that streamingRPHash has comparable clustering accuracy and substantially lower runtime and memory usage.**

***Keywords*-streaming algorithms; clustering; LSH; multi-probe LSH; random projection; min-count sketch**

I. Introduction

The recent expansion in data acquisition has provided nearly endless streams of high-dimensional data. Data streams often undergo rapid changes throughout their lifetime, requiring fast temporally aware tools for data analysis. Data clustering is the principal workhorse of many forms of pilot data analysis, data retrieval and machine learning techniques. In this paper we present an algorithm, named *Streaming Random Projection Hash* (streamingRPHash), to solve the k-means clustering problem on streaming data. Our solution offers comparable clustering accuracy to other stream clustering algorithms and has a better overall complexity bound which makes it considerably faster on high-dimensional data streams than conventional methods. Further, our algorithm provides a linear bounded memory solution to the streaming k-means problem of [1].

Many data streams are inherently high-dimensional [2]. streamingRPHash is created to provide algorithmic scalability while operating on large, high-dimensional data streams. Existing stream clustering algorithms often have issues regarding asymptotic scalability [3], dimensionality limits [4] and robustness to noise. A recent set of algorithms have been proposed to address (with varying degrees of success) many of these problems [1], [5]–[7]. streamingRPHash combines random projection and locality-sensitive hashing in a new way to solve issues of scalability and data security for real-world high-dimensional streaming data.

The remainder of this paper is organized as follows. Section II provides some background information. Section III reviews the recent work with methods for data stream clustering. Section IV describes streamingRPHash algorithm. Section V contains experimental results. Finally, Section VI concludes the paper.

II. Background

The k-means clustering problem can be defined as a partitioning of vectors $x \in X$ among k clusters $\{C_1, C_2, ..., C_k\}$ such that the within-cluster sum of squared error (WCSSE) is minimized over all possible partitions. More formally, this can be stated as $\underset{C}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{x \in C} ||x - \mu_i||^2$, where μ_i is the centroid of cluster C_i. For points in 1-dimension, k-means can be solved in polynomial time with dynamic programming methods, unfortunately this is a special case that does not apply to a general $\mathbb{R}^d$ space.

Random projection is a method for dimensionality reduction, in which d-dimensional data vectors are projected down to an m-dimensional subspace ($m \ll d$). This is done by multiplying the original d-dimensional data by a random $m \times d$ matrix R which is composed of orthogonal vectors sampled from a random or quasi-random distribution. Random projection can achieve a bounded error distortion factor very close to the optimal L_2 norm subspace embedding that usually results from the principal component decomposition [8]. Computationally, random projection is very simple. Forming the random matrix R and projecting the dN data matrix D into m dimensions have the complexity $O(dmN)$, where the data set D has N records in d dimensions [9].

Switching from the continuous spaces of random projections, we now consider the discrete space partitions induced by LSH functions. Optimal implementations of grid based clustering algorithms such as DBSCAN [10] and DStream, require that the data space be partitioned as evenly as possible. For known datasets, a perfect partition of the data space can be produced by the Voronoi diagram [11]. In 2-dimensional space, Voronoi Diagrams can be generated in $\Theta(nlog(n))$-time. However, higher dimensional algorithms for Voronoi partitioning have far less favorable runtime complexities making them inefficient for partitioning arbitrarily high dimensions. To overcome this issue, streamingRPHash borrows heavily from the LSH community which

has derived a variety of partitioning methods. We discuss one such method below, chosen from a set of candidate LSH methods, that proved to be the best among a large scale analysis for a variety of data sets.

The *Spherical LSH* technique of Terasawa [12] is a space partitioning method applicable to data where the vector lies on or near the surface of a hypersphere. Spherical LSH uses an inscribed regular polytope to partition the surface of the sphere where vertexes of the polytope correspond to partition regions. Although other regular d-polytopes such as the d-simplex and d-hypercube exist, we focus on the d-orthoplex, following the favorable collision probability per distance, which results in Table 1 of [12]. The d-orthoplex is a regular d-polytope with $2d$ vertexes corresponding to the positive and negative axis per dimension of a 0 centered unit hypersphere embedding. The nearest vertexes can be searched among the $2d$ vertexes following the max dot product method given in [12], resulting in a search time complexity of $\theta(2*d^2)$. The d-orthoplex has the benefit over Lattice based decoders that allows for an arbitrary projection dimension, while having the disadvantage that it is only applicable to vectors lying on the surface of a hypersphere. To overcome this, we use a variant of SLSH similar to [13] to translate the vector such that it lies very near the surface of a hypersphere.

III. Related Work

Despite an exponential worst-case complexity for k-means, many real-world problems tend to fair much better under Lloyd's type of solutions to k-means optimization than theoretically optimal solutions. For this reason, clustering massive datasets with k-means, although suffering from unbounded complexity guarantees, often yields qualitatively good results close to the optimal k-means solution. Due to the approximate solution's real-world proclivity towards revealing useful results, randomized methods such as sampling and random dimensional reduction are often utilized in overcoming complexity growth. The use of these methods in clustering began with density and grid based scanning algorithms. DBSCAN proceeds in a conceptually similar manner to streamingRPHash in regard to partitioning the data space and counting the number of data vectors within a partitioned region. These methods tend to work well on spatially separated datasets with relatively low dimensions. A common clustering problem for these types of algorithms would be on geo-spatial data, in geographic data systems (GIS), and image segmentation. The algorithms DBSCAN [10], Clique [14], and CLARANS [4], respectively, represent a successful progression of density scanning techniques. Although density scan algorithms are an example of stream clustering algorithms such as streamingRPHash, they often show weakness in accuracy while scaling the number of dimensions.

More recently streaming algorithms for data clustering are also considered [1], [6]. The general setup of these algorithms consists of a diminishing objective bound that tightens as the stream is processed. The processor updates the core-sets of pseudo-centroids when data falls within the objective bound and disregards data outside that bound. Streaming k-means algorithms perform well on data streams but has the drawback of requiring that data be 'well-clusterable'. streamingRPHash uses a similar core-set approach but, instead of choosing core-sets based on computed distances from current centroids, it maintains a count structure similar to CSketch [6] of dense regions. This maintaining of all data replaces the hard margin of windowed data with a slight increase in error due to incorrect hash collisions.

CSketch is a streaming algorithm for generating clusters over massive-domain datasets [6]. It shares much in common with streamingRPHash. In particular, CSketch applies the count-min sketch data structure to update candidate centroids by updating the centroid location and not just a count. Incoming centroids choose the centroid which maximizes the dot-product between the vectors. streamingRPHash uses the LSH decoding of projected points to immediately find the nearest candidate cluster in the count-min sketch data structure. This hashing is intrinsic to the decoding step and does not require a supplementary hashing scheme. In addition, a k-heavy-hitters priority queue is employed to further reduce the memory overhead by storing only likely centroids and removing unlikely ones with error $\frac{\epsilon}{k} \cdot F^{res(k)}$, where $F^{res(k)}$ is the sum of squares of frequencies not in the top k [15].

IV. streamingRPHash Algorithm

streamingRPHash is a stream clustering variant of the original 2-Phase RPHash algorithm of [16] that employs both approximate and randomized techniques to solve the approximate k-means clustering problem. Because streamingRPHash is intended to accept unbounded data streams, it must have quasi-linear complexity growth and a bounded memory requirement. Like the 2-Phase algorithm, streamingRPHash is meant to be an naively parallelizable algorithm requiring minimal communication overhead. In this paper, we will focus on the sequential version of the streaming algorithm for clarity. A parallel version can be achieved using a worker model with a low contention shared resource.

The general idea of streamingRPHash is motivated by a particularly difficult case in LSH based Nearest Neighbor (LSH-NN) search where a single bucket contains an unprecedented number of candidate nearest neighbors. This is a problem in LSH-NN as the bucket then has to be linearly scanned for the optimal solution. However, these degenerate buckets can also be viewed as locally dense regions or density modes in the data. In other words, partitions with high collision rates are good candidates for cluster centroids.

To follow the conventional k-means model, only the top k densest regions are tracked while less dense regions are omitted as noise. streamingRPHash LSH-NN combines LSH-NN methods: multi-probe random projection and discrete space quantization, with a k-Heavy Hitter Centroid tracker.

The streamingRPHash algorithm is discussed below and a pseudo-code outline is presented in Algorithm 1. First, we discuss the generative nature of its region assignment. Clustering region assignments are performed by decoding vectors using our set of LSH functions. In most cases the problem space will not match our locality sensitive hashing subspace. The previously described Johnson-Lindenstrauss (*JL*) lemma offers a solution to this problem at the cost of a matrix vector product. Although the *JL*-lemma requires an $m \times d$ matrix of Gaussian random variables be formed, [17] suggests an approximate method to form the random projection matrix whose elements $r_{ij} \in \mathbf{R}$ have values from $\{1, 0, -1\}$ with probabilities $\{\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\}$ respectively. A further computational reduction is then achieved by treating the matrix-vector product as a linear scan over the non-zero elements of the projection matrix. This allows us to effectively skip $\frac{2}{3}$ of the projection computation with the remaining $\frac{1}{3}$ operations consisting only of scalar additions. Following this computation, a scaling factor of $\frac{1}{\sqrt{n}}$ is also needed to preserve approximate distances between projected vectors. From Panigrahy [18], a requirement of $\Theta(log(n))$ random projections is sufficient for c-approximate hash collisions with its corresponding r-near vectors.

Discrete space quantizers play a central role in the streamingRPHash algorithm. While the code base for streamingRPHash contains implementations for several LSH methods (including E_8, Leech, Pstable Distributions, and Spherical), our experimental studies show that the best overall performance is achieved with a Spherical LSH. Experimentally, the spherical 32-orthoplex decoder of [12] was found to be optimal on a variety of datasets. The spherical orthoplex decoding consists of a projection to $32d$ space followed by a vector normalization step in order to have the vector lie on the surface of a hypersphere. Next, a random rotation is applied to the 64 basis vectors of the 32-orthoplex. The basis vector nearest to the projected vector is then used as the vector's representative and the corresponding bits resulting from the natural ordering of basis vectors are used to define a $log_2(64)$ partial decoding.

At this point streamingRPHash has identified a set of possible partitions in which a given vector might reside. However our goal is to only track the densest candidate regions, sometimes referred to as the k-Heavy Hitters problem. Formally, the k-Heavy Hitters (k-*HH*) problem is the problem of identifying the k most frequent items in a dataset. k-HH can trivially be solved for finite datasets using a hash table and $\theta(n)$ space. However data streams are potentially unbounded, and require a slightly modified approach using approximate methods. One of the earliest streaming solutions for the approximate k-*HH* algorithms is the count-min sketch data structure [19]. The count-min sketch can be used to solve the approximate k-*HH* problem with only the addition of a priority queue using $\theta(\frac{1}{\epsilon} log \frac{1}{\delta})$ space with error $(1-\epsilon)f$, where f is the minimum frequency $\frac{m}{k}$ to be considered frequent.

streamingRPHash uses a k-*HH* tracker to track both the sizes of dense regions as well and accumulates the running centroid for the vectors added to the region. This completes the description of the streaming algorithm for RPHash.

Algorithm 1: Streaming Algorithm

Data
k - number of clusters
$x \in \mathbb{R}^m$ - data vector from stream
$\mathbb{H}(\cdot)$ - LSH Function radius=r, dim=d
$\mathbb{P} = \{p_1, ...p_n\}$ - set of $n, m \times d$ matrices w/ *JL* property
$B = \{b_1, ..., b_{\log_2 d}\}$, $b_i \in \mathbb{R}^d$ - $N(0, r)$ blurring vectors
M- lsh_key $\rightarrow$ centroid map
C- cm-sketch data structure
T- CM-Sketch based ϵk bounded priority queue

forall the $p \in \mathbb{P}$ **do**
  $\tilde{x} := \sqrt{\frac{m}{d}} p^\intercal x$
  forall the $b \in B$ **do**
    $t := \mathbb{H}(\tilde{x} + b)$ C.add(t)
    if $t \in M.keys$ **then**
      $M[t].wadd(x)$
    else
      $M[t]$ =new centroid(x,C.count(t))
      T.insert($M[t]$)
    M.remove(T.pop())

Although a set of dense regions and the corresponding centroids has been identified, there is no direct way of resolving vectors from different projection transformations with one another. A simple solution found in [1] and in other streaming algorithms, often referred to as an offline step, is to merge the so-called micro-clusters using a standard clustering algorithm in the high dimensional space.

From the above steps we get the following complexity for streamingRPHash $\Theta(NP(\frac{Md}{3} + log(d)(log(k) + d + 2d^2)))$, using P database friendly projections from M to d, SLSH decoding complexity ($2d^2$), a balanced priority tree (with $\theta(log(k))$ insertion and constant time removal), and blurring factor $log(d)$. The projection step $\frac{Md}{3}$ tends to dominate when $M >> d$. If we remove the constant factor, and have a small constant number of projection P (< 3) and subspace d (< 24), we get a complexity dependent only on the input $\Theta(NM)$.

V. Experimental Studies

In this section, we evaluate the performance of streamingRPHash and compare it against several other stream clustering algorithms, namely: Streaming k-means [1] implementation from S-Space [20], Damped Sliding Window [21], DStream [22] and Biased Reservoir Sampling [23]. These algorithms are chosen because of their importance and availability (except for streaming k-means) in the R statistical computing framework. The implementation of streamingRPHash and streaming k-means are done in Java. All of these algorithms follow the conventional two-stage (online-offline) approach for data stream clustering. The weighted k-means implementation of Hartigan and Wong (with $nstart = 25$ for algorithms implemented in R) is used in the offline stage. The size of grid cells is set at 0.8 for DStream. Values of the window length in Sliding Window and the number of points to be sampled from the stream in Reservoir Sampling are both set to 100. In streamingRPHash, two random projections are applied to data points with four Gaussian blurring shifts.

We apply the stream clustering algorithms on both real-world and synthetic data streams and evaluate their clustering results based on various external, internal and entropy based clustering validation measures. Runtime and memory usage of all algorithms are also captured on a computer having Intel Xeon X5675 CPU with 6 cores @ 3.07GHz supporting 12 hardware threads. The operating system is Debian 'stretch' (x86_64), running on 28 GB memory.

A. Real-World Datasets

Two real-world labeled datasets are chosen from UCI Machine Learning Repository. We report the values of two external measures: Adjusted Rand Index (ARI) and Cluster Purity, three internal measures: Dunn Index, Silhouette Coefficient and Within-Cluster Sum of Squared Errors (WCSSE), and one entropy based measure: Variation of Information (VI). The algorithms are set up to report clustering results in batches of 500 points.

The first dataset is on 'Human Activity Recognition Using Smartphones' (https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones). It has 10,299 records (the first 10,000 are used in this study), and 561 attributes representing time and frequency domain variables. The dataset contains six clusters which denote the six movement activities performed by each person.

Performance results with this dataset are shown in Figure 1. The plots are organized into two side-by-side graphs. The left graph shows the computed values at each 500 points interval and the right graph is a box and whisker plot that shows the median and quartiles of all of the sample points for each algorithm. The results show that streamingRPHash performs on par with other algorithms. The runtime performance is better than most and on par with streaming k-means. The memory usage is slightly worse than streaming k-means, but both are significantly better (less than half) than the other algorithms studied.

The second dataset is 'UJIIndoorLoc Data Set' (https://archive.ics.uci.edu/ml/datasets/UJIIndoorLoc). It is a multi-building multi-floor indoor localization database to test Indoor Positioning System that rely on WLAN/WiFi fingerprint. The dataset has a total of 21,048 vectors. This study uses first 21,000 vectors each having 520 attributes which represent WiFi fingerprints composed of 520 intensity values detected by Wireless Access Points. Building ID is used as the target variable which has three possible values.

Results from this dataset are shown in Figure 2. The plots have the same format as with the previous study. Again, the clustering accuracy of streamingRPHash is on par with the other algorithms while streaming k-means is the fastest and has a considerably smaller memory footprint.

B. Scalability Study

We study the dimensional scalability of stream clustering algorithms by evaluating their clustering results as the dimensionality of data streams increases. The purpose of this study is to assess and compare the accuracy, runtime, and memory usage of streamingRPHash to those of other stream clustering algorithms with increasing dimensionality. Labeled data streams of 20,000 points and 10 randomly placed clusters with random multivariate Gaussian distributions are generated using a program written in R, which is similar to DSD_Gaussians function from the 'stream' package. The dimensionality of data streams, which increases in a sequence from 100 to 5767, follows two polynomial sequences of the form $d_n = 1.5d_{n-1}$. The first sequence starts from $d_0 = 100$ and the second one from $d_0 = 125$. We do not inject noise points or outliers in these data streams.

The values of two external (ARI and Cluster Purity) and two internal (Silhouette Coefficient and WCSSE) measures are reported for all algorithms along with the average runtime and memory usage per batch of points. We also compute baseline values of internal measures using the ground-truth partitions of input data streams to provide a reference or best possible values for internal measures. The clustering validation measures are computed in batches of 1000 points and are plotted against dimensions of data streams. For ease and clarity of viewing the WCSSE plot, its values are scaled by diving by the number of dimensions.

Figures 3 and 4 show the performance results from this scalability study. The colored lines in each of these plots represent the mean value and shaded regions around the lines represent the variance of a measure over all the batches of data points. The shaded regions on the plots for DStream are wide and clearly visible as its measures have significantly higher variances than those of other algorithms.

From these results we observe that streamingRPHash produces perfect or near-perfect clustering results for most of

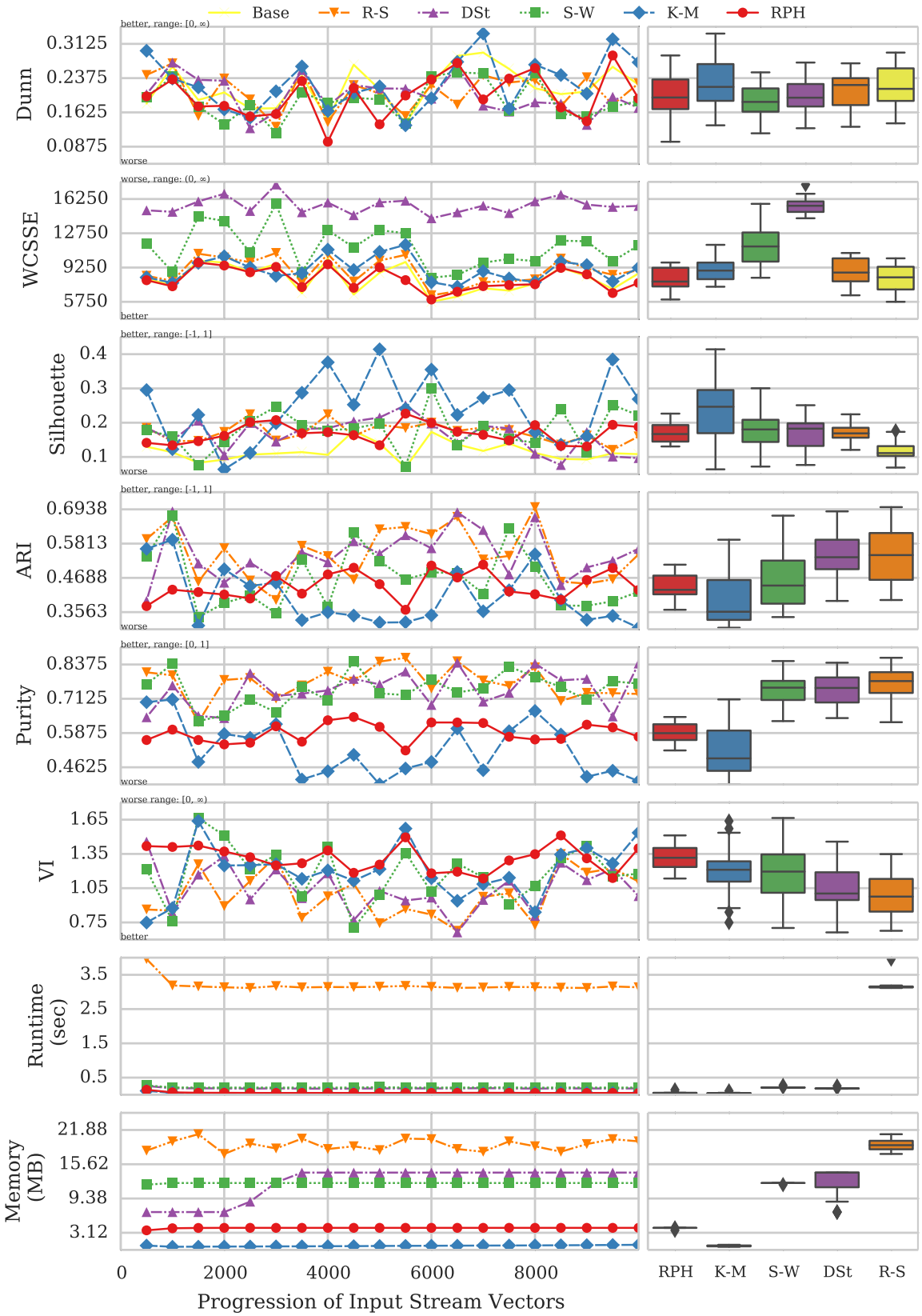

Figure 1. Clustering Results from Smartphone Sensor Data (abbreviations: RPH → streamingRPHash, K-M → Streaming k-means, S-W → Damped Sliding Window, DSt → DStream, R-S → Biased Reservoir Sampling, and Base → Base Value

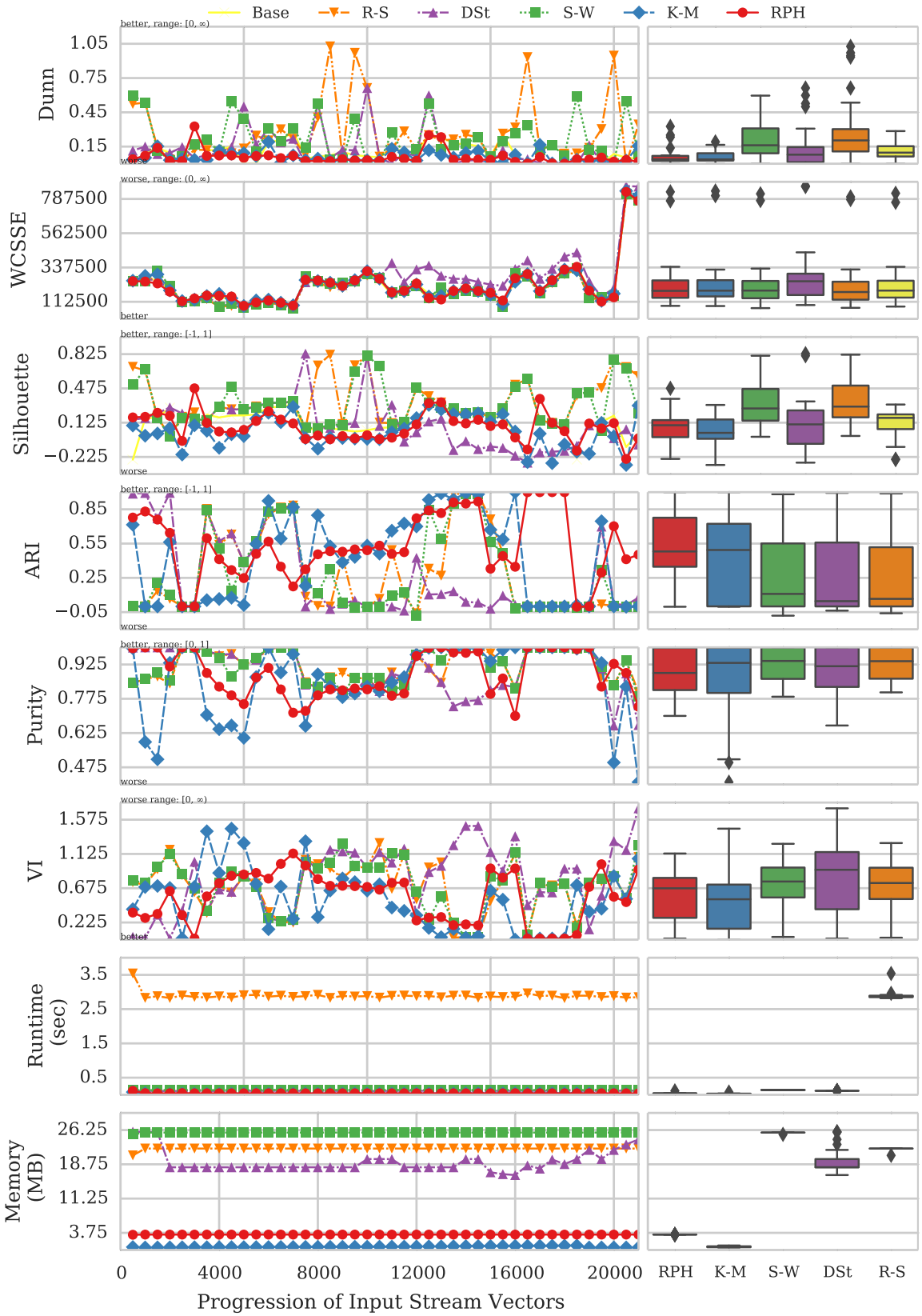

Figure 2. Clustering Results from WiFi Location Data (abbreviations: RPH → streamingRPHash, K-M → Streaming k-means, S-W → Damped Sliding Window, DSt → DStream, R-S → Biased Reservoir Sampling, and Base → Base Value

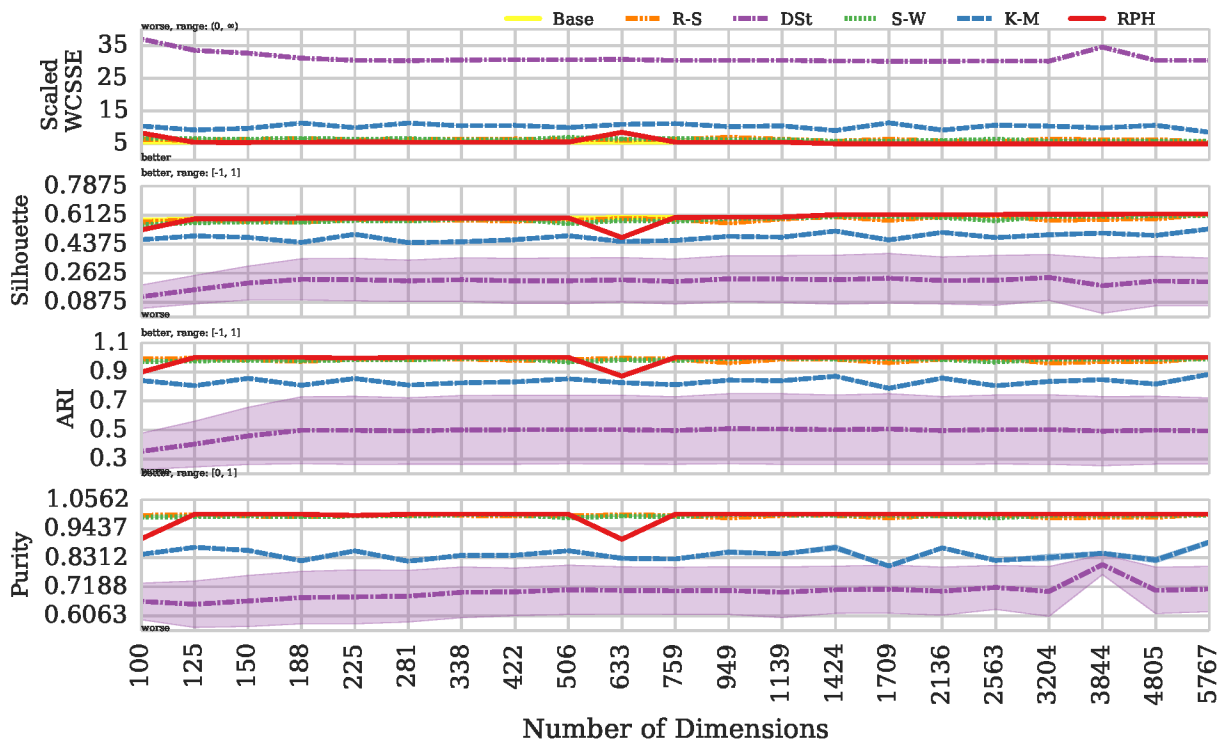

Figure 3. External and Internal Measures from Scaling the Number of Attributes (abbreviations: RPH → streamingRPHash, K-M → Streaming k-means, S-W → Damped Sliding Window, DSt → DStream, R-S → Biased Reservoir Sampling, and Base → Base Value

the data streams and thus exceeds the clustering accuracy of other algorithms. Its ARI and Cluster Purity values steadily remain at 1 and Silhouette Coefficient and WCSSE exactly match the baseline values over all the batches of data points for all data streams. The only exceptions are the slight deviations from perfect clustering at dimensions 100 and 633, which can be attributed to the stochastic nature of our algorithm. One of the biggest advantages of streamingRPHash is that it achieves such clustering accuracy with minimal costs of runtime and memory usage. As with real-world data streams, runtime and memory requirement of streamingRPHash and streaming k-means remain substantially lower than those of other algorithms. This advantage becomes increasingly significant as the dimensionality of the input vectors grows. At dimensions 759 and beyond, the runtime of streamingRPHash becomes even less than that of streaming k-means.

As the runtime and memory usage of streamingRPHash at a certain dimension depend only on the number of points in a batch and remain static for a fixed batch size irrespective of the total number of points in the stream, testing the scalability of streamingRPHash with increasing stream size is not needed. To summarize our findings from the scalability study: we discover a major strength of streamingRPHash and conclude that the algorithm is capable of producing perfect clustering when the data stream consists of a mixture of Gaussians without any noise.

VI. Conclusion

Our experiments show that streamingRPHash leads to fast, accurate and memory efficient clustering of high dimensional streaming data. The initial assumption in this work is that approximate and exact clustering are qualitatively similar due to the effects of noise, redundancy, and the curse of dimensionality. streamingRPHash uses these assumptions to reduce the computational complexity of clustering while still being competitive to many other clustering algorithms.

streamingRPHash combines approximate and randomized methods in a new way to achieve fast, single pass clustering on high dimensional data streams. The growing number of massive, high dimensional streaming data sources would reap the benefits of quick, accurate, scalable and distributed processing of streamingRPHash.

The streamingRPHash Java source code is released under an open source license and is freely available at github.com/wilseypa/rphash-java.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grant ACI–1440420.

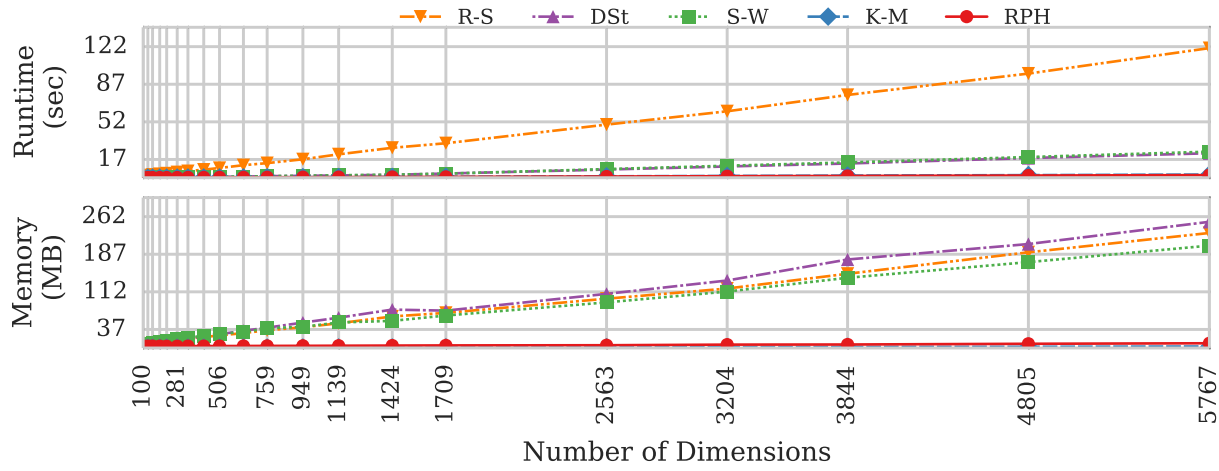

Figure 4. Runtime and Memory usage from Scaling the Number of Attributes (abbreviations: RPH → streamingRPHash, K-M → Streaming k-means, S-W → Damped Sliding Window, DSt → DStream, R-S → Biased Reservoir Sampling, and Base → Base Value

REFERENCES

[1] V. Braverman, A. Meyerson, R. Ostrovsky, A. Roytman, M. Shindler, and B. Tagiku, "Streaming k-means on well-clusterable data," in *Proc of the 22 Annual ACM-SIAM Symp on Discrete Algorithms (SODA '11*, 2011, pp. 26–40.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *Proc of the Thirtieth International Conf on Very Large Data Bases (VLDB '04)*, 2004, pp. 852–863.

[3] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, "Fast algorithms for projected clustering," in *Proc of the 1999 ACM SIGMOD Int Conf on Management of Data (SIGMOD '99)*, 1999, pp. 61–72.

[4] R. T. Ng and J. Han, "Clarans: A method for clustering objects for spatial data mining," *IEEE Trans on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1003–1016, 2002.

[5] S. Har-Peled, "A replacement for voronoi diagrams of near linear size," in *Proc of the 42nd IEEE Symposium on Foundations of Computer Science*, Oct. 2001, pp. 94–103.

[6] C. C. Aggarwal, "A framework for clustering massive-domain data streams," in *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, 2009, pp. 102–113.

[7] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 1, p. 13, 2013.

[8] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, vol. 52, no. 1–2, pp. 46–52, 1985.

[9] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: Applications to image and text data," in *Knowledge Discovery and Data Mining*. ACM Press, 2001, pp. 245–250.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, 1996, pp. 226–231.

[11] R. Klein, "Abstract voronoi diagrams and their applications," in *Computational Geometry and its Applications*. Springer Berlin Heidelberg, 1988, vol. 333, pp. 148–157.

[12] K. Terasawa and Y. Tanaka, "Spherical lsh for approximate nearest neighbor search on unit hypersphere," in *WADS*, 2007, pp. 27–38.

[13] A. Andoni, P. Indyk, H. L. Nguyen, and I. Razenshteyn, "Beyond locality-sensitive hashing," in *Proc of the Twenty-Fifth Annual ACM-SIAM Symp on Discrete Algorithms (SODA '14)*, 2014, pp. 1018–1028.

[14] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," in *Proc of the 1998 ACM SIGMOD Int Conf on Management of Data*, 1998, pp. 94–105.

[15] R. Berinde, P. Indyk, G. Cormode, and M. J. Strauss, "Space-optimal heavy hitters with strong error bounds," *ACM Transactions on Database Systems*, vol. 35, no. 4, p. 26, 2010.

[16] L. A. Carraher, P. A. Wilsey, A. Moitra, , and S. Dey, "Multi-probe random projection clustering to secure very large distributed datasets," in *2nd International Workshop on Privacy and Security of Big Data*, Oct. 2015.

[17] D. Achlioptas, "Database-friendly random projections," in *Proc of the twentieth ACM SIGMOD-SIGACT-SIGART symp on principles of database systems*, 2001, pp. 274–281.

[18] R. Panigrahy, "Entropy based nearest neighbor search in high dimensions," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, ser. SODA '06. New York, NY, USA: ACM, 2006, pp. 1186–1195.

[19] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[20] D. Jurgens, K. Stevens, and M. Dyer, "S-space package," https://github.com/fozziethebeat/S-Space, 2011–2015.

[21] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *Proc of the 28th Int Conf on Very Large Data Bases*, 2002, pp. 358–369.

[22] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *Proc of the 13th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining (KDD '07)*, 2007, pp. 133–142.

[23] C. C. Aggarwal, "On biased reservoir sampling in the presence of stream evolution," in *Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB '06*, Seoul, Korea, 2006, pp. 607–618.