Visualization-Aware Sampling in Streaming Contexts

William Schmitt,
University of Michigan,
schmittw@umich.edu

Abstract
A common way for users consume a high volume of real-time information (eg. sensor data, geographical usage data, etc.) is in scatter plots, maps, and graph in dashboards. This information is queried, processed and prepared across many, highly distributed input streams. Stream processing faces two design goals: it must be (i) low latency and (ii) resource efficient [1]. These goals become increasingly more difficult to satisfy as the throughput of the system grows.

The proposed system will seek to produce low latency, useful visualizations for users as a Kafka and Spark-Streaming application leveraging key contributions from previous work such as Quickr [6], IncApprox [1], VAS [5], and others. Key applications for such a system would real-time monitoring and interactive analysis in high-throughput contexts such as interpreting sensor data, network performance, or IoT analytics [1,5,10].

This project proposes the novel idea of using a small amount of offline learning, in the form of a k-means estimator, in order to provide a low performance overhead. Subsampling in the context of this problem is also explored; however, to much less success.

1. Introduction

In this section in order to define the problem and challenges, I will detail a motivating problem, the core problem, scope, and some relevant background to contextualize this paper.

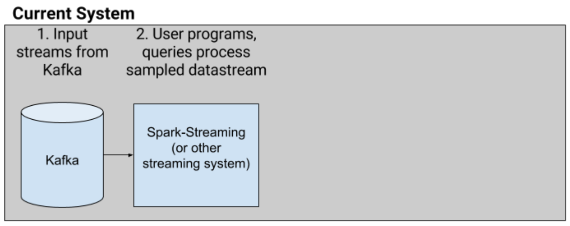
Current System
1. Input streams from Kafka
2. User programs, queries process sampled datastream
Kafka
Spark-Streaming (or other streaming system)

Figure 1: Example Streaming Architecture

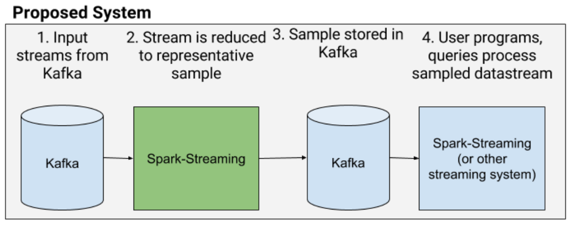
Proposed System
1. Input streams from Kafka
2. Stream is reduced to representative sample
3. Sample stored in Kafka
4. User programs, queries process sampled datastream
Kafka
Spark-Streaming
Kafka
Spark-Streaming (or other streaming system)

Figure 2: Proposed System Architecture

1.1 Motivating Problem

In order to provide a resilient and reliable services, it is common business practice to monitor a distributed application's health by querying application and server logs in order to provide real-time, graphical interfaces to understand system performance and usage. To build a real-time, monitoring application, an engineer will typically will submit an Apache Spark Streaming job that consumes data from a pipeline end-point (eg. Kafka cluster) that performs complex transformations and filters over the data then writes the data to an output endpoint, where the output is consumed graphically.

For a small set of streaming jobs, this presents no challenge or overwhelming cost, however, over time additional jobs will be submitted to the Spark Cluster in order to monitor different parts of the original application. The resources provisioned for this system will become quickly stressed as the number and complexity of programs grows. In order to keep pace with the growing needs, there are two solutions: (1) provision more, costly resources to meet the growing need or (2) limit the number of streaming jobs submitted to the system. The aim of this project is to reduce the resource needs for these jobs so that given the same resource more jobs can be run.

1.2 Problem

Problem Setup

This project seeks to reduce the strain on the system by reducing the amount of work done by each streaming job. For simplicity, this project seeks to reduce work in scenarios that meet these constraints:
- Data stream is high-throughput. The growth of internet scale services has lead to increased stress on backend services such as the one provided in Section 1 and the current trend is that these needs will only continue to grow. Furthermore, approximate computing is most effective in high-volume scenarios.
- Data will be graphed. In order to simplify and the scope of the task, this project will exclusively focus on data streams that will be consumed visually.
- Streaming jobs are resource intensive. This project will assume that most jobs submitted to the system are performing computationally intensive operations and queries on the data stream. While jobs of all types will be included in the evaluation of the system, resource intensive jobs will provide the most resource intensive

Problem Importance
This is an important problem to solve for multiple reasons. First, even if the system is able to achieve marginal performance improvements, then this will greatly reduce the cost running the system. For instance, if we are able to reduce the resource needs of each job by a factor of two, this would lead to half the AWS fees. Second, there is in general a lack of research at the intersection of approximate computing and streaming systems. Some papers have made progress in this field, such as IncApprox; however, overall it remains relatively unexplored. Additionally, current systems do not take advantage of the visual nature of the output.

1.2.1 Hypothesis
The central hypothesis of this project is the question of whether or not visualization aware sampling can introduce a large enough performance benefit during the querying stage for the system to justify the processing and inaccuracy costs. In order to prove this hypothesis true, the proposed system needs to meet the following constraints for the model:
1. **Low cost of sampling.** The cost of running the sampling algorithm should be far less than the value of work saved during querying.
2. **Low cost of inaccuracy**. If the system introduces a large degree of inaccuracy while providing only a small boost of performance, then the proposed system will have very little value.
3. **Non-trivial heuristics exist that do not introduce too much inaccuracy.** The proposed algorithm in [?], runs in O(n^3), which when computed over the entire sample is too expensive for practical use. Current, (simple) techniques such as partitioning the stream sample across multiple nodes have proven somewhat effective in significantly reducing the cost of compute; however, more general approximations will likely have to be made in order to meet desired performance gains. This sub-hypothesis differs from #1 and #2 as overly-practical considerations, such as reducing sample size, increasing scale, etc. are not accounted for here.

1.3 Relevant Background
1.3.1 Visualization Aware Sampling
In [5], the authors note that large databases and current approximate query systems that are used to visualize data do not take advantage of the characteristics of visual output. In order to select a sample of points that will provide high visual quality, the authors propose the following method:

$$\tilde{\varkappa}(x, y) = exp(-\frac{\|x-y\|^2}{\varepsilon}) \text{ [5]}$$

The visual quality of sample can be obtained by summing the above methods across all items in the sample. Higher aggregate values imply a lower quality sample.

This problem deviates slightly from the task of training a kernel machine. The goal of a kernel machine is to learn a set of training data in order to more accurately compute a given task (eg. Binary Classification, Speech Recognition, etc.) - ie the kernel method consumes data to output some optimal result given new input [17]. In [5] the method is used to output a set of representative state vectors given a large input stream. This is a subtle but important difference as many of the techniques that enable large kernel machines to scale, cannot be applied here. Such examples include Multiple Kernel Learning and Kernel Budgeting.

This method works well for batch or offline contexts, however, it is victim to problems similar to encountered to kernel methods in online scenarios and is difficult to scale efficiently. In order to compute a visually representative sample of fixed size for a given input, the computation cost is O(n^3) and O(k) in memory, where k is the desired size of the sample. These problems and proposed solutions around them will be explored in the following section.

1.3.2 Challenges of Scaling Kernel Methods

Due to the rate at which data is scaling, machine learning methods have been forced to adapt in order to solve complex problems over large data sets in over a distributed cluster. While many machine learning techniques, such as Neural Networks, have been able to reasonably scale, kernel based methods have faced many challenges in scaling. Kernel methods are faced with two key problems: high computation cost (O(n^3)) and high memory cost (O(n^2))[1]. These problems are exacerbated in online contexts, where there is little tolerance for high computing costs, high coordination costs, and poor memory efficiency overall. To provide context for these problems: in practice a recurrent neural network (RNN) is able to achieve at (least similar) performance with linear training cost and *sub*-polynomial storage complexity [17, 19].

There has been a great deal of work in scaling up kernel methods to be comparable with Neural Networks. Seminal solutions in this field fall in a variety of categories, ranging from systems engineering in taking advantage of analytics properties. A few relevant topics will be briefly detailed below:
1. The Nystrom Method. In [16] Rahimi and Recht propose that kernel machines can be projected onto a randomized, low-dimensional feature space and fast linear transformations applied in order to improve large-scale kernel machines.
2. Computation Caching. Computing kernel functions are expensive at scale and caching operations reduces the overall computation time [17].
3. Multiple Kernel Learning (MKL). MKL is a set of techniques that learn a set of optimal parameters in order to combine a set of predefined kernels. [17]
4. Dimensionality Reduction. Kernel Methods scale polynomially with the dimension of the data. Many techniques, such as K-Segmentation and Coresets have been proposed to maintain accuracy and reduce the overall cost [17, 18, 20, 21, 24]. In *Coresets for k-Segmentation of Streaming Data*, Rosman et al. propose splitting signals into k-piecewise functions while maintaining a compact representation of all data seen so far [13].
5. Kernel Budgeting. Kernel budgeting is a set of techniques that seek to ensure the scalability of kernel methods by limiting the potential computation and memory complexity of a method. This ranges from removing state vectors, removing and projecting state vectors, to merging state vectors over removal [17].

2. Proposed Methods

In order to scale the visual sampling method while maintaining sample quality, this paper explores a solution tangentially related to [13]. The sampler uses a K-Means classifier trained on a small sample of the data on in order to establish a set of representative centroids. This is an offline preprocessing phase; however, a more suitable, robust, and complex online solution will be discussed later in the paper.

Given a preleant set of centroids, each point encountered in the stream is then compared to each centroid using the visualization kernel method. The sum value of these comparisons is then stored in a priority queue. This priority queue maintains a size invariant - while items are added, the lowest scoring items are dropped in order to maintain size.

This Cluster-Based sampling is then extended by splitting the stream of data among a set of subsamplers. For m-subsamplers, each sampler collects a subsample of size S/m. Each subsampler views only one of every m items in the input stream.

Tradeoffs of Proposed Method

This implementation is easily parallelized and maintains upward-scalability because it vastly reduces the need for global coordination and reduces the runtime to O(K * n log n), where K is the number

[1] Where ‘n’ represents the size of the training set given as input to the kernel.

of centroids, as determined by the user of the system; for this set of experiments, ~21 centroids was found to be an effective number. The added speed comes at the cost of some visualization quality: fewer centroids will mean the sampler collects a poorer sample more quickly than a sampler with more centroids.

4. Evaluation

4.1 Data

In order to explore and evaluate the efficacy of different methods the data chosen must have some key features: (i) the data must be time series based, (ii) be available in large size a reasonable level of accuracy and analysis, (iii) be of high enough dimension to allow sufficient scalability analysis[2]. To meet this challenge, I have chosen a time-series dataset displaying credit card fraud [?]. This dataset meets requirements i - iii and has already been transformed into quantitative features, easing the cost any unrelated preprocessing. The data is batched into hopping windows based on arrival time and desired window length (in seconds).

4.2 Environment

The goal of these experiments is a preliminary evaluation of each type of the proposed methods. As such the scope has been limited to a single node experiments in a controlled docker environment [A1].

4.3 Samplers and Sampler Configurations

In order to study these techniques, five different types of samplers were used. Each sampler was studied across different configurations of: window length (seconds), desired size of sample, and variations of features (eg. different numbers of subsamples and centroids). These samplers are:

1. *Ideal Sampler*: this sampler chooses the visually optimal set. The implementation makes chooses optimal samples at all performance costs.
2. *Random Sampler*: this sampler uses reservoir sampling in order to randomly select points for to fill the requisite sample size. This sampler is not evaluated for performance and only serves as a metric for sample quality comparison.

[2] An ideal set would have hundreds of columns as this would allow greater study of scalability challenges; however, this scale would be impractical for a study of this size.

3. *Cluster-Based Sampler*: this sampler is implemented as described in ??
4. *Parallel Subsampler-Based Sampler*: this sampler is implemented as described in ??. Reservoir sampling is used to randomly shuffle points between subsamples in order to ensure fairness and consistency.
5. *Combination Sampler*: this sampler is a combination of Sampler #4 and #5: it implements cluster- and parallel subsample-based techniques in order to choose items for each sample.

4.3 Experiment

In order to accurately study the proposed methods, two types of experiment were run against in order to evaluate the proposed methods. For each experiment, samplers were explored by varying desired sample size and window size/duration (s) in order to more accurate explore how each sampler scales. While a range of parameters was chosen and explored, in order to present easily consumable results, representative parameters were chosen for each graph.

The first experiment is a performance evaluation: each sampler configuration was fed 30 windows and the average time to compute each window was recorded. Note that the random sampler was ignored in this trial as it only serves as a benchmark of sample quality. The second experiment was to evaluate the effectiveness of each sampler configuration in terms of the visual quality defined in [5].

4.3.1 Evaluating Performance

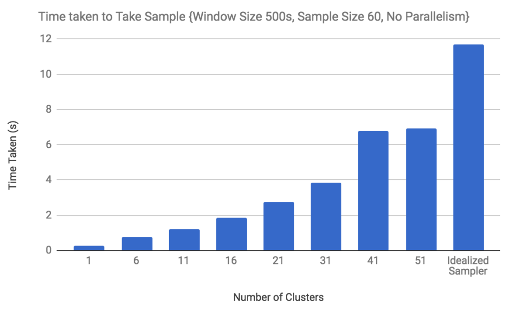

Figure 4.A: Demonstrating the effect of evaluating points against a set of centroids to reduce compute time

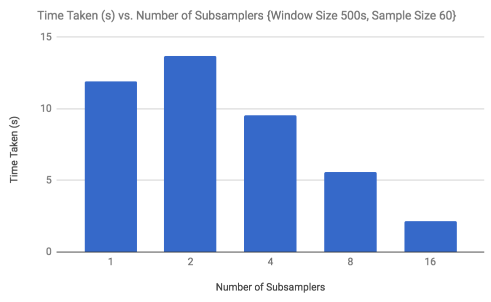

Figure 4.B: The Impact of collecting parallel subsamples for a large window and fixed large sample size.

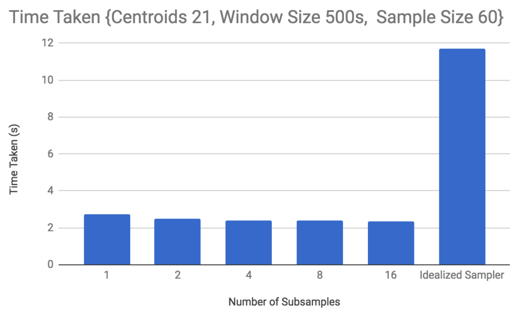

Figure 4.C: Demonstrating the performance of The Combination Sampler

From Figure 4.A it is clear that the Cluster-Based Sampler reduces the amount of time required to produce a sample over the Idealized Sampler. The graph empirically demonstrates the performance benefit of comparing each element in the data stream to a small set of static, representative elements rather than other elements in the collected set. Figure 4.B shows the effect of toggling the number of subsamples in the Subsampler-Based Sampler. The overall trend is that time taken to finish sampling a window reduces with a greater number of subsamplers. The increased amount of time at 2-subsamplers is due to a large cost of coordinating samplers relative to the small amount of time saved by dividing the total computation between two processes.

Figure 4.C is the most interesting of the performance experiments. It demonstrates the performance of the Combination Sampler for 21-centroids and a variable number of subsamplers. It shows that the performance benefit of increasing the number of subsamplers has little impact on the overall run-time. This can be explained by the implementation: changing the number of subsamplers does not reduce the cost of determining the loss for each point in the stream, it only reduces the number of comparisons in the priority queue. This marginal impact is shown in Figure 4.C as the difference between collecting 16 subsamples vs collecting one large sample is fractions of a second. However, it should be noted that the Combination Sampler vastly outperforms the Idealized Sampler with a ~5x reduction in overall compute time.

From these results we see that the use of centroids is far more impactful than simply collecting subsamples. This will be explored further in Section 5.

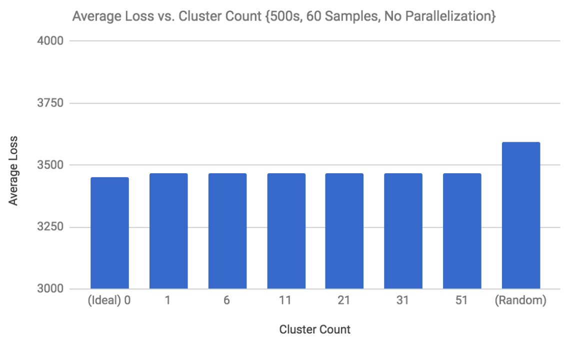

Figure 4.D: Demonstrating the impact of increasing the number of clusters on sample quality for a large window, large sample size and no parallelization

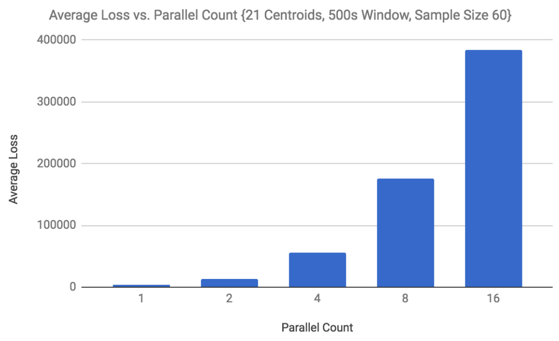

Figure 4.E: Demonstrating the impact of increasing the number of parallel subsamples on sample quality for a large window, large sample size, and fixed number of centroids.

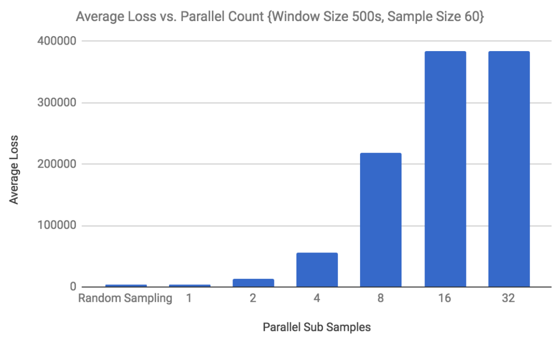

Figure 4.F: Demonstrating the impact of increasing the number of parallel subsamples on sample quality for a large window, large sample size.

4.3.2 Evaluating Quality

This set of experiments display two key findings: (i) subsampling is highly deleterious to sample quality, (ii) the Cluster-Based Sampler is able to reasonably maintain relatively high-quality samples.

Finding (i) is demonstrated in figures 4.E,F. In both cases not only does the introduction of subsampling underperform the Random Sampler, the quality of samples is many times worse than both a randomly collected sample and an ideal sample.

Figure 4.D best demonstrates the quality of the cluster based sampler. As the number of centroids varies, there is only a marginal difference between the quality of the Cluster-Based Sampler's output and Ideal Sampler's output. Here the Random Sampler's performance provides the reader context to understand how well the Cluster-Based Sampler is able to approximate the Ideal Sampler.

5. Discussion

This project evaluates the use of using clustering and subsampling to reduce the cost of collecting a visually representative sample for a given data stream. At the time of writing, the Cluster-Based sampler is the first of its kind to be applied in this specific context and has shown strong performance gains at the cost of a minimal drop in accuracy. In a streaming context, the Cluster-Based Sampler likely provides 'good enough' accuracy.

This remaining subsections will reflect the limitations in the work so far and propose ideas for future work.

5.1 Limitations

Although, there are several limitations with this experiment, this section will highlight the most important of them.

Complete evaluation for the problem identified in S1, would require a set of representative queries and a user study in order to truly evaluate the value of the proposed system. The author was unable to find a large time-series based dataset with a relevant set of corresponding queries during the course of this project. Furthermore, a complete user study of these results is outside the scope and ability of this project.

Current analysis and results are based on microbenchmarks. Higher quality results would be produced by running similar experiments on an many-node, heterogeneous architecture in order to more truly understand each sampler's coordination problems. This project chooses to use micro-benchmarks on a single-node for the benefit of faster iteration and reduced financial cost at the expense of some accuracy because the experimental results serve as good approximations for current work and directions for future work.

5.2 Future Work

A noted problem with offline computation to compute K-centroids is both the cost of training and the rigidity of the model. Future work on this problem should make use of online k-means estimators. There exists a large body of work for large scale and online cluster estimators, such as [22,23], that offer reduced memory and computation efficiency. The introduction of an online cluster estimator would likely reduce the performance benefit, however, it would provide a great deal of robustness, decrease the complexity to use the system, and decrease the cost of system maintenance.

This work leaves some avenues to be explored for this problem. One key avenue of interest lies in reducing the accuracy of each computation. The current experiment would see little benefit from computation caching because operations are very precise.

An additional avenue to explore is addition of a slack term in subsample collection. Enabling each subsampler to collect larger subsamples in order to choose a more accurate sample during the coordination phase would likely increase the accuracy. However, there is little evidence to support the efficaciousness of this and would need to be studied in a more representative environment.

6. Related Work

Work related to this project will largely be aligned to two fields: Stream Processing and Approximate Query Processing. Each will be discussed individually, with an intersecting system.

Stream Processing. This describes the set of applications specifically designed to process time sensitive data consumed from a message passing system like Apache Kafka and output the results further upstream [10]. Apache Spark-Streaming micro-batches data as it receives it in order to create small RDDs (Resilient Distributed Datasets) to build a resilient and performant API for an application that wishes to perform streaming operations [10].

Approximate Query Processing (AQP) Systems. This refers to a class of systems that rely on statistical approximations of an entire dataset in order to vastly improve performance at the cost of reasonable accuracy [1]. There are many popular examples of AQP systems: BlinkDB [3], ApproxHadoop [4], and SnappyData [2].

A related AQP system to this work, is IncApprox. IncApprox is a streaming system that leverages the benefits of incremental and approximate computing to output low latency, reasonably accurate output. IncApprox achieves this by using an “online stratified sampling algorithm to produce an incrementally updated approximate query with bounded error” [1]. It differs from the proposed system because it is actively learning the distribution of past data

Sampling. A difficult problem in engineering AQP systems is determining how to (quickly) choose samples that will be representative of a given dataset within predetermined error bounds and resource needs [1,5]. The sampling technique chosen is heavily reliant on the goals of the system. For example, stratified random sampling is often chosen over simple random sampling because it reduces the error for a given sample size and it can reduce the cost per observation [11].

7. References

[1] Dhanya R. Krishnan, Do Le Quoc, Pramod Bhatotia, Christof Fetzer, and Rodrigo Rodrigues. 2016. IncApprox: A Data Analytics System for Incremental Approximate Computing. In Proceedings of the 25th International Conference on World Wide Web (WWW '16). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1133-1144. DOI: https://doi.org/10.1145/2872427.2883026
[2] Jags Ramnarayan, Sudhir Menon, Sumedh Wale, and Hemant Bhanawat. 2016. SnappyData: a hybrid system for transactions, analytics, and streaming: demo. In Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems (DEBS '16). ACM, New York, NY, USA, 372-373. DOI: https://doi.org/10.1145/2933267.2933295
[3] Agarwal, S., Panda, A., Mozafari, B., Madden, S. and Stoica, I. (2017). BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. [online] Arxiv.org. Available at: https://arxiv.org/abs/1203.5485
[4] GIORI, I., BIANCHINI, R., NAGARAKATTE, S. and NGUYEN, T. (2017). ApproxHadoop: Bringing Approximations to MapReduce Frameworks. [online] Available at: https://dl.acm.org/citation.cfm?id=2694351
[5] Park, Y., Cafarella, M. and Mozafari, B. (2017). Visualization-Aware Sampling for Very Large Databases. [online] Arxiv.org. Available at: https://arxiv.org/abs/1510.03921
[6] Kandula et al. Quickr: Lazily Approximating Complex Ad-Hoc Queries in Big Data Clusters. [online] Arxiv..org. Avaible at: https://www.microsoft.com/en-us/research/publication/quickr-lazily-approximating-complex-ad-hoc-queries-in-big-data-clusters/
[7] Chaudhuri, Surajit & Ding, Bolin & Kandula, Srikanth. (2017). Approximate Query Processing: No Silver Bullet. 511-519. 10.1145/3035918.3056097.
[8] Apache Kakfa Documentation. Available at: https://kafka.apache.org/documentation/

[9] Spark-Streaming Documentation Available at: https://spark.apache.org/docs/latest/streaming-programming-guide.html
[10] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13). ACM, New York, NY, USA, 423-438. DOI: https://doi.org/10.1145/2517349.2522737
[11] STAT 506 Sampling Theory and Methods. (2017). 6.1 How to Use Stratified Sampling. [online] Available at: https://onlinecourses.science.psu.edu/stat506/node/27
[12] BlinkDB Code and Wiki. Available at: https://github.com/sameeragarwal/blinkdb/wiki
[13] Guy Rosman, Mikhail Volkov, Danny Feldman, John W. Fisher III, and Daniela Rus. 2014. Coresets for k-segmentation of streaming data. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'14), Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), Vol. 1. MIT Press, Cambridge, MA, USA, 559-567.
[14] Jing Lu, Steven C. H. Hoi, Jialei Wang, Peilin Zhao, and Zhi-Yong Liu. 2016. Large scale online kernel learning. J. Mach. Learn. Res. 17, 1 (January 2016), 1613-1655.
[15] Quoc Le, Tamás Sarlós, and Alex Smola. 2013. Fastfood: approximating kernel expansions in loglinear time. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28 (ICML'13), Sanjoy Dasgupta and David McAllester (Eds.), Vol. 28. JMLR.org III-244-III-252.
[16] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. In Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07), J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.). Curran Associates Inc., USA, 1177-1184.
[17] Zhiyun Lu, Avner May, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny: "How to Scale Up Kernel Methods to Be As Good As Deep Neural Nets", 2014; arXiv:1411.4000.
[18] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan: "Scalable Kernel Methods via Doubly Stochastic Gradients", 2014; arXiv:1407.5599.
[19] Bill G. Horne, Don R. Hush, Bounds on the complexity of recurrent neural network implementations of finite state machines, In Neural Networks, Volume 9, Issue 2, 1996, Pages 243-252, ISSN 0893-6080, https://doi.org/10.1016/0893-6080(95)00095-X. (http://www.sciencedirect.com/science/article/pii/089360809500095X)
[20] Christopher K. I. Williams and Matthias Seeger. 2000. Using the Nyström method to speed up kernel machines. In Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS'00), T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.). MIT Press, Cambridge, MA, USA, 661-667.
[21] Ninh Pham and Rasmus Pagh. 2013. Fast and scalable polynomial kernels via explicit feature maps. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '13), Rayid Ghani, Ted E. Senator, Paul Bradley, Rajesh Parekh, and Jingrui He (Eds.). ACM, New York, NY, USA, 239-247. DOI: https://doi.org/10.1145/2487575.2487591
[22] Alekh Agarwal, Sham M. Kakade, Nikos Karampatziakis, Le Song: "Least Squares Revisited: Scalable Approaches for Multi-class Prediction", 2013; [http://arxiv.org/abs/1310.1949 arXiv:1310.1949].
[23] Edo Liberty, Ram Sriharsha: "An Algorithm for Online K-Means Clustering", 2014; [http://arxiv.org/abs/1412.5721 arXiv:1412.5721].
[24] Yoshua Bengio, Aaron Courville: "Representation Learning: A Review and New Perspectives", 2012; [http://arxiv.org/abs/1206.5538 arXiv:1206.5538].

Appendix

A1. Environment Details

The environment is an Ubuntu docker container with 8GB of memory and 2 CPUs run on a 2.8Ghz Intel Core i5 processor.