Online Learning under Resource Constraints

Rodolfo S. Villaça*†, Rolf Stadler†
*Dept. of Industrial Technology (DTI), Federal University of Espírito Santo (UFES) – Brazil
†Dept. of Computer Science, Royal Institute of Technology (KTH) – Sweden
Email: rodolfo.villaca@ufes.br, stadler@kth.se

***Abstract*—Data-driven functions for network operation and management are based upon AI/ML methods whose models are usually trained offline with measurement data collected through monitoring. Online learning provides an alternative with the prospect of shorter learning times and lower overhead, suitable for edge or other resource-constraint environments. We propose an approach to online learning that involves a cache of fixed size to store measurement samples and periodic re-computation of ML models. Key to this approach are sample selection algorithms that decide which samples are stored in the cache and which are evicted. We present and evaluate four sample selection algorithms, all of which are derived from well-studied algorithms, and we specifically argue that feature selection algorithms can be used for our purpose. We perform an extensive evaluation of these algorithms for the task of performance prediction using data from an in-house testbed. We find that one of them (RR-SS) leads to models that achieve a prediction accuracy close to that obtained through offline learning, but at a much lower cost.**

***Keywords*—Online Learning, Real-time Learning, Edge Computing, Sample Selection, Performance Prediction.**

I. Introduction

Data-driven network and systems engineering is based upon applying AI/ML (Artificial Intelligence / Machine Learning) methods to measurement data collected from a network or compute infrastructure in order to build novel functionality and management capabilities. This is achieved through learning tasks that use this data for training. Examples of such tasks are KPI prediction using regression models and anomaly detection using clustering techniques.

The ML models are usually trained offline with measurements collected through monitoring [1], [2]. This method, although shown to achieve good results, has several drawbacks. First, the process takes a long time, often hours, since the system must be observed in various states to obtain an accurate model. Second, model training incurs a high computational cost, which increases (at least) linearly with the number of measurements. Third, changes in the system or the infrastructure may require new measurements and model re-computation. In this paper, we propose an online approach for model training, which allows for shorter learning times and lower computational costs, suitable for training in edge or other resource-constrained environments.

Our approach involves a cache of fixed size to store measurement samples and periodic re-computation of ML models based on using the current cache. Key to this approach are sample selection algorithms that decide which samples are stored in the cache and which are evicted. We present and evaluate four sample selection algorithms, all of which are based on well-studied algorithms. We use Reservoir Sampling [3] as a baseline, which maintains the cache with samples equally likely selected from the increasing sequence of collected measurements. Two other sample selection algorithms (RR-SS and LS-SS) are derived from two seminal feature selection algorithms: Redundancy and Relevance Feature Selection (RRFS) [4] and Laplacian Score (LS) [5]. Our argument is that the sample selection problem closely relates to the feature selection problem [6], [7], since in both cases a small subset of elements is selected to represent a larger set in such a way that the trained models are similarly effective. In this paper, effective models are those that achieve a prediction accuracy close to that obtained through offline learning.

We perform an experimental evaluation of these algorithms for the task of performance prediction using data from an in-house testbed that runs a video streaming and a key-value store application under different load patterns. We find that RR-SS leads to effective models in a much shorter time and at a much lower cost when compared to those obtained offline. While more research is needed to validate their effectiveness with learning tasks other than prediction and to identify the conditions for model re-computation, this work demonstrates that online learning allows for fast and low-overhead model training, and that accurate prediction is possible and should be considered for data-driven engineering.

II. The Problem of Online Sample Selection

Consider $\mathbf{X}$ as set of vectors $\mathbf{X}_t \in \mathbb{R}^k$, $t \in \mathbb{N}$, where $\mathbf{X}_t$ is a vector that represents a sample coming from the monitoring system at time t. Each vector has k dimensions representing k features, such as CPU and memory utilization of an application server or packets and Bytes sent/received by a network router. Also, consider $\mathbf{Y}_t \in \mathbb{R}$, $t \in \mathbb{N}$ as a target that represents an application KPI at the same time t. As an example, for a Video on Demand (VoD) application, $\mathbf{Y}_t$ can be the rate of displayed frames per second at time t. A predictor F: $\mathbf{X}_t \mapsto \widehat{\mathbf{Y}}_t$ is a function that maps $\mathbf{X}_t$ to a predicted value $\widehat{\mathbf{Y}}_t$ for some target, which is effective if $\widehat{\mathbf{Y}}_t$ is close to $\mathbf{Y}_t$. For comprehensive background information about Machine Learning (ML), see the seminal works Vapnik [1], Bishop [8], and Goodfellow et al. [2].

978-3-903176-32-4

A. Offline Learning

In this paper, offline learning is understood as a learning method that acquires samples taken during an observation period. After that, some of these samples are randomly chosen to compose a subset, also known as the training set, which is then used to build an ML model. The number of samples in the training set is typically large (about 70% of the entire set of samples) and the remaining 30%, also known as the test set, is used to evaluate the model. Once built and tested, this model is used for prediction and remains unchanged until a new training set is available and a new model is generated. In summary, offline learning uses past observations for learning and relies on the diversity of the samples in the training set to build effective ML models. Both training and test set are randomly selected from the entire set of samples.

B. Online Learning

In this paper, online learning is understood as a learning method in which the training set is continuously updated. In our case, the training set is known as training cache, or simply, cache. The size of the cache is usually much smaller than the size of the training set used in offline learning. It typically ranges from tens to hundreds of samples in online learning and thousands to ten thousands of samples in offline learning. As the cache is continuously updated, the ML model can be periodically recomputed. To keep the cache small with a fixed size and to ensure the diversity of the samples in the cache, a sample selection algorithm is introduced. Figure 1 shows our approach to online learning. In this figure, samples with only features ($\mathbf{X}_t$) are forwarded to the `Sample Selection` algorithm to update the `Cache`, whose content is used as the training set for `Unsupervised Learning`. The result is an unsupervised `ML Model`. Similarly, samples with both features and targets ($\mathbf{X}_t$, $\mathbf{Y}_t$) are forwarded to the `Sample Selection` algorithm to update the `Cache`, which is used as the training set for `Supervised Learning`. The result is a supervised `ML Model`. Once an `ML Model` is available, samples are used for operational purposes, including clustering, prediction, and classification. In the following, we focus on online supervised learning, leaving the investigation of online unsupervised learning for future work.

The use case we study in this paper (see Section V) includes measurements from a computing and networking infrastructure that provides Video-on-Demand (VoD) and Key-Value (KV) applications to a client base. We focus on supervised learning, specifically QoS prediction, and assume that the management system receives information about an application target at predefined observation intervals. During an observation interval, the set of all samples and targets received by the monitoring system increases, but, in contrast to offline learning, only a small portion of them are stored in the cache and used to compute the ML model, which is updated periodically. However, not every new sample inserted in the cache triggers an update, making the system more stable without reacting to transitory peaks in infrastructure measurements. The model update can occur in different ways: after a given time period,

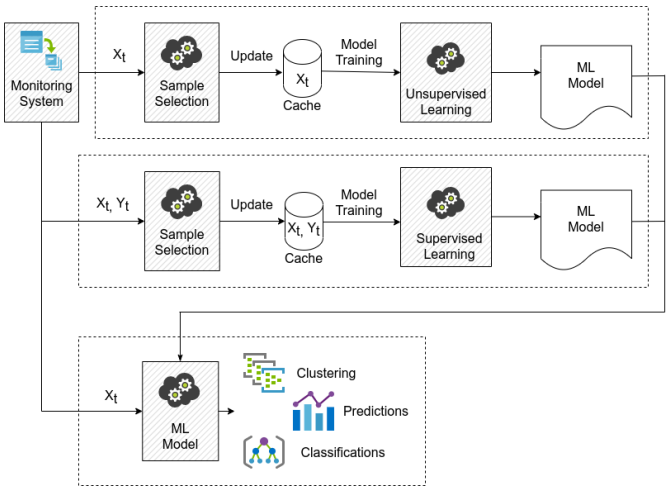


Fig. 1. Online Learning: Samples with both features and targets ($\mathbf{X}_t$, $\mathbf{Y}_t$) from the `Monitoring System` are forwarded to the `Sample Selection` algorithm to update the cache for supervised learning tasks, such as predictions. Samples with only features ($\mathbf{X}_t$) are forwarded to the `Sample Selection` algorithm to update the cache for unsupervised learning tasks, like clustering. Once an `ML Model` is available after the training process (in the first two rows of this figure), $\mathbf{X}_t$ is used as input of this `ML Model` for operational purposes, including clustering, prediction, and classification.

after a given number m of samples has been received (the approach used in this paper), or after a change in system configuration or infrastructure.

In online learning for a prediction task, the cache holds a set of n tuples ($\mathbf{X}_t$, $\mathbf{Y}_t$) and is used to build a predictor F: $\mathbf{X}_t \mapsto \widehat{\mathbf{Y}}_t$. The key difference to offline learning is the need for a cache policy: while in offline learning the training set includes a large number of samples collected offline during an observation period, in online learning the cache is frequently updated and, to keep the cache size fixed and to ensure an effective prediction model, a sample selection algorithm is required. A sample selection algorithm maintains the cache with representative samples in order to minimize the prediction error of the ML model built using this cache. In our use case, the effectiveness of a sample selection algorithm is evaluated by computing the prediction error for a predictor trained on the cache using the Normalized Mean Absolute Error (NMAE) metric. For a set of m samples, it is defined as follows:

$$NMAE = \frac{\sum_{t=1}^{m} |\widehat{Y}_t - Y_t|}{\sum_{t=1}^{m} |Y_t|}$$

C. Research Questions

Considering our approach to online learning depicted in Figure 1, the main contribution of this paper relates to the proposal end experimental evaluation of online sample selection algorithms. A sample selection algorithm makes a decision about the insertion (or not) of each new sample in the cache and, if the insertion is required, which sample must be evicted to keep the cache size fixed. We are interested in a small cache size that leads to predictors with similar accuracy than those trained offline. This leads to the following research questions:

1) Which online sample selection algorithms allow for effective learning using monitored metrics from a networked system?
2) Do different algorithms vary in their effectiveness with respect to the application target or load pattern?
3) How does the cache size relate to the effectiveness of the learning task?

III. Related Work

The use of AI/ML in network management is still a challenge, especially because most ML models are trained offline, and retraining of a model from scratch can be computationally intensive, time-consuming, and prohibitive [9]. Online feature selection algorithms are a well-known research topic in literature [7], [6], [10]. However, the use of these algorithms for sample selection is a novelty. In this paper, two classical feature selection algorithms are adapted and evaluated for the sample selection problem. The first algorithm is based on the Relevance and Redundancy Feature Selection (RRFS) [4], and the second is based on the Laplacian Score (LS) [5].

Perform unsupervised learning with limited-samples and high-dimensions is known as a High-Dimensional Limited-Sample Size (HDLSS) problem. Most works on HDLSS tackles the problem of unsupervised learning with a few high-dimensional samples. In [11] a variational autoencoder for dimensionality reduction and an unsupervised classification model is used to deal with HDLSS data, but it is applied only in unsupervised ML models, with no relation to the sample selection problem. The problem of target prediction using a small number of samples is presented by Fursov [12] et al. with no consideration about the use of cache or sample selection algorithms. Their goal is to remove noised-samples from this small set to reduce the prediction error. Nikolov et al. [13] present a passive data rate estimation method by utilizing an Adaptive Similarity-based Regression (ASR) approach. A sample selection method is presented in ASR, but it is specific for the problem of data rate estimation and cannot be adapted to general prediction tasks.

An approach to improve online Support-Vector Machine (SVM) with unlearning criteria is presented in [14], which proposes a strategy to remove the "least relevant" sample from the training set. The sample selection algorithm used in [14] is applied only for unsupervised learning and they don't consider training sets of small size (caches). Online predictions using reinforced learning is a hot topic in literature [15] but with no relation to the sample selection problem. In summary, the proposal of online sample selection algorithms for resource-constrained infrastructures is the main contribution of this paper.

IV. Online Sample Selection Algorithms

In this section, four different algorithms for online sample selection are presented. The first is based on the classical Reservoir Sampling (RS) algorithm [3] and the second, motivated by the availability of some application target measurements, is a supervised version of the RS algorithm, named Supervised RS. Supervised RS is classified in this paper as a "supervised sample selection algorithm" because it also depends on the values of the target. As the approach used in this paper is to investigate the use of some feature selection algorithms for sample selection, the third and the fourth algorithms are based on modified versions of two classical algorithms for feature selection, specifically the Relevance and Redundancy Feature Selection (RRFS) [4] and the Laplacian Score (LS) [5]. Both RRFS and LS do feature selection based on the intrinsic characteristics of the features, independently from the values of the targets. The third proposal is named Relevance and Redundancy Sample Selection (RR-SS) and the fourth is named Laplacian Score Sample Selection (LS-SS). In this paper, RR-SS, and LS-SS are classified as "unsupervised sample selection algorithms" because they do not rely on the values of the target to decide about cache updates. RS, RR-SS and FS-SS can be used for online supervised and unsupervised learning (see Figure 1).

A. Reservoir Sampling

Algorithm 1: Reservoir Sampling (RS): Iteratively read sample X_t and decide on updating cache of size n

Input: $X = \{X_1, X_2, ..., X_t, ...\}$
Result: Updated $cache$

```
1 t ← 1;
2 while (Monitoring System is Up) do
3 |   X_t ← MonitoringSystem.Acquire(t);
4 |   r ← RandomInt(0, t);
5 |   if (r < n) then
6 |   |   cache[r] ← X_t;
7 |   end
8 |   t ← t + 1;
9 end
```

The Reservoir Sampling [3] is a family of randomized algorithms for choosing uniformly a random set of n samples (also known as "reservoir") from a population of unknown size, in a single pass over each sample. The size of the population is not known to the algorithm in advance and is typically too large to fit into the main memory for post-processing. In the online sample selection problem, the reservoir is the cache and the population is composed of each sample coming from the monitoring system. Algorithm 1 summarizes our implementation of the Reservoir Sampling algorithm for online sample selection. In summary, for each sample $\mathbf{X}_t$ acquired from the `MonitoringSystem` at time t (Line 3) and a $cache$ of size n, a random number r is drawn from the range $[0..t]$ (Line 4) using the `RandomInt` function. If r is lower than n (Line 5), the new sample is inserted in the $cache$ in its corresponding r position (Line 6), replacing the older sample in this position. Otherwise, $\mathbf{X}_t$ is discarded. In the beginning, the first n samples are automatically added to the cache. As it is a one-pass algorithm for each input sample

$\mathbf{X}_t$, with complexity O(1) – which means that its behavior is independent of the size of $\mathbf{X}_t$ – the results of the Reservoir Sampling algorithm are used as the baseline for the other sample selection algorithms presented in this paper.

B. Supervised RS

Once for the training of supervised algorithms the target values $\mathbf{Y}_t$ are available, the Reservoir Sampling algorithm is modified to take advantage of additional information: $\widehat{\mathbf{Y}}_t = F(\mathbf{X}_t)$, which is the prediction of $\mathbf{Y}_t$ using the current ML model, F. This new algorithm is called Supervised Reservoir Sampling, or simply, Supervised RS. Instead of simply using the randomness of the RS algorithm on the decision to insert (or not) a sample and target value in the cache, in the Supervised RS algorithm it is also verified if the prediction error of the current sample using the current prediction model F, $\frac{|\widehat{Y}_t - Y_t|}{|Y_t|}$, is lower than the prediction error of the current prediction model ($currentPredError$) obtained from the evaluation with the current $cache$, both measured as NMAE. The main idea behind this modification is to not increase the error of the current model when inserting a new sample and target in the cache. This process is summarized in Algorithm 2.

Algorithm 2: Supervised RS: Iteratively read sample (X_t, Y_t) and decide on updating $cache$ of size n

Input: $X = \{(X_1, Y_1), (X_2, Y_2), ..., (X_t, Y_t), ...\}$
Result: Updated $cache$

1 $t \leftarrow 1$;
2 **while** *(MonitoringSystem is Up)* **do**
3 $\quad (X_t, Y_t) \leftarrow$ MonitoringSystem.Acquire(t);
4 $\quad r \leftarrow$ RandomInt$(0, t)$;
5 $\quad$ **if** $(r < n)$ *and* $(\frac{|\widehat{Y}_t - Y_t|}{|Y_t|} < currentPredError$ **then**
6 $\quad\quad cache[r] \leftarrow (X_t, Y_t)$;
7 $\quad$ **end**
8 $\quad t \leftarrow t + 1$;
9 **end**

In summary, considering a sequence of samples with both features $\mathbf{X}_t$ and target $\mathbf{Y}_t$ coming from the `MonitoringSystem` at time t (Line 3), and a $cache$ of size n, the Supervised RS algorithm uses both the probabilistic behavior of the RS algorithm and the value of the target $\mathbf{Y}_t$ on the decision about when to update the $cache$ with both ($\mathbf{X}_t$, $\mathbf{Y}_t$). Supervised RS also keeps the cache size fixed. To do this, a random number r is drawn from the range [0..t] (Line 4) for each tuple ($\mathbf{X}_t$, $\mathbf{Y}_t$) from the `MonitoringSystem` at time t. In Lines 5-6, if r is lower than n, and if the prediction error of $\mathbf{X}_t$ using the current model ($\mid \widehat{\mathbf{Y}}_t - \mathbf{Y}_t \mid$) is lower than the prediction error of the current model ($currentPredError$), both ($\mathbf{X}_t$, $\mathbf{Y}_t$) are inserted in the $cache$ in the r position. Otherwise, both $\mathbf{X}_t$, $\mathbf{Y}_t$ are discarded. The complexity of the Supervised RS algorithm depends on the predictor because we need to use the current model F and the new sample $\mathbf{X}_t$ to estimate $\widehat{\mathbf{Y}}_t$. Algorithm 2 runs while the monitoring system is up (Line 2) and can only be used in online supervised learning.

Note that an opposite approach can also be considered: insert a sample in the cache if its prediction error is greater than the prediction error of the current model. This version may be effective after a change of the system configuration that requires major model adjustments.

C. RR-SS Algorithm

Algorithm 3: RR-SS: Iteratively read sample X_t and decide on updating $cache$ of size n

Input: $X = \{X_1, X_2, ..., X_t, ...\}$
Result: Updated $cache$

1 $t \leftarrow 1$;
2 **while** *(Monitoring System is Up)* **do**
3 $\quad \overline{X}_{avg} \leftarrow$ AvgOfAllSamples$(cache)$;
4 $\quad cacheRankList \leftarrow$ Rank$({cache}^T, {cache}^T)$;
5 $\quad rankAvgSample \leftarrow$ Rank$({\overline{X}_{avg}}^T, {cache}^T)$;
6 $\quad X_t \leftarrow$ Monitoring.AcquireSample(t);
7 $\quad rankNew \leftarrow$ Rank$({X_t}^T, {cache}^T)$;
8 $\quad$ **if** $(rankNew > rankAvgSample)$ **then**
9 $\quad\quad i \leftarrow$ GetLowestIndex$(cacheRankList)$;
10 $\quad\quad cache[i] \leftarrow X_t$;
11 $\quad\quad \overline{X}_{avg} \leftarrow$ AvgOfAllSamples$(cache)$;
12 $\quad\quad rankAvgSample \leftarrow$ Rank$({\overline{X}_{avg}}^T, {cache}^T)$;
13 $\quad$ **end**
14 $\quad t \leftarrow t + 1$;
15 **end**
16 **Procedure** `Rank` $(matrix \in \mathbb{R}^{i \times j}$, $cache \in \mathbb{R}^{n \times k})$ **:**
17 $\quad redundanceList \leftarrow$ CosineSim$(matrix, cache)$;
18 $\quad relevanceList \leftarrow$ EuclideanDist$(matrix, cache)$;
19 $\quad rankList \leftarrow \dfrac{relevanceList}{redundanceList}$;
20 $\quad$ return $rankList$;
21 **end**

A modified version of the RRFS algorithm for online feature selection is the Adapted Relevance Redundancy (ARR) algorithm, presented in [16]. As ARR is already adapted to online learning, the RR-SS algorithm for online sample selection is based on it and presented in Algorithm 3. In RR-SS the $cache$ is considered as a $n \times k$ matrix that contains samples X_t in rows. X_t are samples coming from the `MonitoringSystem` at time t. ${cache}^T \in \mathbb{R}^{k \times n}$ is the transposed $cache$ matrix, with features as rows and samples as columns. Also, a sample $\mathbf{X}_t$ is transposed to ${\mathbf{X}_t}^T$ to be seen as a feature vector. Then, it is possible to use well-known feature selection algorithms, such as RRFS, in the sample selection problem. In Algorithm 3, $\overline{\mathbf{X}}_{avg}$ is defined as a vector with the average of each column in the cache, i.e., a vector with the average of each feature considering all samples in the cache (Line 3). Formally, considering $\mathbf{X}_t$ with k features, $\overline{\mathbf{X}}_{avg} = \{\overline{f_1}, \overline{f_2}, \ldots, \overline{f_k}\}$.

A rank list $cacheRankList$ is generated for the cache (Line 4). The elements of the $cacheRankList$ represent the rank of each sample in the cache, calculated by the relation between each value of the relevance list and each value of the redundancy list. The $redundancyList$ is calculated as the average of the cosine similarity (`CosineSim`) between each pair of sample vectors in the cache, while the $relevanceList$ is calculated as the average of the Euclidean distance (`EuclideanDist`) between each pair of sample vectors in the cache. In Lines 5, 7, and 12 the `Rank` procedure is also used to generate a rank value for both $\overline{\mathbf{X}}_{avg}$ ($rankAvgSample$) and $\mathbf{X}_t$ ($rankNew$) vectors. Considering a sequence of samples coming from the `MonitoringSystem` at time t (Line 6), the RR-SS algorithm is used to decide about cache updates. ${cache}^T$, ${\overline{\mathbf{X}}_{avg}}^T$ and ${X_t}^T$ are the transposed versions of $cache$, X_{avg} and X_t. If the rank of the new sample X_t ($rankNew$) is greater than the rank of $\overline{\mathbf{X}}_{avg}$ ($rankAvgSample$), the new sample is added to the cache replacing the sample with the lowest rank in the cache (Lines 9-12). As a design choice, we used the average, but the median, 25% or 75% percentiles could also be used. Algorithm 3 has complexity O(kn^2), where n is the cache size and k is the number of features in each sample.

D. Unsupervised LS-SS

An implementation of the Laplacian Score (LS) algorithm for feature selection [5] is publicly available [17]. A modified version of this algorithm for sample selection is described in Algorithm 4 and called Laplacian Score Sample Selection (LS-SS). In Algorithm 4, every new sample $\mathbf{X}_t$ from the monitoring system (Line 3) is inserted in the cache (Line 4), which has $n+1$ samples. A neighbor graph is constructed with all $n+1$ samples in the cache (Line 5). This graph has samples as its nodes and the links correspond to the k-Nearest Neighbors (kNN, with k=5) of each sample. The Laplacian Score for each sample in the $cache$ is computed (Line 6) and a list ($lapScoreList$) with these scores is generated. The scores are calculated according to a weighted matrix between all nodes (samples) in the graph. The weights are calculated according to the number of links in each node (sample). Then, the index i of the sample with the highest score is identified (Line 7) and the i^{th} sample is removed from the $cache$ (Line 8), that goes back to size n. This procedure is repeated for all samples coming from the monitoring system while the system is up.

The complexity of the LS-SS algorithm is related to the graph construction and the generation of its corresponding weighted matrix. According to [5], [16], adapted for the sample selection problem, LS-SS has complexity O(nk^2), where n is the cache size and k is the number of features.

Algorithm 4: LS-SS: Iteratively read sample X_t and decide on updating $cache$ of size n

Input: $X = \{X_1, X_2, ..., X_t, ...\}$
Result: Updated $cache$

1 $t \leftarrow 1$;
2 **while** *(Monitoring System is Up)* **do**
3 $\quad X_t \leftarrow$ Monitoring.AcquireSample(t);
4 $\quad cache[n+1] \leftarrow X_t$;
5 $\quad graph \leftarrow$ BuildNeighborGraph($cache^T$);
6 $\quad lapScoreList \leftarrow$ CompLaplacianScores($graph$);
7 $\quad i \leftarrow$ GetIndexOfHighestScore($lapScoreList$);
8 $\quad cache \leftarrow$ RemoveSample(i);
9 $\quad t \leftarrow t+1$;
10 **end**

V. Testbed Presentation and Dataset Description

In this section, the experimental infrastructure and the dataset are described, also with the services that run on this infrastructure, namely, a Video-on-Demand (VoD) service and Key-Value (KV) store. Figure 2 outlines the testbed at KTH. It includes a server cluster, an emulated OpenFlow network, and a set of clients. The server cluster is deployed on a rack with 10 servers. All machines run Ubuntu Server 14.04 64 bits, and their clocks are synchronized through NTP. The OpenFlow network includes 14 switches, which interconnect the server cluster with clients and load generators. The load generators emulate clients using the evaluated services.

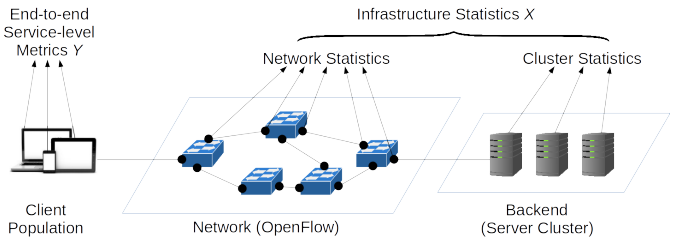
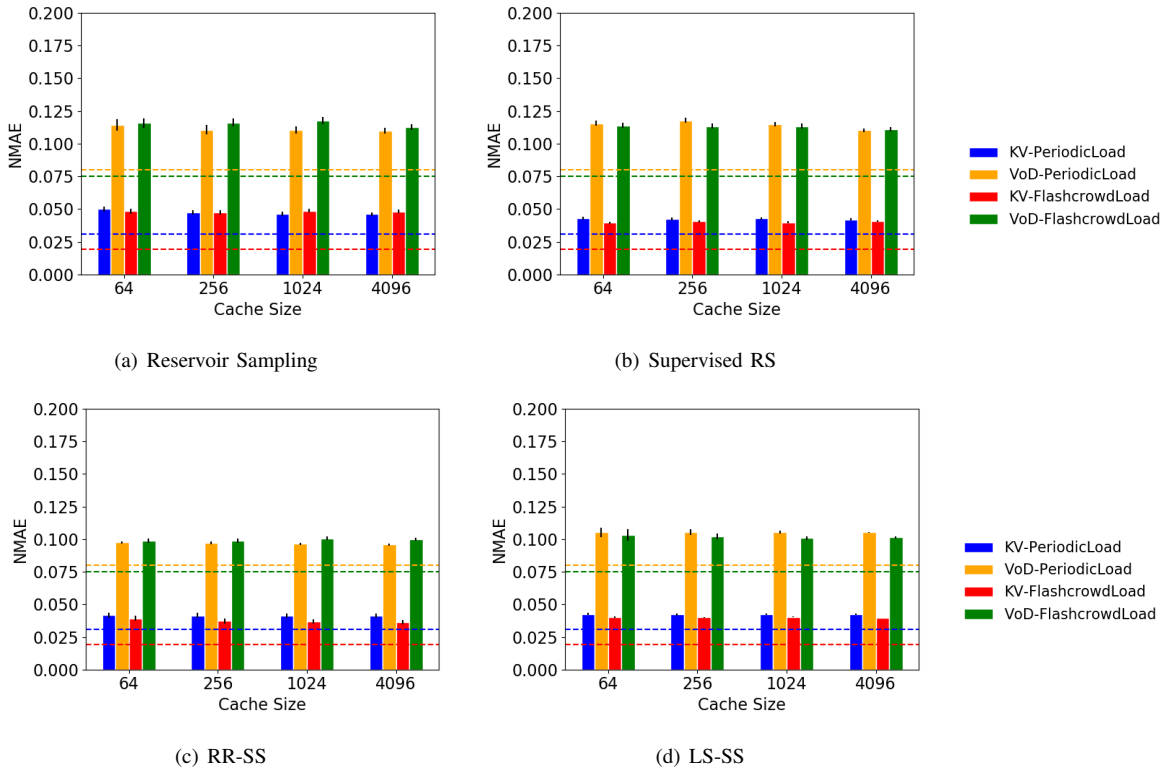


Fig. 2. Testbed used for experiments at KTH. In all scenarios KV and VoD performance metrics from infrastructure measurements are predicted [18]

1) The VoD application: uses VLC media player software, which provides single-representation streaming with varying frame rate with the ten most-viewed YouTube videos in 2013.

2) The KV store application: uses Voldemort software. It executes on the same machines as the VoD service. Six of them act as KV store nodes in a peer-to-peer fashion, running Voldemort version 1.10.22.

3) The Dataset: Metrics from the cluster and network infrastructure, namely, the input feature set $\mathbf{X}_{cluster}$ and $\mathbf{X}_{port}$. The union of these metrics is referenced as $\mathbf{X}$. The $\mathbf{X}_{cluster}$ feature set is extracted from the Linux kernel running on the servers executing the VoD and KV applications. Examples of such statistics are CPU and memory utilization and I/O rate. It includes about 1,700 statistics per server in total and is publicly available [19]. The X_{port} feature set is extracted from the OpenFlow switches at per-port granularity. It includes statistics from all switches in the network: i) Total number of Bytes Transmitted per port; ii) Total number of Bytes Received per port; iii) Total number of Packets Transmitted per port; and iv) Total number of Packets Received per port. In this paper, for the VoD dataset 16 features from both $\mathbf{X}_{cluster}$ and $\mathbf{X}_{port}$ were used, while for the KV dataset 256 features



(a) Reservoir Sampling

(b) Supervised RS

(c) RR-SS

(d) LS-SS

Fig. 3. Prediction error (NMAE) vs cache size for four sample selection algorithms RS, Supervised RS, RR-SS, and LS-SS. The bars show mean values and 95% confidence intervals of 10 runs. Evaluations are performed for the VoD ad KV applications and two different load patterns. The horizontal dashed lines show the results of offline predictions.

were used. These values follow the results presented in [20] with the same applications, features, and targets. $\mathbf{Y}$ targets from both applications are measured on the client device. During an experiment, we capture the Display Frame Rate (`DispFrames`) (frames/sec), i.e., the number of displayed video frames per second for the VoD application, and Read Response Time `ReadsAvg`, i.e., the average read latency.

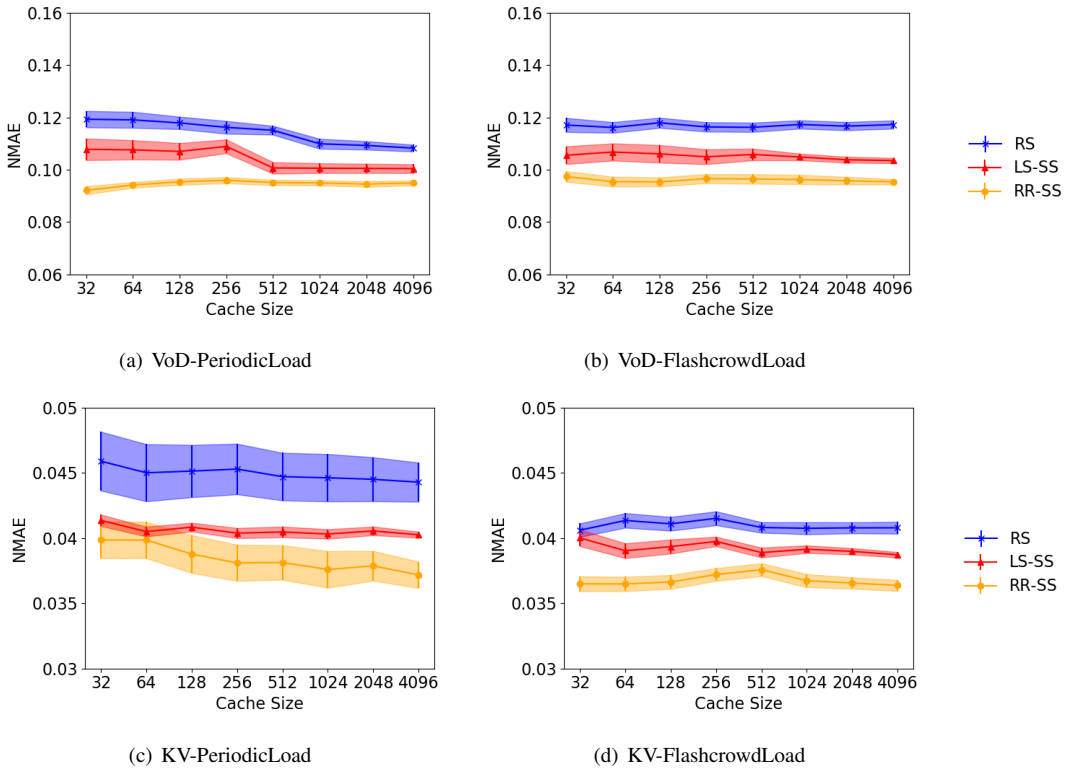
4) Methodology: During experiments, $\mathbf{X}$ and $\mathbf{Y}$ statistics are collected every $1s$ on the testbed during $10h$. Each trace has about 36,000 samples and targets. For each service running on the testbed, the data collection framework produces a trace in form of a time series $(\mathbf{X}_t, \mathbf{Y}_t)$. We interpret this time series as a set of samples $\{(\mathbf{X}_1, \mathbf{Y}_1), ..., (\mathbf{X}_t, \mathbf{Y}_t)\}$.

5) Service Load Patterns and Generators: The VoD load generator dynamically controls the number of active VoD sessions, spawning and terminating VLC clients. The KV load generator controls the rate of KV read operations issued per second. The VoD and the KV services run different load patterns. In the *Periodic load pattern*, the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function with a period of 60 min. In the *Flashcrowd load pattern*, the load generator produces requests following a Poisson process whose arrival rate is modulated by a Flashcrowd model [21], whose arrival rate starts at an initial load and peaks at flash events, which are randomly generated, and then decreases to the initial load.

VI. Evaluation and Results

In this section, the prediction errors of the four sample selection algorithms are presented and discussed, also with an evaluation of the time to perform the sample selection and to build a new prediction model. For reference, the results of the predictions from an offline learning model are presented in Table 1, which summarizes the results from [20]. It is important to highlight that in offline learning, both training and test sets are randomly selected from all samples in the same dataset.

In all evaluation scenarios, the Random Forest [22] algorithm is used to build the prediction models, which are updated at every round of m samples. Each prediction model is used to predict the next m samples, which is set to 100, i.e., at every 100 samples the prediction model is updated and used to predict the next 100 samples. n is evaluated from values starting in 32 to 4096, growing exponentially with base 2 to explore a large state space in a few steps. We perform 10 executions for each scenario with different random seeds of the Random Forest algorithm. The number of trees (estimators) is set to 20 and the depth of each tree is set to 10. In summary, taking 1 sample at each 1s during approximately 10h we have 36,000 samples for each load pattern and application. Using m=100, a new model is generated approximately 360 times in each scenario (36,000/100). For each new model, its corresponding NMAE is calculated.



(a) VoD-PeriodicLoad (b) VoD-FlashcrowdLoad

(c) KV-PeriodicLoad (d) KV-FlashcrowdLoad

Fig. 4. Prediction error (NMAE) vs cache size for three unsupervised sample selection algorithms RS, RR-SS, and LS-SS. The mean values and 95% confidence intervals of 10 runs are shown. Evaluations are performed for the VoD ad KV applications and two different load patterns.

TABLE I
PREDICTION ERROR (NMAE) USING OFFLINE LEARNING FOR DIFFERENT APPLICATIONS AND LOAD PATTERNS [20].

Dataset	Target	NMAE
VoD-PeriodicLoad	DispFrames	0.080
VoD-FlashcrowdLoad	DispFrames	0.075
KV-PeriodicLoad	ReadsAvg	0.031
KV-FlashcrowdLoad	ReadsAvg	0.019

A. Evaluation of Supervised Algorithms

In this section, both the RS and the Supervised RS algorithms for online sample selection are evaluated regarding their prediction errors (NMAE). It is important to highlight that, while the Reservoir Sampling (Algorithm 1) is an unsupervised algorithm, this section contains a comparison with the Supervised RS. In Figure 3 each bar represents the average of the prediction error for the VoD and KV applications with different load patterns and cache sizes. The 95% confidence interval is also plotted. The colored horizontal dashed lines represent the results using offline learning, as listed in Table I.

As expected, according to Figure 3(b) the Supervised RS algorithm achieves better results when compared to the original Reservoir Sampling, depicted in Figure 3(a). The prediction error using the Supervised RS is closer to the offline results represented by the dashed horizontal lines. It is also possible to see that, in the KV-Flashcrowd load pattern, the prediction error using the Reservoir Sampling algorithm doubles the error of the offline results. Regarding the cache size, it is possible to notice that there is no significant difference in choosing the lowest or the highest ones. This fact is better discussed later, in Section VI-C.

B. Evaluation of Unsupervised Algorithms

Two unsupervised online sample selection algorithms are evaluated in this section, RR-SS, and LS-SS. The evaluation is related to their prediction errors, also measured using the NMAE. We call these algorithms "unsupervised" because the sample selection doesn't depend on the value of the target nor on the current evaluation of the prediction error.

According to Figure 3(c), considering the prediction error for the VoD application under both load patterns, the proposed RR-SS algorithm achieves better results than those using the LS-SS algorithm, depicted in Figure 3(d). On the other hand, when looking at the KV application, there is no significant difference between the RR-SS (Figure 3(c)) and the LS-SS (Figure 3(d)) which can lead us to conclude that there is no winner between them and it depends on the application and load pattern being used. Anyway, it is also possible to see that, in all cases, the prediction error when using RR-SS or LS-SS algorithms is lower than when using the original Reservoir Sampling algorithm, used as our baseline. This error

for both algorithms is close to the offline results, represented by the four colored dashed lines in the figure, except for the KV application under the Flashcrowd load pattern, which reinforces our perception that the choice of the best algorithm depends on the application and the load pattern. Due to this, in the next section, the proposed unsupervised algorithms are compared explicitly for each application and load pattern, for an extended range of cache sizes.

C. The influence of the cache size

Figures 4(a)-(d) show the average prediction error (NMAE) for the unsupervised algorithms (including Reservoir Sampling) for the different applications and load patterns. In these evaluations, we use an extended range of cache sizes, from 32 to 4096. In each figure, the average value is plotted as a colored line, and the shaded area represents the 95% confidence interval. We observe that, in all cases, Reservoir Sampling has the worst performance, while the RR-SS achieves the best results among all unsupervised sample selection algorithms.

Surprisingly, we find that the cache size does not have a strong influence on the prediction error. The theory of statistical learning suggests that the error declines exponentially with increasing sample size if samples are chosen uniformly at random [1]. When we train a Linear Regressor instead of Random Forest on the same data, we find a much clearer decline of the error when the cache size increases (results are not included in this paper). Our finding merits further investigation, since it suggests that a small cache size can achieve similar prediction accuracy than a significantly larger one.

Some reflection regarding the cache size: remember that when using the offline approach we take samples from the entire dataset (70% ratio) for training and use other samples from the same dataset (30% ratio) for evaluating the prediction accuracy. In offline learning, the model is built with samples from the entire monitored time period. When using an online algorithm, as proposed in this paper, we use samples from the past to predict targets with samples in the future. Our results suggest that sample selection itself is of more importance than the number of samples in the cache, at least for our data sets. Maybe the diversity of samples in the cache is of more importance than the cache size for prediction accuracy.

D. Sample Selection and Model Build Time

The sample selection time is measured as the time between a new sample is acquired and the decision is made about performing (or not) the cache update. It is independent of the cache size but is highly dependent on the sample selection algorithm. For both RR-SS and LS-SS, the number of features is also of high importance, as a cache transpose operation is performed for both Algorithms 3 and 4: the greater the number of features, the greater the cache size in the transposition, and the larger the computational cost. Due to space restrictions, these results are not detailed here, but on average they vary from about 10ms for Reservoir Sampling to 200ms for RR-SS on KV trace data. A more efficient implementation of both RR-SS and LS-SS algorithms is left as future work.

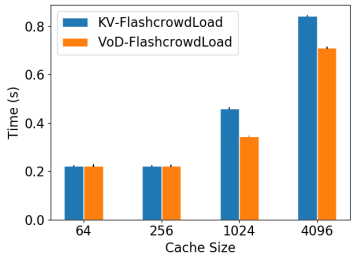


Fig. 5. Model training time vs cache size for a Random Forest predictor. The mean values and 95% confidence intervals of 10 runs are shown. Time is measured on an Intel i7-5500U CPU 2.40GHz laptop with 16GB RAM.

The time to build a new prediction model using the Random Forest algorithm is presented in Figure 5. We observe that this time is highly influenced by the cache size n and the number of features k, which agrees with the complexity of the Random Forest algorithm, namely $O(kn*log(n))$. All computations are performed in less than 1s, which is the monitoring interval on the KTH testbed.

VII. Conclusion

This paper makes a contribution to data-driven engineering for resource-constrained systems. It demonstrates, using testbed results, that online learning can allow for fast and low-overhead model training, and that accurate prediction is possible using a small cache. The key to our online learning approach is sample selection, and we evaluated four algorithms: RS, Supervised RS, RR-SS, and LS-SS. Among the unsupervised algorithms, RR-SS achieves the best accuracy for a Random Forest predictor, across different applications and load patterns.

Regarding the impact of the cache size on the effectiveness of the model, we find that the cache size does not have a strong influence on the prediction error. This is surprising and suggests that the diversity of samples in the cache is of higher importance than the cache size. Overall, we find that, using our approach, sufficiently accurate predictions can be made with a cache size that is 2-3 orders of magnitude smaller than the number of samples used for offline prediction on the same data traces.

As future work, we plan to further investigate the influence of the cache size on the effectiveness of ML models. We also want to study and evaluate the use of sample selection algorithms for unsupervised online learning including clustering.

Acknowledgment

This research has been supported by Brazilian Council for Scientific and Technological Development (CNPq), FAPES, the Swedish-Brazilian Research and Innovation Centre (CISB), SAAB, and the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects AutoDC and ANIARA.

REFERENCES

[1] V. N. Vapnik, *Statistical Learning Theory*, 2nd ed. New York: Springer, 11 1999.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer, 1 2016.

[3] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, p. 37–57, Mar. 1985. [Online]. Available: https://doi.org/10.1145/3147.3165

[4] A. J. Ferreira and M. A. T. Figueiredo, "An unsupervised approach to feature discretization and selection," *Pattern Recogn.*, vol. 45, no. 9, p. 3048–3060, Sep. 2012. [Online]. Available: https://doi.org/10.1016/j.patcog.2011.12.008

[5] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Proc. of the 18th Int. Conf. on Neural Information Processing Systems*, ser. NIPS'05. Cambridge, MA, USA: MIT Press, 2005, p. 507–514.

[6] J. Wang, P. Zhao, S. C. H. Hoi, and R. Jin, "Online feature selection and its applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 698–710, 2014.

[7] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, Dec. 2017.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. New York: Springer, 8 2006.

[9] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, 05 2018.

[10] A. U. Haq, D. Zhang, H. Peng, and S. U. Rahman, "Combining multiple feature-ranking techniques and clustering of variables for feature selection," *IEEE Access*, vol. 7, pp. 151 482–151 492, 2019.

[11] M. S. Mahmud and X. Fu, "Unsupervised classification of high-dimension and low-sample data with variational autoencoder based dimensionality reduction," in *2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2019, pp. 498–503.

[12] V. A. Fursov, A. V. Gavrilov, and A. P. Kotov, "Prediction of estimates' accuracy for linear regression with a small sample size," in *2018 41st Int. Conf. on Telecomm. and Signal Processing (TSP)*, 2018, pp. 1–7.

[13] G. Nikolov, M. Kuhn, A. McGibney, and B. Wenning, "Asr - adaptive similarity-based regressor for uplink data rate estimation in mobile networks," *IEEE J. on Sel. Areas in Communications*, pp. 1–1, 2020.

[14] D. M. J. Tax and P. Laskov, "Online svm learning: from classification to data description and back," in *2003 IEEE XIII Workshop on Neural Networks for Signal Processing (IEEE Cat. No.03TH8718)*, 2003, pp. 499–508.

[15] A. G. Joseph and S. Bhatnagar, "An online prediction algorithm for reinforcement learning with linear function approximation using cross entropy method," *Mach. Learn.*, vol. 107, no. 8–10, p. 1385–1429, Sep. 2018. [Online]. Available: https://doi.org/10.1007/s10994-018-5727-z

[16] X. Wang, F. S. Samani, and R. Stadler, "Online feature selection for rapid, low-overhead learning in networked systems," in *IEEE 16th International Conference on Network and Service Management (CNSM 2020)*, ser. CNSM 2020, 2020.

[17] J. Li. (2018) scikit-feature. [Online]. Available: https://github.com/jundongl/scikit-feature

[18] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *J. Network Syst. Manage.*, vol. 25, no. 4, pp. 672–698, 2017. [Online]. Available: https://doi.org/10.1007/s10922-017-9426-z

[19] R. Pasquini. (2017) traces-jnsm-2017. [Online]. Available: https://github.com/rafaelpasquini/traces-jnsm-2017

[20] F. S. Samani, H. Zhang, and R. Stadler, "Efficient learning on high-dimensional operational data," in *2019 15th International Conference on Network and Service Management (CNSM)*, 2019, pp. 1–9.

[21] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. E. Long, "Managing flash crowds on the internet," in *11th IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer Telecomm. Systems. MASCOTS 2003.*, 2003, pp. 246–249.

[22] Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.