Online Feature Selection for Efficient Learning in Networked Systems

Xiaoxuan Wang, *Graduate Student Member, IEEE*, and Rolf Stadler, *Senior Member, IEEE*

***Abstract*— Current AI/ML methods for data-driven engineering use models that are mostly trained offline. Such models can be expensive to build in terms of communication and computing costs, and they rely on data that is collected over extended periods of time. Further, they become out-of-date when changes in the system occur. To address these challenges, we investigate online learning techniques that automatically reduce the number of available data sources for model training. We present an online algorithm called Online Stable Feature Set Algorithm (OSFS), which selects a small feature set from a large number of available data sources after receiving a small number of measurements. The algorithm is initialized with a feature ranking algorithm, a feature set stability metric, and a search policy. We perform an extensive experimental evaluation of this algorithm using traces from an in-house testbed and from two external datasets. We find that OSFS achieves a massive reduction in the size of the feature set by 1-3 orders of magnitude on all investigated datasets. Most importantly, we find that the accuracy of a predictor trained on a OSFS-produced feature set is somewhat better than when the predictor is trained on a feature set obtained through offline feature selection. OSFS is thus shown to be effective as an online feature selection algorithm and robust regarding the sample interval used for feature selection. We also find that, when concept drift in the data underlying the model occurs, its effect can be mitigated by recomputing the feature set and retraining the prediction model.**

***Index Terms*—Data-driven engineering, machine learning (ML), dimensionality reduction, online learning, online feature selection.**

I. Introduction

DATA-DRIVEN network and systems engineering is based upon applying AI/ML methods to data collected from an infrastructure in order to build novel functionality and management capabilities. This is achieved through learning tasks that are trained on this data. Examples are KPI prediction and forecasting through regression and anomaly detection through clustering techniques.

The way AI/ML methods are currently applied in data-driven engineering has several drawbacks. For instance, model training is often expensive in terms of communication and computing costs, and it relies on data that must be collected over extended periods of time. Second, since training is performed offline, models generally become out-of-date and the performance of learning tasks degrades after changes in the system or in the environment. To address these challenges, we advocate investigating online learning techniques and performing model recomputation after detecting a change in the data distribution underlying the model.

Creating a model for a learning task, for example a model for KPI prediction, includes (1) the extraction of metrics from devices in a network or an IT system, (2) the transfer of this data to a processing point, (3) the preprocessing of this data (removing outliers, etc.), and (4) the training of the model. The first two steps are part of the monitoring process. Each of these four steps incurs computing and/or communication costs that increase at least linearly with the number of sources from which data are extracted and with the number of observations that are used for training the model.

This paper focuses on automatically reducing the number of data sources involved in creating an effective model with the goal of significantly reducing monitoring cost and model training cost. We propose a novel online source-selection method that requires only a small number of measurements to significantly reduce the number of data sources needed for training models that are effective for learning tasks.

Using the terminology of machine learning, we call a (one-dimensional, scalar) data source also *a feature*, and we refer to measurements taken from a set of data sources at a specific time as *a sample*.

Our approach consists of (1) ranking the available data sources using (unsupervised) feature selection algorithms and (2) identifying *stable feature sets* that include only the top k features. We call a feature set stable if it remains sufficiently similar when additional samples are considered.

We present an online algorithm, which we call Online Stable Feature Set Algorithm (OSFS). It selects a small feature set from a large number of available data sources using a small number of measurements, which allows for efficient and effective learning. The algorithm is initialized with a feature ranking algorithm, a feature set stability metric, and a search policy.

We perform an extensive experimental evaluation of this algorithm using traces from an in-house testbed (the KTH dataset), the FedCSIS Challenge dataset [1], and the Poli Torino dataset [2]. We find that OSFS achieves a massive reduction in the size of the feature set by 1-3 orders of

Manuscript received 7 December 2021; revised 12 April 2022; accepted 24 May 2022. Date of publication 8 June 2022; date of current version 12 October 2022. This research has been partially supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects AutoDC and ANIARA, as well as by Digital Futures through project Democritus. The associate editor coordinating the review of this article and approving it for publication was N. Zincir-Heywood. *(Corresponding author: Xiaoxuan Wang.)*

The authors are with the Department of Computer Science, Digital Futures, KTH Royal Institute of Technology, 100 44 Stockholm, Sweden (e-mail: xiaoxuan@kth.se; stadler@kth.se).

Digital Object Identifier 10.1109/TNSM.2022.3180936

magnitude on all investigated datasets. The number of samples needed to compute the feature sets averages around 400, which means that OSFS produces a stable feature set within an average of 7 minutes on the KTH testbed, where monitoring data is collected every second. Most importantly, we find that the performance of a predictor trained on a OSFS-produced feature set is similar to the one that uses offline feature selection. OSFS is thus shown to be effective as an online feature selection algorithm and robust regarding the sample interval used for feature selection.

We also find that, if concept drift occurs, its effect can be effectively mitigated by recomputing the feature set and retraining the prediction model.

Our results suggest that many data-driven functions in networked systems can be trained rapidly and with low overhead if they are retrained after a change is detected. In particular, they do not require a lengthy monitoring phase and expensive offline training before prediction can begin.

This paper is a significant extension of earlier work published in CNSM 2020 [3]. In this paper, we present OSFS in a new, generic form that allows instantiation with feature ranking algorithm, feature stability metric, and search policy. We introduce and evaluate an additional stability metric which can be used with OSFS. The evaluation in this paper is more in-depth and more extensive. It includes studying the performance of OSFS on two additional dataset, namely, the FedCSIS challenge dataset and the Poli Torino dataset. Also, we study the effect of concept drift on prediction accuracy and how to mitigate the increasing error through feature set recomputation and model retraining. Finally, we revise and extend the related work section.

The rest of the paper is organized as follows. Section II formulates the problem we address in the paper. Section III describes the feature selection methods we use to obtain ranked feature lists. Section IV introduces the concept of the stable feature set. Section V presents our online feature selection algorithm OSFS. Section VI discusses change detection and model recomputation. Section VII details our testbed, the traces we generate, the FedCSIS challenge dataset, and the Poli Torino dataset. Sections VIII and IX contain the evaluation results. Section X surveys related work. Finally, Section XI presents the conclusions and future work.

II. Problem Formulation and Approach

We consider a monitoring infrastructure that collects readings from a set $\boldsymbol{F}$ of n distributed data sources (or features). Each feature has a one-dimensional, numerical value that changes over time. We collect readings at discrete times $t = 1, 2, 3, \ldots$ and store them in sample vectors $\boldsymbol{X}_t \in \mathbb{R}^n$. Our objective is to identify a subset $\boldsymbol{F}_k \subset \boldsymbol{F}$ with $k \ll n$ features using the samples $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_{t_k}$. Second, we consider a learning task, like KPI prediction or anomaly detection, whose model is trained using the samples $\boldsymbol{X}_t \in \mathbb{R}^k$ with the features from $\boldsymbol{F}_k$.

Figure 1 illustrates the process how the subset $\boldsymbol{F}_k$ is created. Central to the process is the algorithm OSFS which will be introduced in Section V.

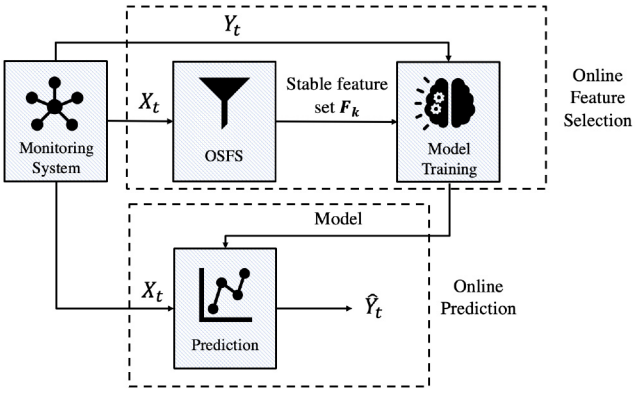


Fig. 1. Online learning: feature selection and prediction. OSFS reads a stream of samples ($\boldsymbol{X}_t$) and extracts a stable feature set $\boldsymbol{F}_k$.

In order to keep the monitoring overhead low for collecting the samples and the computational overhead low for training the model associated with the learning task, the values for k and t_k should be small. Note that k indicates the number of data sources that need to be monitored to train the model and t_k refers to the number of measurements that are needed to compute $\boldsymbol{F}_k$. Assuming periodic readings, t_k further indicates the time it takes until the feature set $\boldsymbol{F}_k$ is available.

Our goal thus is to select k and t_k as small as possible, while enabling the models trained using $\boldsymbol{F}_k$ to be effective.

The task of selecting a subset of features from a larger set is called *feature selection* in machine learning and data mining and is a well-studied topic area (See Section X). We are specifically interested in unsupervised feature selection methods, whereby the values of the target are not known during the feature selection process, i.e., the process to compute $\boldsymbol{F}_k$. This allows us to keep the feature selection process independent from the learning task and will enable different learning tasks in a system to share the same feature subset.

The specific problem we address in this paper is to design an online algorithm that reads a sequence of n-dimensional sample vectors $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots$ one by one, computes k and the feature set $\boldsymbol{F}_k$, and terminates after step t_k. The values for k and t_k must be small, while $\boldsymbol{F}_k$ must be equally effective as the feature set selected by the same ranking algorithm in an offline setting on the complete data stream. We measure the effectiveness of a feature set by computing the prediction error of the predictor trained with the feature set.

In our approach, we choose an unsupervised feature selection method that ranks the n features at every step t of the online algorithm and checks how the top i features ($i = 1, \ldots, n$) change with increasing t. We introduce two metrics to measure the stability of the feature set in Section IV. If the feature set of the top k features has become sufficiently stable, the algorithm terminates and $\boldsymbol{F}_k$, the values for k and t_k are returned (see Section V).

Note that we assume here that the feature set $\boldsymbol{F}$ is fixed. In a real system that runs over some time, changes to the physical configuration or the virtualization layer occur, which result in changes to the set of available measurement points, i.e., the feature set. In such a case, the online algorithm must

TABLE I
TABLE OF NOTATION

$\boldsymbol{X}$	data set
n	number of features in $\boldsymbol{X}$
m	number of samples in $\boldsymbol{X}$
$\boldsymbol{X}_{i,:}$ or $\boldsymbol{X}_i (i = 1, ..., m)$	i-th row or i-th sample of $\boldsymbol{X}$
$\boldsymbol{X}_{:,j}$ $(j = 1, ..., n)$	j-th column or j-th feature vector of $\boldsymbol{X}$
$X_{i,j}$	element of the i-th row and the j-th column of $\boldsymbol{X}$
k	number of selected features
t_k	number of samples used for feature selection
$\boldsymbol{F}$	set of all available features
f_j $(j = 1, ..., n)$	single feature in $\boldsymbol{F}$
$\boldsymbol{F}_k$	subset with k selected features
$\boldsymbol{F}_{k,t}$	the top k features computed on t samples

be restarted. Note also that we do not investigate how many samples are needed to train an accurate model for the learning task. This is left for future work.

The solution approach we develop in Sections IV, V, VI uses concepts from statistical learning [4] and is based on assumptions about the system under consideration. Fundamentally, we consider a distributed system whose state evolves over time. We assume that time is discrete and global. We monitor the system at a given frequency, which means that we receive a sample with statistics of the system state at constant time intervals. We model the sample as a vector with fixed dimensionality.

We further assume that the system state evolves as a sequence of epochs. During each epoch the monitoring samples are drawn from a fixed distribution. Epochs can vary in duration. At each time step the current epoch can end and a new epoch can start, in which monitoring samples are drawn from a different, fixed distribution. Changes of epochs are "rare", and epochs generally continue for a "long time". With changes of epochs we model events that modify the behavior of a networked system, such as a change in load pattern, a failure of a system component, a change in system configuration, etc. During an epoch the system is stationary.

The above assumptions are not strictly valid for the systems and scenarios we study, but our experience suggests that they often lead to practical and sufficiently accurate results. In fact, many works that apply machine learning techniques to engineering networked systems make these or similar assumptions explicitly or (oftentimes) implicitly, for example [5], [6], [7], [8].

Table I shows the notation we use in the paper. The available data for computing the feature set $\boldsymbol{F}_k$ is presented as a design matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, whose n columns represent the feature vectors and m rows represent the samples in the dataset. Since we assume that the samples arrive in sequence one-by-one, m is increasing over time and can be interpreted as a time index.

III. Creating Ranked Feature Lists

In this section, we describe two algorithms (ARR, LS) that produce a ranked feature list from a list of samples. They are based on unsupervised feature selection methods from the literature. We will evaluate the suitability of these algorithms for our online feature selection method OSFS in Section VIII. We present these two algorithms as offline algorithms since they appear in this form in the literature. In a practical implementation of OSFS, one would use an online feature ranking algorithm, for instance, an online version of ARR or LS.

Algorithm 1: Adapted Relevance Redundancy Feature Selection (ARR)

Input: Data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$
Output: Ranked feature list $\boldsymbol{F'}$
1 $\boldsymbol{F'}$=[];
2 **for** $i = 1; i \leq n; i++$ **do**
3 $\quad relevance_i = \sum_{j=1}^{m} \left| X_{j,i} - \overline{\boldsymbol{X}_{:,i}} \right|$;
4 $\quad sim\_sum_i = 0$;
5 $\quad$ **for** $l = 1; l \leq n; l++$ **do**
6 $\quad\quad sim\_sum_i += cosim(\boldsymbol{X}_{:,i}, \boldsymbol{X}_{:,l})$;
7 $\quad score_i = \frac{relevance_i}{sim\_sum_i}$;
8 $\boldsymbol{F'}$ = a list of all features sorted by $score_i$ in descending order;
9 return $\boldsymbol{F'}$;

The first algorithm is Adapted Relevance Redundancy Feature Selection (ARR). It is based on the Relevance Redundancy Feature Selection (RRFS) method [9], which we adapted to compute a ranked feature list. ARR uses two criteria in assessing the rank of a feature: (a) relevance, which relates to the distance of a feature vector to the mean of all feature vectors, and (b) redundancy, which relates to the cosine similarity between a feature vector and the vectors of all other features. High relevance and low similarity result in a high score. The pseudocode of ARR is given in Algorithm 1. First, the relevance of a feature is computed as the mean absolute difference of its feature vector from the mean (line 3). Then, the redundancy of a feature is computed as the sum of the cosine similarity of its feature vector and each feature vector of the dataset (lines 5-6). The score for ranking a feature is the relevance value divided by the redundancy value (line 7). The computational complexity of ARR is $O(n^2 m)$. Recall that the cosine similarity between two features vectors $\boldsymbol{X}_{:,a}$ and $\boldsymbol{X}_{:,b}$ is calculated as:

$$cosim(\boldsymbol{X}_{:,a}, \boldsymbol{X}_{:,b}) = \left| \frac{\sum_{i=1}^{m} (X_{i,a} X_{i,b})}{\left(\sqrt{\sum_{i=1}^{m} {X_{i,a}}^2} \right) \left(\sqrt{\sum_{i=1}^{m} {X_{i,b}}^2} \right)} \right| \quad (1)$$

The second algorithm is Laplacian Score (LS) [10]. It follows the so-called filter method which examines intrinsic properties of the data to evaluate the features. LS ranks those features high that preserve locality with respect to a neighborhood graph. Algorithm 2 shows the pseudocode of LS. Input parameters for this algorithm are the design matrix and the number of local neighbors K. First, using the m sample vectors as the nodes of the graph, a neighborhood graph is constructed with the K nearest neighbors of each node as the links of the graph (line 1). Then, the graph connectivity and the distance between node pairs are used to compute the weight matrix (line 2). The graph Laplacian matrix is computed in line 4. The Laplacian score of all n features is obtained in the for loop (lines 5-7). A low score for a feature signifies

Algorithm 2: Laplacian Score (LS)

Input: Data matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, $K \in \mathbb{N}$ nearest neighbors
Output: Ranked feature list $\boldsymbol{F'}$

1 Construct graph $\boldsymbol{G}$ of K nearest neighbors from nodes $\boldsymbol{X}_{i,:}, i = 1, \ldots, m$;
2 Compute weight matrix $\boldsymbol{S} \in \mathbb{R}^{m \times m}$ from $\boldsymbol{G}$

$$S_{ij} = \begin{cases} e^{-\|\boldsymbol{X}_{i,:} - \boldsymbol{X}_{j,:}\|^2} & \textit{if nodes } i \textit{ and } j \textit{ are connected,} \\ 0 & \textit{otherwise} \end{cases}$$

3 $\boldsymbol{D} = diag(\boldsymbol{S1})$ where $\boldsymbol{1} = [1, \ldots, 1]^T$, $\boldsymbol{D} \in \mathbb{R}^{m \times m}$;
4 $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{S}$;
5 **for** $i = 1; i \leq n; i++$ **do**
6 $\quad \boldsymbol{V}_i = \boldsymbol{X}_{:,i} - \frac{\boldsymbol{X}_{:,i}^T \boldsymbol{D1}}{\boldsymbol{1}^T \boldsymbol{D1}} \boldsymbol{1}$ where $\boldsymbol{V}_i \in \mathbb{R}^{m \times 1}$;
7 $\quad lscore_i = \frac{\boldsymbol{V}_i^T \boldsymbol{L} \boldsymbol{V}_i}{\boldsymbol{V}_i^T \boldsymbol{D} \boldsymbol{V}_i}$;
8 $\boldsymbol{F'}$ = a list of all features sorted by $lscore_i$ in ascending order;
9 return $\boldsymbol{F'}$;

high locality preservation (see [10] for justification), and the features are ranked according to increasing score (line 8). The computational complexity of the LS algorithm is $O(nm^2)$.

In the evaluations reported in Sections VIII and IX of this paper, we choose K in relation to the value of m: $K = 2$ for $0 < m \leq 16$; $K = 5$ for $16 < m \leq 128$; $K = 10$ for $m > 128$.

IV. Computing Stable Feature Sets

In Section III we discussed the algorithms ARR and LS, which produce ranked feature lists after reading m samples $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_m$. In an online setting, where samples become available one-by-one at discrete times $t = 1, 2, \ldots$, the value of m can be interpreted as time. Let's assume we run a feature ranking algorithm at time t_1 and compute the set $\boldsymbol{F}_{k,t_1}$ with the top k features. This set will generally be different from the set $\boldsymbol{F}_{k,t_2}$ produced by the same algorithm at a later time t_2 for statistical reasons. Using the standard assumption from statistical learning that samples are drawn from static distributions [4], we expect that the sequence $\boldsymbol{F}_{k,t}$ converges to a set $\boldsymbol{F}_k^*$ with increasing time t. Our objective for this section is to identify heuristic criteria that determine at which time the sequence $\boldsymbol{F}_{k,t}, t = 1, 2, 3, \ldots$, has sufficiently converged or, as we also say, $\boldsymbol{F}_{k,t}$ has become "stable".

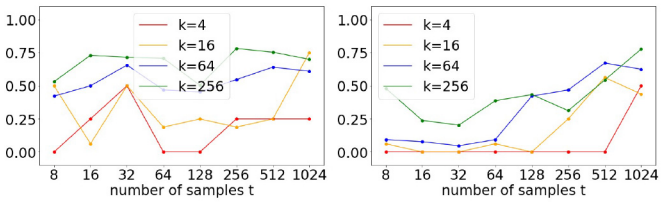
We define two criteria for feature set stability, which use frequency-based stability methods [11]. The first is based on the similarity of two feature sets, the second on the variance of feature frequencies in feature sets.

A. A Stability Condition Based on Set Similarity

Given two feature sets $\boldsymbol{F}_{k,t_1}$ and $\boldsymbol{F}_{k,t_2}$ ($k \in \mathbb{N}$), we define the similarity between them as the fraction of joint features:

$$sim(\boldsymbol{F}_{k,t_1}, \boldsymbol{F}_{k,t_2}) := |\boldsymbol{F}_{k,t_1} \cap \boldsymbol{F}_{k,t_2}|/k \tag{2}$$

The values of the metric *sim* lie between 0 and 1. 0 means no similarity, i.e., $\boldsymbol{F}_{k,t_1}$ and $\boldsymbol{F}_{k,t_2}$ have no common features, while 1 means the maximum similarity, i.e., $\boldsymbol{F}_{k,t_1}$ and $\boldsymbol{F}_{k,t_2}$ are identical.



(a) $sim(\boldsymbol{F}_{k,t/2}, \boldsymbol{F}_{k,t})$ vs t in ARR (b) $sim(\boldsymbol{F}_{k,t/2}, \boldsymbol{F}_{k,t})$ vs t in LS

Fig. 2. The similarity of consecutive feature sets over time using ARR and LS with samples from the KV flash-crowd dataset and start time $t = 1$.

When we study the evolution of $sim(\boldsymbol{F}_{k,t}, \boldsymbol{F}_{k,2t})$ using the traces described in Section VII, we observe that the similarity metric tends to grow with the increase of k and t. Also, this effect becomes stronger the larger k and t are. Figure 2 shows an example with data from the KTH testbed running a KV service.

We introduce the first heuristic notion of a *stable feature set*:

$$\begin{aligned} &\boldsymbol{F}_{k,t} \textit{ is stable, iff } sim(\boldsymbol{F}_{k,t}, \boldsymbol{F}_{k,2t}) > \eta \\ &\quad \textit{and } sim(\boldsymbol{F}_{k,t}, \boldsymbol{F}_{k,2t}) > sim(\boldsymbol{F}_{k,2t}, \boldsymbol{F}_{k,4t}) \end{aligned} \tag{3}$$

Based on our experience with operational data, we set $\eta = 0.5$ for this work. This means that $\boldsymbol{F}_{k,t}$ must share at least half of its features with $\boldsymbol{F}_{k,2t}$, which is computed with double the number of samples. The second condition in equation (3) is met if the sequence of similarity values reaches a local maximum. In Figure 2 (a) $\boldsymbol{F}_{16,1024}$ and in Figure 2 (b) $\boldsymbol{F}_{64,512}$ are examples of stable feature sets.

B. A Stability Condition Based on Feature Frequency

In [12] the authors propose a metric for feature set stability based on statistical concepts. They measure the selection frequency of each feature and estimate its variance for a sequence of consecutive feature sets. They model the frequency of a feature using a Bernoulli process. The stability criterion we introduce requires that the variance becomes sufficiently small.

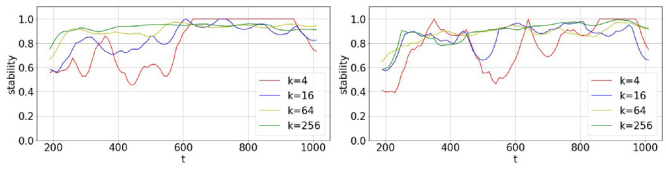
Given a feature set $\boldsymbol{F}_{k,t}$ and the original feature set $\boldsymbol{F} = \{f_1, f_2, \ldots, f_n\}$ ($k \in \mathbb{N}$, $k \leq n$), we define the feature representation vector $\boldsymbol{A}_{k,t} \in \{0, 1\}^n$ as follows

$$\boldsymbol{A}_{k,t}[j] = \begin{cases} 1, & \textit{if } f_j \in \boldsymbol{F}_{k,t} \\ 0, & \textit{otherwise} \end{cases} \tag{4}$$

We consider r consecutive vectors and set the matrix $\boldsymbol{Z}_{k,t,r} := [\boldsymbol{A}_{k,t}, \ldots, \boldsymbol{A}_{k,t+r-1}]^T$. We define the stability of $\boldsymbol{F}_{k,t}$ for r consecutive feature sets as:

$$stab(\boldsymbol{Z}_{k,t,r}) := 1 - \frac{\frac{1}{n} \sum_{j=1}^{n} s_j^2}{\frac{k}{n}\left(1 - \frac{k}{n}\right)} \tag{5}$$

To understand the right side of equation (5), we need to know that p_j is the parameter of the Bernoulli process that models the occurrence of feature f_j. If we denote the empirical feature frequency with $\widehat{p_j}$, we can write $\widehat{p_j} = \frac{1}{r} \sum_{i=1}^{r} z_{ij}$ (z_{ij} is an



(a) $stab(\mathbf{Z}_{k,t,r})$ vs t in ARR (b) $stab(\mathbf{Z}_{k,t,r})$ vs t in LS

Fig. 3. The feature set stability of current feature sets over time using ARR and LS with samples from the KV flash-crowd dataset and start time $t = 1$.

element of $\boldsymbol{Z}_{k,t,r}$). The empirical sample variance is given by $s_j^2 = \frac{r}{r-1}\widehat{p_j}(1-\widehat{p_j})$. We know from theory that s_j^2 is an unbiased estimator of the variance of the feature frequency.

The stability metric $stab(\boldsymbol{Z}_{k,t,r})$ has values in $[-\frac{1}{r-1}, 1]$ [11]. If no feature shows any variance, i.e., each feature is either always selected or always left out, the value is 1. Otherwise, the value is less than 1.

Similar to $sim(\boldsymbol{F}_{k,t_1}, \boldsymbol{F}_{k,t_2})$ we observe using the data from our testbed that $stab(\boldsymbol{Z}_{k,t,r})$ tends to grow with the increase of k and t. This effect becomes stronger the larger k and t are. Figure 3 shows an example with data from the KTH testbed running a KV service.

We introduce the second heuristic notion of a *stable feature set*:

$$\boldsymbol{F}_{k,t} \textit{ is stable for } r, \textit{ iff } stab(\boldsymbol{Z}_{k,t,r}) > \eta \textit{ for } w \textit{ consecutive timesteps} \tag{6}$$

Based on our experience with operational data, we set $r = 10$, $w = 10$ and $\eta = 0.9$ for this work. This means that when 10 consecutive stability values are all larger than 0.9, then the feature set $\boldsymbol{F}_{k,t}$ is stable. In Figure 3 (a) $\boldsymbol{F}_{4,610}$ and in Figure 3 (b) $\boldsymbol{F}_{4,840}$ are examples of stable feature sets.

C. Feature Clustering

We also investigated a different approach to determine whether a feature set $\boldsymbol{F}_{k,t}$ is stable. This approach includes clustering the features after t samples have been obtained. The two conditions we formulated to identify a stable feature set were then applied on the feature clusters instead of on individual features. The motivation for this approach stems from the idea that features in the same cluster have the same information content for constructing the prediction model. Specifically, we have used DBSCAN [13], which is based on the cosine similarity metric to cluster the features. However, the evaluation showed that the accuracy of the obtained predictors has been only marginally better than when clustering was not used. For this reason, we have not included the detailed results in this paper.

V. OSFS: Online Feature Selection With Low Overhead

In this section, we introduce the Online Stable Feature Set Algorithm (OSFS), which reads a stream of samples $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3, \ldots$, and returns the number of features k, the number of samples t_k needed to determine k, and the stable feature set $\boldsymbol{F}_{k,t_k}$.

Algorithm 3: Online Stable Feature Set (OSFS)

Input: Sample sequence $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$,...;
Feature ranking algorithm A;
Metric and condition for stable feature set C;
Search space S for (k, t);
Search policy P to traverse S.
Output: Stable feature subset $\boldsymbol{F}_{k,t}$, k, t.

```
1 Initialize (k, t);
2 while True do
3     Compute F_{k,t} using A on X_1,..., X_t;
4     if F_{k,t} is stable or S is exhausted then
5         return F_{k,t}, k, t;
6     else
7         choose next (k, t) following P;
```

Recall that the purpose of OSFS is to select a subset of available data sources (i.e., features), in order to reduce the monitoring costs and the training overhead for learning. To keep costs and overhead low, we want k and t_k to be small while still allowing for effective learning and prediction.

The standard use case for OSFS plays out as follows. We start monitoring the values of n features at time $t = 1$ and collect a sequence of samples with index $t = 1, 2, \ldots,$ We use the collected samples to find values for $k << n$ and t_k so that the first t_k samples allow us to compute a stable feature set $\boldsymbol{F}_{k,t_k}$. At time t_k+1, we reduce the number of data sources from n to k and continue monitoring only sources from the set $\boldsymbol{F}_{k,t_k}$.

We first present OSFS in a generic form, which allows for a range of instantiations and configurations. Algorithm 3 shows the options for configuration. First, we choose a feature ranking algorithm. Section III contains two examples of such algorithms, namely, ARR and LS. Second, we choose a condition for a feature set $\boldsymbol{F}_{k,t}$ to be stable. Section IV presents two possibilities of such conditions, namely, (3) and (6). Third, we choose the search space S, i.e., the space of possible values for (k, t). An example of S is a two-dimensional grid where t has values [16, 64, 256, 1024] and k has the values [4, 16, 64, 256]. A search policy P is an algorithm to traverse S. We have experimented with different search policies. For instance, policy *k–small* starts with the smallest k, traverses all possible t values, continues to the next larger k, traverses all possible t values, etc. This way the grid is traversed from bottom to top (see Figure 4 red path). Similarly, the policy *t–small* starts with the smallest t, traverses all possible k values, continues to the next larger t, traverses all possible k values, etc. This way the grid is traversed from left to right (see Figure 4 blue path). Many other policies are possible, for instance, a policy based on random walk.

Once the execution of instantiated OSFS algorithm starts, initial values for k and t are selected (line 1). It then enters a loop where, during each iteration, one point in the grid is evaluated (line 2-7). First, the feature set $\boldsymbol{F}_{k,t}$ is computed using the ranking algorithm A and available samples $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_t$. Then, the stability condition is evaluated for $\boldsymbol{F}_{k,t}$. If $\boldsymbol{F}_{k,t}$

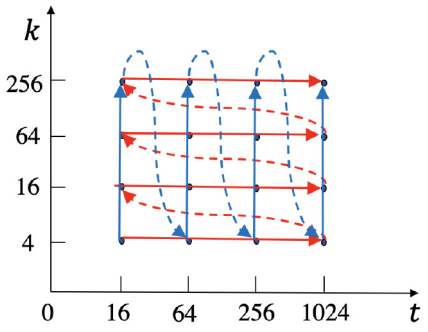


Fig. 4. Search policy on one search space. The red path follows the policy *k–small* and the blue path follows the policy *t–small*.

is stable or the grid has been completely searched, the algorithm terminates and returns $\boldsymbol{F}_{k,t}$, k, t. Otherwise, the loop continues with the next (k, t) pair.

When discussing the evaluations of OSFS in Sections VIII and IX, we present results obtained with feature ranking algorithms ARR and LS and with both feature set stability conditions. Regarding the search policy, most of the results in this paper are obtained with *k–small*, which gives priority to a feature set of small size at the possible expense of larger running time of OSFS. We also perform evaluations with policy *t–small*, which prioritises obtaining a stable feature set within a short amount of time at the possible expense of a larger feature set.

Algorithm 4 is an example of an instantiated OSFS algorithm, which we use for evaluation. The algorithm takes as input the sequence of arriving samples $\boldsymbol{X}_t$ and is initialized with the feature ranking algorithm ARR. ARR is used in the function *subset*(), which takes as input k and the first t samples and returns a set with the top k features ranked by ARR.

It has two main loops: an outer loop (lines 3-23) that iterates over a subspace of k and an inner loop (lines 9-22) that iterates over a subspace of t. The index values of k and t increase exponentially to enable the exploration of a large space with a small number of evaluations. The algorithm performs a grid search in the space of tuples (k, t) with the termination condition A: $sim_{kt} < sim_{k12}$ *and* $sim_{k12} > \eta$ (line 15) or B: $sim_{kt} > \eta$ *and* $t == 1024$ (line 17). In case the conditions A or B are never met, the algorithm terminates after the search on the grid [4 − 256]*[8 − 1024] has been completed. The key termination condition A expresses the case where (1) the similarity of two consecutive feature sets is above the threshold η and (2) the similarity declines when the subsequent feature set is considered.

The algorithm reads at least 32 samples in order to ensure statistical viability and terminates after at most 1024 samples. Since the outer loop is indexed by increasing k, it favors a smaller k at the possible expense of a larger t. This means that we prefer reducing the monitoring overhead over reducing the time to compute $\boldsymbol{F}_{k,t}$.

Let us clarify the time window over which the feature set $F_{k,t}$ is computed in Algorithm 4. The algorithm starts at time step $t = 1$. At every time step t a new sample X_t is read. The stability test of the feature set $F_{k,t}$ is performed on the

Algorithm 4: OSFS-ARR-*sim*-k − *small*

Input: Sample sequence $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, $\boldsymbol{X}_3$,...;
Feature ranking algorithm ARR;
Metric *sim* and condition (3) for stable feature set;
Search space $[4 - 256] * [8 - 1024]$ for (k, t);
Search policy $k - small$.
Output: Feature subset $\boldsymbol{F}_{k,t_k}$, k, t_k
$subset(k, t, ARR)$ returns top k features computed with ARR using samples with index $1, \ldots, t$.

1 $\eta = 0.5$ (threshold for stable feature set);
2 $read = false$;
3 **for** *k in [4,16,64,256]* **do**
4 **if** $not\ read$ **then**
5 read and store $\boldsymbol{X}_t, t = 1, \ldots, 16$
6 $\boldsymbol{F}_{k1} = subset(k, 8, ARR)$;
7 $\boldsymbol{F}_{k2} = subset(k, 16, ARR)$;
8 $sim_{k12} = sim(\boldsymbol{F}_{k1}, \boldsymbol{F}_{k2})$;
9 **for** $t = 17, \ldots, 1024$ **do**
10 **if** $not\ read$ **then**
11 read and store $\boldsymbol{X}_t$
12 **if** *t in [32,64,128,256,512,1024]* **then**
13 $\boldsymbol{F}_{kt} = subset(k, t, ARR)$;
14 $sim_{kt} = sim(\boldsymbol{F}_{k2}, \boldsymbol{F}_{kt})$;
15 **if** $sim_{kt} < sim_{k12}\ and\ sim_{k12} > \eta$ **then**
16 return $\boldsymbol{F}_{k1}, k, t/4$;
17 **else if** $sim_{kt} > \eta\ and\ t == 1024$ **then**
18 return $\boldsymbol{F}_{k2}, k, t/2$;
19 **else**
20 $\boldsymbol{F}_{k1} = \boldsymbol{F}_{k2}$;
21 $\boldsymbol{F}_{k2} = \boldsymbol{F}_{kt}$;
22 $sim_{k12} = sim_{kt}$;
23 $read = true$
24 return $\boldsymbol{F}_{k2}, 256, 1024$

samples X_i with time index $i = 1, \ldots, t$. This takes place in line 15. The time window in which the samples are evaluated is thus expanding with t.

Note that in Algorithm 4 we choose a specific range for the number of samples and the number of features in the search space of the algorithm. These are configuration parameters and can be changed. When experimenting with datasets from three different sources described in this paper, we found that the chosen values enable good performance of OSFS.

VI. Recomputing Feature Set After Concept Drift

The distribution from which the samples are drawn for training prediction models can evolve over time, and this phenomenon is referred to as *concept drift*, which is a well-studied topic in data analysis [14]. When concept drift occurs, the prediction model must generally be recomputed in order to maintain prediction accuracy. In addition, the feature set from

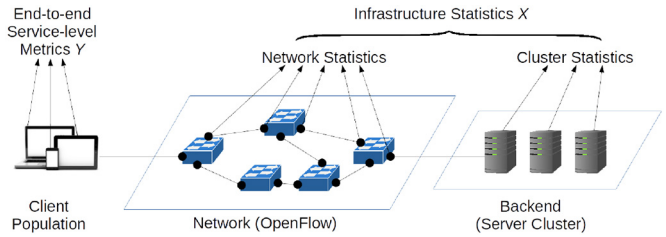


Fig. 5. The testbed at KTH, providing the infrastructure for experiments. In various scenarios we predict end-to-end service-level metrics from low-level infrastructure measurements [19].

which the prediction model is computed should be adapted as well for the same reason.

In the context of network systems and their operation, concept drift can occur when a system is reconfigured or when the resource allocation policy for services or applications is changed. This means that, when concept drift occurs, OSFS must be run again to identify the updated set of features for recomputing the prediction model.

In this work, we use a state-of-the-art concept drift detection algorithm called STUDD (Student-Teacher Method for Unsupervised Concept Drift Detection) [15] on our data traces. STUDD detects concept drift in an unsupervised manner by using a student-teacher paradigm [16]. The basic idea of this paradigm is that in addition to the primary predictor, which is called *teacher*, a second predictor, which is called *student*, is trained. The student model is learned on the same input dataset $\boldsymbol{X}$ as the teacher, but the target set $\boldsymbol{Y}$ is replaced with the values predicted by the teacher. During the prediction phase, the discrepancy between the teacher's prediction and the student's prediction for the same target values is monitored. The Page-Hinkley test [17] is applied to detect changes in the time series of the discrepancy values. Such a change indicates a concept drift event.

We use a random forest regressor from the *scikit-learn* library [18] to compute the teacher as well as the student models.

In Section IX we study the effect of recomputing the stable feature set on the prediction accuracy using the KTH dataset. Specifically, we compare the accuracy values for scenarios where the stable feature set is recomputed after change detection, with scenarios where the stable feature set is computed only at the beginning of the trace.

VII. Traces for Evaluation

A. KTH Dataset

Figure 5 outlines our laboratory testbed at KTH. It includes a server cluster, an emulated OpenFlow network, and a set of clients. A more detailed description of the KTH testbed setup and the services running on it is given in [19].

The online feature selection method proposed in this paper has been evaluated using data from four experiments. Two of them involve running a Video-on-Demand (VoD) service and two a Key Value store (KV) service.

1) *VoD periodic*: In this experiment, we run the VoD service under a periodic load pattern on the testbed. Data is collected every second over a period of 50'000 seconds. After cleaning the dataset, it contains 50'000 samples and 1'296 features. More details about the experiment are given in [20].
2) *VoD flash-crowd*: This experiment relies on the same setup as VoD periodic, except that the testbed is loaded with a flash-crowd pattern. After cleaning the dataset, 50'000 samples and 1'255 features remain. More details about the experiment are given in [20].
3) *KV periodic*: In this experiment, we run the KV service under a periodic load pattern on the testbed. Measurements are collected every second over a period of 28'962 seconds. The dataset contains 10'374 features collected from the server cluster and 176 features from the network switches. After cleaning the data set, 1'751 features remain. More details about the experiment are given in [19].
4) *KV flash-crowd*: This experiment relies on the same setup as KV periodic, except that the testbed is loaded with a flash-crowd pattern. After cleaning the dataset, 19'444 samples and 1'723 features remain. More details about the experiment are given in [19].

TABLE II
Datasets From KTH Testbed

Dataset	Number of features	Number of samples	Target
KV flash crowd	1'723	19'444	$'ReadsAvg'$
KV periodic	1'751	28'962	$'ReadsAvg'$
VoD flash crowd	1'255	50'000	$'DispFrames'$
VoD periodic	1'296	50'000	$'DispFrames'$

We describe the metrics we collect on the testbed, namely, the input feature sets $\boldsymbol{X}_{cluster}$ and $\boldsymbol{X}_{port}$ — the union of which we refer to as $\boldsymbol{X}$. The $\boldsymbol{X}_{cluster}$ *feature set* is extracted from the kernel of the Linux operating system that runs on the servers executing the services. To access the kernel data structures, we use System Activity Report (SAR), a popular open-source Linux library [21]. SAR in turn uses procfs [22] and computes various system statistics over a configurable interval. Examples of such statistics are CPU core utilization, memory utilization, and disk I/O. $\boldsymbol{X}_{cluster}$ includes only numeric features from SAR. The $\boldsymbol{X}_{port}$ *feature set* is extracted from the OpenFlow switches at per-port granularity. It includes statistics from all switches in the network, namely 1) Total number of Bytes Transmitted per port, 2) Total number of Bytes Received per port, 3) Total number of Packets Transmitted per port, and 4) Total number of Packets Received per port.

Table II summarizes the basic information about these four traces, which are available at [23]. For the KV service, the prediction target is '*ReadsAvg*', which represents the average response time per second of read operations. '*DispFrames*' is the prediction target for the VoD service and represents the number of displayed video frames per second on the client side.

B. FedCSIS 2020 Challenge Dataset

The FedCSIS 2020 Challenge dataset [1] is a public dataset provided by EMCA Software, an analytics company.

TABLE III
DATASETS FROM FEDCSIS 2020 CHALLENGE DATASET

Dataset	Number of features	Number of samples	Target
Dataset1	30'438	1'917	$'host0958\_cpu\_1m'$
Dataset2	10'163	1'917	$'host4846\_cpuusagebyproc'$

TABLE IV
INFORMATION OF POLI-TORINO DATASETS

Dataset	Number of features	Number of samples	Number of target classes
Jitsi	95	377'831	5
Webex	95	1'019'678	7

It contains around 2'000 samples collected from December 2019 to February 2020. Each sample aggregates one hour of measurements, and there are 24 samples per day. The data are gathered from 3'728 hosts. Hosts can have different numbers of attributes. For each attribute, such as CPU utilization, several feature columns are included, for example, the mean value or the standard deviation. The detailed information of features can be found in [24]. From the challenge dataset, we construct two smaller datasets for evaluation. In each of these two datasets, all samples share the same features.

- Dataset1 contains samples from hosts that have the following six attributes [*'cpu_1m'*, *'cpu_5m'*, *'cpu_5s'*, *'memoory_free'*, *'memory_total'*, *'memory_used'*]. For every attribute, four feature columns are included with the mean, the standard deviation, the minimum and the maximum value, respectively.
- Dataset2 contains samples from hosts that have the following six attributes [*'cpuusagebyproc'*, *'memoryallocatedbyproc'*, *'in_traffic'*, *'out_traffic'*, *'error_in'*, *'error_out'*]. For every attribute, four feature columns are included with the mean, the standard deviation, the minimum and the maximum value, respectively.

In these two datasets, many feature values are missing. We remove all feature columns where more than 30% of the values are missing. The remaining missing values are estimated through linear interpolation. After these steps, the number of features and the number of samples in each dataset are listed in Table III. For both datasets, we select one of the feature columns as the prediction target. For dataset1 we select $'host0958\_cpu\_1m'$, which denotes the mean 1-minute CPU usage over 1 hour on host0958. For dataset2 we select $'host4846\_cpuusagebyproc'$, which denotes the mean CPU time used by processes over 1 hour on host4846.

C. Poli Torino Dataset

A research group from Politecnico di Torino has published two datasets that contain measurements from two video conferencing applications, Webex and Jitsi [2]. During videoconferencing sessions, transport-level protocol statistics have been captured on a per-second basis. The statistics relate to the length of UDP packets, the difference in length between two consecutive packets, and the delay between two consecutive RTP packets, etc. Table IV summarizes these datasets.

TABLE V
THE NUMBER OF SAMPLES IN EACH TRAFFIC CLASS OF POLI-TORINO DATASETS

	Jitsi	Webex
Audio	153'925	305'076
Screen Sharing (SS)	36'086	49'970
Low Quality Video - 180p (LQ)	104'326	277'205
Medium Quality Video - 360p (MQ)	42'525	73'268
High Quality Video - 720p (HQ)	40'969	78'590
FEC-audio	-	187'814
FEC-video	-	47'755

The datasets were collected to predict the type of real-time communication stream a measurement sample belongs to. Table V lists the number of samples in each class. We can see that the datasets are roughly balanced and there is no obvious minority class. The Webex application uses FEC to mitigate packet losses, which explains the two additional classes in the Webex dataset.

VIII. EVALUATION OF OSFS ON DIFFERENT DATASETS

We evaluate OSFS by asking two main questions:

1) How effective is OSFS in reducing communication and computing overhead as well as the data collection time, i.e., the number of samples needed for feature selection? What are typical values for k and t_k for the three datasets?
2) How effective is OSFS in producing feature sets for training accurate prediction models?

To obtain the evaluation results, we perform two preprocessing steps on the datasets. First, we linearly scale the values of each feature vector to the range [0, 1]. Second, we remove the feature vectors with a variance below 0.0001. For prediction, we use a random forest regressor or classifier. As for hyperparameters, we use 100 trees and the default values for the other parameters.

The prediction targets in the KTH and FedCSIS datasets are numerical values. We express the prediction error for these targets as the Normalized Mean Absolute Error (NMAE), which is computed as follows:

$$NMAE = \frac{1}{\overline{y}}\left(\frac{1}{q}\sum_{i=1}^{q}|y_i - \widehat{y_i}|\right) \tag{7}$$

where $\overline{y}$ is the mean value of the target in the test set, q is the number of samples and $\widehat{y_i}$ is the predicted value.

The prediction target in the Poli Torino dataset is categorical. We choose the prediction error of the trained classifier as the proportion of misclassified instances over the set of instances under consideration, which is computed as follows:

$$Classification\ Error = \frac{1}{q}\sum_{i=1}^{q}\mathbb{1}\{y_i \neq \widehat{y_i}\} \tag{8}$$

Table VI shows the evaluations for the KTH dataset, Table VII for the FedCSIS dataset, and Table VIII for the Poli Torino dataset. All three tables have the same column structure. The first column identifies the dataset, the second column lists the feature ranking algorithm, the third column gives the feature set stability metric, the fourth column gives

TABLE VI
EVALUATION OF OSFS ON KTH DATASETS. THE PREDICTION METHOD IS RANDOM FOREST REGRESSION. THE ERROR IS EXPRESSED IN NMAE (7). EACH ROW SHOWS THE AGGREGATED RESULT FROM AN EVALUATION SCENARIO WITH TEN RUNS. THE VALUES PRINTED IN BOLDFACE RELATE TO PREDICTORS TRAINED USING A FEATURE SET COMPUTED THROUGH ONLINE FEATURE SELECTION

Dataset	Method	Metric	Search	k	t_k	**Pred Error (Online FS)**	Pred Error (Offline FS)
KV flash-crowd	ARR	Similarity	$k-small$	25.6 ± 25.6	251.2 ± 224.0	**0.0280** ± 0.0056	0.0304 ± 0.0049
		Stability	$k-small$	17.2 ± 16.5	744 ± 159.6	**0.0299** ± 0.0054	0.0322 ± 0.0033
			$t-small$	192.4 ± 98.4	472 ± 56.9	**0.0233** ± 0.0060	0.0227 ± 0.0043
	LS	Similarity	$k-small$	10 ± 6	64 ± 72.3	**0.0232** ± 0.0014	0.0255 ± 0.0007
		Stability	$k-small$	6.4 ± 4.8	724 ± 244.7	**0.0289** ± 0.0054	0.0322 ± 0.0037
			$t-small$	186.4 ± 107.4	472 ± 83	**0.0235** ± 0.0060	0.0234 ± 0.0054
	No FS			1'723			0.0184
KV periodic	ARR	Similarity	$k-small$	10 ± 6	249.6 ± 194.4	**0.0325** ± 0.0071	0.0434 ± 0.0006
		Stability	$k-small$	8.8 ± 5.9	660 ± 179.5	**0.0294** ± 0.0015	0.0435 ± 0.0006
			$t-small$	211.6 ± 89.8	415 ± 50.2	**0.0253** ± 0.0011	0.0263 ± 0.0059
	LS	Similarity	$k-small$	12.4 ± 5.5	195.2 ± 177.3	**0.0268** ± 0.0016	0.0264 ± 0.0001
		Stability	$k-small$	14.8 ± 17.3	514 ± 180.5	**0.0273** ± 0.0014	0.0262 ± 0.0005
			$t-small$	122.8 ± 111.2	451 ± 68.3	**0.0245** ± 0.0025	0.0249 ± 0.0010
	No FS			1'751			0.0214
VoD flash-crowd	ARR	Similarity	$k-small$	12.4 ± 5.5	131.2 ± 142.7	**0.1629** ± 0.0139	0.1328 ± 0.0083
		Stability	$k-small$	16 ± 17.0	724 ± 198.1	**0.1340** ± 0.0067	0.1349 ± 0.0087
			$t-small$	140.8 ± 94.1	387 ± 74.6	**0.1081** ± 0.0255	0.1129 ± 0.0222
	LS	Similarity	$k-small$	17.2 ± 16.5	226.4 ± 198.2	**0.0969** ± 0.0405	0.1469 ± 0.0257
		Stability	$k-small$	7.6 ± 5.5	749 ± 213.2	**0.1170** ± 0.0338	0.1657 ± 0.0135
			$t-small$	167.2 ± 110.4	520 ± 85.2	**0.0817** ± 0.0320	0.0899 ± 0.0425
	No FS			1'255			0.0771
VoD periodic	ARR	Similarity	$k-small$	10 ± 6	363.2 ± 196.5	**0.2085** ± 0.0203	0.1795 ± 0.0041
		Stability	$k-small$	5.2 ± 3.6	781 ± 168.2	**0.2123** ± 0.0189	0.1827 ± 0.0024
			$t-small$	174.4 ± 100.8	356 ± 59.2	**0.1413** ± 0.0196	0.1384 ± 0.0259
	LS	Similarity	$k-small$	25.6 ± 25.6	108 ± 85.5	**0.1162** ± 0.0295	0.1426 ± 0.0718
		Stability	$k-small$	17.2 ± 16.5	686 ± 145.9	**0.1139** ± 0.0605	0.1166 ± 0.0739
			$t-small$	211.6 ± 89.8	541 ± 115.7	**0.1130** ± 0.0084	0.1243 ± 0.0344
	No FS			1'296			0.1190

TABLE VII
EVALUATION OF OSFS ON FEDCSIS 2020 CHALLENGE DATASETS. THE PREDICTION METHOD IS RANDOM FOREST REGRESSION. THE ERROR IS EXPRESSED IN NMAE (7). EACH ROW SHOWS THE AGGREGATED RESULT FROM AN EVALUATION SCENARIO WITH TEN RUNS. THE VALUES PRINTED IN BOLDFACE RELATE TO PREDICTORS TRAINED USING A FEATURE SET COMPUTED THROUGH ONLINE FEATURE SELECTION

Dataset	Method	Metric	Search	k	t_k	**Pred Error (Online FS)**	Pred Error (Offline FS)
Dataset1	ARR	Similarity	$k-small$	26.8 ± 24.9	217.6 ± 169.2	**0.701** ± 0.124	0.593 ± 0.243
		Stability	$k-small$	10 ± 6	600 ± 135.2	**0.829** ± 0.132	0.730 ± 0.218
	LS	Similarity	$k-small$	10 ± 18	8 ± 0	**0.291** ± 0.024	0.884 ± 0.170
		Stability	$k-small$	44.8 ± 73.8	876 ± 66.7	**0.569** ± 0.197	0.715 ± 0.249
	No FS			10'162			0.230
Dataset2	ARR	Similarity	$k-small$	168.4 ± 108.7	153.6 ± 145.1	**0.221** ± 0.128	0.208 ± 0.103
		Stability	$k-small$	42.4 ± 26.7	882 ± 65.2	**0.275** ± 0.121	0.272 ± 0.116
	LS	Similarity	$k-small$	4 ± 0	92.8 ± 36.3	**0.055** ± 0.004	0.218 ± 0
		Stability	$k-small$	4 ± 0	337 ± 69.7	**0.055** ± 0.003	0.218 ± 0
	No FS			30'437			0.058

TABLE VIII
EVALUATION OF OSFS ON POLI TORINO DATASETS. THE PREDICTION METHOD IS RANDOM FOREST CLASSIFICATION. THE ERROR IS EXPRESSED IN CLASSIFICATION ERROR (8). EACH ROW SHOWS THE AGGREGATED RESULT FROM AN EVALUATION SCENARIO WITH TEN RUNS. THE VALUES PRINTED IN BOLDFACE RELATE TO PREDICTORS TRAINED USING A FEATURE SET COMPUTED THROUGH ONLINE FEATURE SELECTION

Dataset	Method	Metric	Search	k	t_k	**Pred Error (Online FS)**	Pred Error (Offline FS)
Jitsi	ARR	Similarity	$k-small$	5.2 ± 3.6	140 ± 150.7	**0.0666** ± 0.0342	0.0896 ± 0.0145
	LS	Similarity	$k-small$	4 ± 0	144 ± 188.6	**0.0650** ± 0.0289	0.1165 ± 0
	No FS			95			0.0150
Webex	ARR	Similarity	$k-small$	4 ± 0	98.4 ± 88.1	**0.1031** ± 0.0656	0.1066 ± 0
	LS	Similarity	$k-small$	4 ± 0	116.0 ± 138.3	**0.0877** ± 0.0558	0.1998 ± 0
	No FS			95			0.0058

the search policy. The fifth and sixth columns give the output of OSFS, the seventh and eighth columns provide the prediction error. The values of the seventh column are printed in boldface, which means that they relate to predictors trained using a feature set computed through online feature selection.

Each row in Tables VI, VII, and VIII shows the result from an evaluation scenario for a given dataset, feature ranking algorithm, feature set stability metric, and search policy. A scenario contains both the mean values and the standard deviations from ten evaluation runs of OSFS with different start times (whereby one start time is t = 1 and the other 9 are chosen uniformly at random from the trace).

We observe that OSFS produces a range of feature set sizes k, from 4 to 212. The same applies to the number of samples

t_k (i.e., the data collection time) with a range from 8 to 882. As expected, the search policy *k–small* produces small values for k and larger values for t_k, and for the search policy *t–small*, the opposite applies. In all scenarios, we see a significant reduction in the size of the feature sets compared with the total number of features. We also find that, while the values for k and t_k are strongly dependent on the search policy, they are less influenced by other parameters, namely, the type of dataset and the feature ranking algorithm.

The column Pred Error (Online FS) shows the error of a prediction model trained with the feature set of size k that has been produced with t_k samples. These values reflect the capability of OSFS to produce an effective feature set for model training. We compare the results with two baseline methods. The first baseline method uses all samples of the dataset for feature selection. (When applying offline feature selection using LS on the Poli Torino datasets, we use only 10% random samples of the Jisti dataset or 5% of the Webex dataset, since these datasets are so large and smaller sample size is sufficient to train accurate predictors.) The values for this baseline are listed in column Pred Error (Offline FS). The second baseline method does not rely on feature selection and uses all features to train the prediction model. The result for this method is listed in the lowest row of each dataset and indicated by No FS.

Comparing the values in column Pred Error (Offline FS) with the error values in No FS shows us the cost of reducing the feature set. Depending on the scenario, the difference in error ranges from 2% to 115% for the KTH datasets, from 158% to 369% for the FedCSIS datasets, and from 497% to 3'345% for Poli Torino datasets. While the difference in the error can be quite significant, we have shown in earlier research that it can be effectively reduced by using a supervised feature ranking algorithm instead of an unsupervised one [25].

Most importantly, the values in the column Pred Error (Online FS) tend to be somewhat smaller than those in Pred Error (Offline FS), 6% on KTH datasets, 25% on FedCSIS datasets, and 32% on Poli Torino datasets in average. This result suggests that OSFS is effective as an online feature selection algorithm and is robust regarding the sample interval on the trace that is used for feature selection.

Other conclusions we draw from Tables VI, VII, and VIII:

- OSFS can achieve a massive reduction in the size of the feature set, 1-3 orders of magnitude.
- The number of samples needed to compute the feature sets averages around 400 on the investigated datasets. For the KTH testbed, where metrics are monitored once per second, this means that OSFS produces a stable feature set within an average of 7 minutes.
- For KTH and FedCSIS datasets, The feature ranking algorithm LS generally provides much better results than ARR, although at the cost of higher computational complexity when the number of samples used for feature selection becomes large (see Section III).
- Consistent with our earlier results (e.g., [19], [26]), we find that the type of service and the load pattern significantly affect the prediction error.
- We find that neither the feature set similarity metric nor the feature set stability metric outperforms the other on the datasets we investigated. Since the feature set similarity has lower computational complexity, we prefer this metric for future investigations.
- On the investigated datasets, OSFS achieves the best results, on average, with feature ranking algorithm LS, search policy *k–small*, and the feature set similarity metric.

IX. Evaluation of OSFS With Online Training and Change Detection

In Section VIII, we evaluate OSFS by studying the prediction error of a random forest regressor or classifier. In this case, feature selection is performed online, but model training is performed offline. The column Pred Error (Online FS) in Tables VI, VII, and VIII gives the evaluation results.

Building on these results, we investigate two key questions in this section. First, which prediction accuracy can be achieved when not only feature selection is performed online, but also model training, and how do the results compare to offline training? Second, can better prediction accuracy be achieved if the feature set is adapted over time? We address the second question through detecting concept drift, i.e., a change in the conditional distribution $P(Y|X)$ over time (see Section X). We use the STUDD algorithm for the concept drift detection (see Section VI).

The implementation of the Page-Hinkley test in STUDD uses the *scikit-multiflow* library [27] with the change parameter *delta* set to 0.05. Regarding OSFS, we use the *k–small* search policy and the *similarity* metric.

Table IX summarizes the results of these evaluations. The first column indicates the dataset, the second shows the feature ranking algorithm, and the third column (Offline train) gives the prediction error when using online feature selection and offline training. (The values are computed the same way as in Tables VI, VII, and VIII column Pred Error (Online FS).) The fourth column (Online train) shows the prediction error when using online feature selection and online training. The fifth column (Model retrain) gives the results when using online feature selection, online training, and model retraining after concept drift is detected. The sixth column (Model retrain and feature recomp) provides the most important results. It shows the prediction error when using online feature selection, online learning, as well as feature set recomputation, and model retraining after detecting concept drift. Finally, the last column (number of changes) indicates the number of changes in concepts discovered by STUDD on the trace.

Each row in Table IX lists the result from a run with start time $t = 1$. Figure 6 describes how an experiment takes place. At the start time t_0, the system begins with periodically collecting samples with values from the entire feature set $\boldsymbol{F}$. At time t_0', OSFS returns a stable feature set $\boldsymbol{F}_0$ (based on the samples in the interval $[t_0, t_0']$). At this point, the monitoring system begins to collect samples with values only from the reduced feature set $\boldsymbol{F}_0$. Based on this feature set, the prediction model M_0 is trained and the system starts predicting

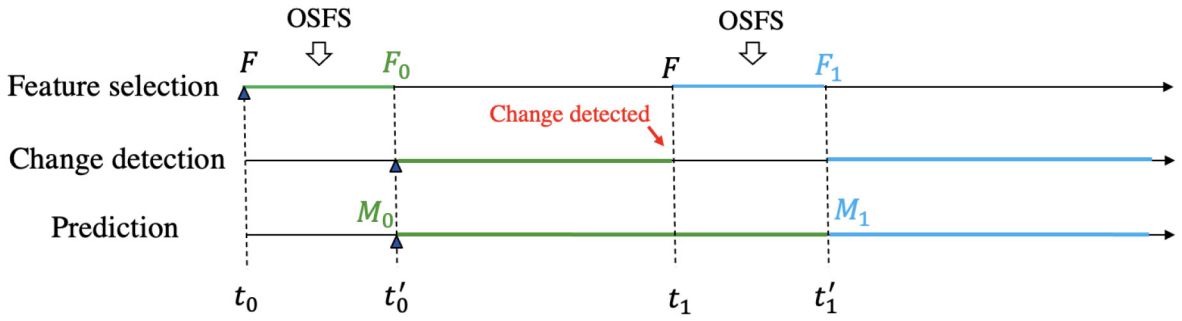


Fig. 6. OSFS selects subset F_0 from the entire feature set F using samples from interval $[t_0, t_0']$. When a change is detected at time t_1, OSFS selects subset F_1 from F using samples from interval $[t_1, t_1']$.

TABLE IX
EVALUATION OF OSFS FOR ONLINE MODEL TRAINING AND MODEL RETRAINING UNDER CHANGE DETECTION ON KTH DATASETS. THE PREDICTION METHOD IS RANDOM FOREST REGRESSION. THE ERROR IS EXPRESSED IN NMAE (7). THE MOST IMPORTANT COLUMN IS 'MODEL RETRAIN AND FEATURE RECOMP', WHICH SHOWS THE PREDICTION ERROR WHEN USING ONLINE FEATURE SELECTION, ONLINE LEARNING AS WELL AS FEATURE SET RECOMPUTATION AND MODEL RETRAINING AFTER DETECTING CONCEPT DRIFT

Dataset	Method	Offline train	Online train	Model retrain	Model retrain and feature recomp	number of changes
KV flash-crowd	ARR	0.0238	0.0387	0.0387	0.0387	0
	LS	0.0232	0.2447	0.0389	0.0393	1
KV periodic	ARR	0.0436	0.1554	0.0579	0.0531	2
	LS	0.0261	0.0985	0.0588	0.0534	2
VoD flash-crowd	ARR	0.1499	0.1163	0.1163	0.1163	0
	LS	0.0654	0.1157	0.1157	0.1157	0
VoD periodic	ARR	0.2280	0.2700	0.1789	0.1789	1
	LS	0.1844	0.1840	0.1687	0.1684	1

target values. In the case of online training, the samples in the interval $[t_0, t_0']$ are used. At time t_1, the change detection algorithm detects a concept drift and triggers OSFS to recompute the feature set. At this point, the monitoring system switches to monitor the values from the entire feature set $\boldsymbol{F}$. The newly computed feature set $\boldsymbol{F}_1$ becomes available at time t_1'. In the case of model retraining, the new predictor M_1 is trained with the samples from the interval $[t_1, t_1']$. For the reason of simplicity, the training time is omitted in Figure 6.

From Table IX, we gain the following insights regarding the performance of OSFS on the studied datasets:

- Feature set recomputation is effective as it can achieve a reduction in error by 3% to 6%, in addition to the reduction achieved by model retraining (8% to 80%). (The scenarios with feature recomputation are those that have values larger than 0 in the last column.)
- Online training (Online train), compared with offline training (Offline train), incurs 18% to 955% higher prediction error. This error can be reduced by considering additional samples for training. Note that decreasing the difference in prediction error between online training and offline training is not studied in this work, as the focus is on online feature selection.

In summary, the error of a predictor can be significantly reduced by recomputing the feature set and retraining the model after a concept drift.

X. RELATED WORK

In the context of this paper, we understand *online learning* as an iterative learning method where training samples arrive sequentially as a data stream. Periodic monitoring of infrastructure metrics in a networked system creates such a stream of samples. The learning algorithms that enable learning on a stream of samples are often referred to as *incremental learning*. Reference [28] formalizes the concept of incremental learning and discusses algorithms that require only constant memory. In [29], the authors compare eight incremental learning algorithms for supervised classification. Instead of the term incremental learning, many authors use the expression *learning on data streams*. Reference [30] is a textbook that provides a state-of-the-art description of this subject. Algorithms for data stream mining including classification, regression, clustering, and frequent pattern mining are discussed. Most of these algorithms are adapted from well-known offline learning algorithms.

Most works that address learning on data streams assume that the samples in a data stream are drawn from a distribution $P(X, Y)$, which is also referred to as a concept [31]. The change of such a concept is called a *concept drift*. For our research, it is important to detect a change in the conditional distribution $P(Y|X)$, which is called a *real concept drift* [31]. In this paper, we use the STUDD algorithm [15] for detecting a change in the conditional probability distribution $P(Y|X)$ (see Section VI). Reference [14] provides a survey on concept drift adaptation methods, which detect concept drift and adapt the prediction model. The paper contains a taxonomy of such methods. It also lists metrics for the evaluation of change detection methods.

A key challenge in learning on operational data from networked systems, which are distributed by nature, is respecting the resource constraints. The extraction of data on the devices,

their transportation, the data preprocessing, and the model training require network, computing, and memory resources. Much research has been conducted to reduce resource consumption while producing sufficiently accurate prediction models. One research direction is *online sample selection* where samples are kept in a cache of constant size for later processing and model training. A sample selection algorithm decides which samples should be stored in the cache and which should be dropped from the data stream. A well-known and efficient algorithm is reservoir sampling, which chooses a sample set of fixed size uniformly at random from a population of unknown size in a single pass [32]. In [33] the authors aim at solving the label budget problem and the class imbalance problem on stream data. They propose an active sampling scheme which involves biased sampling, biased archiving, and genetic programming to identify the champion classifier. If the predicted label relates to a minority class, the instance is prioritized for selection. Then, an archived instance which belongs to a majority class will be replaced with a high probability. In earlier work [34] we have evaluated four online sampling algorithms which have been derived from known algorithms. We specifically argue that feature selection algorithms can be adapted for sample selection and provide experimental evidence that the resulting algorithms can be effective.

A second research direction focuses on reducing the dimensionality of the feature space, which corresponds to the number of features. Many methods have been developed that map a high-dimensional feature space into a low-dimensional space. Principal Components Analysis (PCA) [35] is a linear and popular method that is often used for this purpose. A second method, which is based on neural networks, is auto-encoder [35]. Another approach to reduce the dimensionality of the feature space is feature selection. In this paper, we study feature selection methods, because they allow us to significantly reduce monitoring costs, in addition to saving model training costs. The other dimensionality reduction methods described above only reduce model training costs.

Feature selection has been studied as an effective data preprocessing strategy in both the machine learning and data mining fields for several decades. Many survey papers cover and compare feature selection methods, e.g., [36], [37], [38], [39], [40]. For instance, [38] provides a useful categorization of feature selection methods along several dimensions, such as supervised and unsupervised methods; wrapper, filter, and embedded methods; and static or streaming methods. In [40], the authors evaluate several supervised feature selection methods on security datasets. They investigate the prediction performance and computing time of these algorithms. They find that the selected feature set varies with the type of attack recorded in the dataset. In [41] the authors propose a policy-induced unsupervised feature selection method based on a concrete auto-encoder. The policies are expressed in form of must-have features, which are pre-selected for the feature set. KL-divergence is used as a similarity measure to identify features which are significantly different from the pre-selected features. This method requires the size of the feature set as input and does not provide a ranked feature list.

Most feature selection methods described in the literature have been developed for offline learning where all data fits into memory. In recent years, however, *online feature selection* methods have received increasing attention. In this case, the input to the algorithm is either a stream of feature vectors (e.g., [42], [43], [44], [45], [46], [47]) or a stream of samples (e.g., [47], [48], [49]). Interestingly, most published online feature selection methods fall into the first category and process a stream of feature vectors. Such methods are not suitable for the problem we study in this paper, since, in our case, the data becomes gradually available as time progresses. Our interest thus is in methods that process streams of samples. Examples of such works are [48], [49], and [47].

In [48], the authors perform online supervised feature selection as a part of training the perceptron algorithm. During a training step, the values of the selected features are used to update the weights of the perceptron. Features with large weights are selected with high probability and features with small weights are selected with low probability. More closely related to our work is [49], which presents an unsupervised online feature selection algorithm based on clustering. The algorithm uses constant memory. In [47], the authors introduce an online feature selection algorithm called Geometric Online Adaption (GOA). A characteristic of this algorithm is that it works for both streams of samples and streams of feature vectors.

Note that a large part of the online feature selection algorithms we found in the literature require the number of features to be selected as an input parameter. The algorithm we present in this paper attempts to find a small but sufficiently large number of features that form a stable feature set.

The authors of [50] provide a recent survey of work on the *stability* of feature selection algorithms. By stability, they mean that an algorithm selects similar feature sets from similar sample sets. Another survey is given in [11], which additionally proposes a novel metric for measuring feature set stability. We are using this metric in Section IV.

XI. Conclusion and Future Work

In this paper, we introduced OSFS, an online algorithm that selects a small set from a large number of available data sources, which allows for rapid, low-overhead learning and prediction. OSFS is instantiated with a set of options and performs a grid search that terminates when a stable feature set has been identified.

Using datasets from three different sources, we found that OSFS achieves a massive reduction in the size of the feature set while maintaining its performance for training predictors. We have shown the effectiveness of OSFS as an online feature selection algorithm which is robust regarding the sample interval used for feature selection. In addition, we found that, if concept drift occurs, its effect can be mitigated by recomputing the feature set with OSFS and retraining the prediction model.

Regarding future work, there are many ways our method can be improved or extended. We can evaluate OSFS with additional instantiations of feature ranking algorithms, stability

conditions, and search policies. Also, the tradeoffs between the three metrics k, t_k, and the prediction error warrant further study.

Online learning is only one part of efficiently producing a learning model using online techniques. Other parts include online sample selection [34], and the integration of online feature selection, online sample selection, and online training into a joint framework. Finally, we believe it should be investigated how such a framework can be distributed, since we envision both training and prediction as distributed processes in future networked systems.

ACKNOWLEDGMENT

The authors are grateful to Andreas Johnsson, Hannes Larsson, and Jalil Taghia with Ericsson Research for fruitful discussion around this work, as well as to Forough Shahab Samani, Kim Hammar, and Rodolfo Villaça for comments on an earlier version of this paper.

REFERENCES

[1] A. Janusz, M. Przyborowski, P. Biczyk, and D. Ślezak, "Network device workload prediction: A data mining challenge at knowledge pit," in *Proc. 15th Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, 2020, pp. 77–80.

[2] M. Trevisan. *Real-Time Classification of Real-Time Communications*. 2021. [Online]. Available: https://smartdata.polito.it/rtc-classification/

[3] X. Wang, F. S. Samani, and R. Stadler, "Online feature selection for rapid, low-overhead learning in networked systems," in *Proc. 16th Int. Conf. Netw. Service Manag. (CNSM)*, 2020, pp. 1–7.

[4] V. N. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.

[5] S. Handurukande, S. Fedor, S. Wallin, and M. Zach, "Magneto approach to QoS monitoring," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM) Workshops*, 2011, pp. 209–216.

[6] O. Izima, R. de Fréin, and M. Davis, "Video quality prediction under time-varying loads," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, 2018, pp. 129–132.

[7] X. Tao, Y. Duan, M. Xu, Z. Meng, and J. Lu, "Learning QoE of mobile video transmission with deep neural network: A data-driven approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1337–1348, Jun. 2019.

[8] G. Perna *et al.*, "Online classification of RTC traffic," in *Proc. IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2021, pp. 1–6.

[9] A. J. Ferreira and M. A. Figueiredo, "An unsupervised approach to feature discretization and selection," *Pattern Recognit.*, vol. 45, no. 9, pp. 3048–3060, 2012.

[10] X. He, D. Cai, and P. Niyogi, "Laplacian score for feature selection," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Assoc., 2006, pp. 507–514.

[11] S. Nogueira, K. Sechidis, and G. Brown, "On the stability of feature selection algorithms," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6345–6398, 2017.

[12] J. Haug, M. Pawelczyk, K. Broelemann, and G. Kasneci, "Leveraging model inherent variable importance for stable online feature selection," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2020, pp. 1478–1502.

[13] "Scikit-Learn Developers, DBSCAN." 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–37, 2014.

[15] V. Cerqueira, H. M. Gomes, A. Bifet, and L. Torgo, "STUDD: A student-teacher method for unsupervised concept drift detection," 2021, *arXiv:2103.00903*.

[16] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2006, pp. 535–541.

[17] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, nos. 1–2, pp. 100–115, 1954.

[18] "Scikit-Learn Developers, sklearn.ensemble.RandomForestRegressor." 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[19] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *J. Netw. Syst. Manag.*, vol. 25, no. 4, pp. 672–698, 2017.

[20] R. Yanggratoke *et al.*, "Predicting service metrics for cluster-based services using real-time analytics," in *Proc. 11th Int. Conf. Netw. Service Manag.*, 2015, pp. 135–143.

[21] "SAR." 2016. [Online]. Available: http://linux.die.net/man/1/sar

[22] T. Bowden, B. Bauer, J. Nerin, S. Feng, and S. Seibold. "The /Proc Filesystem." 2000. [Online]. Available: http://www.kernel.org/doc/Documentation/filesystems/proc.txt (Accessed: Jun. 5, 2022).

[23] F. Shahab. "Data Traces for 'Efficient Learning on High-Dimensional Operational Data' Paper, CNSM 2019." 2018. [Online]. Available: https://github.com/foroughsh/CNSM2019-traces

[24] C. Gao, "Online learning with sample selection," M.S. thesis, School Electr. Eng. Comput. Sci., KTH Royal Inst. Technology, Stockholm, Sweden, 2021.

[25] F. S. Samani, H. Zhang, and R. Stadler, "Efficient learning on high-dimensional operational data," in *Proc. 15th Int. Conf. Netw. Service Manag. (CNSM)*, 2019, pp. 1–9.

[26] R. Yanggratoke *et al.*, "Predicting real-time service-level metrics from device statistics," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, 2015, pp. 414–422.

[27] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *J. Mach. Learn. Res.*, vol. 19, no. 1, pp. 2915–2914, 2018.

[28] A. Gepperth and B. Hammer, "Incremental learning algorithms and applications," in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, 2016, pp. 1–12.

[29] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, pp. 1261–1274, Jan. 2018.

[30] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer, *Machine Learning for Data Streams: With Practical Examples in MOA*. Cambridge, MA, USA: MIT Press, 2018.

[31] J. Haug and G. Kasneci, "Learning parameter distributions to detect concept drift in data streams," in *Proc. 25th Int. Conf. Pattern Recognit. (ICPR)*, 2021, pp. 9452–9459.

[32] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, pp. 37–57, 1985.

[33] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "On botnet detection with genetic programming under streaming data label budgets and class imbalance," *Swarm Evol. Comput.*, vol. 39, pp. 123–140, Apr. 2018.

[34] R. S. Villaça and R. Stadler, "Online learning under resource constraints," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM)*, 2021, pp. 134–142.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[36] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Mar. 2003.

[37] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, 2014.

[38] J. Li *et al.*, "Feature selection: A data perspective," *ACM Comput. Surveys*, vol. 50, no. 6, pp. 1–45, 2017.

[39] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "A review of unsupervised feature selection methods," *Artif. Intell. Rev.*, vol. 53, no. 2, pp. 907–948, 2020.

[40] M. Di Mauro, G. Galatro, G. Fortino, and A. Liotta, "Supervised feature selection techniques in network intrusion detection: A critical review," *Eng. Appl. Artif. Intell.*, vol. 101, May 2021, Art. no. 104216.

[41] J. Taghia, F. Moradi, H. Larsson, X. Lan, M. Ebrahimi, and A. Johnsson, "Policy-induced unsupervised feature selection: A networking case study," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2022. [Online]. Available: https://infocom2022.ieee-infocom.org/program/accepted-paper-list-main-conference

[42] P. Zhou, X. Hu, P. Li, and X. Wu, "Online feature selection for high-dimensional class-imbalanced data," *Knowl.-Based Syst.*, vol. 136, pp. 187–199, Nov. 2017.

[43] N. AlNuaimi, M. M. Masud, M. A. Serhani, and N. Zaki, "Streaming feature selection algorithms for big data: A survey," *Appl. Comput. Inform.*, vol. 18, nos. 1–2, pp. 113–135, 2019.

[44] X. Wu, K. Yu, W. Ding, H. Wang, and X. Zhu, "Online feature selection with streaming features," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 5, pp. 1178–1192, May 2013.

[45] S. Perkins and J. Theiler, "Online feature selection using grafting," in *Proc. 20th Int. Conf. Mach. Learn. (ICML)*, 2003, pp. 592–599.
[46] S. C. Hoi, J. Wang, P. Zhao, and R. Jin, "Online feature selection for mining big data," in *Proc. 1st Int. Workshop Big Data, Streams Heterogeneous Source Min. Algorithms Syst. Program. Models Appl.*, 2012, pp. 93–100.
[47] S. Y. Sekeh, M. R. Ganesh, S. Banerjee, J. J. Corso, and A. O. Hero, "A geometric approach to online streaming feature selection," 2019, *arXiv:1910.01182*.
[48] J. Wang, P. Zhao, S. C. Hoi, and R. Jin, "Online feature selection and its applications," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 698–710, Mar. 2014.
[49] W. Shao, L. He, C.-T. Lu, X. Wei, and S. Y. Philip, "Online unsupervised multi-view feature selection," in *Proc. IEEE 16th Int. Conf. Data Min. (ICDM)*, 2016, pp. 1203–1208.
[50] U. M. Khaire and R. Dhanalakshmi, "Stability of feature selection algorithm: A review," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1060–1073, 2022.

Xiaoxuan Wang (Graduate Student Member, IEEE) received the M.Sc. degree in communication systems from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2020, where she is currently pursuing the Ph.D. degree. Before starting her Ph.D., she worked as a Research Engineer with KTH for one year. Her main research interests include online learning, feature, and sample selection.

Rolf Stadler (Senior Member, IEEE) received the M.Sc. degree in mathematics and the Ph.D. degree in computer science from the University of Zurich. He is a Professor with the KTH Royal Institute of Technology, Stockholm, Sweden, and the Head of the Division of Network and Systems Engineering. Before joining KTH in 2001, he held positions with the IBM Zurich Research Laboratory, Columbia University, and ETH Zürich. His group made contributions to real-time monitoring, resource management, and automation for large-scale networked systems. His current interests include data-driven methods for network engineering and management, as well as AI techniques for cybersecurity. He was the Editor-in-Chief of IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT from 2014 to 2017.