Efficient Learning on High-dimensional Operational Data

Forough Shahab Samani †‡ Hongyi Zhang † and Rolf Stadler†‡
† Dept. of Network and Systems Engineering, KTH Royal Institute of Technology, Sweden
‡ RISE AI, Sweden
Email: {foro, hongyiz, stadler}@kth.se

***Abstract*—In networked systems engineering, operational data gathered from sensors or logs can be used to build data-driven functions for performance prediction, anomaly detection, and other operational tasks. The number of data sources used for this purpose determines the dimensionality of the feature space for learning and can reach millions for medium-sized systems. Learning on a space with high dimensionality generally incurs high communication and computational costs for the learning process. In this work, we apply and compare a range of methods, including, feature selection, Principle Component Analysis (PCA), and autoencoders with the objective to reduce the dimensionality of the feature space while maintaining the prediction accuracy when compared with learning on the full space. We conduct the study using traces gathered from a testbed at KTH that runs a video-on-demand service and a key-value store under dynamic load. Our results suggest the feasibility of reducing the dimensionality of the feature space of operational data significantly, by one to two orders of magnitude in our scenarios, while maintaining prediction accuracy. The findings confirm the Manifold Hypothesis in machine learning, which states that real-world data sets tend to occupy a small subspace of the full feature space. In addition, we investigate the tradeoff between prediction accuracy and prediction overhead, which is crucial for applying the results to operational systems.**

***Index Terms*—Data-driven engineering, Machine learning, ML, Dimensionality reduction,**

I. Introduction

In networked systems engineering, operational data gathered from sensors or logs can be used to build data-driven functions for performance prediction, anomaly detection, and other operational tasks. The number of data sources or monitored variables used for this purpose determines the dimensionality of the feature space for learning: each (univariate) source or variable defines a dimension in this space. Considering that a single server can generate several hundred metrics that can be monitored [1], the total number of metrics can reach millions for a system with thousands of components.

Learning on a space with high dimensionality generally incurs high communication and computational costs for the learning process. The overhead increases at least linearly with the number of dimensions. To achieve efficient learning in networked systems, reducing the dimensionality of the feature space thus becomes crucial.

An obvious approach is to use (human) domain experts that select, at the design or configuration stage of a system, the sources and metrics that should be monitored. In fact, almost all work in performance prediction using machine learning follows this approach (see, e.g. [2]). Our experience has been that this strategy becomes increasingly difficult with growing system complexity and system size, if the goal is to achieve high prediction accuracy.

We follow a different path in this work. We consider all possible (or at least a very large number of) data sources for prediction and automate the reduction of the feature space through learning algorithms. This approach is motivated by the so-called *Manifold Hypothesis*, which states that real data (as opposed to artificially created data sets) tends to occupy a small subspace of the total feature space[3]. The hypothesis has been validated for image and speech data sets, but not (to our knowledge) for measurements from networked systems. A classical result in this context is reported in [4] where images of hand-written digits (MNIST dataset), each represented as a vector in a 784 dimensional space, are mapped into a 30 dimensional subspace with minimal distortion by using a deep neural network.

In this work, we apply and compare a range of known machine-learning methods, including, feature selection, Principle Component Analysis (PCA), and autoencoder with the objective to reduce the dimensionality of the feature space in such a way that learning on the lower-dimensional space can achieve the same prediction accuracy than learning on the full space. We conduct the study using traces gathered from a testbed at KTH that runs a video-on-demand service and a key-value store under dynamic load.

The chosen use case for our investigation is the *prediction of end-to-end performance metrics*. Using statistical learning, we measure device-level metrics in the infrastructure and predict service-level end-to-end metrics (see Figure 2). Specifically, from measuring metrics like CPU utilization on servers and packet counters on network devices we predict quality-of-service parameters like video frame rates and query response times of the services on the testbed. For details, see our earlier work in this area [5], [6], [1].

This paper makes the following contributions. First, we demonstrate, in a systematic way, that it is feasible to automatically and significantly reduce the dimensionality of the feature space of operational data while maintaining (and sometimes improving) prediction accuracy. In our case, we reduce the full features space of some 1500 dimensions by one to two orders magnitude. Second, by studying the tradeoff between prediction accuracy and overhead of dimensionality reduction,

978-3-903176-24-9

we show how the particular choice of a dimensionality-reduction technique depends on the particular use case and the constraints of the target technology.

The remainder of this paper is organized as follows. Section II formalizes the problem and describes the models we use and the concepts we apply for dimensionality reduction. Section III details our testbed, the experiments we are conducting, the metrics we are collecting during experiments and the traces we generate from this data. Sections IV, V, and VI evaluate three dimensionality-reduction methods applied on the testbed traces. Section VII surveys related work. Finally, Section VIII presents conclusions and future work.

II. Problem Setting and Machine Learning Methods Used in This Work

This section formalizes the problem of learning on low-dimensional subspaces and describes the models we use and the concepts we apply for mapping the original input space onto a lower-dimensional subspace. The discussion and formalism we use are informed by three classical textbooks on machine learning: Vapnik [7], Bishop [8], and Goodfellow et al. [3].

First, we set the context. We map low-level infrastructure statistics $x \in X$ to service-level metrics $y \in Y$ using supervised learning. The infrastructure statistics x include measurements from a server cluster and a network. In this work, the service-level metrics y refer to performance indicators on a client, for example, frame rate or response time. Details regarding the composition of X and Y are given in Section III-B.

The metrics x and y evolve over time, influenced, e.g., by the load on the servers, operating system dynamics, network traffic, and the number of active clients that access the service. Assuming a global clock that can be read on the machines in the server cluster, the network devices, and the clients, we model the combined evolution of the metrics x and y as a time series $\{(x^{(t)}, y^{(t)})\}$.

We predict the estimated value of the service-level metric y conditioned on the infrastructure metric x. Using the framework of statistical learning [7], we model x and y as random variables. We assume that the measurements $(x^{(t)}, y^{(t)})$ are i.i.d. samples drawn from the joint hidden distribution $p_{\text{data}}(x, y)$. Further, we assume $x^{(t)} \in \mathbb{R}^d$. (In the context of this work, d is in the order of thousands.) Lastly, we assume $y^{(t)} \in \mathbb{R}$, i.e., $y^{(t)}$ is one-dimensional (or univariate), which simplifies the formal description.

We now find a model (or function) $f(\theta) : x \rightarrow \hat{y}$ with parameter θ so that $\hat{y}$ closely approximates y for all $x \in X$, given the samples $(x^{(t)}, y^{(t)})$ with $t = 1, ..., m$. This is achieved by minimizing a loss function $\mathcal{L}(\{f(\theta, x^{(t)}, y^{(t)}\})$ with respect to θ.

In this work, we use *random forest* as our model for prediction, since it provides accurate prediction in our testbed environment [1], [5].

Our objective is to map X to a new space X' with much smaller dimensionality, i.e., $dim(X') \ll dim(X)$, while maintaining the prediction accuracy. In other words, learning a function $f'(\theta') : x' \rightarrow \hat{y}$ should result in the same prediction accuracy than learning the function $f(\theta) : x \rightarrow \hat{y}$ whereby $x'^{(t)}$ is the result of mapping $x^{(t)}$ from X to X'. With "the same prediction accuracy" we mean $\mathcal{L}(\{f'(\theta', x'^{(t)}), y^{(t)}\}) \simeq \mathcal{L}(\{f(\theta, x^{(t)}), y^{(t)}\})$. Figure 1 illustrates our approach.

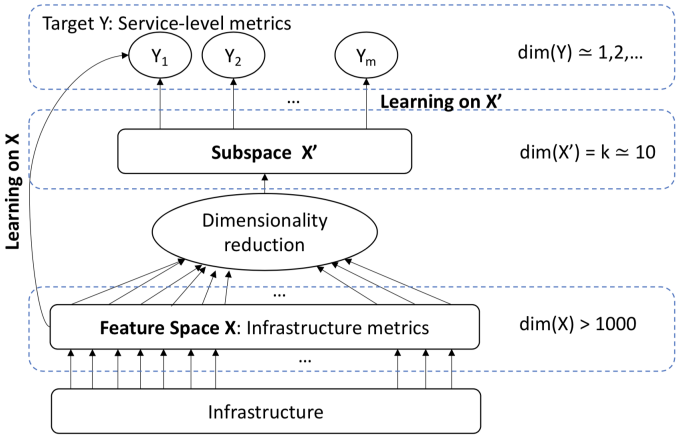


Fig. 1. Instead of learning on the feature space X to predict Y, this work proposes to learn on the low-dimensional subspace X' by first mapping the observations $x^{(t)}$ into X' using a dimensionality reduction method.

There are a range of techniques and methods that can be applied to decrease the dimensionality of the X space (or feature space). In this work we investigate the effectiveness of three methods, namely, *tree-based feature selection*, *Principle Component Analysis (PCA)*, and autoencoder. These three methods produce increasingly general mappings in the following sense. Feature selection reduces the feature space through projection, PCA achieves this through a general linear mapping, and autoencoder through a general nonlinear mapping.

We use regression tree as the basis for our feature selection method [9]. This method ranks all features based on their relationship with the target variable. Regression tree recursively partitions the feature space, which produces the ranking of the features. The method is an example of a *supervised* feature selection method, since it requires (x, y) samples (instead of (x) samples) to reduce the X space. (In our earlier work, we used a semantic feature selection method where we compared learning from the complete set of the infrastructure features to learning from networking features alone [5] (see Figure III). The results show that end-to-end service-level performance statistics can sometimes be predicted from network features alone with almost the same accuracy than from the complete set of infrastructure features.)

The second method is Principal Component Analysis (PCA). PCA is a traditional way of producing a linear mapping that iteratively finds the directions of largest variance in a data set and represents data points as coordinates in these directions. The directions coincide with the eigenvectors of the covariance matrix of the data set and form an orthogonal basis. PCA minimizes the distortion between the original and

the transformed data points among all possible linear mappings [8].

The third method we investigate in this work is autoencoder. An autoencoder is a specialized deep neural network which copies an input vector to an output vector with minimal distortion. Its architecture has two parts: the encoder, which transforms the input vector to a code vector of lower dimensionality. The second part is decoder, which reconstructs the input vector from the code vector [3].

III. Testbed and Traces

A. Testbed and services

In this section, we describe the experimental infrastructure and the structure of the data traces that we create. Further, we describe the services that run on this infrastructure, namely, a Video-on-Demand (VoD) service and Key-Value (KV) store. Lastly, we explain the load patterns we use and the experiments we run to obtain the traces.

Figure 2 outlines our laboratory 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 ten high-performance machines interconnected by a Gigabit Ethernet. Nine machines are Dell PowerEdge R715 2U servers, each with 64 GB RAM, two 12-core AMD Opteron processors, a 500 GB hard disk, and four 1 Gb network interfaces. The tenth machine is a Dell PowerEdge R630 2U with 256 GB RAM, two 12-core Intel Xeon E5-2680 processors, two 1.2 TB hard disks, and twelve 1 Gb network interfaces. All machines run Ubuntu Server 14.04 64 bits, and their clocks are synchronized through NTP [10].

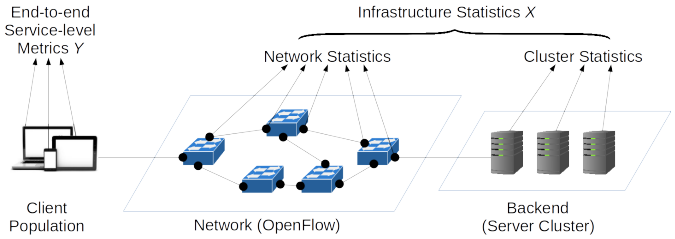


Fig. 2. The testbed at KTH, providing the infrastructure for experiments. In various scenarios we predict end-to-end service-level metrics from low-level infrastructure measurements [5].

The VoD service uses VLC media player software [11], which provides single-representation streaming with varying frame rate. It is deployed on six PowerEdge R715 machines —one HTTP load balancer, three web server and transcoding machines, and two network file storage machines. The load balancer runs HAProxy version 1.4.24 [12]. Each web server and transcoding machine runs Apache version 2.4.7 [13] and ffmpeg version 0.8.16 [14]. The network file storage machines run GlusterFS version 3.5.2 [15] and are populated with the ten most-viewed YouTube videos in 2013, which have a length of between 33 seconds and 5 minutes. The VoD client is deployed in another PowerEdge R715 machine and runs VLC [11] version 2.1.6 over HTTP.

The KV store service uses the Voldemort software [16]. 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 [16]. The OpenFlow network includes 14 switches, which interconnect the server cluster with clients and load generators. The load generators emulate client populations.

A more detailed description of the testbed setup is given in [5].

B. Collected data and traces

We describe the metrics we collect on the testbed, namely, the input feature sets $X_{cluster}$ and X_{port} —the union of which we refer to as X —as well as the specific service-level metrics Y_{VoD} and Y_{KV}.

The $X_{cluster}$ feature set is extracted from the kernel of the Linux operating system that runs on the servers executing the applications. To access the kernel data structures, we use System Activity Report (SAR), a popular open-source Linux library [17]. SAR in turn uses procfs [18] and computes various system statistics over a configurable interval. Examples of such statistics are CPU core utilization, memory utilization, and disk I/O. $X_{cluster}$ includes only numeric features from SAR, about 1 700 statistics per server.

The 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.

The Y_{VoD} service-level metrics are measured on the client device. During an experiment, we capture the following metrics: *1) Display Frame Rate (frames/sec)*, i.e., the number of displayed video frames per second; *2) Audio Buffer Rate (buffers/sec)*, i.e., the number of played audio buffers per second. These metrics are not directly measured, but computed from VLC events like the display of a video frame at the client's display unit. We have instrumented the VLC software to capture these events and log the metrics every second.

The Y_{KV} service-level metrics are measured on the client device. During an experiment, we capture the following metrics: *1) Read Response Time* as the average read latency for obtaining responses over a set of operations performed per second; *2) Write Response Time* as the average write latency for obtaining responses over a set of operations performed per second. These metrics are computed using a benchmark tool of Voldemort, which we modified for our purposes. The read and write operations follow the request–reply paradigm, which allows for tracking the latency of individual operations. We instrumented the benchmark tool to log the metrics every second.

Generating the traces: During experiments, X and Y statistics are collected every second on the testbed. For each application running on the testbed, the data collection framework produces a trace in form of a time series $(x^{(t)}, y^{(t)})$. We interpret this time series as a set of samples

$\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$. Assuming that each sample in the set is drawn uniformly at random from a joint distribution of (x, y), we obtain predictions models using methods from statistical learning.

C. Generating load on the testbed

We have built two load generators, one for the VoD application and another for the KV application. 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 operations issued per second. Both generators produce load according to two distinct load patterns.

1) Periodic-load pattern: the load generator produces requests following a Poisson process whose arrival rate is modulated by a sinusoidal function with starting load level P_S, amplitude P_A, and period of 60 minutes;

2) Flash-crowd load pattern: the load generator produces requests following a Poisson process whose arrival rate is modulated by the flash-crowd model described in [19]. The arrival rate starts at load level F_S and peaks at flash events, which are randomly generated at rate F_E events/hour. At each flash event, the arrival rate increases within a minute to a peak load F_R. It stays at this level for one minute and then decreases to the initial load within four minutes.

Table I shows the configurations of the load generators during the experiments reported in Section III-A. We used a single load generator for the VoD experiments (see [1]) and three for the KV experiments (see [5]).

All traces we used have been created by stochastic models in an attempt to approximate real scenarios. The flash-crowd load pattern, for instance, is based on a model from the research literature and produces arrival patterns that are quite hard to predict. We plan to use in future work traces from operational environments in addition to synthetic traces.

TABLE I
CONFIGURATION PARAMETERS OF VOD AND KV LOAD GENERATORS.

Application	Load Generator	Periodic-load		Flash-crowd-load		
		P_S	P_A	F_S	F_E	F_R
VoD	1	70	50	10	10	120
KV	1	1 000	800	200	10	1 800
	2, 3	350	150	200	3	500

D. The scenarios chosen for this paper

The prediction method proposed in this paper has been evaluated using data from four experiments. Two of them involve running the VoD service and two the KV service.

1) *VoD periodic*: In this experiment, we run the VoD service and generate a periodic load pattern on the testbed. Load generator and client are directly connected to the server cluster, and the testbed does not include the network (see Figure 2). Data is collected every second over a period of 50 000 seconds. The X feature set contains 4 984 features. After cleaning the dataset, 1 409 features remain for processing. Using tree-based feature selection [20], the input space is further reduced to the top 30 features. From the service metrics Y only the video frame rate is used in this investigation. For model training, we use the first 21 600 samples of the trace. More details about the experiment are given in [1], and the trace is available at [21].
2) *VoD flash-crowd*: This experiment relies on the same setup as VoD periodic, except that the testbed is loaded using the flash-crowd pattern. We use the first 3 600 samples of the trace to train the model. We process the trace the same way as described above and the given references contain more information.
3) *KV periodic*: In this experiment, we run the KV service to generate a periodic load pattern on the testbed. Unlike the VoD periodic experiment, we connect load generator and clients to the server cluster via an OpenFlow network (seeIII-A). Measurements are collected every second over a period of 28 962 seconds. The $X\_cluster$ feature set contains 10 374 features. After cleaning the data set, 1 649 features remain for the processing. We use univariate feature selection [20] to decrease number of features to the 200 top features. (The fact that we use two different methods for feature selection in different scenarios has historical, not methodological reasons. We could have used a single method throughout.) The X_port feature set contains 176 features, and after cleaning 134 features remain. From the service metrics Y only the response time for read operations is used in this work. For model training, we use the first 3 600 samples of the trace. More details about the experiment are given in [5], and the trace is available at [22].
4) *KV flash-crowd*: This experiment relies on the same setup as KV periodic, except that the testbed is loaded using the flash-crowd pattern. We process the trace the same way as described above and the given references contain more information.

E. Evaluation metrics for mean estimates

To evaluate the model accuracy in predicting mean values we use two metrics. The first is Normalized Mean Absolute Error (NMAE) which is defined as $\frac{1}{\bar{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y_i}|)$

where $\hat{y_i}$ is the mean value of the target variable, $\bar{y}$ is the average value of the target variable, and m is the number of samples. NMAE is an intuitive and useful measure in the application domain.

The second metric is the Coefficient of Determination (R^2), which is defined as $1 - (\frac{\sum_{i=1}^{m}(y_i - \hat{y_i})^2}{\sum_{i=1}^{m}(y_i - \bar{y})^2})$. R^2 provides theoretical insight: $R^2 = 1$ means perfect prediction, $R^2 = 0$ is the accuracy of a naïve estimator that predicts the sample mean. R^2 takes values in $(-\infty, 1]$.

IV. Dimensionality Reduction through Feature Selection

In this section we apply a (supervised) feature selection method to reduce the dimensionality of the feature space (see section II). Specifically, we use a regression tree model that produces as output a ranked list of all features according to their relative importance with respect to the target. This means that the first k features of this list are the k features with most affinity to the target.

Feature selection allows us to create subspaces of the original feature space X (Figure 1). The top k features span the subspace with k dimensions. This way, we generate subspaces for $k = 1, 2, 3, ..., 1409$. (1409 is the dimensionality of X.) Then we predict the target Y on each of these subspaces using random forest regression.

In this investigation, we answer two questions.

Question 1: How does prediction accuracy depend on k?

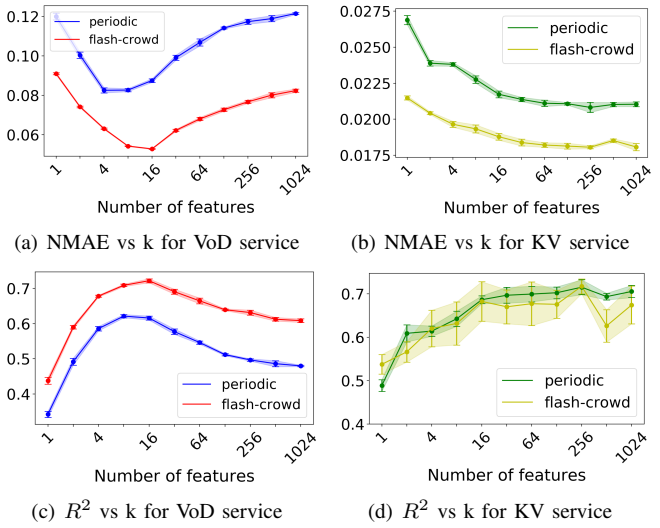
Figure 3 shows the evaluation results for all four scenarios. Throughout this paper, we use blue curves for the scenario VoD periodic load, red for VoD flash-crowd load, green for KV periodic load, and yellow for KV flash-crowd load. Prediction accuracy is measured in $NMAE$ and R^2 (see section III-E). One point on a curve represents the average value from ten-fold cross validation. The vertical bar represents the standard deviation. In most cases, the standard deviation is small and barely visible.

The curves in Figure 3(b) follow our expectation: when the number of dimensions increases, the prediction error falls monotonically (at least up to k=256). In contrast, Figure 3(a) shows that, for the VoD service, the prediction error can be significantly lower for small values of k than for the full feature space X where $k = 1409$. This is surprising and suggests that many features do not contribute to an accurate prediction of the frame rate and actually reduce accuracy. For VoD, the optimal value of k with the smallest prediction error is between 8 and 16, while for KV the optimal k is around 256.

Figure 3 demonstrates that the effect of feature selection differs among services, which is reflected in the fact that the top k feature sets for both services have minimal overlap. At the same time, we observe that, for the same service, the curves for different load patterns show very similar derivatives and thus have minima and maxima for very similar values of k.

Figures 3(c) and 3(d) contain the results of the same evaluation than Figures 3(a) and 3(b), but for the error metric R^2 instead of $NMAE$. The observations we can draw from Figures 3(c) and 3(d) are analogous and the conclusions are the same as above. One advantage of using R^2 is that the error values are in the same range for both services. In contrast, the $NMAE$ values for both services are very different. We generally work with $NMAE$ as an error measure, because it is better interpretable by network engineers. In the following we provide the plots with the popular R^2 metric for information purposes, but we do not further comment on them.

Question 2: What is the tradeoff between prediction accuracy and overhead for learning the prediction model?



(a) NMAE vs k for VoD service (b) NMAE vs k for KV service

(c) R^2 vs k for VoD service (d) R^2 vs k for KV service

Fig. 3. Feature selection: Frame rate and response time prediction on the subspace spanned by the top k features. Vertical axis is accuracy, horizontal axis is dimensionality of subspace.

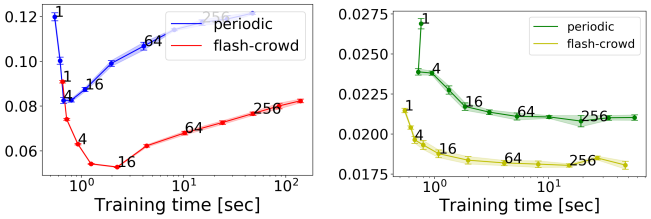
Figure 4 provides the evaluation results. The k value indicates the monitoring overhead, and the training time indicates the computational overhead. The answer to the question depends on two aspects: first, on the constraints on monitoring and computing overhead for a particular use case, and second, on the accuracy objective. Considering the KV service, we observe that there is only a marginal gain in accuracy when increasing k beyond 32 or 64. Limiting k to such a number incurs a gain in computing time of more than an order of magnitude. In the case of the VoD service, the value for k should be chosen between 8 and 16 if the constraints permit this. The computing time given in the Figure was measured on the "tenth machine" described in Section III.

Table II shows the value of k that we chose for our testbed environment, together with accuracy, training time for computing the prediction model (random forest), and the time for computing the dimensionality reduction (feature selection for all values of k). Following the argument given above, we chose one value of k for the VoD service, and another value for the KV service.

For reference and comparison, Table III gives accuracy and training time for prediction on the full feature space X. The training time in this table is same as the reduction time in Table II since the feature selection method we use in this work is based on the same algorithm (see Section II).

If we compare the information in Table II with the information in Table III, we can see that it is possible to achieve a reduction in prediction error by applying feature reduction before learning the prediction model. As we have seen before, the reduction in prediction error is larger for the VoD service than for the KV service. When comparing the computational overhead, we have to compare the sum of the model training time and the dimensionality reduction time in Table II with the model training time in Table III. For the VoD service, the

computational overhead is comparable, for the KV service, the overhead is twice as much, if we choose feature selection than if we perform model computation on the full feature set. For both services, the monitoring overhead is significantly reduced if feature selection is applied.



(a) VoD (b) KV

Fig. 4. Feature selection: Tradeoff between accuracy and overhead in learning the prediction model. Vertical axis is prediction error (NMAE), horizontal axis is computing time. Labels on the curves indicate subspace dimensionality k.

TABLE II
FEATURE SELECTION: VALUE OF K CHOSEN FOR OUR TESTBED, TOGETHER WITH ACCURACY AND TRAINING TIME, FOR ALL SCENARIOS. K WAS CHOSEN THROUGH FEATURE SELECTION.

Trace	k	Error		Model train time [sec]	Dim reduce time [sec]
		NMAE	R^2		
VoD periodic	16	0.08	0.62	0.79	37
VoD flash-crowd	16	0.05	0.71	1.24	41
KV periodic	256	0.02	0.72	19.18	21
KV flash-crowd	256	0.018	0.72	15.12	17

TABLE III
ACCURACY AND TRAINING TIME FOR PREDICTING FRAME RATE AND RESPONSE TIME ON THE FULL FEATURE SPACE X.

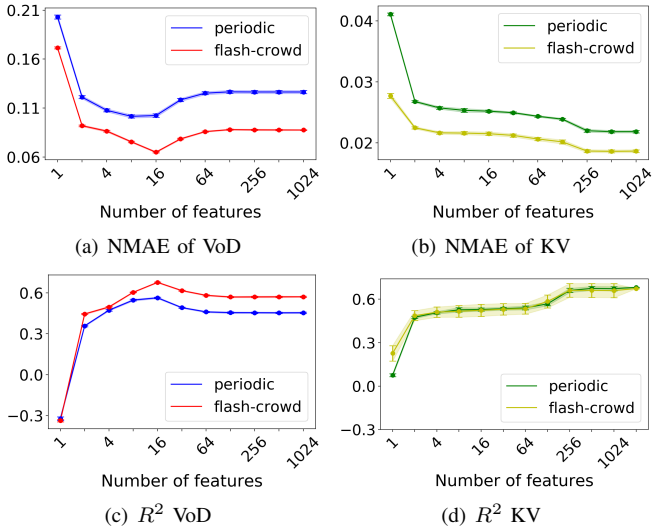
Trace	Error		Model train time [sec]
	NMAE	R^2	
VoD periodic	0.12	0.43	36.6
VoD flash-crowd	0.08	0.56	41.2
KV periodic	0.022	0.68	20.9
KV flash-crowd	0.02	0.67	17.4

V. DIMENSIONAL REDUCTION THROUGH PRINCIPAL COMPONENT ANALYSIS (PCA)

In this section we apply PCA to reduce the dimensionality of the feature space (see section II) and learn the prediction model on the lower dimensional space. For computation, we use the PCA class of the Scikit library. The class is based on the LAPACK implementation of full SVD or a randomized truncated SVD, depending on the input data and the value of k [23].

Question 1: How does prediction accuracy depend on k?

Figure 5 gives the result. When comparing this figure to Figure 3, we observe that the curves in both figures are very similar: the VoD service and the KV service exhibit very different performance properties; for the KV service, the error monotonically decreases with increasing k; in contrast, for the VoD service, there is small k which provides better accuracy



(a) NMAE of VoD (b) NMAE of KV

(c) R^2 VoD (d) R^2 KV

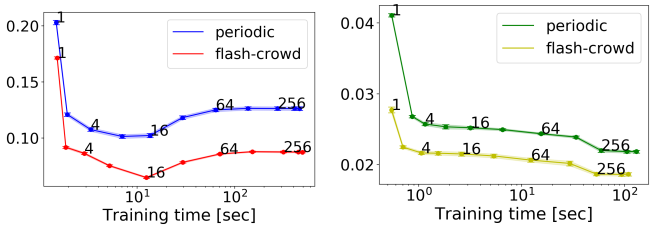
Fig. 5. PCA: Frame rate and response time prediction on the subspace spanned by the top k features. Vertical axis is accuracy, horizontal axis is dimensionality of subspace.

than the full feature set; for different load patterns of the same service, the curves show similar derivatives.

Question 2: What is the tradeoff between prediction accuracy and overhead for learning the prediction model?

Figure 6 provides the results, which show the same qualitative behavior for PCA as we observed for feature selection (see Figure 4). Table IV gives the values of k which we consider suitable for our testbed. Compared to the results from feature selection (Table II), we observe a small decrease in accuracy, which we explain by the fact that our feature selection method explicitly considers the target, while PCA does not.

When comparing Table IV with Table III, we find an increase in accuracy, but no gain in computational overhead, similar to what we observed for the feature selection method (see section IV).



(a) VoD (b) KV

Fig. 6. PCA: Tradeoff between accuracy and overhead in learning the prediction model. Vertical axis is a prediction error (NMAE), horizontal axis is computing time. Labels on the curves indicate subspace dimensionality k.

VI. DIMENSIONALITY REDUCTION THROUGH AUTOENCODER

The third method we apply for dimensionality reduction is autoencoder (See section II). We use the Keras library to implement an autoencoder neural network [24]. The architecture of this neural network is simple: it includes a sequence of

TABLE IV
PCA: VALUE OF k CHOSEN FOR OUR TESTBED, TOGETHER WITH ACCURACY AND TRAINING TIME, FOR ALL SCENARIOS. k WAS CHOSEN THROUGH FEATURE SELECTION.

Trace	k	Error		Model train time [sec]	Dim reduce time [sec]
		NMAE	R^2		
VoD periodic	16	0.10	0.56	13.56	11
VoD flash-crowd	16	0.06	0.68	12.48	12
KV periodic	256	0.022	0.66	59.15	8
KV flash-crowd	256	0.019	0.66	53.35	6

layers, namely, input layer, hidden layer, code layer, hidden layer, and output layer. All five layers are fully connected. The size of the input and output layers corresponds to the dimensionality of the feature space X, and the size of code layer is k. The size of the hidden layer is 16 nodes for the VoD service and 256 nodes for KV service. The activation function of the code layer and second hidden layers is *tanh*, and the activation function for the first hidden layer and the output layer is *linear*. We initialize the weights of the first and last layer with the computed transformation matrix from the PCA method. During the experiments, we vary the size of code layer $k = 1, 2, ..., dim(X)$, where $dim(X)$ denotes the dimensionality of X space. As part of this work, we investigated various architectures for autoencoders, different types of activation functions and several methods for initialization. All these alternatives produced very similar results than those given in the figures and tables below.

Question 1: How does prediction accuracy depend on k?

Figure 7 shows the accuracy of predicting the frame rate and the response time for different values of k. The figure shows a monotonic decline of the prediction error when k increases, for both the VoD and the KV service. This behavior suggests that increasing dimensionality improves the knowledge of the prediction model and thus improves accuracy. Keep in mind that autoencoders produce nonlinear mappings which can describe a larger class of systems than linear mappings can.

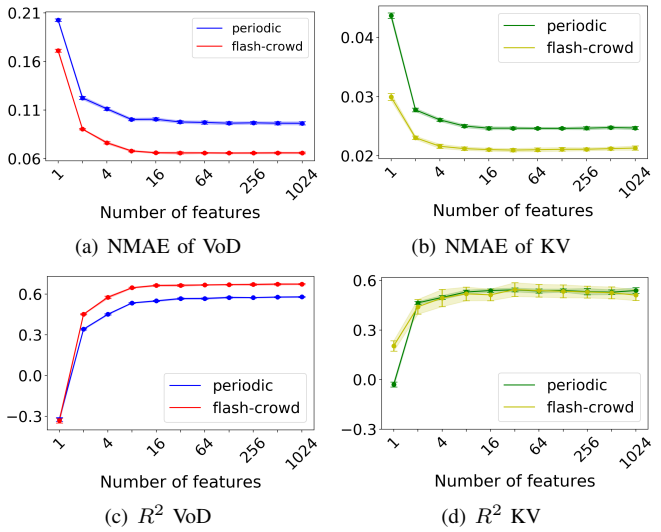
Question 2: What is the tradeoff between prediction accuracy and overhead for learning the prediction model?

Figure 8 shows the graphs of accuracy versus training time of the prediction model. We observe that increasing k causes the training time to grow significantly for both services. An increase of k beyond 8 or 16 results in a very small gain in accuracy and incurs a significant increase in computation time.

Table V contains the value of k that is suitable for our testbed. Comparing the results of the prediction accuracy of these models with those of the feature selection and PCA methods, we find that the latter two methods are more accurate. The reason for this is the fact that autoencoders are known to be hard to train [4].

VII. RELATED WORK

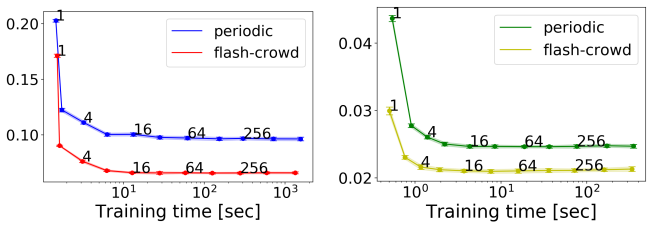
As we mentioned before, dimensionality reduction techniques have been intensively studied in the fields of machine learning and data science, and their application is well understood for cognitive processes like image recognition, natural



(a) NMAE of VoD (b) NMAE of KV

(c) R^2 VoD (d) R^2 KV

Fig. 7. Autoencoder: Frame rate and response time prediction on the subspace spanned by the top k features. Vertical axis is accuracy, horizontal axis is dimensionality of subspace.



(a) VoD (b) KV

Fig. 8. Autoencoder: Tradeoff between accuracy and overhead in learning the prediction model. Vertical axis is prediction error (NMAE), horizontal axis is computing time. Labels on the curves indicate subspace dimensionality k.

TABLE V
AUTOENCODER: VALUE OF k CHOSEN FOR OUR TESTBED, TOGETHER WITH ACCURACY AND TRAINING TIME, FOR ALL SCENARIOS. k WAS CHOSEN THROUGH FEATURE SELECTION.

Trace	k	Error		Model train time [sec]	Dim reduce time [sec]
		NMAE	R^2		
VoD periodic	16	0.10	0.57	13.27	538
VoD flash-crowd	16	0.07	0.67	12.82	511
KV periodic	256	0.025	0.53	76.38	507
KV flash-crowd	256	0.021	0.53	72.19	332

language processing, etc. The manifold hypothesis brought up in the introduction section reflects this understanding. In networked systems engineering however, these techniques have rarely been used to date. However, we expect they will become more prominent as network functions, driven by operational data and machine-learning techniques, will be gradually introduced.

Tao et al. predict a quality-of-experience metric from network statistics for a video service [25]. The feature space they consider has 89 metrics. On this space, they perform (supervised) feature selection using a heuristic search method on the graph of all subsets of the complete feature set. The

stated reason for feature selection is to eliminate redundancy of information and non-relevant features. Their method is shown to be effective regarding accurate target prediction. Compared to our work however, they do not investigate the tradeoff between prediction accuracy and overhead. Also, they do not investigate the relationship between prediction accuracy and the dimensionality of the subspace.

In [26] the authors propose an unsupervised feature selection method called REC-FSA for predicting the end-to-end performance of a network path from network features in a multi-hop wireless sensor network. The features relate to physical and MAC layer measurements, the path length, the temperature of the originating node, etc. The feature reduction method is based on computing the representation entropy of the feature matrix and on clustering and has a complexity of $O(m^3)$ where m is the number of features. They evaluate the method using small deployments in a rural and an industrial environment. They show that their method can achieve a reduction of from the original feature set of 17 to a reduced set of 3-5 features, while still maintaining good prediction accuracy. Compared to our work, the authors do not systematically investigate the relationship between the dimensionality of the reduced feature space and the achievable prediction accuracy.

PCA methods have been successfully used for detecting traffic anomalies in networked systems (see e.g., [27]). In contrast to our paper, the goal of these works was not dimensionality reduction, but the identification of patterns in subspaces of low rank.

In addition to autoencoders, there is a range of techniques available that provide non-linear mappings to a smaller feature space for various error objectives. A solid overview is provided in [28].

VIII. Conclusions and Future Work

Figure 9 illustrates well some of the insights we gained from the work reported in this paper. First, we find that we can perform more accurate prediction of service-level metrics from (low-level) infrastructure measurements if we learn on a significantly smaller subspace than on the original feature space X. This applies to both services, all scenarios, and all dimensionality reduction methods we investigated. (Figure 9 includes only the curves for VoD periodic, but the three other scenarios support this observation.) Second, due to the capability to learn general non-linear mappings, autoencoder allows for more accurate prediction than feature selection or PCA (for $k \geq 32$). The exception is $k < 32$ where our autoencoder configuration leads to worse prediction. As we discussed in Section VI, the reason for bad performance in this instance stems from the well-known difficulty to configure and train autoencoders [4]. We expect that additional efforts by us and others will eventually produce autoencoder configurations that perform better than PCA across all values of k. Third, the computational overhead for dimensionality reduction is highest for autoencoder, namely, some ten times larger than PCA or feature selection in our case. While PCA and feature selection incur comparable overhead, note though that feature selection relates to a specific target and PCA is target-independent. As a consequence, in a use case where the targets are not known at design time or are numerous, PCA should be preferred over (supervised) feature selection.

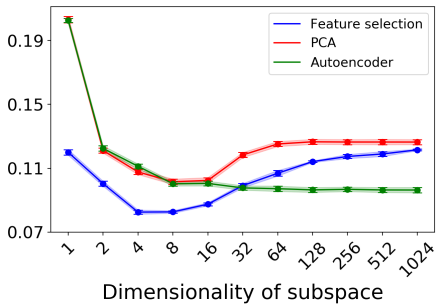


Fig. 9. Comparing prediction accuracy for different dimensionality reduction methods. Curves relate to VoD service and periodic load pattern. Vertical axis shows prediction error (NMAE).

Future work will include additional efforts in studying and refining dimensionality-reduction techniques which improve on the performance reported in this paper and which prove suitable for networked systems.

More importantly, our future work will take into consideration the architecture for computing prediction models and for performing real-time predictions, as well as the constraints of the the specific use case. Such an architecture may be centralized in a cloud-computing case, a one-level hierarchy in an edge computing scenario, or completely decentralized in the case of an urban sensor network. The potential benefits of automated dimensionality reduction for data-driven network functions have been shown in this paper, but the choice of a specific technique and its integration into a learning architecture will depend on the particular use case and requires thorough investigation.

Acknowledgements

The authors are grateful to Erik Ylipää with RISE AI, as well as to Andreas Johnsson, and Christofer Flinta with Ericsson Research for fruitful discussion around this work. This research has been partially supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through project AutoDC and by the KTH Software Research Center CASTOR.

References

[1] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "Predicting Service Metrics for Cluster-based Services using Real-time Analytics," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 135–143.

[2] S. Handurukande, S. Fedor, S. Wallin, and M. Zach, "Magneto approach to qos monitoring," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 209–216.

[3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] R. Stadler, R. Pasquini, and V. Fodor, "Learning from network device statistics," *Journal of Network and Systems Management*, vol. 25, no. 4, pp. 672–698, 2017.

[6] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "Predicting Real-time Service-level Metrics from Device Statistics," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 414–422.

[7] V. N. Vapnik, *Statistical learning theory*. John Wiley, 1998.

[8] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York Inc., 2006.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[10] NTP, 2016. [Online]. Available: http://www.ntp.org/

[11] VLC, 2016. [Online]. Available: http://www.videolan.org/vlc/

[12] HAProxy, 2016. [Online]. Available: http://www.haproxy.org/

[13] Apache HTTP Server, 2016. [Online]. Available: http://httpd.apache.org/

[14] FFmpeg, 2016. [Online]. Available: https://www.ffmpeg.org/

[15] Gluster FS, 2016. [Online]. Available: http://www.gluster.org/

[16] Voldemort, 2016. [Online]. Available: http://www.project-voldemort.com/voldemort/

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

[18] T. Bowden, B. Bauer, J. Nerin, S. Feng, and S. Seibold, "The /proc Filesystem," *Linux Kernel Documentation*, 2000.

[19] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing Flash Crowds on the Internet," in *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*. IEEE, 2003, pp. 246–249.

[20] D. Cournapeau, "Scikit learn," 2018. [Online]. Available: http://scikit-learn.org/stable/modules/feature_selection.html

[21] Mldata, 2018. [Online]. Available: http://mldata.org/repository/data/viewslug/realm-cnsm2015-vod-traces/

[22] R. Pasquini, "traces-jnsm-2017," 2018. [Online]. Available: https://github.com/rafaelpasquini/traces-jnsm-2017

[23] "Scikit learn," 2018. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[24] Keras, 2018. [Online]. Available: https://keras.io/

[25] 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 Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1337–1348, 2019.

[26] A. Panousopoulou, M. Azkune, and P. Tsakalides, "Feature selection for performance characterization in multi-hop wireless sensor networks," *Ad Hoc Networks*, vol. 49, pp. 70–89, 2016.

[27] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *ACM SIGCOMM computer communication review*, vol. 34, no. 4. ACM, 2004, pp. 219–230.

[28] J. A. Lee and M. Verleysen, *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.