IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796928

Policy-Induced Unsupervised Feature Selection: A Networking Case Study

Jalil Taghia*, Farnaz Moradi*, Hannes Larsson*, Xiaoyu Lan*, Masoumeh Ebrahimi*‡, and Andreas Johnsson*†
* Ericsson Research, Sweden, Email: *{jalil.taghia, farnaz.moradi, hannes.larsson, xiaoyu.lan, andreas.a.johnsson}@ericsson.com*
† Uppsala University, Department of Information Technology, Sweden, Email: *andreas.johnsson@it.uu.se*
‡ KTH, Division of Electronics and Embedded Systems, Sweden, Email: *mebr@kth.se*

***Abstract*—A promising approach for leveraging the flexibility and mitigating the complexity of future telecom systems is the use of machine learning (ML) models that can analyze the network performance, as well as taking proactive actions. A key enabler for ML models is timely access to reliable data, in terms of features, which require pervasive measurement points throughout the network. However, excessive monitoring is associated with network overhead. Considering domain knowledge may provide clues to find a balance between overhead reduction and meeting requirements on future ML use cases by monitoring just enough features. In this work, we propose a method of unsupervised feature selection that provides a structured approach in incorporation of the domain knowledge in terms of policies. Policies are provided to the method in form of must-have features defined as the features that need to be monitored at all times. We name such family of unsupervised feature selection the *policy-induced unsupervised feature selection* as the policies inform selection of the latent features. We evaluate the performance of the method on two rich sets of data traces collected from a data center and a 5G-mmWave testbed. Our empirical evaluations point at the effectiveness of the solution.**

I. Introduction

Telecommunication operators deliver services under strict service-level agreements. It is well-known that operations and management of such systems are challenging and demanding. A promising approach that is receiving extensive attention in academia and industry is the use of methods and models, trained using machine learning (ML), that can predict and analyze the network performance, as well as taking reactive and proactive re-configuration actions [1], [2]. The ability to learn such models simplifies and improves a vast number of management and operational tasks in a telecom infrastructure ranging from spectrum management and beamforming, resource and slice orchestration, service assurance, energy efficiency optimization, and root-cause analysis. Real-time requirements on the models will guide where they will be positioned, and hence executed, as illustrated by the network sketch in Figure 1.

A key enabler for machine learning models is timely access to reliable data both for model training and real-time inference, thus requiring pervasive measurement points throughout the network; for example in the remote-radio heads, the basebands, the core network, and central data centers, as exemplified by Figure 1. Furthermore, there is a clear research trend in addressing the need for data. For example, data centers are equipped with far more measurement points than ever

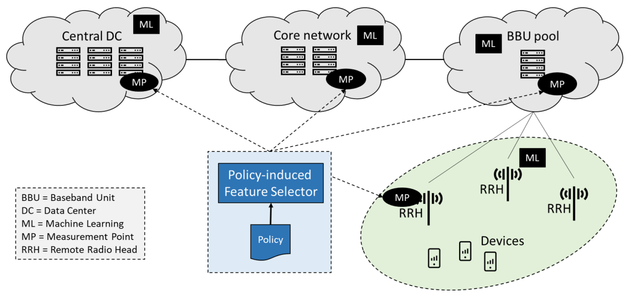


Fig. 1. Schematic view of an edge telecom network, including measurement points and AI/ML functions for operations and management. The contribution of the paper, the policy-induced feature selector, is colored in blue.

before [3] in an attempt to enable more effective, efficient, and environmentally friendly operations. Unfortunately, excessive data collection for fulfilling the needs of an increasing number of ML functions and transfer of data from a measurement point to its consumer come with overhead cost. Processing of many features, which may contain redundant features, increases the complexity of ML models. The added complexity increases the need for extra compute resources which can adversely affect the network and performance of its services [4]. Furthermore, the process of training machine learning models may also be negatively affected by an excessive amount of metrics (or features), leading to a reduced model performance as well as increased model complexity imposing challenging requirements on data availability and compute power on nodes in the infrastructure that may not have such capabilities.

Statistical feature selection learning methods have been extensively studied in the literature for supervised and unsupervised problems which can potentially mitigate some of the aforementioned challenges [5]. However, for the purpose of overhead reduction, supervised feature selection (SFS) may not be suitable. This is due to the fact that the selected features are specific to the task underlying the use case, and they may not generalize well to other use cases - effectively implying that it will be necessary to continuously measure all features. The goal of feature selection for overhead reduction shall be defined as selecting features that can support existing use cases and future use cases, for example by re-configuring measurement points. In this regard, unsupervised feature selection (UFS) is well positioned as the lack of an explicit task can help selection of the features that are potentially generalizable for a wide range of use cases. Success of a UFS method in selecting generalizable features depends on the choice of the objective function based upon which the method has been constructed. Defining an objective function for the UFS which could lead to selection of generalizable features remain an open research problem. However, we argue here that the effective inclusion of domain knowledge in UFS can help reduce the difficulty of the problem. Domain knowledge can be seen in the form of policies that guide the learning towards selection of generalizable features. The policies can be reflective of our domain knowledge in managing a network, where the domain knowledge may be provided by domain experts or acquired from previous supervised ML experiences. Policies can alternatively be reflective of certain regulations. In any case, the assumption here is that these policies are either useful for the support of the existing use cases or potentially useful for the support of future use cases.

In light of this, we introduce the family of policy-induced unsupervised feature selection (policy-induced UFS) where polices inform selection of the latent features. Specifically, we propose a method of policy-induced UFS named policy-induced Concrete autoencoder (PI-CAE) which is based on the standard Concrete autoencoders [6]. PI-CAE takes a set of policies dictating the features that must be monitored in the infrastructure, named *must-have* features. It then selects the latent features that are complementary to the policies - the complementary features are the ones that do not carry redundant information and together with the policies enable solving various ML use cases. The method objective is to ensure that the set of selected features supports existing and potentially future use cases while allowing inclusion of domain-specific features deemed relevant. The selected features are then communicated for configuration of the multiple measurement points across the infrastructure as indicated by Figure 1.

The main contributions of this paper are: (1) introducing the family of policy-induced UFS and a method in this family named *policy-induced Concrete autoencoder* (PI-CAE), and (2) an extensive evaluation of the method properties and performance using two rich set of data traces collected from a central data center environment and a 5G-mmWave testbed environment, representing two important parts of the telecom network as sketched in Figure 1.

The rest of the paper is organized as follows. Section II formally describes the problem formulation. Section III provides a brief background on Concrete autoencoders which is the basis of our method. Section IV describes the proposed method. Section V describes the experiments and corresponding results. Section VII covers related work on feature selection in networking. The paper ends with discussion in Section VI and conclusions in Section VIII.

II. Problem Formulation

The problem of feature selection has been studied in the context of supervised and, more recently, unsupervised feature selection (UFS). In supervised feature selection (SFS), the objective is fairly clear and that is to find an optimal set of features which are most useful for solving the supervised learning task underlying a use use. In contrast, in UFS, there is no explicit task. UFS can be seen as the feature selection problem where the task has been marginalized out (averaged out). That being said, selected features by a UFS method will be used for solving various real-world problems, including supervised learning problems.

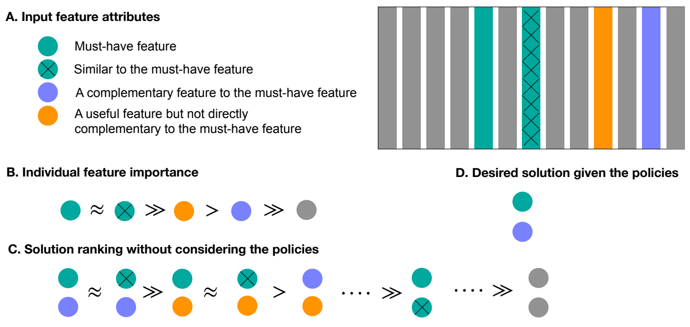

A. Input feature attributes
Must-have feature
Similar to the must-have feature
A complementary feature to the must-have feature
A useful feature but not directly complementary to the must-have feature
B. Individual feature importance
≈ ≫ > ≫
C. Solution ranking without considering the policies
≈ ≫ ≈ > .... ≫ .... ≫
D. Desired solution given the policies

Fig. 2. A conceptual figure describing unsupervised feature selection given policies. Given a must-have feature as the policy, the goal is to select a second important feature that best complements the must-have feature. (A) Input features and their descriptions. The green color-coded feature corresponds to the must-have feature that must be monitored. For the remaining features, there is no prior knowledge about their importance. (B) Individual importance of the features without taking into account interactions among them. (C) Ranking of the solutions considering the interactions of the features into account. (D) The desired solution. The solution contains the purple color-coded feature which complements the must-have feature.

The lack of an explicit task in UFS makes the problem challenging. Incorporation of domain knowledge can help reduce the difficulty of the problem. Domain knowledge in this context is in the form of policies that specify the features that must be monitored, named *must-have* features. The must-have features are specified, for example, by a domain expert or from previous supervised ML experiences.

In practice, our domain knowledge may be limited. We may know of individual features that are important and should be monitored. However, we may not know which combination of the features are complementary to each other - as this is the interactions between a subset of features that often prove useful for solving real-world problems, such as in network management. Furthermore, primarily relying on domain experts might lead to a biased selection of the features.

An ideal feature selection method should be able to use the provided policies by domain experts as “cues” in order to discard the latent features that carry similar information as the must-have features, and infer latent features that are complementary to them. Refer to Figure 2 for an illustrative conceptual example.

Problem statement: Let $\mathcal{D}$ of cardinality D denote a set that includes all features, and let $\mathbf{x} \in \mathbb{R}^D$ be a D-dimensional vector of the measured features on the real continuous space $\mathbb{R}$, shown as $\mathbf{x} = (\mathrm{x}_d : d \in \mathcal{D})^\top$ where x_d is the input measurement corresponding to the feature d. We have collected N such measurements in a $D \times N$-dimensional matrix $\mathbf{X}_\mathcal{D} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$, where $\mathbf{x}^n, \forall n \in \{1, \ldots, N\}$, is a

D-dimensional observation vector at the n-th time instance.

Let $\mathcal{M} \subset \mathcal{D}$ of cardinality $M < D$ be the *observed* set of features corresponding to the policies. The set $\mathcal{M}$ includes all *must-have* features. The goal is to find a *latent* set $\mathcal{K}$ of features of cardinality K where $\mathcal{K} \subset \mathcal{D}$ and $\mathcal{K} \cap \mathcal{M} = \emptyset$.

There are two ways to approach this problem:

(i) unsupervised feature selection given policies where the *policies $\mathcal{M}$ do not play a role in selection of the latent features $\mathcal{K}$*, and

(ii) policy-induced unsupervised feature selection where *policies $\mathcal{M}$ guide selection of the latent features $\mathcal{K}$.*

The former may be seen as the standard approach and the latter is the proposed approach of policy-induced unsupervised feature selection for which we develop a method named policy-induced Concrete autoencoders. Accordingly, in the experiments, we directly compare the above two approaches where the primary goal is to show the importance of effective incorporation of domain knowledge in the learning.

III. Background: Concrete Autoencoders

Recently, authors in [6] proposed a method of UFS, referred to as the *Concrete autoencoder*, which resembles to that of a standard autoencoder [7]. While in autoencoders, the goal is representation learning from input features, in the Concrete autoencoder, the goal is feature selection. Instead of the encoder, the Concrete autoencoder uses a *Concrete selector layer* which selects a stochastic linear combination of the input features as opposed to encoding the input features as in the encoder unit of the autoencoder. Furthermore, just as in autoencoders, the Concrete autoencoder has a decoder which takes as the input the filtered version of the input features and outputs a reconstructed version of the input features.

The objectives of UFS using Concrete autoencoders are finding a latent feature set $\mathcal{K}$ and a reconstruction function $f_\theta : \mathbb{R}^K \to \mathbb{R}^D$ with the parameter set θ, that minimize a loss ℓ defined as

$$\ell = \mathfrak{L}\left(\mathbf{X}_\mathcal{D}, f_\theta(\mathbf{X}_\mathcal{K})\right), \tag{1}$$

where $\mathbf{X}_\mathcal{K}$ is the filtered version of the input features $\mathbf{X}_\mathcal{D}$, and $\mathfrak{L}$ is a *reconstruction loss* that measures dissimilarities between $\mathbf{X}_\mathcal{D}$ and its reconstructed version $f_\theta(\mathbf{X}_\mathcal{K})$.

In the following, we briefly describe the functionality of the Concrete selector layer, as this will serve as a basis for our proposed solution. Refer to [6] for additional details on the Concrete autoencoders.

Concrete selector layer: The selector layer makes an explicit use of the Concrete latent variables [8]. Seen as a module, it takes as its inputs the features $\mathbf{X}_\mathcal{D}$ and a user-specified number of latent features K. It then outputs a filtered version of the input features $\mathbf{X}_\mathcal{K}$, where $\mathcal{K} \subset \mathcal{D}$.

Let $\mathbf{Z} = \left(\mathbf{z}^1, \ldots, \mathbf{z}^K\right)$ be a $D \times K$-matrix of *Concrete latent variables* where $\mathbf{z}^k = \left(\mathrm{z}_d^k : d \in \mathcal{D}\right)^\top$ denotes the k-th latent variable vector and z_d^k denotes its d-th element. A Concrete latent variable $\mathbf{z}$ follows a *Concrete distribution* with a location vector $\boldsymbol{\alpha} = \left(\alpha_d : d \in \mathcal{D}\right)^\top, \forall \alpha_d \in \mathbb{R}_{>0}$, and a scalar temperature $\tau \in \mathbb{R}_{>0}$, that is $\mathbf{z} \sim \mathrm{Concrete}(\boldsymbol{\alpha}, \tau)$,

$$p(\mathbf{z}; \tau, \boldsymbol{\alpha}) = (D-1)!\tau^{D-1} \prod_{d=1}^{D} \left(\frac{\alpha_d \mathrm{z}_d^{-\tau-1}}{\sum_{d'=1}^{D} \alpha_{d'} \mathrm{z}_{d'}^{-\tau}} \right). \tag{2}$$

Use of the Concrete latent variables crucially allows for differentiation with respect to $\boldsymbol{\alpha}$ via the *reparametrization trick* [8]. Accordingly, the Concrete selector layer is constructed as:

$$\widetilde{\mathrm{x}} = \sum_{d \in \mathcal{D}} \mathrm{z}_d \mathrm{x}_d, \quad \mathrm{z}_d \overset{\text{dist}}{=} \phi_d(\boldsymbol{\alpha}, \tau), \tag{3}$$

where $\overset{\text{dist}}{=}$ denotes the equality in distribution, z_d is the d-th element of a sample vector from the stochastic representation of the Concrete distribution given by:

$$\phi_d(\boldsymbol{\alpha}, \tau) = \frac{\exp\left(\log \alpha_d + \xi_d / \tau\right)}{\sum_{d' \in \mathcal{D}} \exp\left(\left(\log \alpha_{d'} + \xi_{d'}\right) / \tau\right)}, \tag{4}$$

and ξ is an i.i.d. sample vector from a standard Gumbel distribution, that is $\xi \sim \mathrm{Gumbel}(0, 1)$.

For temperature values greater than zero, the expression in Equation (3) is a stochastically weighted linear combination of the input features $\mathbf{x} = \left(\mathrm{x}_d : d \in \mathcal{D}\right)^\top$. However, as the temperature approaches zero, the selector layer essentially outputs only one of the input features,

$$\tau \to 0 \Rightarrow \begin{cases} \mathrm{z}_{d^*} \to 1, \\ \mathrm{z}_{d'} \to 0, \ \forall d' \neq d^*, \quad \forall d', d^* \in \mathcal{D}, \end{cases} \Rightarrow \widetilde{\mathrm{x}} \approx \mathrm{x}_{d^*},$$

where d^* is the index of the selected feature. This is known as the zero-temperature property of the Concrete random variables [8].

The selector layer is applied K times. The indices of the selected features are stored in $\mathcal{K}$. Accordingly, the filtered version of the input features is a $K \times N$-dimensional matrix which is shown as $\mathbf{X}_\mathcal{K} = (\mathbf{x}^1, \ldots, \mathbf{x}^N)$ where $\mathbf{x}^n = \left(\mathrm{x}_k^n : k \in \mathcal{K}\right)^\top$.

IV. Policy-Induced Unsupervised Feature Selection

Returning to our problem formulation in Section II, given the set of observed policies $\mathcal{M} \subset \mathcal{D}$, the objective is to infer the latent feature set $\mathcal{K} \subset \mathcal{D}$ from the input features $\mathbf{X}_\mathcal{D}$ such that the policies guide selection of latent features. For this purpose, we introduce an adaptation of the Concrete autoencoder, named *policy-induced Concrete autoencoder* (PI-CAE).

The policy-induced Concrete autoencoder consists of a selector layer named, *policy-induced Concrete selector* , and a decoder. The policy-induced Concrete selector is an adaptation of the Concrete selector layer described in Section III. We describe this unit in details in Section IV-A. Seen as a module, it takes as its inputs the set of observed policies $\mathcal{M}$, the input features $\mathbf{X}_\mathcal{D}$, and a user-specified number of latent features K. It then outputs a filtered version of the input features $\mathbf{X}_{\mathcal{K} \cup \mathcal{M}}$ where $\mathcal{K} \subset \mathcal{D}$ and $\mathcal{K} \cap \mathcal{M} = \emptyset$. Figure 3-(A) illustrates a high-level block-diagram representation of PI-CAE and the functionality of the policy-induced Concrete selector layer.

The decoder plays the role of a reconstruction function, $f_\theta : \mathbb{R}^{K+M} \to \mathbb{R}^D$, which takes as its input the filtered version

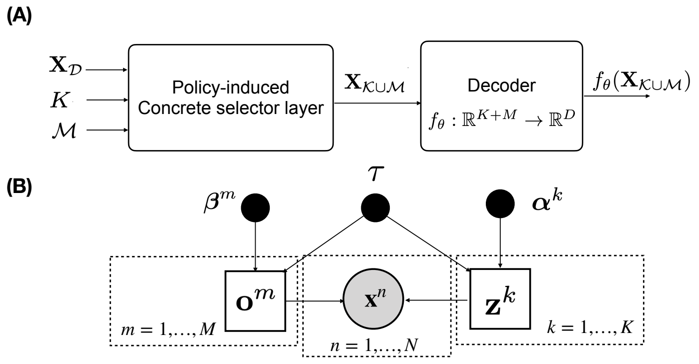

(A)
$\mathbf{X}_{\mathcal{D}}$
K
$\mathcal{M}$
Policy-induced Concrete selector layer
$\mathbf{X}_{\mathcal{K}\cup\mathcal{M}}$
Decoder
$f_\theta : \mathbb{R}^{K+M} \to \mathbb{R}^D$
$f_\theta(\mathbf{X}_{\mathcal{K}\cup\mathcal{M}})$
(B)
$\boldsymbol{\beta}^m$
τ
$\boldsymbol{\alpha}^k$
$\mathbf{o}^m$
$\mathbf{x}^n$
$\mathbf{z}^k$
$m = 1,\ldots,M$
$n = 1,\ldots,N$
$k = 1,\ldots,K$

Fig. 3. (A) Block diagram representation of the policy-induced Concrete autoencoder; (B) Probabilistic graphical representation of the policy-induced Concrete selector layer. Plates indicate replications, observations $\mathbf{x}^n$ are shown with a shaded circle, and latent variables $(\mathbf{z}^k, \mathbf{o}^m)$ are shown with open squares. The hyper-parameters $(\boldsymbol{\alpha}^k, \boldsymbol{\beta}^m, \tau)$ are shown with black circles.

of the input features $\mathbf{X}_{\mathcal{K}\cup\mathcal{M}}$ and produces a reconstructed version of the inputs. The decoder functionality and implementation are identical to the one in autoencoders.

In order to guide the selection of the latent features by the policies, we introduce a loss that discourages selection of the latent features with similar underlying distribution to the must-have features, by introduction of a regularization term to the reconstruction loss. The modified loss ℓ is formally defined in Section IV-B.

A. Policy-induced Concrete Selector Layer

Following a similar construction as the Concrete selector layer, let $\mathbf{Z} = \left(\mathbf{z}^1, \ldots, \mathbf{z}^K\right)$ be a $D \times K$-matrix of Concrete latent variables. Additionally, let $\mathbf{O} = \left(\mathbf{o}^1, \ldots, \mathbf{o}^M\right)$ be a $D \times M$-matrix of Concrete variables corresponding to the must-have policies, where $\mathbf{o}^m = (\mathrm{o}_1^m, \ldots, \mathrm{o}_D^m)^\top$ denotes the m-th feature vector and o_d^m denotes its d-th element. The Concrete variable $\mathbf{o}^m$ follows a Concrete distribution $\mathbf{o}^m \sim \mathrm{Concrete}(\boldsymbol{\beta}^m, \tau)$ with the temperature parameter τ and a location vector $\boldsymbol{\beta}^m$ defined as

$$\beta_d^m \to \begin{cases} 1, & \forall d = m, \\ 0, & \forall d \neq m. \end{cases} \tag{5}$$

The Concrete variables, $\mathbf{o}$, share the same temperature as in $\mathbf{z}$. In other words, all Concrete latent variables are tied to the same temperature τ.

We now construct the policy-induced Concrete selector layer as:

$$\widetilde{\mathrm{x}}^i = \begin{cases} \sum_{d\in\mathcal{D}} \mathrm{z}_d^i \mathrm{x}_d, \ \mathrm{z}_d \overset{\mathrm{dist}}{=} \phi_d(\boldsymbol{\alpha}^i, \tau), & i \in \mathcal{K}, \\ \sum_{d\in\mathcal{D}} \mathrm{o}_d^i \mathrm{x}_d, \ \mathrm{o}_d \overset{\mathrm{dist}}{=} \phi_d(\boldsymbol{\beta}^i, \tau), & i \in \mathcal{M}, \end{cases} \tag{6}$$

where $\phi(\cdot,\cdot)$ is given by Equation (4).

The filtered version of the input features is a $(K+M)\times N$-dimensional matrix which is shown as $\mathbf{X}_{\mathcal{K}\cup\mathcal{M}} := \widetilde{\mathbf{X}}$ where $\widetilde{\mathbf{X}} = (\widetilde{\mathbf{x}}^1, \ldots, \widetilde{\mathbf{x}}^N)$ and $\widetilde{\mathbf{x}}^n = (\widetilde{\mathrm{x}}_i^n : i \in \mathcal{K}\cup\mathcal{M})^\top$.

Figure 3-(B) illustrates the main components of the policy-induced Concrete selector and their relations using a probabilistic graphical representation.

B. Loss Computation

The lack of an explicit task in unsupervised feature selection makes defining the loss function challenging - more so than in supervised feature selection. One may consider the objective of the unsupervised feature selection as finding the optimal feature set that includes most interesting patterns in data as in [9]. Alternatively, one can take a coding-like approach and define the objective as finding the feature set that best approximates the data [6]. Last but not least, one can consider information theoretic approaches in defining the objective [10].

Each approach has its own strengths and weaknesses, and it ultimately depends on the application domain. In this work, without loss in generality, we adopt a coding-based loss similar to the one employed in autoencoders [7] and standard Concrete autoencoders [6]. However, we introduce an adaptation of this loss which includes a regularization term that discourages selection of the latent features with similar underlying distributions to the must-have features.

Accordingly, our choice of loss function in a general form is expressed as:

$$\ell = \mathfrak{L}_{\mathrm{rec}}\left(\mathbf{X}_{\mathcal{D}}, f_\theta(\mathbf{X}_{\mathcal{K}\cup\mathcal{M}})\right) + \mathfrak{L}_{\mathrm{reg}}(\mathbf{Z}, \mathbf{O}), \tag{7a}$$

where, as introduced in Section IV-A, $\mathbf{Z}$ is the matrix of Concrete variables responsible for the selection of the latent feature set $\mathcal{K}$, and $\mathbf{O}$ is the Concrete variable responsible for generation of the must-have features $\mathcal{M}$. The first term, $\mathfrak{L}_{\mathrm{rec}}$, is the reconstruction loss which measures how well the input features are being reconstructed for the selected subset of features. The second term, $\mathfrak{L}_{\mathrm{reg}}$, is a regularization loss which penalizes selection of the features with similar underlying distributions to those of the must-have features. The regularization loss is a non-positive quantity, that is $\mathfrak{L}_{\mathrm{reg}} \le 0$.

1) Reconstruction loss: Assuming a zero-mean Gaussian noise with the same variance σ^2 across all features and all measurements, the reconstruction loss is proportional to:

$$\ell_{\mathrm{rec}}\left(\mathbf{X}_{\mathcal{D}}, f_\theta(\mathbf{X}_{\mathcal{K}\cup\mathcal{M}})\right) = \frac{1}{2\sigma^2}\sum_{i=1}^{N} \left\|\mathbf{x}_i - f_\theta(\widetilde{\mathbf{x}}_i)\right\|^2, \tag{7b}$$

where $\mathbf{x}_i \in \mathbf{X}_{\mathcal{D}}$ and $\widetilde{\mathbf{x}}_i \in \mathbf{X}_{\mathcal{K}\cup\mathcal{M}}$. The noise variance in a general form can be assumed to be data-and-feature dependent and expressed via a covariance matrix that is learned from data. However, for simplicity, in all experiments, we assume $\sigma^2 = 1$.

2) Regularization loss: The regularization loss measures dissimilarities between the distribution of a selected feature $k \in \mathcal{K}$ and the policies $\mathcal{M}$. For this purpose, we use the Kullback-Leibler (KL) divergence as the choice of metric. The regularization loss is expressed as:

$$\begin{aligned} \mathfrak{L}_{\mathrm{reg}}(\mathbf{Z}, \mathbf{O}) &= -\frac{1}{K}\sum_{k\in\mathcal{K}}\sum_{m\in\mathcal{M}} \mathfrak{D}_{\mathrm{KL}}\left(p(\mathbf{o}^m) || p(\mathbf{z}^k)\right) \\ &\approx -\frac{1}{KR}\sum_{k\in\mathcal{K}}\Big(\sum_{m\in\mathcal{M}}\sum_{r=1}^{R} \log p(\mathbf{o}^{(m,r)}; \tau, \boldsymbol{\beta}^m) - \\ &\qquad \log p(\mathbf{z}^{(k,r)}; \tau, \boldsymbol{\alpha}^k)\Big), \end{aligned} \tag{7c}$$

Algorithm 1 Policy-Induced Concrete Autoencoder (PI-CAE)

```
1: Inputs:
     - Input features $\mathbf{X}_{\mathcal{D}}$.
     - The set of observed policies corresponding to the must-have-features $\mathcal{M}$.
     - A user-specified number of latent features $K$.
2: Initialization:
     - Initialization of the reconstruction function $f_\theta : \mathbb{R}^{K+M} \to \mathbb{R}^D$ with the parameter set $\theta$ (i.e., the decoder net). Typically the decoder net can be in form of a neural network or any other differentiable model.
     - Initialization of the location vector $\boldsymbol{\alpha}^k = (\alpha_d^k : d \in \mathcal{D})^\top, \forall \alpha_d^k \in \mathbb{R}_{>0}, \forall k$, with small random positive values.
     - Initialization of the initial temperature $\tau \in \mathbb{R}_{>0}$, shown as $\tau_0$, with a relatively large scalar positive value (e.g., $\tau_0 = 10$). Setting the final value of the temperature, shown as $\tau_T$, to a small positive value (e.g., $\tau_T = 10^{-2}$).
     - Initialization of the optimizer of choice $g_\lambda(\nabla_\theta \ell, \nabla_{\boldsymbol{\alpha}} \ell)$ where the $\lambda$ includes all optimizer parameters such as the learning rate. The optimizer $g$ takes as its inputs the gradients of the loss $\ell$ with respect to the parameters $\theta$ and $\boldsymbol{\alpha}$, and produces updated version of the parameters.
3: for $t = 1, \ldots, T$ do
4:     Adjust the temperature value $\tau$ using: $\tau \leftarrow \tau_0(\tau_T/\tau_0)^{t/T}$.
5:     for $k = 1, \ldots, K$ and $m \in \mathcal{M}$ do
             - Draw a random sample from $\mathbf{z}^k \sim \text{Concrete}(\boldsymbol{\alpha}^k, \tau)$.
             - Draw random samples from $\mathbf{o}^m \sim \text{Concrete}(\boldsymbol{\beta}^m, \tau)$.
             - Construct the policy-induced Concrete selector layer for the $i$-th input feature $\mathbf{x}_i^n$ by application of Equation (6). Repeat this step for all $n = 1, \ldots, N$, and next construct the filtered version of the input features $\widetilde{\mathbf{X}} = (\widetilde{\mathbf{x}}^1, \ldots, \widetilde{\mathbf{x}}^N)$ where $\widetilde{\mathbf{x}}^n = (\widetilde{\mathbf{x}}_i^n : i \in \mathcal{K} \cup \mathcal{M})^\top$.
6:     end for
7: end for
8: Compute loss $\ell$ using Equation (7a) given $f_\theta$ with the parameter set $\theta$.
9: Compute gradients of the loss with respect to $\theta$ and $\boldsymbol{\alpha}^k$, i.e., $\nabla_\theta \ell$ and $\nabla_{\boldsymbol{\alpha}^k} \ell, \forall k \in \mathcal{K}$.
10: Update the parameters: $\theta, \{\boldsymbol{\alpha}^k : k \in \mathcal{K}\} \leftarrow g_\lambda(\nabla_\theta \ell, \{\nabla_{\boldsymbol{\alpha}^k} \ell : k \in \mathcal{K}\})$.
11: Outputs: The set $\mathcal{K}$ which includes the index of the latent features and updated parameters, $\theta$ and $\{\boldsymbol{\alpha}^k : k \in \mathcal{K}\}$.
```

where $\mathfrak{D}_{\mathrm{KL}}(q_1||q_2)$ measures the deviation of q_2 from q_1. In this equation, R is the number of Monte-Carlo samples, and $p(\mathbf{o}^{(m,r)}; \tau, \boldsymbol{\beta}^m)$ and $p(\mathbf{z}^{(k,r)}; \tau, \boldsymbol{\alpha}^k)$ are computed using Equation (2). The KL-divergence $\mathfrak{D}_{\mathrm{KL}}$ is a non-negative quantity. The negative sign in Equation (7c) encourages selection of latent features which differ in their underlying distributions to the observed must-have features given by the policies. The interplay between the two competing objectives is similar to the one in variational autoencoders [11].

An algorithmic description of the policy-induced Concrete autoencoder is given in Algorithm 1.

V. Experiments

Experiments are in connection to the problem statement in Section II. The primary goal of the experiments is to empirically verify the effectiveness of the proposed solution in incorporation of domain knowledge in unsupervised feature selection. Hence, experiments are designed with that goal in mind.

A. Datasets

We consider two sets of data for the experiments, namely, data center (DC) traces from a testbed at KTH University [12] [13], referred to as the *KTH traces* and traces from an in-house 5GmmWave testbed, referred to as the *5G traces*.

1) KTH traces: The publicly-available KTH traces are collected from a testbed environment. The testbed consists of a server cluster and a set of clients. There are two services running on these machines, namely Video-on-Demand (VoD) and a Key-Value (KV) store (database). Traces are generated by executing testbed experiments with different configurations where statistics are collected every second; specifically it includes features, and service-level metrics. Features are collected from the server cluster and the service level metrics are collected on the client machines. Service-level metrics serve as the machine learning tasks.

TABLE I
Trace configurations and specifications for the KTH traces.

Trace Name	Service	Load Pattern	Number of Services	Number of Tasks
VoD-SingleApp-FlashCrowd	VoD	FlashCrowd	1	5
VoD-BothApps-FlashCrowd	VoD	FlashCrowd	2	5
VoD-SingleApp-Periodic	VoD	Periodic	1	5
VoD-BothApps-Periodic	VoD	Periodic	2	5
KV-SingleApp-FlashCrowd	KV	FlashCrowd	1	2
KV-BothApps-FlashCrowd	KV	FlashCrowd	2	2
KV-SingleApp-Periodic	KV	Periodic	1	2
KV-BothApps-Periodic	KV	Periodic	2	2

Table I summarizes traces used in the experiments. The trace name is encoded according to service under investigation (KV store or VoD services), number of concurrent services (either SingleApp or BothApps), and load pattern (either periodic or flashcrowd load). As an example, the following naming VoD-BothApps-Flashcrowd corresponds to a trace where both services, VoD and KV, are running under the flashcrowd load, while the service level metrics of VoD are being measured on the client side.

Features are collected from the Linux kernels that run on the server cluster machines. There are about 1700 features and examples of such are CPU utilization per core, memory

TABLE II

Summary of the tasks for KV and VoD services of KTH traces.

Acronym	Service	Description
noDispFrames	VoD	Number of display video frames
noAudioPlayed	VoD	Number of audio buffers
NetReadAveDelay	VoD	Net value of read average delay
NetReadOperations	VoD	Net value of read operations
NetReadBytes	VoD	Net value of read bytes
ReadsAve	KV	Average read latency
WritesAve	KV	Average write latency

TABLE III

Trace configurations and specifications for the 5G traces.

Trace Name	Traffic Load	RTT Averaging Interval	Number of Tasks
NoTraffic-RTT1000	None	1000 (ms)	2
NoTraffic-RTT100	None	100 (ms)	2
ConstantTraffic-RTT1000	Constant Bit Rate	1000 (ms)	2
ConstantTraffic-RTT100	Constant Bit Rate	100 (ms)	2

utilization, network utilization, and disk I/O.

The ML tasks using these traces are predicting service-level metrics given the features collected from the Linux kernels. ML tasks that are considered in the experiments for KV and VoD services are summarized in Table II.

2) 5G traces: The 5G traces are collected from an in-house 5G-mmWave testbed in which the equipment corresponds to a 5G non-standalone (NSA) system, where the control plane is served through a 4G LTE base station (eNB) and the user-plane is served through a 5G NR base station (gNB). The 4G LTE eNB operates on B3, 1800 MHz, with 5 MHz bandwidth. The 5G NR gNB operates on n257, 28 GHz, with 100 MHz bandwidth. The spectrum is time-shared between DL and UL using a 4:1 TDD pattern (*DDDSU* [14]). In this testbed we have two user equipments (UEs), one for performance measurements and the other for traffic load generation.

The traces obtained from this testbed contain round-trip time (RTT) values as experienced by a UE in the network, and approximately 200 metrics and events related to the analogue beamforming function, the UEs connected to the base station, and the uplink (UL) and downlink (DL) events [15]–[17].

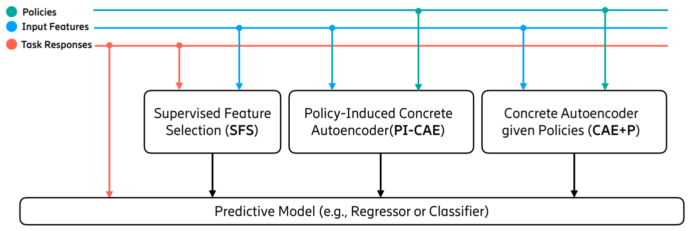
Two different experiments are performed, corresponding to *NoTraffic* and *ConstantTraffic* trace categories. In both experiments, the UEs were moving for a duration of 10 minutes. In one of the experiments, a constant bit rate traffic load (UL) was generated, while in the other experiment, no traffic load was added. The end-to-end round trip time (RTT) was measured every 10ms using ICMP ping [18] with a measurement packet size of 1400 bytes. Further, as a pre-processing step, the RTT and the base station metrics are averaged with different time intervals (e.g., every 100ms, 1000ms). In total, 636 features were generated from around 200 metrics based on different averaging intervals. The traces used in this paper are summarized in Table III. The trace name is encoded according to traffic load (either constant traffic or no traffic) and the averaging intervals (either 100ms or 1000ms).

In this paper, we consider scenarios where the data traces are used to predict end-to-end RTT as experienced by the UE, based on traces from the 5G-mmWave base station. The use

TABLE IV

Summary of the ML tasks for the 5G traces.

Acronym	Description
Low Threshold	A low RTT threshold value
High Threshold	A high RTT threshold value



Policies
Input Features
Task Responses
Supervised Feature Selection (SFS)
Policy-Induced Concrete Autoencoder(PI-CAE)
Concrete Autoencoder given Policies (CAE+P)
Predictive Model (e.g., Regressor or Classifier)

Fig. 4. A block diagram of the evaluation framework. Refer to Section V-B2 for additional details.

case is formalized as a binary classification problem based on different service-level metrics (low, high) corresponding to RTT thresholds[1] corresponding to service violations.The ML tasks using these traces are predicting service-level metrics. Table IV summarizes the ML tasks considered in the experiments.

B. Experiment Setup

1) Choosing the must-have features:

a) KTH traces: In real world scenarios, must-have features may be identified by an expert based on domain knowledge. However, here, we identify the must-have features for a given task via solving a supervised feature selection problem where the must-have features are the features with the highest importance. This means that must-have features are task specific and trace specific. This procedure reduces uncertainty around usefulness of the must-have features and facilitates validation of the proposed method.

b) 5G traces: In this case, must-have features are identified by the domain experts, and they remain the same across all traces and all tasks. There are four features identified as must-have, namely: uplink wide-beam RSRP, uplink narrow-beam RSRP, downlink wide-beam RSRP, and downlink narrow-beam RSRP [15]–[17].

2) Evaluation framework: The evaluation framework is designed in relation to the problem statement in Section II. The following methods are compared against each other:

(CAE+P) Latent features are inferred using the standard Concrete auto-selector as the method of unsupervised feature selection. A subset of features are constructed from the features representing the must-have policies and the latent features inferred by the Concrete auto-selector. Selected features are fed into the predictive model. In this construction, the policies do not play any role in selection of the latent features.

(PI-CAE) The proposed solution where features are inferred via the policy-induced Concrete autoencoder. Unlike CAE+P, the selection of the latent features in PI-CAE

[1]The exact values are confidential. However, the difference between low and high is large enough to lead to different distributions across outputs.

is guided by the policies. Selected features are fed into the predictive model.

(SFS) For defining an approximate upper bound on the performance, we use a supervised feature selection technique which makes use of the task response in order to select the most relevant features. We use Random Forest as the feature selection method. Once the most relevant features are selected for a given task, they are fed into the predictive model. Note that in the case of SFS, there are no constraints on the budget so that the number of relevant features is task-and-trace specific and optimized for achieving the best performance, which means that the SFS technique is not suitable for reducing measurement overhead.

Figure 4 visualizes the evaluation framework. The methods CAE+P and PI-CAE are initialized similarly using the same random seed.

3) Performance evaluation:

a) KTH traces: Experiment is repeated per task and per trace for 5 times. Data is divided into a training and a test set (about 50% of data). Regression performance is evaluated on the test set using mean-absolute error (MAE) between the true responses and predicted responses. The MAE is normalized by the mean of the true task responses and the normalized MAE (nMAE) is reported.

b) 5G traces: Experiment is repeated per task and per trace for 20 times. Data is divided into a training and a test set (about 30% of data). Classification performance is evaluated on the test set using AUC (area-under-the-curve using trapezoidal rule) between the true responses and the predicted responses.

C. Experimental Results

As discussed in Section V-B1, for the case of KTH traces, must-have features are trace-and-task specific, and they are chosen via a supervised technique. While for the case of 5G traces, must-have features are chosen by an expert, and the same set of features are used across all traces and all tasks. When must-have features are provided by a supervised technique, we have some quantitative measure of their importance. However, when they are given by a domain expert, we may have limited understanding of their importance, e.g., we may not know if they are the most important ones. One goal of the experiments is to examine how effectively PI-CAE can benefit from domain knowledge under different conditions.

1) KTH traces: We consider different scenarios by varying the budget specified in terms of the user-defined number of latent features K and the set of must-have features corresponding to the policies $\mathcal{M}$. Figure 5 summarizes the performance across all tasks per trace. Note that, here, the average nMAE values across tasks are reported[2].

The main observation is that for smaller number of latent features K, there is a clear advantage in using PI-CAE as within the limited budget it selects the features that are complementary to the must-have features. However, as the budget increases, that is for larger values of K, the advantage becomes less

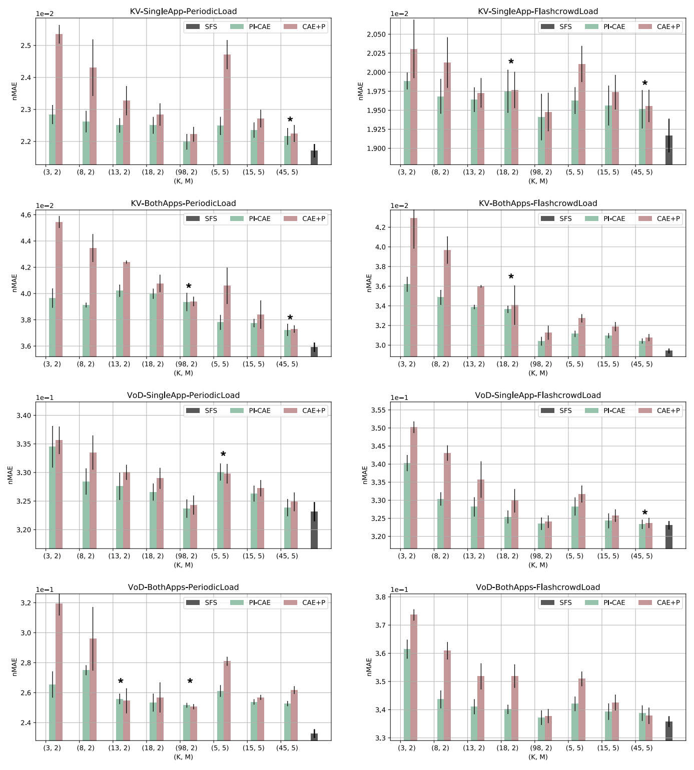


Fig. 5. Performance evaluation of methods SFS, PI-CAE and CAE+P for various number of latent features K and must-have features M. Performance is evaluated using nMAE scores between the true and predicted responses (lower values are preferred). The pair of (K, M) on the x-axis specify the condition at which PI-CAE and CAE+P are evaluated. Note that SFS serves as an approximate upper bound on the performance and its performance does not depend on K and M. The figure shows the average results across tasks per trace. Pairs with no statistical significance are annotated by a star symbol (*).

significant. The result is consistent for both scenarios of $M = 2$ and $M = 5$, and across all traces.

The result suggests that PI-CAE can achieve similar performance figures as the method CAE+P but crucially for smaller budgets. As an example, consider the trace KV-SingleApp-PeriodicLoad. For the case of $(K, M) = (98, 2)$, the performance of PI-CAE and CAE+P are fairly similar but as the budget decreases to $(K, M) = (3, 2)$, CAE+P performs poorly while PI-CAE performs reasonably well. Similar observations can be made across majority of the traces evaluated here.

A secondary observation is that, as expected, as the budget increases, the performance of both unsupervised methods PI-CAE and CAE+P improves, and given high enough budget, in some cases, their performance can approach to that of the supervised method SFS.

2) 5G traces: We consider different scenarios by varying the number of latent features K for the selected set of must-have features by the domain expert which remains the same across all traces and tasks. Figure 6 summarizes the performance across all tasks and per trace. In this figure, the AUC values across tasks are reported. The main observation is that in most scenarios PI-CAE can effectively make use of the domain knowledge provided by the policies. As in the case of the KTH

[2]Due to the lack of space, performance per task and per trace are not shown.

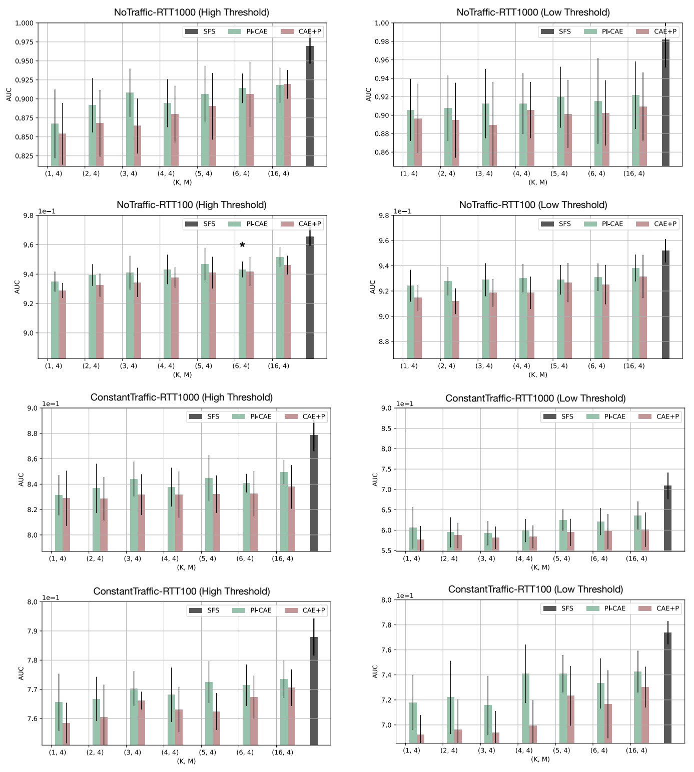


Fig. 6. Performance evaluation of methods SFS, PI-CAE and CAE+P for various number of latent features K and must-have features M. Classification performance is evaluated using AUC scores between the true and the predicted labels (higher values are preferred). The pair of (K, M) on the x-axis specify the condition at which PI-CAE and CAE+P are evaluated. Note that SFS serves as an approximate upper bound on the performance and its performance does not depend on K and M. The figure shows the average results per task (High Threshold and Low Threshold) and per trace. Pairs with no statistical significance are annotated by a star symbol (*).

traces, as the budget reduces the advantage of PI-CAE over CAE+P becomes more noticeable.

We also compute the average gain that PI-CAE achieves over CAE+P for each task across all traces and scenarios. For the task labeled as High Threshold, the average gain across 20 runs is $6.8\%(\pm1.1\%)$ and for the task labeled as Low Threshold, the average gain is $8.8\%(\pm0.8\%)$. The average gain for Low Threshold is statistically significantly larger than that of for High Threshold.

VI. Discussion

A. Policy-induced Concrete autoencoder

In Section I, we motivated the need for efficient incorporation of domain knowledge in unsupervised feature selection. Accordingly, we introduced the family of policy-induced unsupervised feature selection where domain knowledge in the form of policies inform the inference. Specifically, we developed policy-induced Concrete autoencoder, PI-CAE, which provides a structured way for incorporation of domain knowledge. PI-CAE takes a set of policies in the form of must-have features, it then selects the latent features that are importantly complementary to the must-have features.

From the method point of view, PI-CAE uses domain knowledge as cues and aims at finding the most complementary features to them by discarding both redundant features, i.e., features with similar underlying distributions to those of the must-have features, and features with minimal contributions in explaining data. This property of PI-CAE is useful in overhead reduction in network monitoring, in particular when we have resource constrained budgets, where the budget is specified by the maximum number of features that can be monitored.

Although, we verified the effectiveness of PI-CAE in networking use cases, we believe it can be applied to other case studies where efficient incorporation of domain knowledge is needed.

Finally, PI-CAE, as in most feature selection methods, does not have a data-driven mechanism for automatically inferring the number of latent features. Indeed the number of latent features are specified by the user. Such a problem is known as the automatic determination of the model complexity which can be seen as an exciting future research.

B. Systems view

As discussed in Section I, timely access to reliable data is critical for model training and real-time inference. Unfortunately, excessive monitoring leads to a situation where the network and other infrastructure components become overloaded thus reducing the capacity for services providing a business value. Thus there is a need for intelligent feature selection. On the other hand, there are anecdotal evidence indicating that black-box automation in general, including feature selection and its impact on the configuration of a monitoring system, meets resistance when incorporated into systems and processes for network management and operations. Trustworthiness, robustness, and explainability are often key aspects of criticism. This black-box challenge encountered in networking is also consistent with observations when using ML models for decision making in other fields [19]. The PI-CAE approach makes an attempt at balancing the need for intelligent feature selection that can meet future ML use cases while at the same time providing a level of control to the network operators, that is propagated into the running system.

From an architectural viewpoint, the proposed PI-CAE method could be integrated as a key functionality in the Open Radio Access Network (O-RAN) architecture, which is a concept based on interoperability and standardization [20]. The function would have an obvious place in e.g. the Network Management System and the Intelligent Controller layers to control the measurement points based on operator-defined policies. Note however that O-RAN is just one possibility, and thus, Figure 1 is very generic in its design.

VII. Related Work

Feature selection is an important step in building machine learning models with the aim of removing irrelevant, redundant, or misleading features from a given dataset while achieving a good generalization capability [5], [21], [22]. Feature selection techniques broadly can be categorized into supervised (e.g.,

[23], [24]) and unsupervised (e.g., [6], [9]) methods. While supervised learning incorporates information about the labels (or output responses) in the selection process, unsupervised learning does not rely on existence of such information. Supervised and unsupervised feature selections are used widely in application domains such as pattern recognition, and image processing, while they are receiving increasing attention in the areas of computer systems and networks. In this domain, feature selection could leverage network operations and service management while at the same time mitigate the overhead challenges related to the increasing amount of data created throughout the system.

Feature selection for networking: In general, the works discussed below can be divided into either feature selection or representation learning, where the former set of approaches maintains the explainability of a data-driven model. Further, these approaches can be used for controlling the monitoring infrastructure. Representation learning techniques on the other hand, such as principle component analysis, fail with respect to both these two important properties.

In [4], the authors describe a low-cost method of obtaining a sparse representation of the data collected at each individual server while preserving a specified fidelity with respect to the original signal. Further, in [10], [25], the authors introduce an unsupervised algorithm for online feature selection. The method is instantiated by a feature ranking algorithm. The algorithm does not consider changes in feature importance over time, which is a drawback in case the system enters a new unknown state. The work in [26] provides a comparison of multiple dimensionality reduction techniques with the objective to reduce the learning complexity while maintaining the prediction accuracy of a supervised task for network management. However, none of these techniques are specifically designed for reducing the network monitoring overhead.

In [27], the authors explore feature extraction and dimensionality reduction techniques for cellular networks in a fault diagnosis problem. The intended use case is deployment as a component between monitoring of network features and their usage in performance analysis in mobile networks.

A framework for dimensionality reduction, based on both feature selection and reduction, is presented in [28]. The framework is targeting radio networks and the self-healing processes. The method presented in this paper is complementary to this type of frameworks, giving the operator necessary tools for control in the dimensionality reduction process.

Classification of network traffic and flows has been an active research area for over two decades, e.g., [29], [30]. This type of functionality improves, for example, routing algorithms and decisions. Feature selection can improve the model performance also in this field. In [31], which is a recent example, the authors explore properties of sequential feature selection methods for training ML models for the classification of the traffic flows.

In [32], the authors propose a method for real-time prediction of QoE-relevant metrics for encrypted video traffic. They experimental evaluation showed that a feature selection method based on information gain ranking and manually selected threshold using expert knowledge can lead to similar prediction performance compared to the full feature set.

In [33], an ensemble feature selection method for malware detection is presented. The method selects the most discriminative features using labeled data and the most representative features using unlabeled data.

Importantly, none of the above discussed works consider the novel aspects of this paper, namely the capability of specifying operator requirements, using policies, on *must-have* features. Existing feature selection methods often ignore domain knowledge which can lead to removal of critical features for the application. A related work that considers domain knowledge in feature selection is discussed in [24]. The authors propose a multi-layer supervised feature selection method which integrates knowledge from domain experts by means of weighted scoring. After assigning importance scores to the features by the human experts, the features are evaluated with respect to sparsity, correlation, and redundancy to select features that are distinct and correlated with the target features. The method is heuristic and at times limited as the importance scores are computed based on the observational data from the features and not the underlying distributions of the features. As an example, observed instances of two arbitrary features might seem different from each other while in fact they share similar underlying distributions.

VIII. Conclusion

We introduced a new family of methods for unsupervised feature selection named *policy-induced unsupervised feature selection* in which domain knowledge guides selection of the latent features. Domain knowledge is in the form of a set of policies which are provided by domain experts or previous experiences. The policies dictate which features must be monitored in the infrastructure, referred to as the must-have features. In this regard, we introduced a method named policy-induced Concrete autoencoder PI-CAE. Conceptually, PI-CAE takes the must-have features as cues and selects the latent features that are complementary to the must-have features. We verified the effectiveness of our theoretical contributions on data traces collected from a central data center environment and a 5G-mmWave testbed environment. We showed that PI-CAE can effectively make use of the domain knowledge provided by the policies. Importantly, the benefit of using PI-CAE becomes more apparent when we have limited budget, that is when only a few number of features can be monitored. This property of the PI-CAE makes the method in particular attractive for overhead reduction in network management. Finally, although we discussed the method in a networking case study, we believe the method can be applied in other studies where effective incorporation of domain knowledge is needed.

Acknowledgment

This research has been supported by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through projects ITEA3 AutoDC and Celtic ANIARA, and Swedish Foundation for Strategic Research, SSF Strategic Mobility.

References

[1] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[2] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6g wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Vehicular Technology Magazine*, vol. 15, no. 4, pp. 122–134, 2020.

[3] J. Gustafsson, S. Fredriksson, M. Nilsson-Mäki, D. Olsson, J. Sarkinen, H. Niska, N. Seyvet, T. B. Minde, and J. Summers, "A demonstration of monitoring and measuring data centers for energy efficiency using opensource tools," in *Proceedings of the Ninth International Conference on Future Energy Systems*, 2018, pp. 506–512.

[4] S. DeCelles, M. Stamm, and N. Kandasamy, "Data reduction, compression, and recovery for online performance monitoring," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 2019, pp. 256–263.

[5] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[6] A. Abid, M. F. Balin, and J. Zou, "Concrete autoencoders for differentiable feature selection and reconstruction," in *ICML*, 2019.

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

[8] C. J. Maddison, A. Mnih, and Y. Teh, "The Concrete distribution: A continuous relaxation of discrete random variables," in *ICLR*, 2017.

[9] J. G. Dy and C. E. Brodley, "Feature selection for unsupervised learning," *Journal of Machine Learning Research*, vol. 5, no. 12, pp. 845—889, 2004.

[10] X. Wang, F. S. Samani, and R. Stadler, "Online feature selection for rapid, low-overhead learning in networked systems," *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1–7, 2020.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *ICLR*, 2014.

[12] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "A service-agnostic method for predicting service metrics in real time," *International Journal of Network Management*, vol. 28, no. 2, p. e1991, 2018.

[13] F. S. Samani, "Data traces for efficient learning on high-dimensional operational data," 2021.

[14] 3GPP, "NR; Physical layer procedures for control," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021, version 16.5.0.

[15] ——, "NR; Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021, version 16.5.0.

[16] ——, "NR; Physical layer measurements," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021, version 16.5.0.

[17] ——, "NR; Medium Access Control (MAC) protocol specification ," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), 2021, version 16.5.0.

[18] J. Postel, "Internet control message protocol," IETF RFC 792, 1981. [Online]. Available: https://www.rfc-editor.org/rfc/rfc792.txt

[19] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[20] O-RAN, "O-RAN Architecture Description," O-RAN Working Group 1, Technical Specification, 2021, version 4.0.

[21] J. Han, M. Kamber, and J. Pei, *Data Mining (Third Edition)*. Boston: Morgan Kaufmann, 2012.

[22] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014, 40th-year commemorative issue. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790613003066

[23] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proceedings of the Ninth International Workshop on Machine Learning*, ser. ML92. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, p. 249–256.

[24] Y. Liu, J.-M. Wu, M. Avdeev, and S.-Q. Shi, "Multi-layer feature selection incorporating weighted score-based expert knowledge toward modeling materials with targeted properties," *Advanced Theory and Simulations*, vol. 3, no. 2, p. 1900215, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/adts.201900215

[25] X. Wang, F. S. Samani, and R. Stadler, "Online feature selection for rapid, low-overhead learning in networked systems," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–7.

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

[27] I. de-la Bandera, D. Palacios, J. Mendoza, and R. Barco, "Feature extraction for dimensionality reduction in cellular networks performance analysis," *Sensors*, vol. 20, no. 23, p. 6944, 2020.

[28] D. Palacios, S. Fortes, I. de-la Bandera, and R. Barco, "Self-healing framework for next-generation networks through dimensionality reduction," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 170–176, 2018.

[29] M. Soysal and E. G. Schmidt, "Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison," *Performance Evaluation*, vol. 67, no. 6, pp. 451–467, 2010.

[30] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*. IEEE, 2005, pp. 250–257.

[31] A. Pasyuk, E. Semenov, and D. Tyuhtyaev, "Feature selection in the classification of network traffic flows," in *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*. IEEE, 2019, pp. 1–5.

[32] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019, pp. 688–695.

[33] X.-Y. Zhang, S. Wang, L. Zhang, C. Zhang, and C. Li, "Ensemble feature selection with discriminative and representative properties for malware detection," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 674–675.