Improving System Configurations using Domain Knowledge Assisted Semi-Supervised Learning

Negar Mohammadi Koushki, Sanjeev Sondur, and Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, USA
{koushki|sanjeev.sondur|kkant}@temple.edu

***Abstract*—Given the difficulty in obtaining adequate data from production systems, characterizing performance as a function of configuration variables (CVs) via supervised learning is difficult, and the use of standard semi-supervised learning (SSL) techniques may or may not help. In this paper, we describe a knowledge-assisted (KA) SSL algorithm that determines the confidence level of the generated data independently based on the domain knowledge. We demonstrate that such an approach outperforms plain SSL with the most popular SSL algorithms for all the workloads used in this study.**

***Index Terms*—configuration, semi-supervised learning, data argumentation, pseudo-labeled data, confidence measure**

I. Introduction

In an enterprise IT infrastructure, every major entity (e.g., application, virtual machine/container, infrastructure services, and hardware) has many *Configuration Variables* (CVs), which can be set at run-time to some value within their specified range. Although many instances of an entity (e.g., servers, VMs, application instances, etc.) have identical configurations, there is still a lot of variety in configuration, often necessitated by different roles, resource amounts, and workloads. There is often a tendency to define a large number of CVs for each subsystem without a clear description of what they do. Thus they are mostly left at their default values, which itself may be inappropriate. Also, because of the interaction between CVs and the workload, inadequate/incorrect CV settings may not be discovered until exposed to unseen workload patterns.

ML models can, in theory, capture various types of correlations across CVs and their interactions with the workload. However, their accuracy and reliability entirely depend on the adequacy of the data to cover the Configuration-Workload Space (CWS) reasonably well. The ML model could well be trained on a large amount of available configuration data, but if it does not cover the relevant CWS, the model is unlikely to be accurate or reliable. Also, since misconfigurations can take the system to unexpected places in the CWS, a purely data-driven approach would very likely lead to plainly wrong results since it has no notion of what behavior is sensible or possible. (We have demonstrated this in [1]). Furthermore, collecting a large set of configuration data is often a demanding task [2] leading to a *data bottleneck problem*. In addition, ML and AI are generally domain-agnostic and treat the data-sets in the same way [3].

This research was supported by NSF grant CNS-2011252

One way to partially address these issues is by generating new *pseudo-labeled data* artificially and using it to improve the training of the ML model. This is clearly the space of the much-explored *Semi-Supervised Learning* (SSL) for which numerous model training methods have been developed [2]. However, the focus in this paper is not to devise a new SSL method, but instead to explore how well the domain knowledge-assisted SSL methods (henceforth called KA-SSL) can work in the configuration management context to deal with the paucity of data. Thus, the main contributions of this paper are as follows:

- We introduce a domain knowledge-based method to (a) generate variants of existing configuration data that preserves the essential dependencies between them, and (b) a mechanism to evaluate the *quality* of generated pseudo-labeled data.
- We demonstrate that the resulting KA-SSL approach provides definitive improvements in performance predictions over the plain SSL approach that has been used in the past.

To the best of our knowledge, this is the first contribution towards extending KA-SSL techniques to address the limitations of data-bottleneck in configuration studies.

II. Data Driven Modeling of Configurations

A. Configuration Management Problem

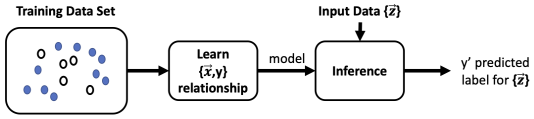
Consider a system with M CVs denoted by the set $C = \{C_j = j = 1..M\}$ where C_j is the name or ID of the jth CV. Each C_j has a current value between the lower and upper bounds ("lb", "ub") $[lb_{C_j}, ub_{C_j}]$. We assume that these CVs are relevant w.r.t some measurable system behavior O (e.g., throughput, latency, etc.) under some workload W. Thus, the specific values of the triplet (W, C, O) provide the data for data-driven models, where (W, C) are inputs and O is the output (or "label" in ML terms). For notational simplicity, we denote the data as a pair (x_i, y_i) where y_i is the output and x_i is a vector $x_i = \{x_{ij}, j = 0..M\}$ where x_{i0} denotes the workload measure[1] and $x_{ij},\ j > 0$ the selected value of the CV C_j. The entire data-set then can be denoted as $D = (x_i, y_i), i = 1..N$. Note that even if N is large, many of the data points are likely to be for the same or similar

[1]The use of a single parameter x_{i0} to denote workload is for notational simplicity – one could surely represent workload by several aspects such as intensity, burstiness, and composition of different types of transactions.

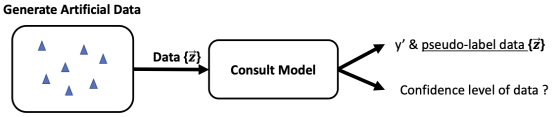
configuration driven by different workload parameter values. That is, the number of distinct configurations, say N_c, is generally quite limited.

B. Supervised and Semi-Supervised Models

A supervised ML algorithm attempts to implicitly learn the dependence of the output y_i on the input vector x_i. It could then predict the label for unseen configuration $\vec{z}$, as shown in Fig. 1a. The notion of SSL extends supervised learning by augmenting the labeled data with additional data to increase the accuracy of the model [2, 4]. The *transductive learning* directly determines the labels for all unlabeled data points without learning a model first, typically by using the labels of nearby points [4]. On the contrary, inductive learners assign *pseudo-labels* to the unlabeled data and use them (in addition to labeled data) to learn a model. In this paper, we focus entirely on inductive methods.



(a) Typical machine learning model



(b) Uncertainty in pseudo labeled data

Fig. 1: Challenges in generating pseudo labels.

C. Characterizing Pseudo-labels

It is clear that a pure-data-driven SSL cannot derive accurate pseudo-labels for the unlabeled data unless the underlying marginal data distribution $p(x)$ over the input space x provides some information about the posterior distribution $p(y|x)$ [2]. Generally, this is stated as a "smoothness" assumption, i.e., if inputs x_i and x_j are close, then their labels y_i and y_j should also be close. The close relationship is usually not transitive, which makes it difficult to draw conclusions based on it. Yet another way to state the same thing is through the clustering assumption, which states that high-density areas of inputs are likely to share the same label. Conversely, the boundaries in the input space that separate the output labels should pass through the low-density areas of the input.

In applying such properties to the configuration problem, we first need to transform the input parameters so that one can meaningfully speak of the closeness. For example, the storage space (including caches, memory, and secondary storage) generally (but not necessarily) increases in multiplicative steps (e.g., 32, 64, or 128 GB memory), and the performance increase, if any, from doubling is far less than double. Thus a log compression of the inputs becomes necessary before considering closeness. Real configurations are also likely to be limited to discrete values rather than continuous [1]. For example, based on the available DIMMs, the memory could be 32GB, 48GB, or 64GB, but unlikely to assume other values between 32 and 64. Similarly, the NICs may support 10, 40, or 100 Gb/sec bandwidth. Also, each parameter will have a range that is sensible based on the domain knowledge, even if other values are feasible (e.g., minimum memory of 32GB).

On the output side, although the performance would generally vary smoothly with the resources, this is not true when bottlenecks are involved. For example, if the bottleneck in the memory bandwidth causes long access delays for the CPU, a slight increase in the memory bandwidth could increase the processing rate quite considerably. Similarly, as the resource amount goes below some minimum level, an extreme resource contention (e.g., thrashing) may kick in, thereby killing off the performance. Another crucial issue in configuration studies is that the data points (configurations) are far from representative of the configuration space. Most of the configurations for which we have the data are those that were chosen simply because they perform well enough; others are likely to be much more sparse. Thus, a pure data-driven pseudo-labelling is likely to fail in configuration problems.

The SSL literature uses the notion of *low-dimensional manifold*, which can be thought of as the surface where points with similar output values (labels) cluster together. In the configuration context, a manifold often arises because of the compensatory effect of different CVs; for example, a configuration of a small amount of DRAM and a fast IO device may provide similar performance as the one with a large amount of memory and a slow IO device. The manifold is generally an approximate, low-dimensional representation of such behavior, and in practice like the rules of thumb that the administrators already know.

Consider the 3D configuration space consisting of CPU speed, memory bandwidth, and memory size. One could then identify a 3D surface for each performance level (or output label), which is the true manifold. If the low dimensional approximation to this manifold is approximately correct, for any given target performance, very few points should lie outside the corresponding surface. That is, the density of such points should be low, and that can be used as a means for identifying the boundaries of regions with different performance levels (or labels).

D. Generating Pseudo-labels

The clustering and related assumptions above provide one way to produce the pseudo-labels [2, 4]. Another one, known as a wrapper methodology, starts with some *base learners* that are initially trained only on the labeled data. These learners could then be used to predict the pseudo-labels and the *pseudo-labeled* data could then be recycled back for further training of the learners. There are several methods to do this and we shall discuss them in the following.

It is generally a poor idea to admit all generated data from the base learners; instead, we need a reliable measure of *confidence* in the label prediction, and retain only those data items for which the confidence level is above some threshold (e.g., 90%). Although most supervised learning (SL) algorithms produce a confidence level measure in the last step (usually using softmax), it is not necessarily a reliable

measure. The manifold and clustering ideas assisted by the domain knowledge can provide an alternate and more reliable measure instead. The pseudo-labeled data could thus be used like the labeled data for additional training of the base learners, either starting with the result of original training or from scratch with original and pseudo-labeled data intermixed.

III. Generating and Labelling Pseudo-labeled Data

A. Generating Artificial Data-set

To generate artificial (or "fake") data, we first generate the input data (z) and then put labels (y') on it. An obvious approach is to randomly choose a value for z_{ij} in the range $[lb_{C_j}, ub_{C_j}]$ for each j. However, doing so would ignore the dependencies between the CV's and may result in unrealistic configurations. To address this, we can make use of the domain knowledge in the form of sensible ranges for CVs relative to some key CV (e.g., memory relative to CPU capacity or number of TCP connections for the target throughput). Domain knowledge enables the creation of appropriate groupings either manually or by guiding the clustering algorithm toward preferred groupings as explored in our earlier work [5].

B. Data Labeling and Estimating Label Confidence

With the typical black-box ML models, it is difficult to *qualify* the prediction on the previously unseen data-set (i.e. confidence on z as shown in Fig. 1b), particularly if the unseen data is quite different from the data used for training [6]. Although most neural nets provide or can be coaxed to provide a confidence level for the output label, the probabilities are generally inaccurate. In particular, the decision trees and related models (e.g., random forest) are known to provide a rather poor estimate of the label probability since they focus on minimizing the tree size and enhancing classification accuracy rather than the probabilities [7]. Augmenting ML with explicit domain knowledge can enhance the quality of the results [3]. In particular, we have examined the notion of *quality* of configuration by defining a *Configuration Health Index* (CHI) that quantifies how well the system is configured based on the expected behavior of the output y with respect to the different components of the configuration vector $\vec{x}$ [1].

Intuitively, the CHI measures the variation of y_i's (i.e., the output) as a function of x_{ij}'s (i.e., jth CV or C_j). For example, if index j refers to CPU cores, the CHI may indicate performance vs. #cores. For generality, CHI measure is regarded as a *weight* $W_j(z_j)$ in the range 0..1 and indicates how well the system is configured for desired output (usually performance) with respect to the normalized value z_j of CV_j in the range 0..1. We assume a specific form for $W_j(z_j)$ based on the domain knowledge. For example, if z_j is the resource amount and $W_j()$ relates to performance, we generally expect $W_j(z_j)$ to rise at a slowing rate w.r.t z_j and eventually either saturate or even droop (due to bottlenecks/overhead). The parameters of such behavior are determined from the available data. This method implicitly accounts for the dependencies across the CVs in addition to the dependency of output on each specific CV. In addition to being independent of any ML model, we found it to be extremely successful in accurate performance prediction even in cases considered to be difficult [1].

$$\varepsilon = 1 - \|y^{(a)} - y_e^{(a)}\|^2 \tag{1}$$

IV. Methods for Using Pseudo-labeled Data

Pseudo-labeled data can be used in many ways [4]. In the following, we describe and explore some of the key methods in the configuration context.

A. Self-training

Self-training [8] is the basic method wherein the original SL classifier is trained further using both original and pseudo-labeled data. Here one could use an expectation-maximization (EM) type of approach wherein we include all of the unlabeled data at once and then iteratively determine the parameters of the labeling process such that we maximize the likelihood that the observed data belongs to the assumed distribution of the labels. For example, Wu et al. [9] use Naive Bayes for the classification version of the problem. The implicit assumption here is that each dimension (or configuration variable) k contributes independently to the label. Furthermore, by assuming that the contribution of each CV has a normal distribution, they estimate its mean and variance to maximize the likelihood that the observed data (both labeled and unlabeled) comes from the computed distribution. The Gaussian distribution is reasonable for continuous variables; for discrete values, a binomial model can be used. In Self-training pseudo-labeled data may be given a lower weight than the original data; either directly (if the confidence level of each data-point is known) or collectively through a single hyperparameter.

B. Co-training (Multiview)

Unlike Self-training, Co-training involves training $K \geq 2$ largely independent classifiers on the labeled data-set and then generates pseudo-labeled data for one another. This collaborative approach can lead to better accuracy in comparison with the baseline. In Co-training, multiple models are trained on different subsets or views of the data, allowing them to capture diverse patterns and potentially correct errors in the training data. When one model makes a confident prediction, its prediction can be used to label the data for the other model, leading to improved accuracy.

Additionally, Co-training can leverage diversity in the data due to the use of different subsets or views. For example, in the case of using Random Forest classifiers, each may capture different patterns in the data. By combining their predictions, we benefit from the ensemble effect, which often leads to better generalization and higher accuracy. However, it's essential to note that if the classifiers are strongly correlated, they will likely generate very similar labels, and thus the advantage of having K classifiers is lost. The required data diversity may be either natural (e.g., multi-modal data-sets such as audio and video) or induced by having different classifiers focus on different features [10]. Nevertheless, the selected features for

each classifier should be adequate for a good prediction of the output; otherwise, Co-training could actually hurt the accuracy.

V. Experimental Evaluation

A. Data-set Overview: A Closer Look at the Data

We applied SSL models to datasets listed in Table I. Most existing datasets focus on time-series observations of system parameters with little information on configuration settings. Our self-collected ES dataset is the only true configuration dataset [11], while the public BB dataset provides limited system-specific details [12].

Cloud/Edge Storage Data-set (ES): Edge Storage (ES) is vital for fast local access at the "edge" nodes by using a small local storage as a cache for the essentially unlimited remote Cloud storage. However, proper configuration of hardware and software CVs is essential to prevent high latencies and IO timeouts. Our experiments with ES collected 991 valid data points with different CV settings (see Table I) where N^O refers to the size of the original data set and N^A is the size of the generated pseudo-data set.

BitBrains Data-set (BB): The BB data-set[2] contains performance logs from 1,750 virtual machines (VMs) in BitBrains's cloud data center, covering a 5-month period. It was analyzed by Technical University, Delft, and comprehensively characterizes requested and used resources, including CPU, memory, disk, and network resources. See Table I for initial VM configurations.

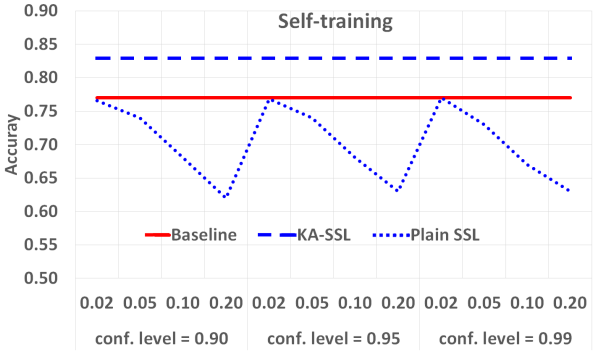


Fig. 2: Self-Training results for ES dataset.

B. Experimental Setup

For each data-set, we followed a standard split[3] where approximately 20% of the labeled data (original data) was set aside as test samples, while the remaining labeled data, along with artificially generated unlabeled data, was used for training. Our ML model concerns the prediction of performance as a function of the identified CVs, which we pose as a classification problem. That is, the performance is divided into 10 equally spaced levels. A similar stratification is done for the input CV's to deal with all CVs in a uniform way.

[2][BB] http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains (RND500) :
[3]Commonly referred as k-fold validation in ML terminology, k=5

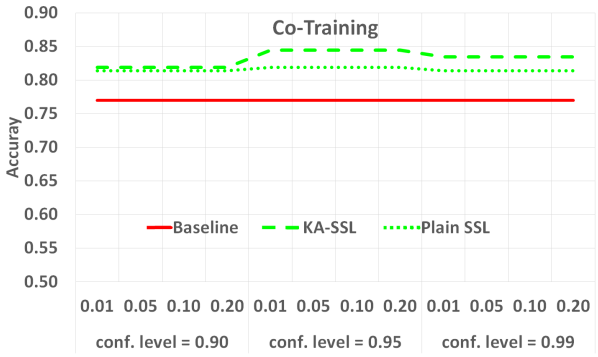


Fig. 3: Co-Training results for ES dataset.

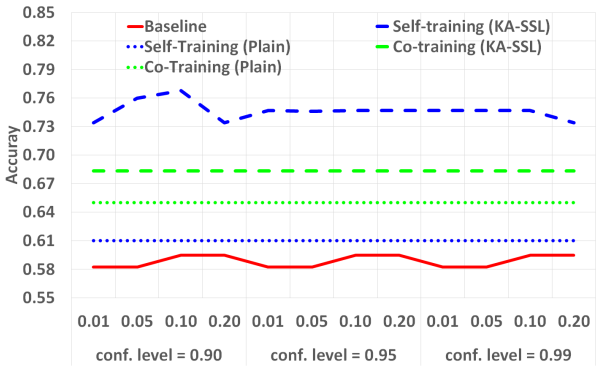


Fig. 4: KA-SSL vs. baseline for BB for different fractions of fake data added (first line) and confidence levels (second line)

It is well proven that the applicability and performance of ML algorithms depend on the data characteristics and the domain [11, 13]. We explored a wide range of ML algorithms to find the best fit for the configuration problem domain. We found that the RandomForest works best for ES and BB data-set.

For evaluation, we start with the *baseline* model trained entirely on the labeled data. The goal then is to investigate the impact of gradually adding unlabeled data at varying levels of confidence (i.e. Eq. 1) to the labeled data. We added the unlabeled data in increments of 10% (of the labeled data size)

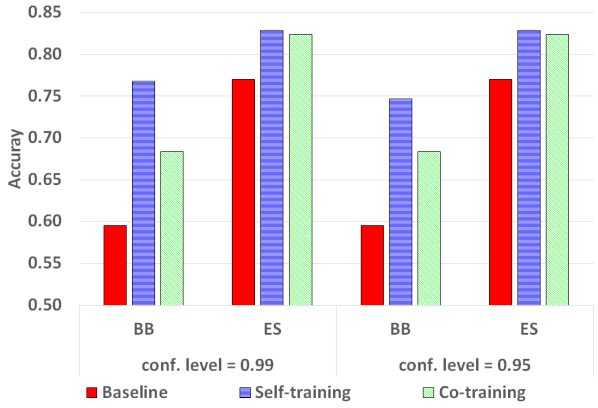


Fig. 5: Results summary.

TABLE I: Data-sets: configuration variables and output.

Data-Set	Domain	Size (N^O, N^A)	Configuration Variables (CVs) $\vec{x}$	Output (Label y)
ES [11]	Cloud Storage	991, 14717	No. of CPU Cores, Core Speed, Memory Capacity, Memory Bandwidth, Disk IO Rate, Request Arrival Rate, Request Size, Metadata Size	Performance
BB [12]	Virtual Machines	391, 3504	No. of CPU Cores, Core Speed, Memory Capacity, Network Data Rcvd., Network Data Transmit, Disk Read Throughput, Disk Write Throughput	CPU Usage (%)

and considered three confidence levels for adding unlabeled data: 90%, 95%, and 99%. We explored this in the context of the SSL algorithms described above.

We used two flavors of SSL: (a) the KA-SSL as described above, and (b) plain SSL, typical of how SSL is generally applied. The data generation for the latter has two differences: (i) All CV values are selected randomly within their specified ranges (i.e. no domain knowledge), and (ii) The confidence level is the probabilities directly generated from the ML model (i.e. dependency on ML again). In particular, we generated 10,000 random data points, which were then used as inputs to a Random Forest (RF) Classifier trained on the labelled data. The RF classifier first generates class probabilities through a softmax layer, whose maximum probability was taken as the confidence level.

C. Experimental Results

Fig. 2 presents the self-training results for the ES dataset, comparing KA-SSL, plain SSL, and the baseline model, which employs Random Forest as the base estimator. The findings reveal that the intelligent augmentation of artificial data to the original dataset can significantly enhance accuracy, underscoring the benefits of KA-SSL. However, the indiscriminate addition of noisy data may have an adverse impact on accuracy. Fig. 3 shows the co-training results for the same dataset and with the same baseline Random Forest model. Interestingly, adding noisy data to the model in a co-training framework may lead to improved accuracy compared to relying solely on Random Forest as the baseline. This improvement is attributed to co-training's ability to reduce or mitigate noise through the collaboration of diverse models, including Random Forest, which collectively averages out errors stemming from noisy data. This ensemble approach often yields superior accuracy by producing more robust and precise predictions. Nevertheless, with high confidence level data addition, KA-SSL outperforms plain SSL.

Fig 4 shows the comparisons for the BB workload. It is seen that while SSL improves performance over baseline in all cases, the KA-SSL provides significantly better results. We also found that co-training and self-training generally outperform other SSL methods (not discussed here for brevity). However, all methods lead to improvement in accuracy. These results underscore the potential and applicability of KA-SSL methods in real-world scenarios.

VI. Conclusions

In this paper, we explored the problem of artificially generating pseudo-labeled data to enhance machine learning models to predict performance as a function of configuration parameters. This is necessitated by the difficulty in obtaining adequate performance data from the production systems. We propose a domain knowledge-assisted (KA) method for estimating the confidence level for the generated data in the context of semisupervised learning (SSL) algorithms. Such a KA-SSL approach outperforms the Plain SSL approach and improves the model accuracy even with rather small amounts of fake data addition. In the future, we will explore a more strategic way of selecting the fake data points to better fill the uncovered space.

References

[1] Sanjeev Sondur and Krishna Kant. Performance Health Index for Complex Cyber Infrastructures. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 2022.

[2] Van Engelen et al. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

[3] Tirtharaj Dash et al. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1040, 2022.

[4] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. *CS Technical Reports*, 2005.

[5] Negar Mohammadi Koushki, Sanjeev Sondur, and Krishna Kant. Automated configuration for agile software environments. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 511–521. IEEE, 2022.

[6] Maksims Ivanovs et al. Perturbation-based methods for explaining deep neural networks: A survey. *Pattern Recognition Letters*, 150:228–234, 2021.

[7] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *ML*, 52:199–215, 2003.

[8] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd ACL*, 1995.

[9] Zhiang Wu, Junjie Wu, and et al. Hysad: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation. In *18th ACM SIGKDD*, 2012.

[10] Zhi-Hua Zhou and Ming Li. Semi-supervised learning by disagreement. *KAIS*, 24:415–439, 2010.

[11] Sanjeev Sondur et al. Towards automated configuration of cloud storage gateways: A data driven approach. In *Cloud Computing*, pages 192–207. Springer, 2019.

[12] Iosup Alexandru et al. The grid workloads archive. *FGCS*, 24(7):672–686, 2008.

[13] Mohammad S Sorower. A literature survey on algorithms for multi-label learning. *OSU, Corvallis*, 18, 2010.