

An extensive comparison of preprocessing methods in the context of configuration space learning

Damian Garber^{1,*}, Alexander Felfernig¹, Viet-Man Le¹, Tamim Burgstaller¹ and Merfat El-Mansi¹

¹Graz University of Technology, Inffeldgasse 16b, Graz, Styria, Austria

Abstract

One of the core goals in the research field of configuration space learning is building precise predictive models that allow for reliably estimating the performance of a configuration without requiring costly tests. The models used for this purpose are usually machine learning-based. However, the models show significant deviations in their performance depending on the investigated Software Product Line (SPL), the applied data preprocessing, and the number of sample configurations collected. Thus, we investigate the impact of different preprocessing methods and their behavior when using different SPLs, machine learning models, and sample sizes. Performance comparisons on this scale are usually not conducted due to their prohibitively expensive execution time requirements, even for smaller SPLs. Thus, we used three fully enumerated spaces as our training data, which allows for more generalized results. Our results show that the average factors between the worst and best-performing preprocessing methods are 2.05 (BerkeleyDBC), 1.17 (7z), and 1.84 (VP9). Further, no single preprocessing method tested was able to outperform all others, nor was this the case within one specific SPL or model type. This underlines the importance of testing new approaches with multiple preprocessing methods.

Keywords

Configuration Space Learning, Machine Learning, Preprocessing

1. Introduction

The discovery of configurations that optimize the performance of any given Software Product Line (SPL) is one of the core goals of configuration space learning. The performance of a model can take many forms and rely heavily on the use case. For instance, one may optimize a SPL to perform a core task very efficiently or optimize for the size of the compiled SPL binary. This optimization usually takes place in steps. The first step is sampling configurations from the configuration space of the SPL and measuring the target property, which often entails compiling and running tests or benchmarks, a very time and resource-intensive undertaking. One can use these samples to train a prediction model, which is then used to find a configuration that optimizes the target property. In this paper, we focus on the creation and training of the prediction model. Many factors can impact the performance of a performance prediction model for SPLs, from the SPL itself to the sampling approach used to collect the training data. However, our focus lies on one

of the factors often neglected in SPL performance prediction: Preprocessing. We will define any necessary terms used in this paper in Section 2. Preprocessing has proven itself in many other domains that employ machine learning-based prediction models. However, literature reviews such as Gong and Chen [1] show that less than half of the investigated studies within the field of configuration performance learning use preprocessing, further discussed in Section 3. Accordingly, we thus conduct an in-depth investigation on the influence of preprocessing on performance prediction models for SPLs. To this end, we measure the performance of 4 preprocessing methods in the context of 3 SPLs, 5 machine learning models, and 20 different sizes of training sets. We discuss the details of the experimental evaluation in Section 4, followed by a discussion of the results in Section 5.

2. Definitions

Software Product Line (SPL). SPLs, as a concept started to gain widespread popularity at the beginning of the 2000s [2]. Engström and Runeson [3] describe SPLs as the paradigm of forming derivative products from a set of generic components. A SPL has multiple features, each supporting an individual domain of values, which allows for the generation of diverse products using the same components.

Configuration. In the context of a SPL, a configuration defines for each feature the corresponding feature value. However, there may exist additional constraints within

ConfWS'24: 26th International Workshop on Configuration, Sep 2–3, 2024, Girona, Spain

*Corresponding author.

✉ dgarber@ist.tugraz.at (D. Garber);
alexander.felfernig@ist.tugraz.at (A. Felfernig);
vietman.le@ist.tugraz.at (V. Le); tamim.burgstaller@ist.tugraz.at
(T. Burgstaller); merfat.elmansi@un.org (M. El-Mansi)

ORCID: 0009-0005-0993-0911 (D. Garber); 0000-0003-0108-3146
(A. Felfernig); 0000-0001-5778-975X (V. Le); 0009-0007-4522-8497
(T. Burgstaller); 0009-0005-2695-4210 (M. El-Mansi)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

the SPL. Thus, we speak of a valid configuration if none of the assigned values is inconsistent with any of the constraints.

Configuration Space. The configuration space of a SPL describes the space spanned by all valid configurations of the SPL. The size of configuration spaces commonly grows exponentially with the number of features, and we speak of colossal configuration spaces if its size is $\gg 10^{10}$ [4].

3. Related Work

The three fully enumerated configuration spaces provided by Oh et al. [4] facilitate the comprehensive performance analysis and comparison we conducted. They use them to show that relatively simple approaches like uniform random sampling can outperform well-established tools like SPL Conquerer [5] to find near-optimal configurations for SPLs. We build on this idea and conduct a comparison of different preprocessing methods. The reasoning behind conducting this comparison is the alarming result of Gong and Chen [1]. They performed a literature review on deep configuration performance learning and reported that 44 out of 85 investigated studies used the data as it was without any preprocessing. This limited utilization implies a lack of awareness of the impact of preprocessing methods. This lack of awareness may then aggravate the difficulty of reproducing and validating the results of published works. Dacrema et al. [6], for example, investigated 18 new approaches published recently, of which they could only reproduce 7. Of the 7 reproduced approaches, they showed that they can outperform 6 by using relatively simple other approaches. The importance of preprocessing methods in many domains has long since been established. Wu et al. [7], for example, shows that preprocessing improves the performance of streamflow forecasts. Rasekhi et al. [8] report improvements in the prediction of epileptic seizures by using preprocessing.

We further include in our tests different sample sizes, which allows us to investigate the reaction of the preprocessing methods to changing sample sizes. Acher et al. [9] sampled and measured 95854 Linux configurations, a minute fraction of the configuration space of 2^{15000} ($2.818 * 10^{4515}$) configurations spanned by Linux. They reported to have needed 15000 hours of computation time to collect the samples. Guo et al. [10] uses tree-based models to predict configuration performances. Martin et al. [11] focus on using transfer learning across different versions of the Linux kernel to predict the performances of the versions. They mention preprocessing only for encoding configurations into formats compatible with their machine-learning approach.

The literature reviews of Gong and Chen [1] and Pereira

Name	Value
Vendor	Lenovo
Product	20N6001GGE
CPU	Intel Core i7-8665U (4x 1,90 GHz)
RAM	32GB (DDR4)
OS	Manjaro Linux
Kernel version	6.1.80-1-MANJARO

Table 1

Specifications of the machine used in the experiments

et al. [12] named multiple data sampling approaches used in configuration performance learning. However, both identified random sampling as the most popular approach. Pereira et al. [13] conducted a dedicated study on sampling approaches for learning configuration spaces. They suggest using uniform random sampling as long as it is computationally feasible. Accordingly, we adapted it for our comparison.

4. Experimental Setup

This section will discuss the exact experimental setup for data collection and which machine learning models, preprocessing methods, and datasets we were using. All measurements were collected using the same machine with specifications as they are listed in Table 1. We use Mean Absolute Percentage Error (MAPE) to evaluate the model performances, which is one of the most commonly used metrics in literature [1] [12] [13]. The code was implemented in Python using the widely used scikit-learn¹ library [14]. We used Uniform Random Sampling (URS) to generate the training sets of different sizes for model learning. We can perform URS by selecting configurations randomly from the set of valid configurations. The size of the training sets range from 50 to 1000 in steps of 50. However, the tests for all models and preprocessing methods use, within the same iteration, the same training set of a specific size. After the performance of all models using the preprocessing applied to the training sets is measured, these measurements are repeated 15 times, each time with new training sets selected with URS. The average of the resulting MAPE values in the 15 iterations is the value we use when we discuss the results.

4.1. Datasets

For our comparison, we use a dataset of fully enumerated configuration spaces. Thus, the dataset includes all valid configurations for a given SPL. In addition, a value, like execution times of benchmarks or similar, representing the performance of each configuration was measured. We use three such datasets based on three configurable

¹<https://scikit-learn.org/stable/index.html>

software projects: **BerkeleyDBC**², **7z**³, and **VP9**⁴. The datasets are used in the work of Oh et al.[4], and they were made available in their resources⁵. Oh et al.[4] provided the following description of the three datasets.

BerkeleyDBC is an embedded database system with 9

Name	Domain
DIAGNOSTIC	0 1
HAVE_STATISTICS	0 1
HAVE_REPLICATION	0 1
HAVE_CRYPTO	0 1
HAVE_SEQUENCE	0 1
HAVE_VERIFY	0 1
HAVE_HASH	0 1
CACHESIZE	CS16MB CS32MB CS64MB CS512MB
PAGESIZE	PS1K PS4K PS8K PS16K PS32K

Table 2
BerkeleyDBC variable names and their respective domains

variables and 2560 configurations. Benchmark response times were measured. We visualize the variable names and their domains in Table 2.

7z is a file archiver with 9 variables and 68640 configurations. Compression times were measured. We visualize the variable names and their domains in Table 3.

VP9 is a video encoder with 12 variables and 216000 configurations. Video encoding times were measured. We visualize the variable names and their domains in Table 4.

Although these three configuration spaces do not approach the sizes of colossal configuration spaces like Linux, which spans a configuration space with 2^{15000} configurations[9], they still have sizes where an enumeration is no longer an option, and thus fall in the purview of the research field of configuration space learning. Depending on the complexity of the tests and the underlying system, procuring very few samples may already be very costly. Acher et al.[9], for example, reported 15000 hours of computation to build and measure 95854 Linux configurations. We selected these datasets due to two main advantages. The first is avoiding the extreme computation times of collecting such data, and the second is that using them allows us to test multiple iterations of training sets of different sizes.

4.2. Models

We selected five different types of machine-learning models, each representing a different general approach to

²<https://www.oracle.com/database/technologies/related/berkeleydb.html>

³<https://www.7-zip.org/download.html>

⁴<https://www.webmproject.org/vp9/>

⁵<https://zenodo.org/records/7776627>

Name	Domain
root	0 1
CompressionMethod	LZMA LZMA2 PPMd BZip2 Deflate
x	x_0", "x_2 x_4 x_6 x_8 x_10
BlockSize	BlockSize_1", "BlockSize_2 BlockSize_4 BlockSize_8 BlockSize_16 BlockSize_32 BlockSize_64 BlockSize_128 BlockSize_256 BlockSize_512 BlockSize_1024 BlockSize_2048 BlockSize_4096
Files	Files_0 Files_10 Files_20 Files_30 Files_40 Files_50 Files_60 Files_70 Files_80 Files_90 Files_100
tmOff	0 1
mtOff	0 1
HeaderCompressionOff	0 1
filterOff	0 1

Table 3
7z variable names and their respective domains

maximize the usefulness of our results. In our implementations, we used models from the scikit-learn⁶ python library [14]. For the sake of reproducibility, we did not perform any parameter tuning on the models and used their respective default settings if not explicitly stated otherwise.

The first model is a Multi-Layer Perceptron (**MLP**) model, a feedforward neural network approach. We set the maximum of iterations to 1000 and activated early stopping for our tests. MLPs are, according to Gong and Chen [1], the most popular approach when conducting deep configuration performance learning. However, it is a very data-intensive approach that needs comparatively large training sets to perform well.

The second model is a K-Nearest Neighbors (**KNN**) model, a memory-based approach. The model finds the k-nearest neighbors to a configuration from the training set, in our case the default value 5. The KNN model predicts the performance of the configuration by calculating the average of the performances of the configuration's k nearest neighbors. In our case, the average was weighted by the distance between the neighbor and the configuration.

The third model is a Random Forest (**RF**), an ensemble method employing several decision trees generated using the training data to predict the performance of an un-

⁶<https://scikit-learn.org/stable/index.html>

Name	Domain
root	0 1
lagInFrames	lagInFrames_0 lagInFrames_8 lagInFrames_16
endUsage	variableBitrate constantBitrate constrainedQuality
AdaptiveQuantizationMode	off variance complexity cyclicRefresh
TileColumns	TileColumns_0 TileColumns_3 TileColumns_6
cpuUsed	cpuUsed_0 cpuUsed_2 cpuUsed_4 cpuUsed_6 cpuUsed_8
Threads	Threads_2 Threads_4 Threads_6 Threads_8 Threads_10
bitRate	bitRate_300 bitRate_600 bitRate_900 bitRate_1200 bitRate_1500
FrameBoost	0 1
lossless	0 1
AutoAltRef	0 1
Quality	good realtime

Table 4
VP9 variable names and their respective domains

known data point. We use bagged trees, which means we train all underlying decision trees to solve the problem using all features. The final result is in the context of classification decided based on a majority vote. However, in our context of regression, we calculate the final result by taking the mean of all results produced by the decision trees.

The fourth model is a Support Vector Machine (**SVM**), a well-established model based on statistical learning frameworks. We use a radial basis function as our kernel type.

The final model is an ElasticNet (**EN**) model, a derivative of linear regression models. The model combines L1 and L2 priors as a regularizer.

4.3. Preprocessing

We used several preprocessing methods to test their impact on the different models and training sizes. For the sake of this comparison, we do not distinguish between actual preprocessing methods like Standardization and encodings such as the One Hot Encoding.

The first preprocessing method discussed we call default (**DEF**). It provides a baseline for mostly unaltered data and leaves numeric values untouched. The boolean values are, however, encoded with 0 and 1 for false and true, respectively. If all values of a domain can be converted

into numbers this is done (e.g. CS32MB = 32, Table 2). If this is not possible, the string values are encoded using label encoding [15, 16], which assigns an increasing numeric value for each unique string in a domain. This format is the default state of the data. Thus, we apply all preprocessing methods mentioned hereafter to the data in this format.

The second preprocessing method is Min Max Scaling (**MMS**) [17, 18], which reduces the scale of a given feature to be between 0 and 1. We achieve this by applying Equation 1 on every feature of the configuration, where min and max are the minimum and maximum recorded numbers for this feature, respectively. When we apply this to the features encoded using label encoding, the result is a derivative of the former called scaled label encoding [19, 20].

$$f_i(x_i) = \frac{x_i - \min_i}{\max_i - \min_i} \quad (1)$$

The third preprocessing method is Standardization (**STD**) [21, 22], which is achieved by calculating the mean and standard deviation of each feature and applying Equation 2

$$f_i(x_i) = \frac{x_i - \mu_i}{\sigma_i} \quad (2)$$

This results in the mean of every feature in the training set being now 0 and the standard deviation being 1.

The final preprocessing method is One Hot Encoding (**OHE**) [23, 24], which changes the domain of all features to a boolean domain. We achieve this by increasing the dimensions of the data by an encoding of the domain. Thus, if, for example, feature f has the domain 0, 6, 12, it would have been replaced with the features f_0, f_6, f_{12} each of the three resulting boolean features are mutually exclusive and encode one possible value assigned to feature f .

5. Results

In this section, we showcase the measurements collected as described in the experimental setup section and discuss them. To this end, we will discuss the results of each dataset separately and what observations we made.

Firstly, we start with a discussion of our smallest SPL, BerkeleyDBC. All performance results are visualized in Figure 1. In the results for MLP, we see that OHE is performing best among all preprocessing methods regardless of sample size. However, we can also see a shift in the performances of the preprocessing approaches. STD started as the worst-performing preprocessing method. Despite that, with increasing sample size, it outperformed MMS and DEF. Accordingly, the results of STD approached the results of the best performer OHE for the larger sample sizes. However, when looking at Figure 2 and Figure

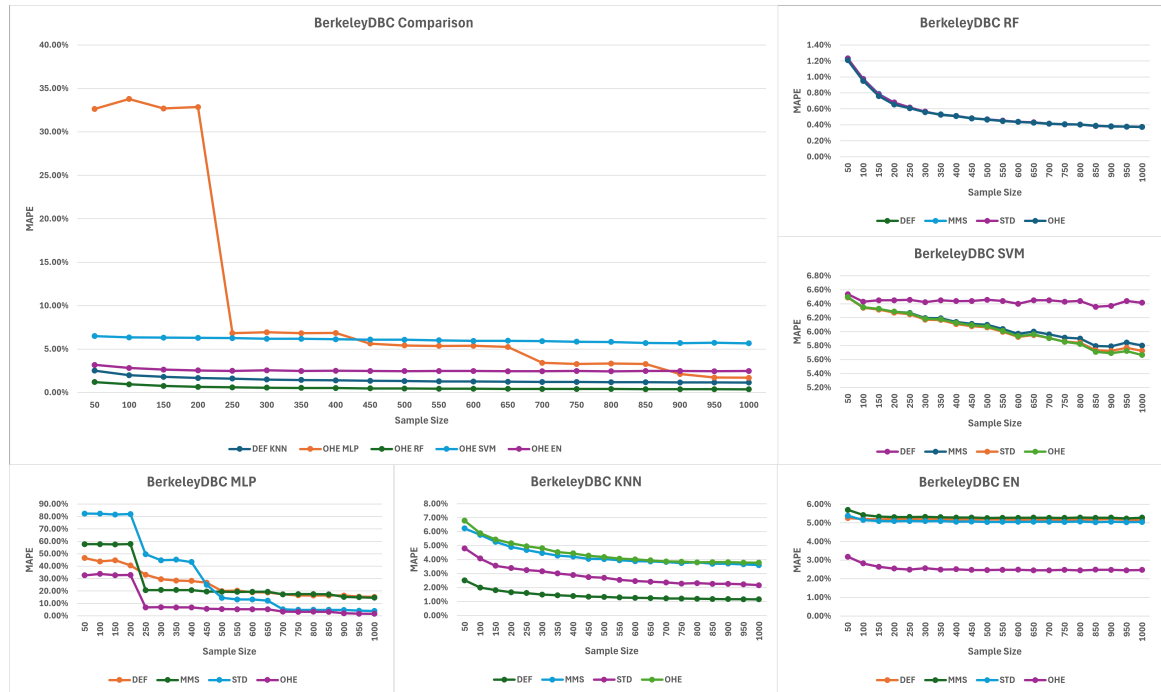


Figure 1: The performances of each preprocessing method applied to each model and a comparison between the top performers of each model applied to the BerkeleyDBC dataset.

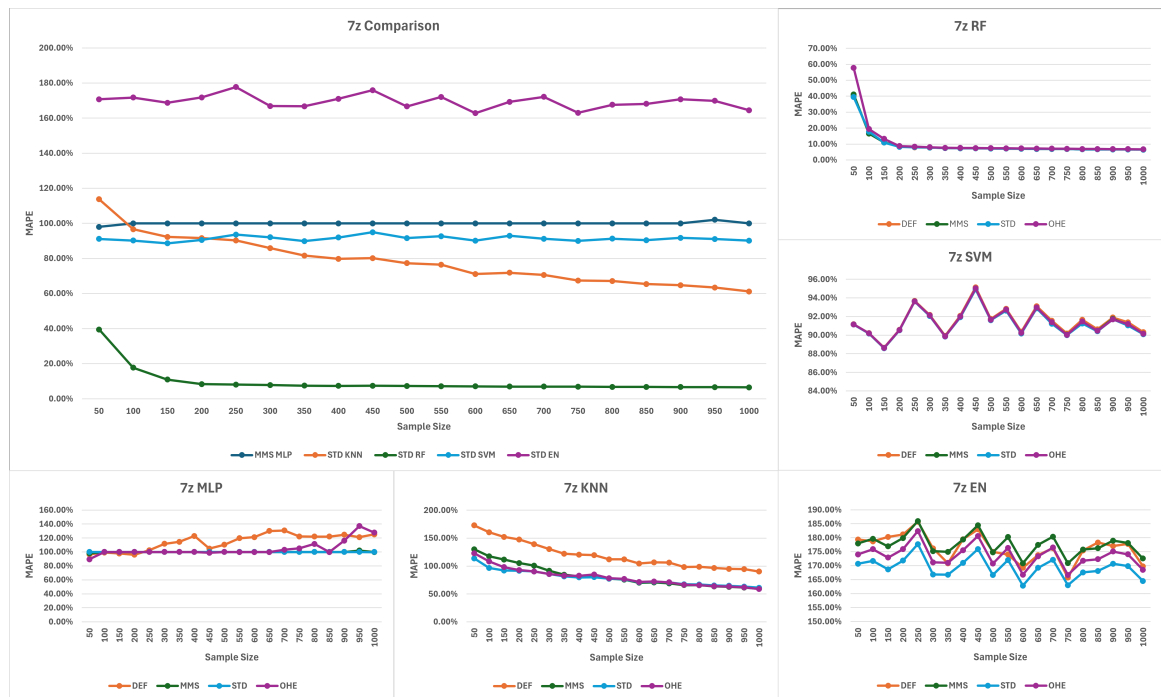


Figure 2: The performances of each preprocessing method applied to each model, and a comparison between the top performers of each model applied to the 7z dataset.

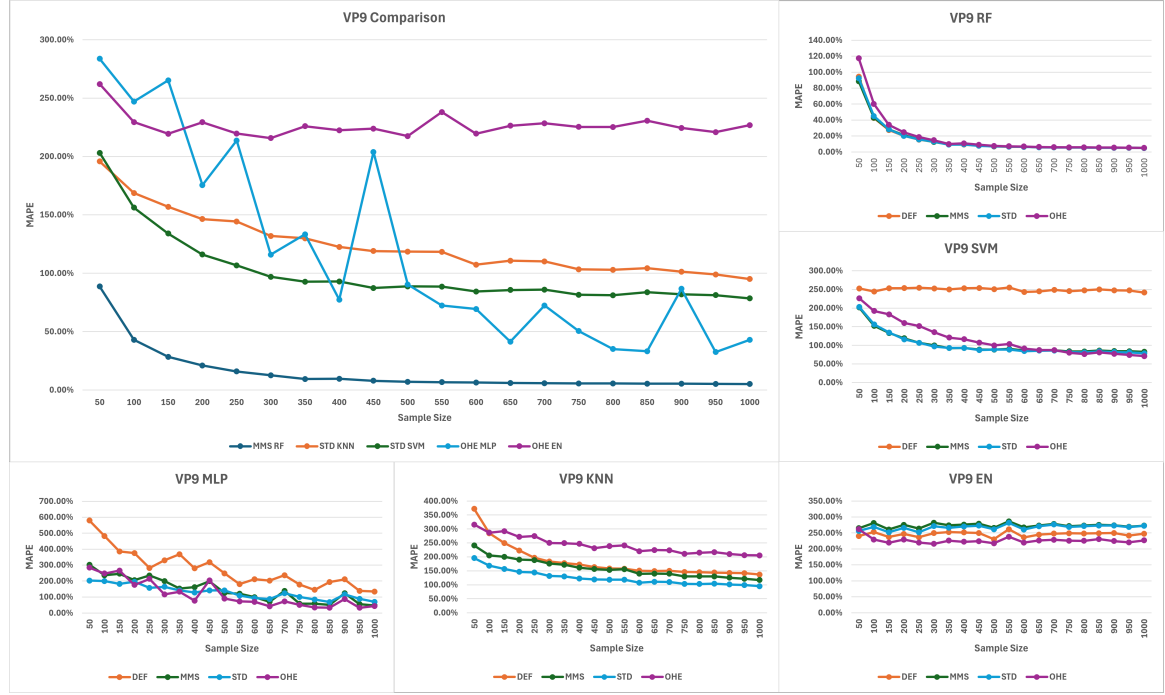


Figure 3: The performances of each preprocessing method applied to each model, and a comparison between the top performers of each model applied to the VP9 dataset.

3, we see that this behavior does not occur in the other datasets, but, in contrast to other approaches, the results of STD remain either relatively stable or improve with increasing sample sizes. We see a similar behavior on a smaller scale with MMS and DEF. MMS performed initially worse than DEF, overtaking it as soon as sample sizes became larger than 200 and achieving similar results from sample sizes 500 and larger. The other models, in contrast, showed more pronounced preferences for preprocessing methods. For the KNN model, DEF was the best-performing preprocessing method, followed by STD, MMS, and OHE. RF showed the best performance of all models with almost indistinguishable differences of 0.01% between the preprocessing methods on average. For the SVM model, DEF performed the worst with comparatively little improvement with larger sample sizes. The remaining preprocessing methods, from worst to best, MMS, STD, and OHE show relatively similar results, improving with increasing sample sizes. For the EN model, OHE performs best, and the remaining three, from worst to best, MMS, DEF, and STD show relatively similar results. The sample size has a comparatively small impact on the performances. Results with sample sizes of 200 and larger only show a minor oscillation and remain otherwise stable. The comparison between models shows that RF outperforms the other models. All models

except for EN show significantly better performances with larger sample sizes. However, MLP was impacted the most by the sample size. It overtook the performance of SVM at a sample size of 450 and EN at 900. Secondly, we will discuss the next larger SPL, 7z. We provide the results for this dataset in Figure 2. The first observation we can make is that the overall quality of the predicted results decreased. This matches our expectations, since we are predicting the performance of a larger SPL using an equivalent setup. Another observation we can make is that three of the five tested models showed strong oscillations in their performances or, for some preprocessing methods, a worsening of the performance with increasing sample size. The MLP model, for example, showed for the best and second best performing preprocessing methods, MMS and STD, respectively, no significant changes with increasing sample sizes. DEF and OHE showed meanwhile a decrease in performance with increasing sample sizes. The EN and SVM models showed strong oscillations with increasing sample sizes. However, the performances of the preprocessing methods all follow that same pattern, which suggests that the cause for this may lie in the model or the SPL rather than the preprocessing approaches. In the case of the SVM, the performances remained very similar. The preprocessing performances in the EN model follow the same oscillation

Model	Preprocessing	BerkeleyDBC	7z	VP9
MLP	DEF	25.68%	114.76%	273.94%
MLP	MMS	26.29%	99.97%	144.24%
MLP	STD	31.12%	99.98%	129.80%
MLP	OHE	10.27%	104.41%	117.11%
KNN	DEF	1.44%	119.01%	182.27%
KNN	MMS	4.29%	82.30%	158.79%
KNN	STD	2.85%	78.42%	124.31%
KNN	OHE	4.47%	79.99%	241.55%
RF	DEF	0.54%	9.51%	15.14%
RF	MMS	0.55%	9.54%	14.97%
RF	STD	0.55%	9.50%	15.23%
RF	OHE	0.54%	10.82%	18.32%
SVM	DEF	6.44%	91.46%	249.59%
SVM	MMS	6.07%	91.33%	101.45%
SVM	STD	6.04%	91.29%	100.35%
SVM	OHE	6.03%	91.36%	116.08%
EN	DEF	5.19%	176.36%	246.24%
EN	MMS	5.31%	177.54%	273.19%
EN	STD	5.09%	169.40%	267.77%
EN	OHE	2.54%	173.59%	226.60%

Table 5

Average MAPE value over all tested sample sizes from 50 to 1000 with steps of 50

pattern while being displaced with a relatively constant margin along the y-axis, with STD performing best. The KNN model performed as expected, showing constant improvements with increasing sample sizes for all preprocessing methods. However, it is notable that the best performer on the BerkeleyDBC dataset DEF performs the worst now, with the former second-best performing STD taking its place as the best performer. The RF model remains again the best performer with a significant margin. The preprocessing performances are again very similar, but STD performs significantly better for the smallest tested sample size, thus outperforming the others.

Thirdly, we will discuss the largest SPL we investigated, VP9. The results collected for VP9 are shown in Figure 3. Our first observation is that the results for VP9 are closer to the results from BerkeleyDBC. There are again some oscillations in the results of MLP, but they are comparatively minor and show a clear trend to improvement with increasing sample sizes. We see again that the performance of the EN model remains unaffected by increasing sample sizes, except for some minor oscillations. The SVM model shows a similar pattern as it did with the BerkeleyDBC dataset. The DEF preprocessing method performs once more the worst and shows as the only method with no significant improvement with increasing sample size. KNN shows to be once more consistent, showing stable improvement with increasing sample size, STD performing best once more. The RF model performs once more best by a significant margin. The preprocessing methods have little impact on its performance, but some improve the prediction performance earlier, the

best performing being MMS.

Finally, we will discuss the results and our observations in general. To this end, we provide the average performances of all models and preprocessing approaches in Table 5. The first observation must be that preprocessing methods have a significant impact on the prediction performances. BerkeleyDBC has an average factor of 2.05 between the best and worst-performing preprocessing methods. In comparison, 7z and VP9 have an average factor of 1.17 and 1.84 respectively. This observation holds for all tested models, even for the best-performer RF. However, RF shows this impact only with the larger SPLs like 7z and VP9. In general, the differences are maximized at low sample sizes and become then smaller with increasing sample sizes. We observe a similar situation with the SVM model, except DEF, which was largely unsuited. DEF was in two out of three tested SPLs performing the worst, showing insignificant improvement with increasing sample sizes. For MLP, KNN, and EN, on the other hand, we can see significant performance differences on every sample size tested, with, in general, more pronounced differences when applying smaller sample sizes. We also observe multiple occasions where misrepresentation of performances could occur when conducting tests with only one preprocessing method. For instance, one can conclude that SVMs outperform MLPs on the BerkeleyDBC dataset for sample sizes smaller or equal to 1000 when conducting tests only with DEF or MMS. However, when testing with STD or OHE, we see that MLP outperforms SVMs on the BerkeleyDBC dataset for sample sizes greater than 650 or 400, respectively. From

this, we conclude that a sound comparison between two or more predictive models should compare their performances when using their best-performing preprocessing methods. Omitting the preprocessing method used may, by extension, lead to poorly reproducible results.

MLP showed to work on average best with OHE. The performance of this model strongly correlated with the sample size, and it usually started with comparatively high MAPE scores that became more competitive with increasing sample sizes. Furthermore, it is prone to oscillation. KNN showed to work on average best with STD. It was one of the most stable and robust models, achieving constant improvement with increasing sample sizes, even in the context of SPLs like 7z that triggered oscillation in most other models. However, its prediction quality places it in the middle field. RF showed to work on average best with MMS. This model outperformed every other model significantly in every aspect we measured. Its worst performance using the smallest tested sample size of 50 outperforms, in all but two cases, the best performances of all other models. This performance is then improved further with increasing sample size. The model usually reaches a plateau relatively early on average at a sample size of 350, after which its improvement slows significantly. SVM showed to work on average best with STD. The model improves like RF on average with a sample size up to 600 steadily, after which the model starts to plateau in its improvement, except for the already mentioned DEF. EN showed to work on average best with OHE. This model showed, on average, comparatively minor improvements with increased sample size.

6. Threats to validity

This paper compared multiple machine learning-based models and explicitly did not perform any parameter tuning for any one of the models. We used, if not stated explicitly differently, always the default parameters defined by the scikit-learn⁷ library [14]. Thus, we must acknowledge that fine-tuning the model parameters, especially for the more complex models like MLP, likely will improve the performances of the models employed. However, the measured results are still valid and valuable for comparing the model performances concerning the preprocessing methods and the sizes of the training sets employed.

7. Conclusion

We tested 15 scenarios of machine learning-based performance prediction in the context of SPLs by measur-

ing the performance of five different machine learning models on three SPLs with training sets of increasing sizes. Except for two, all scenarios tested showed, in part, radical changes in prediction quality depending on the preprocessing method used. These changes were most pronounced when we measured the model performances with only a few samples to use as training sets and became less distinctive with training sets of increased size. On average, the disparity between the worst and the best performing preprocessing method were factors of 2.05 (BerkeleyDBC), 1.17 (7z), and 1.84 (VP9). While we identified the on average best performing preprocessing methods for each model we tested, we also see, as visualized in Table 5, that no single method outperforms all others for each dataset, which holds as well if we only focus on a single model. Thus, having shown both the significant impact and the inconsistency in the performance of preprocessing methods, we draw the following conclusions. Results that do not state which, if any, preprocessing method was employed become hard to reproduce. Further, the disregard of preprocessing methods may pose a threat to the validity of results. In summary, preprocessing methods are a high-impact, low-effort, and inconsistent part of the field of SPL performance prediction, and all these properties make them essential to be considered and tested.

References

- [1] J. Gong, T. Chen, Deep configuration performance learning: A systematic survey and taxonomy, 2024. arXiv:2403.03322.
- [2] P. Clements, L. Northrop, Software product lines, Addison-Wesley Boston, 2002.
- [3] E. Engström, P. Runeson, Software product line testing – a systematic mapping study, *Information and Software Technology* 53 (2011) 2–13. URL: <https://www.sciencedirect.com/science/article/pii/S0950584910001709>. doi:<https://doi.org/10.1016/j.infsof.2010.05.011>.
- [4] J. Oh, D. Batory, R. Heradio, Finding near-optimal configurations in colossal spaces with statistical guarantees, *ACM Trans. Softw. Eng. Methodol.* 33 (2023). URL: <https://doi.org/10.1145/3611663>. doi:10.1145/3611663.
- [5] N. Siegmund, M. Rosenmuller, C. Kastner, P. G. Garruso, S. Apel, S. S. Kolesnikov, Scalable prediction of non-functional properties in software product lines, in: 2011 15th International Software Product Line Conference, IEEE, 2011, pp. 160–169.
- [6] M. F. Dacrema, P. Cremonesi, D. Jannach, Are we really making much progress? A worrying analysis of recent neural recommendation approaches,

⁷<https://scikit-learn.org/stable/index.html>

- CoRR abs/1907.06902 (2019). URL: <http://arxiv.org/abs/1907.06902>. arXiv:1907.06902.
- [7] C.-L. Wu, K.-W. Chau, Y.-S. Li, Predicting monthly streamflow using data-driven models coupled with data-preprocessing techniques, *Water Resources Research* 45 (2009).
 - [8] J. Rasekhi, M. R. K. Mollaei, M. Bandarabadi, C. A. Teixeira, A. Dourado, Preprocessing effects of 22 linear univariate features on the performance of seizure prediction methods, *Journal of Neuroscience Methods* 217 (2013) 9–16. URL: <https://www.sciencedirect.com/science/article/pii/S0165027013001246>. doi:<https://doi.org/10.1016/j.jneumeth.2013.03.019>.
 - [9] M. Acher, H. Martin, J. A. Pereira, A. Blouin, J.-M. Jézéquel, D. E. Khelladi, L. Lesoil, O. Barais, Learning very large configuration spaces: What matters for Linux kernel sizes, Ph.D. thesis, Inria Rennes-Bretagne Atlantique, 2019.
 - [10] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, H. Yu, Data-efficient performance learning for configurable systems, *Empirical Software Engineering* 23 (2018) 1826–1867.
 - [11] H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J.-M. Jézéquel, D. E. Khelladi, Transfer learning across variants and versions: The case of linux kernel size, *IEEE Transactions on Software Engineering* 48 (2022) 4274–4290. doi:10.1109/TSE.2021.3116768.
 - [12] J. A. Pereira, M. Acher, H. Martin, J. Jézéquel, G. Botterweck, A. Ventresque, Learning software configuration spaces: A systematic literature review, *Journal of Systems and Software* 182 (2021) 111044.
 - [13] J. Alves Pereira, M. Acher, H. Martin, J.-M. Jézéquel, Sampling effect on performance prediction of configurable systems: A case study, in: *Proceedings of the ACM/SPEC International Conference on Performance Engineering, ICPE '20*, Association for Computing Machinery, New York, NY, USA, 2020, p. 277–288. URL: <https://doi.org/10.1145/3358960.3379137>. doi:10.1145/3358960.3379137.
 - [14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* 12 (2011) 2825–2830.
 - [15] M. Acher, H. Martin, L. Lesoil, A. Blouin, J.-M. Jézéquel, D. E. Khelladi, O. Barais, J. A. Pereira, Feature subset selection for learning huge configuration spaces: the case of linux kernel size, in: *Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A, SPLC '22*, Association for Computing Machinery, New York, NY, USA, 2022, p. 85–96. URL: <https://doi.org/10.1145/3546932.3546997>. doi:10.1145/3546932.3546997.
 - [16] L. Bao, X. Liu, F. Wang, B. Fang, Actgan: Automatic configuration tuning for software systems with generative adversarial networks, in: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 465–476. doi:10.1109/ASE.2019.00051.
 - [17] J. Gong, T. Chen, Predicting software performance with divide-and-learn, in: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023*, Association for Computing Machinery, New York, NY, USA, 2023, p. 858–870. URL: <https://doi.org/10.1145/3611643.3616334>. doi:10.1145/3611643.3616334.
 - [18] S. Fu, S. Gupta, R. Mittal, S. Ratnasamy, On the use of ML for blackbox system performance prediction, in: *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, USENIX Association, 2021, pp. 763–784. URL: <https://www.usenix.org/conference/nsdi21/presentation/fu>.
 - [19] Q. Cao, M.-O. Pun, Y. Chen, Deep learning in network-level performance prediction using cross-layer information, *IEEE Transactions on Network Science and Engineering* 9 (2022) 2364–2377. doi:10.1109/TNSE.2022.3163274.
 - [20] J. Cheng, C. Gao, Z. Zheng, Hinnperf: Hierarchical interaction neural network for performance prediction of configurable systems, *ACM Trans. Softw. Eng. Methodol.* 32 (2023). URL: <https://doi.org/10.1145/3528100>. doi:10.1145/3528100.
 - [21] K. Zhu, S. Ying, N. Zhang, D. Zhu, Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network, *Journal of Systems and Software* 180 (2021) 111026. URL: <https://www.sciencedirect.com/science/article/pii/S0164121221001230>. doi:<https://doi.org/10.1016/j.jss.2021.111026>.
 - [22] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, A. Cristal, A machine learning approach for performance prediction and scheduling on heterogeneous cpus, in: *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2017, pp. 121–128. doi:10.1109/SBAC-PAD.2017.23.
 - [23] K.-T. Ding, H.-S. Chen, Y.-L. Pan, H.-H. Chen, Y.-C. Lin, S.-H. Hung, Portable fast platform-aware neural architecture search for edge/mobile computing ai applications, *ICSEA 2021* (2021) 108.
 - [24] Y. Gao, X. Gu, H. Zhang, H. Lin, M. Yang, Runtime performance prediction for deep learning

models with graph neural network, in: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2023, pp. 368–380. doi:10.1109/ICSE-SEIP58684.2023.00039.