To Tune or not to Tune: Hyperparameter Influence on the Learning Curve

Prajit Bhaskaran
Supervisor(s): Tom Viering, Marco Loog
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

A learning curve displays the measure of accuracy/error on test data of a machine learning algorithm trained on different amounts of training data. They can be modeled by parametric curve models that help predict accuracy improvement through curve extrapolation methods. However, these learning curves have only been mainly generated from default learning algorithms. Research into tuning the machine learning algorithm and its effect on the learning curve has not been adequately researched. This research aims to look at the influence of hyperparameter tuning on the learning curve. This regards not only how the learning curve shape changes in general but also how different parametric models are affected when a learner undergoes tuning. We experiment with the decision tree and KNeighbors classifier which undergo significant hyperparameter tuning. We find that the tuned learner performs marginally better than the default learner for anchors past 25% of the data for the majority of the tested datasets. We also observe that the tuned learner displays a smoothing behaviour that makes ill-behaved curves more well-behaved. In terms of the curve fitting, the tuned learner does not uncover any curve models nor does it show any statistical significance, and instead performs very similarly to the default learners.

1 Introduction

In the context of machine learning, scarce data is the unavailability of data that could increase a system's accuracy [1]. The size of data is naturally one of the primary factors that influence the precision of an ML (machine learning) algorithm. However, data collection and preparation is a significant bottleneck in ML [2] . This suggests that further data sampling may not be a viable option considering the unknown improvement in accuracy it may offer-even if the gain in accuracy may outweigh the difficult process of collecting data.

Learning curves may offer an answer to this problem of unknown accuracy improvement. A learning curve displays the measure of accuracy/error on test data of an ML algorithm trained on different amounts of training data [3]. Modelling learning curves effectively can offer insight into the amount of data one may need to attain a certain level of classification accuracy. Furthermore, learning curves can also be extrapolated by parametric learning curve models [4] which have the potential to display the trade-off between collecting more data and the gain in the accuracy improvement.

However, there is no concluding evidence of a one size fits all curve model [4]. To have better fitting learning curve models, extensive hyperparameter tuning may be required [5]. Even if it is at the expense of worse performance, it may lead to a better estimation of the error improvement rate. To our knowledge, there has been limited to no research on hyperparameter tuning and its effect on learning curves. This paper aims to answer the question:

What is the influence of hyperparameter tuning on the learning curve?

This question gives further insight into two sub-questions:

1. How does the general learning curve of a particular ML algorithm change when hyperparameter tuned?
2. How does the curve fitting differ for hyperparameter tuned models against the default models?

These questions may not only help understand how hyperparameter tuning improves the learning curve but it can also give insight into the relationship between ML algorithms and learning curve models. This research aims to answer these questions by experimenting and analysing two classification ML algorithms tested on six different curve models across 20 datasets.

Section 2 covers the related works and presents the current findings of hyperparameter tuning and the current optimal parametric curve fitting models. Section 3 discusses the methodology behind this paper's contribution; it introduces the methods used to generate and fit the learning curve. Section 4 covers the specific implementation choices for the experimental setup. Section 5 gives an overview of the results obtained. Section 6 is a reflection of the paper and evaluates the validity of the results. Section 7 gives concluding remarks from the results obtained as well as motivation for future work. Lastly, section 8 gives insight into the ethics and reproducibility of the project.

2 Related Work

There remains a substantial amount of uncertainty with regard to hyperparameter tuning and its effect on the learning curve. However, sizeable research has been done with regard to analysing the predominant hyperparameters of a learner. Van Rijn et al. [6] have found that the most influential tuning parameters for the random forest are the minimal samples per leaf, maximal features for determining a split, and the split criterion. Thus, when tuning learners such as decision trees, the significance of these hyperparameters can be appropriately taken into account (such as using a wider range of values for tuning).

In the deep learning field, hyperparameter tuning has been investigated to a certain extent. Kornblith et al. [7] have found that tuned deep learning models marginally outperform the standard model as the gap is often small and narrows as the training size grows. Hoiem et al. [5] have found that a non-pretrained but finetuned deep learning model performs significantly better than a non-pretrained default model. This research indicates that patterns like these may arise with other machine learning algorithms as well.

A considerable amount of research has been done with regard to analysing the ideal parametric curve model across a range of learners. Viering et al. [4] have reviewed the empirical and analytic experiments on learning curves and have found that there continues to be contention between the optimal learning curve model. For example, Frey et al.[8] and Last [9] have claimed that the power law fits best for decision

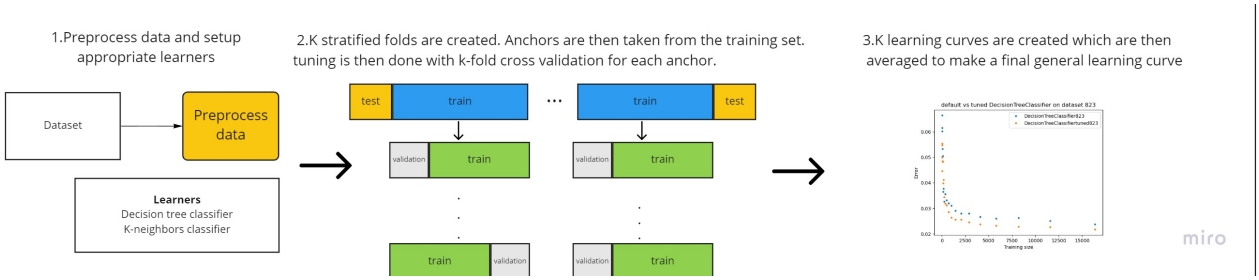

Figure 1: Experiment setup with regards to creating a general learning curve.

tree algorithms, while Singh [10] tests four learning algorithms (Decision trees, K-nearest neighbors, SVM, ANN) and finds that using an exponential function is ideal. Booonyanunta's [11] analytical research supports Singh's paper by claiming the exponential function is best suited for fitting a learning curve. Thus, it seems there is no concluding evidence of a one size fits all learning curve model. Not only do the results remain indecisive, but they also come with a few limitations. Firstly, none of the aforementioned papers tune the learners at all. Instead, the default implementations are used. Secondly, Singh et al. [10] only test the four learners on four datasets, while Last et al. [9] test one learner on 5 datasets. This brings to light that perhaps an inadequate number of datasets have been used to make such a generalisation on the best fitting model. This paper will experiment with 20 datasets (varied in class and feature distribution) which may help give a more concrete result in the best fitting model. Recently, Mohr et al. [12] did an empirical analysis on learning curves and model fitting with 20 different learners and 246 datasets. It was found that the four parameter models performed one of the best(pow4, mmf4, exp4) followed by pow3 with regards to the mean squared error. However, the paper takes an average of all learners across all datasets which may lose or miss partially fine detail that a certain dataset or learner displays. A key difference in this work is that we focus on only two supervised learners and analyse them independently and together in terms of the general learning curve performance and the curve fitting.

3 Learning Curve Estimation

This section discusses the motivation for using certain methods and tools in the experiment. We first describe the methods for constructing the general learning curve and collecting the data. We then describe the methods for extrapolating the estimated general learning curve.

3.1 General learning curve

The general learning curve is carefully constructed through *nested cross validation* [13]. The outer nest is generated through k stratified folds. For each of these folds, a training set and a test set is generated. The stratified folds help ensure the class distribution of the dataset is represented appropriately in the sets [13].

Figure 1 displays the step-by-step process of generating a learning curve. The first step is to preprocess the data and set the hyperparameter grid for the appropriate learners. The second step constructs the training and testing sets through cross-validation, and the final step displays the resulting learning curve of the default learner compared to the tuned learner. Steps 2 and 3 will be outlined below in more detail.

Within the training set, we create training anchors. These anchors are defined as being a certain number of data points that the learner trains on. We vary the training anchors by S_n and they are computed by the formula

$$S_k = \lceil 2^{\frac{7+k}{2}} \rceil \text{ [12]}$$

where the smallest anchor is 16 for $k = 1$. The range of k is defined as $[1, \lfloor 2 \cdot \log_2 N - 7 \rfloor]$ where N is the size of the dataset.

Each training anchor is monotonically increasing. This means that if we have training samples S_1 and S_2 and $|S_1| < |S_2|$ then $S_1 \subset S_2$. With the established anchors, training and testing can be done. Once the learner is trained on the training set, all unused training samples are appended to the test set. The advantage that comes with this is that no data is being discarded/unused which would have been wasteful.

For each anchor within each fold, the tuned learner undergoes cross-validation (inner nest) to estimate the optimal hyperparameters while the default learner is just trained on the anchor. Since there are k stratified folds, k learning curves are generated and averaged to make a final learning curve as seen in step 3 of Figure 1.

3.2 Extrapolation

The curve fitting was done using the existing LCDB code which is explained in the ECML paper [12]. The implementation utilises a curve optimise function to analytically estimate the curves for a certain parametric curve model. For curve fitting, the Levenberg-Marquadt [14] algorithm is used. This algorithm is used to solve non-linear least squared problems which tries to fit a set of m observations to n unknown parameters where $m \geq n$. Using random initial parameters, the optimisation runs numerous times to try and find the best parameters. If no parameters can be found after a certain number of attempts then the particular fit is discarded from the analysis. More detail can be found in the supplement of the paper.

4 Experimental Setup

This section gives an overview of the setup of the experiment. It will discuss the specific algorithms, values, and iterations

used for constructing the general learning curve as well as extrapolating and fitting the curve.

4.1 Datasets

The datasets were all retrieved from an open-source platform called OpenML [15]. 20 different datasets were used in this experiment and details of the datasets can be viewed in the appendix in section A. The learning curves generated by the decision tree and KNeighbors learners can also be viewed in the appendix in section B.

4.2 General learning curve

For the preprocessing of dataset, missing values are imputed by the median of the corresponding attribute. All categorical attributes are mapped to a one hot encoding. The preprocessing is carried out per stratified split to avoid bias. For the outer (stratified) folds and inner (cross-validation) folds, $k = 10$ and $k = 5$ were used, respectively. The learners used were the KNeighbors classifier (KN) and the decision tree classifier (DT). Table 1 and table 2 display the hyperparameters used for the KNeighbors and decision tree classifier. It shows the hyperparameters used for the default learners as well as the hyperparameter ranges used for the tuned learner. Numbers within square brackets denote the interval of numbers used. The code was made in collaboration with Nguyen et al. [16] and can be found in the Github [1].

Hyperparameters	KN default	KN tuned
neighbors	5	[1,20]
weights	uniform	uniform, distance
p	2	2
metric	Minkowski	Minkowski

Table 1: Hyperparameter used for the default and tuned KNeighbors learner

Hyperparameters	DT default	Dt tuned
criterion	gini	gini, entropy, log-loss
splitter	best	best, random
max depth	None	None, [1,10]
min samples split	2	2
min samples leaf	1	[1,12]
random state	42	42
min weight leaf	0	0
max features	None	None

Table 2: Hyperparameter used for the default and tuned decision tree learner

To clarify some of the lesser-known hyperparameters uses, for the Kneighbors, the Minkowski metric along with the p value of 2 means that euclidean distance is used to compare the data points. For the decision tree, min samples leaf refers to the minimum number of samples required to be considered a leaf node. Min weight leaf refers to the minimum weighted fraction of the sum of total weights required to be at a leaf node. This is only applied if samples are given unequal weights. More detail about each hyperparameter can be found in the Sci-kit documentation for the KNeighbors [2] and decision tree [3] learners.

4.3 Extrapolation

model	Formula
POW2	$-an^{-b}$
POW3	$an^{-b} + c$
EXP2	$a\text{exp}(-bn)$
EXPP3	$c-\text{exp}((n-b)^{\alpha})$
log2	$c - \frac{a}{log(n)}$
logpow3	$a/((n\text{exp}(-b))^{c} + 1)$

Table 3: Parametric learning curve models and their respective formulas where n is the varying training sizes

Table 3 displays the six different parametric models taken into consideration.Here a, b, c denote the parameters of the function and n denotes the training sizes. For the curve fitting, initial parameters were randomly generated at most 1000 times per fit, and the optimisation was repeated 25 times with random initial parameters. Extrapolation performance was measured through ranks, which display the average rank of a model relative to other models and is tested against the Wilcoxon signed rank test. The ranks , that this report focuses on, are determined by the average MSE values of a fitted model on a particular dataset. This means that different training anchors are used to fit the curve and then the MSE is averaged across the rest of the anchors (test anchors). The lower the MSE value is for a model in a certain dataset, the lower it will rank. Its rank is then averaged across all datasets. A Friedman test [17] is also conducted to determine whether the difference in model performance is significant with $\alpha = 0.05$. The ranks are also visualised with the critical diagrams (CD) that display the average ranking of the parametric models and red lines connected between two models display statistically non-significant pairs.

In order to speed up the process of generating the learning curves and fitting the models, we used Delft's high performance computing centre [18]. This allows for parallel computing and can speed up parts of the experiment such as the cross-validation, and curve fitting.

5 Results

How does the general learning curve of a particular ML algorithm change when hyperparameter tuned?

With regards to the general learning curve, numerous important features come to the surface. Figure 2 displays the performance of the default KNeighbors classifier against the tuned Kneighbors classifier on a particular dataset. The vertical lines show the standard error at each anchor. As

[1] https://github.com/pbhaskaran/Research-Project

[2] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

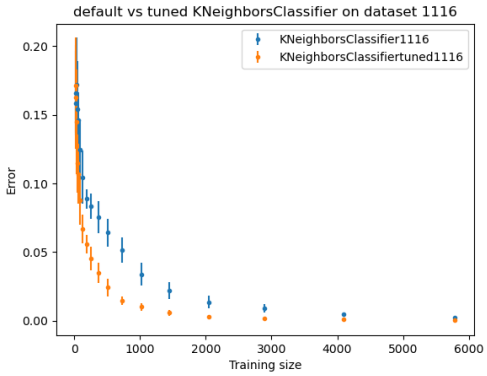

Figure 2: Learning curves of tuned and default KNeighbors classifier with standard error

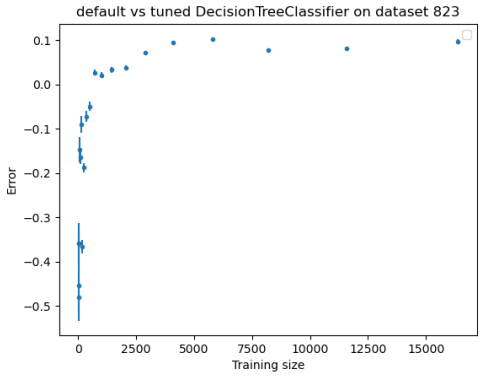

Figure 3: performance difference of tuned and default Decision tree classifier

represented in figure 2, the performance of the default learner against the tuned learner for small anchors does not indicate better performance for the tuned learner. The main reason for this is due to the construction of the experiment. For small anchors in fairly complex datasets (datasets with more than two classes or having more than 5 important features), even though the splits are stratified, the tuned learner does not have enough training samples that properly represent the full dataset, and so it does not optimally classify the data points correctly. This is also reflected by the large standard errors. The figure shows that the tuned learner has greater standard error than the default learner for the smaller anchors. This occurs because the training set anchors are quite varied within the different stratified folds so there is a large fluctuation in the error estimate. In figure 2, the standard error in the early anchors(until 300) for the tuned classifier is at least as large as the standard error in the default classifier.

However, the previous figure also displays that as the training anchors increase, the tuned learner starts performing better than the default learner. In fact, from the 15/20 datasets, the tuned learners performed better than the default learners after around the 25% anchor mark (25% of all data being used for training). This is expected because, with more training data, the tuned learner can get a much better representation of the class and feature distribution within the dataset and find ideal hyperparameters through minimising the loss function and maximisng the model performance.

One question that arises from the previous findings is whether this improvement in accuracy is significant. Qualitatively, looking at figure 2, we can see that the standard error lines do not overlap for the later anchors. This means that the tuned learner will always perform better than the default learner for later anchors in this particular dataset. Furthermore, we can also perform a paired t-test to confirm the results. The reason we can do a paired t-test is because the data are paired by having the same training and testing data. Assuming all the conditions of the paired t-test hold, we get a t-value of 4.56. This is larger than the t-value, against α = 0.05, of 2.11. We can therefore conclude that the difference is significant. Another thing to note is that when looking at error performance differences between the default and tuned learner, certain properties arise. Figure 3 displays the difference in accuracy between the tuned decision tree classifier and the default decision tree classifier. The accuracy, d, is computed as $d = \frac{E_{default} - E_{tuned}}{E_{default}}$. The standard errors are also shown as the vertical lines. The graph displays that the tuned learner does not outperform the default learner at the start (as seen by a negative difference), but as the anchors increase, the performance of the tuned learner also increases. The shape of the difference curve in figure 3 also resembles a log shape with plateauing. In fact, 6/20 datasets also seemed to display this log behaviour with plateauing. This may indicate that a certain parametric function may be able to model the difference between the default and tuned learner. If an appropriate model is found, then we have a more concrete idea of error improvement a tuned learner may offer at certain anchors. Out of the 20 datasets tested, 16 displayed positive accuracy at the later training anchors for both the decision tree and the KNeighbors classifier.

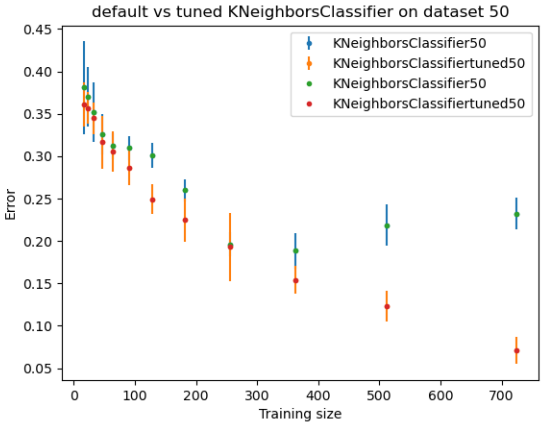

Figure 4: Default vs Tuned KNeighbors classifier

One interesting thing to note is that tuning learners can also smooth out unexpected behaviour and make a curve

model	DT D	DT T	KN D	KN T
POW2	0.091 ± 0.240	0.066 ± 0.193	0.065 ± 0.178	0.042 ± 0.134
POW3	0.011 ± 0.016	0.009 ± 0.014	0.006 ± 0.017	0.005 ± 0.015
EXP2	0.262 ± 0.670	0.225 ± 0.330	0.231 ± 0.670	0.180 ± 0.662
EXPP3	0.206 ± 0.324	0.012 ± 0.018	0.012 ± 0.013	0.007 ± 0.010
LOG2	0.012 ± 0.017	0.016 ± 0.031	0.089 ± 0.019	0.010 ± 0.018
logpower3	0.062 ± 0.016	0.013 ± 0.020	0.012 ± 0.036	0.007 ± 0.018

Table 4: MSE values averaged across all test anchors against default and tuned learners. D stands for the default learner. T stands for the tuned learner

well-behaved. Viering et al.[4] define a curve as being well behaved if it shows a reducing error with increasing training sample sizes (for all training sizes). This phenomenon can be seen in figure 4 which displays the default KNeighbors classifier against the tuned KNeighbors classifier on dataset 50. The default learner's learning curve displays a few fluctuations. Firstly, there is a slight increase in error in the early anchors (from anchor 45 to anchor 64), then up until 250, it behaves as expected, and then right after the error seems to increase as the training sizes increase which clearly violates the property of being well behaved. On the other hand, the tuned learner continues to display the expected decreasing relationship between the error and training size and smooths out the unexpected behaviour. 4/20 datasets displayed this smoothing phenomenon that occurs in figure 4. This behaviour is important because the curves follow a more sensible shape (error decreases as training sizes increase) and can also produce better curve fits. However, the question arises of how much better the curve fitting becomes for the tuned learners compared to the default learners for the 4 datasets. This can be seen by measuring the average MSE of the test anchors.

How does the curve fitting differ for hyperparameter tuned models against the default models?

We can analyse how the curve fit MSE changes for the default learner against the tuned learner for the four datasets that display this smoothing behaviour. Table 4 displays the average MSE of the decision tree and KNeighbors classifier against its tuned counterparts. The table shows that the MSE for the tuned decision tree classifier is lower for 5/6 of the curve models. Furthermore, the tuned KNeighbors classifier has MSE values lower for all of the curve models. The standard deviation for both tuned learners is also considerably lower for each parametric curve fit. This suggests that when dealing with ill-behaved learning curves shown in table 4, tuning may offer much better curve fits for the different parametric models.

To compare the curve performances for the 20 datasets, ranks are used as it helps deal with outlier bias. Table 5 displays the curve performances on the decision tree classifier of the six parametric models for different anchor percentages. Table 6 displays the same attributes but for the tuned decision tree classifier. Table 5 displays that the pow3 model outperforms all other models for anchors all and 5%. The expp3 model outperforms for anchors 10% and 40%, logpow3 outperforms for anchors 20%, while log2 outperforms at 80%. The tuned decision tree learner, in table 6, shows that the pow3 model does not outperform all models at all. Instead, the expp3 and logpow3 models have the best rank for the corresponding anchors. This displays that hyperparameter tuning for the decision tree does not uncover a certain parametric model. Furthermore, the 2 parameter models(pow2, exp2, log2) perform much worse than the three parameter models-for both the default and tuned decision tree. Although the tuned table has different results to the default table, the question arises of whether the change in model performance is significant. Figure 5 displays that for the test MSE, even as the default DT learner and the DT tuned learner have different top three rankings for anchors up to 10%, the results are not statistically significant. The first critical diagram displays that pow3 is not better than the three models ranked below it, and the second critical diagram displays that expp3 is not statistically better than any of the other models.

Decision tree default

curve	pow2	exp2	log2	pow3	expp3	logpow3
all	4.09	5.59	3.11	2.54	2.91	2.77
5%	4.00	5.83	3.00	2.00	2.67	3.50
10%	4.17	5.83	3.25	2.58	2.17	3.00
20%	3.67	5.67	3.27	2.73	3.00	2.67
40%	4.42	5.58	3.26	2.58	2.53	2.63
80%	3.68	5.11	2.74	2.89	3.47	3.11

Table 5: Ranks for extrapolating to all test anchors of default decision tree

Decision tree tuned

curve	pow2	exp2	log2	pow3	expp3	logpow3
all	3.93	5.66	3.34	2.76	2.77	2.54
5%	3.33	5.83	3.00	3.33	2.67	2.83
10%	3.86	5.43	3.14	2.71	2.29	3.57
20%	3.60	5.93	3.20	3.80	1.93	2.53
40%	3.94	5.61	3.61	2.83	2.78	2.22
80%	4.42	5.32	3.37	2.68	2.95	2.26

Table 6: Ranks for extrapolating to all test anchors of tuned decision tree

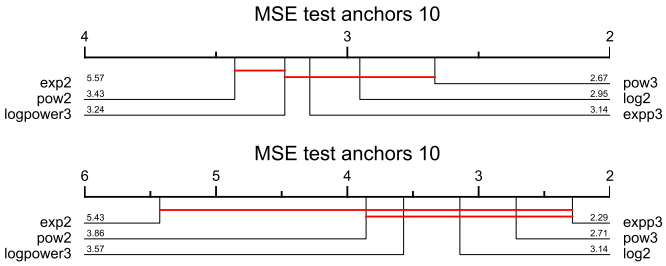

Figure 5: Critical diagram of decision tree default(top) and tuned(bottom) for MSE test anchor up to 10% of total dataset. Curve models connected with a red line are statistically insignificant

Similar behaviour is reflected in the rank tables for the

KNeighbours classifier. For the ranking table which extrapolates all test anchors, the default learner has the pow3 model as the best for 5/6 anchor percentages while the tuned learner has the pow3 model as the best for 4/6 anchors percentages. Once again, the tuned KN learner does not offer an improvement to a certain curve fitting model.However, with the KN default and tuned learner, the pow3's performance is statistically significant compared to the other models for certain anchor percentages. This is shown in figure 6 where the pow3 model is significantly better than the other models. Furthermore the two parameter models for the KN default and tuned learner continue to perform badly by consistently ranking at the bottom.

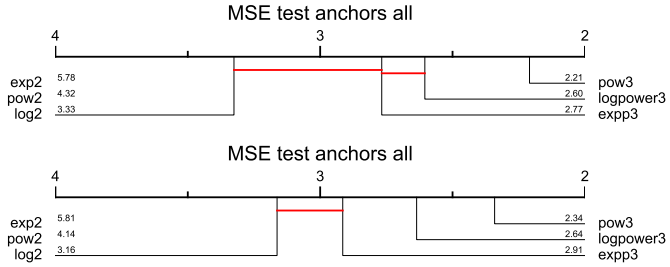

Figure 6: Critical diagram of KNeighbors default(top) and tuned(bottom) for MSE test anchor up to 100% of total dataset. Curve models connected with a red line are statistically insignificant

6 Discussion

One thing to note is that Kornblith's et al. [7] deep learning hyperparameter tuning observation was almost the same as the observation noted in this paper. Both papers display that as the training size grows, the tuned model performs consistently better than the default model. This suggests that the learning curve behaviour of deep learning models and classical models may be quite similar.

With regards to the parametric models, we use pow2, pow3, exp2, expp3, log2, and logpower3. The reason the six models were chosen is that, as found in section 2, they are models that have been empirically analysed as being strong models. Therefore, comparing them is suitable as we can observe which models outperform others when hyperparameter tuned. Although Mohr et al. have found that the three and four parameter models perform best(aside from the lastone model which creates a horizontal line from the last training anchor) [12], it is important to include the two parameter models as well because it has the ability to prevent over fitting. The nature of the two parameter model keeps the generality which could be a more viable option compared to the three parameter models that may overfit. However, we have seen that the two parameter models perform extremely weakly to the three parameter model displaying that it cannot fully capture the extrapolation anchors.

During the test for significance between the learners, we also assumed the conditions of the paired t-test hold. Although this may not be the case. For example, the dependent variable should be approximately normally distributed. However, after making QQ plots of the tuned and default leaner in figure 2, normality cannot be visually seen as shown in section C of the appendix.

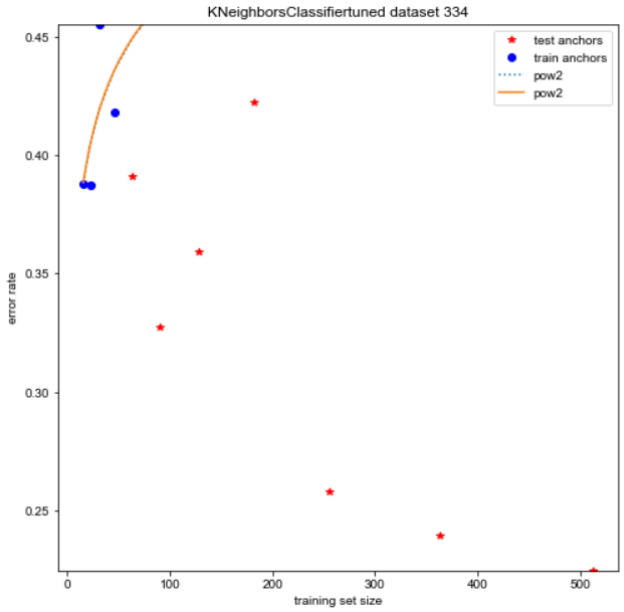

Figure 7: pow2 model fit on tuned Kneighbors classifier

One reason why the results acquired for the curve fitting may not be fully accurate is due to the LCDB database [12] not producing optimal curves. This affects not only the MSE values for the different anchors but also the ranking. Figure 7 displays the tuned KNeighbors classifier trying to model the pow2 curve. It can be seen that the curve fit is clearly not optimal which means better curve fitting models must be researched. Donghwi et al.[19] have analysed different methods for curve fitting so using those methods may result in more accurate results.

7 Conclusion and Future Work

The purpose of this paper was to design and implement methods to analyse the learning curve performance of default and tuned learners. We will list each sub-question and give a brief description of the findings.

How does the general learning curve of a particular ML algorithm change when hyperparameter tuned?

When a learner is hyperparameter tuned, there are a few properties that change the learning curve. Firstly, at early anchors, the standard error for the tuned learner is higher than the standard error of the default learner because of the construction of the experiment. The error for early anchors also tend to be higher for the tuned learners. Secondly, as the anchor sizes increase, the tuned learner starts performing better than the default learner at each training anchor. Thirdly, the tuned learner also displays a smoothing effect such that

it transforms an ill-behaved curve into a more well-behaved curve.

How does the curve fitting differ for hyperparameter tuned models against the default models?

The tuned learner does not uncover any new parametric curve results. The parametric curve fits of the default and tuned learner perform quite similarly. Even the slight changes in curve models the tuned learner shows are not statistically significant as it does not pass the Friedman test. There was no clear optimal parametric model for the default and tuned decision tree, however, the pow3 model was the best performing for the default and tuned KNeighbors. For curves that display ill-behaved properties, the curve fitting for the average MSE values across all test anchors are lower for tuned learners compared to the default learners.

Future work

To further solidify the results, the experiment should also be conducted with more datasets, learners, and curve models. This will not only create more reliable results, but it will also show statistically significant results. Perhaps with more datasets, the tuned DT or KN learner may uncover a parametric model that is statistically significant to other models.

Another piece of work that can be done in the future is to model the difference of accuracy improvement as shown in figure 3. This could be helpful in predicting the gain in improvement a learner may offer when it undergoes hyperparameter tuning and whether it is worth to tune the learner in question.

8 Responsible Research

Ethical considerations must always be accounted for in any research paper. This paper discusses the issue of reproducibility. Being able to reproduce results is important as it can not only solidify or contest observations made in the paper, but it can also add further insight into the observations. However, there are many instances of experiments that cannot be reproduced due to unshared code/data. Or in the context of ML, undisclosed model parameter conditions [20].

Numerous steps have been taken to ensure reproducibility in this experiment. Firstly, a random seed of 42 was used for not only the hyperparameter tuning for certain learners, such as the decision tree classifier, but also for the creation of the stratified splits. Secondly, the curve fitting has also been implemented with a random seed of 42, so the random point selection, and corresponding calculations can also be replicated. Thirdly, all the source code will be published on GitHub as an open-source software (Github linked in section 4). Lastly, there will be an attached Readme document that will contain the following pieces of information; a brief outline of what the code entails and a small description explaining how to generate plots and figures.This allows for the experiments to be easily replicated and also modified for future implementation.

During the entirety of the research, all relevant revelations and results are displayed in section 5. Furthermore, there was no modification of the results that change the projected performance of the different learners.

References

[1] I. Global, "What is data scarcity," 2019.

[2] Y. Roh, G. Heo, and S. E. Whang, "A survey on data collection for machine learning: A big data-ai integration perspective," pp. 1328–1347, 4 2021.

[3] G. I. Webb, C. Sammut, C. Perlich, T. Horváth, S. Wrobel, K. B. Korb, W. S. Noble, C. Leslie, M. G. Lagoudakis, N. Quadrianto, W. L. Buntine, G. Namata, L. Getoor, J. H. Xin Jin, J.-A. Ting, S. Vijayakumar, S. Schaal, and L. D. Raedt, "Learning curves in machine learning," pp. 577–580, 2011.

[4] T. Viering and M. Loog, "The shape of learning curves: a review," 3 2021.

[5] D. Hoiem, T. Gupta, Z. Li, and M. M. Shlapentokh-Rothman, "Learning curves for analysis of deep networks," 10 2020.

[6] J. N. V. Rijn and F. Hutter, "Hyperparameter importance across datasets." Association for Computing Machinery, 7 2018, pp. 2367–2376.

[7] S. Kornblith, J. Shlens, and Q. V. Le, "Do better imagenet models transfer better?" 5 2018.

[8] L. J. Frey and D. H. Fisher, "Modeling decision tree performance with the power law," in *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, D. Heckerman and J. Whittaker, Eds., vol. R2. PMLR, 03–06 Jan 1999, reissued by PMLR on 20 August 2020.

[9] M. Last, "Predicting and optimizing classifier utility with the power law," 2007, pp. 219–224.

[10] S. Singh, "Project report modeling performance of different classification methods: Deviation from the power law," 2005.

[11] N. Boonyanunta and P. Zeephongsekul, "Predicting the relationship between the size of training sample and the predictive power of classifiers," in *KES*, 2004.

[12] F. Mohr, T. Viering, M. Loog, and J. V. Rijn, "Lcdb 1.0: An extensive learning curves database for classification tasks," under review.

[13] 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, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] H. P. Gavin, "The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems," 2020.

[15] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013.

[16] D. Nguyen, T. Viering, and M. Loog, "In search of best learning curve model," unpublished.

[17] M. Friedman, "A comparison for alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, pp. 86–92, 1940.

[18] Delft High Performance Computing Centre (DHPC), "DelftBlue Supercomputer (Phase 1)," https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[19] D. Kim, T. Viering, and M. Loog, "Different approaches to fitting and extrapolating the learning curve," unpublished.

[20] M. Hutson, "Artificial intelligence faces reproducibility crisis," *Science*, vol. 359, no. 6377, pp. 725–726, 2018.

A Dataset information

OpenML ID	Name	# of Classes	# of Instances	Class distribution
6	Letter	26	20000	Almost equal distribution
31	credit -g	2	1000	[700, 300]
37	diabetes	2	768	[268,500]
42	soybean	19	683	[20,20,20,88,44,20,20,92,20, 20,20,44,20,91,91,15,14,16,8]
50	tic-tac-toe	2	958	[626,332]
61	iris	3	150	equal distribution
299	libras_move	15	360	equal distribution
333	monks-problem-1	2	556	[278,278]
334	monks-problem-2	2	601	[206,395]
715	fri_c3_1000_25	2	1000	[443,557]
737	space_ga	2	3107	[1541,1566]
823	houses	2	20640	[8914,11726]
923	visualizing_soil	2	8641	[4753,3888]
1116	musk	2	6598	[1017, 5581]
1120	MagicTelescope	2	19020	[12332,6688]
1462	banknote-authentication	2	1372	[762,610]
1464	blood-transfusion-service-center	2	748	[570,178]
1466	cardiotocography	10	2126	[384,579,53,81,72,332,252, 107,69,197]
1504	steel-plates-fault	2	1941	[1268,673]
40536	SpeedDating	2	8378	[1380,6998]

B Learning curves

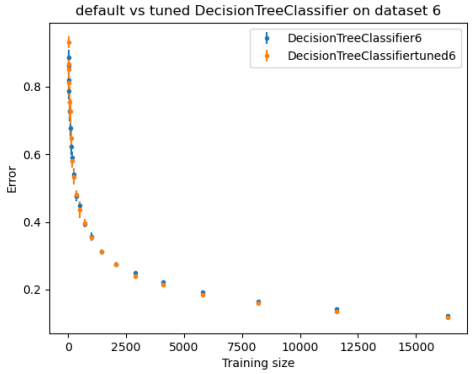

Figure 8: Learning curves of tuned and default Decision classifier with standard error

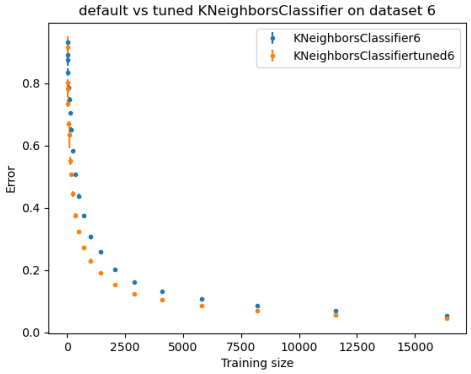

Figure 9: performance difference of tuned and default KNieghbors classifier

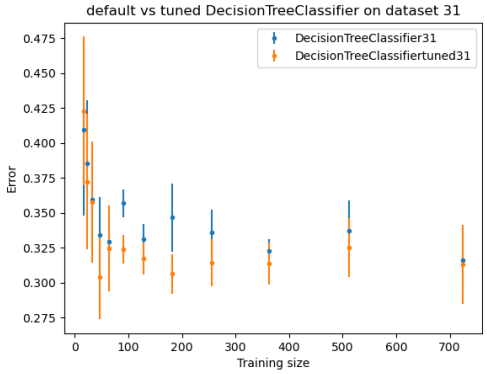

Figure 10: Learning curves of tuned and default Decision classifier with standard error

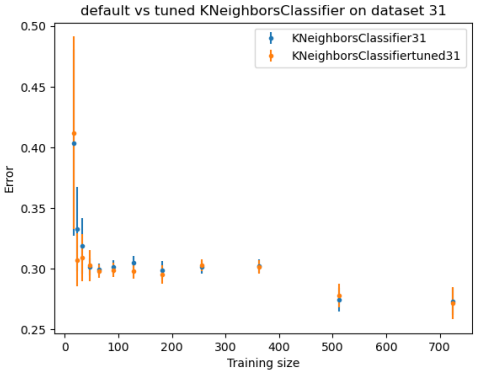

Figure 11: performance difference of tuned and default KNieghbors classifier

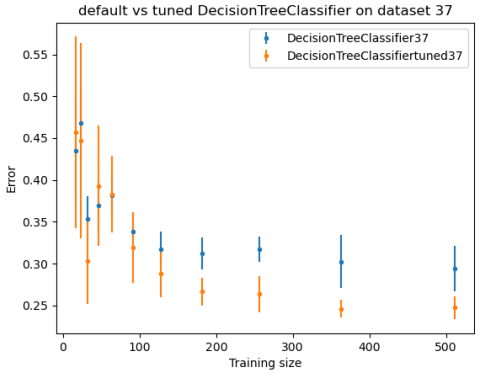

Figure 12: Learning curves of tuned and default Decision classifier with standard error

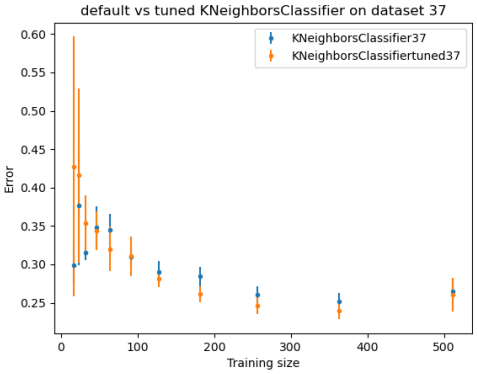

Figure 13: performance difference of tuned and default KNieghbors classifier

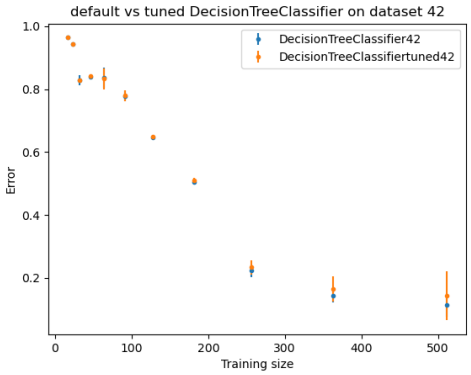

Figure 14: Learning curves of tuned and default Decision classifier with standard error

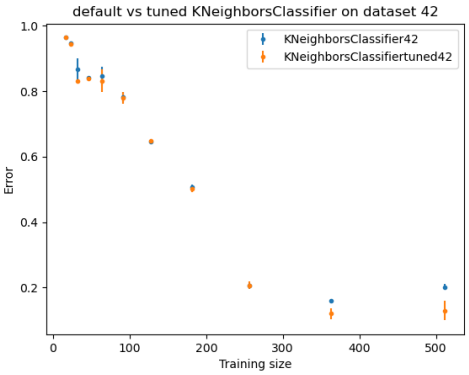

Figure 15: performance difference of tuned and default KNieghbors classifier

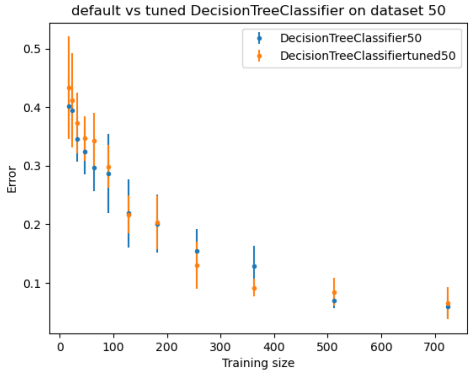

Figure 16: Learning curves of tuned and default Decision classifier with standard error

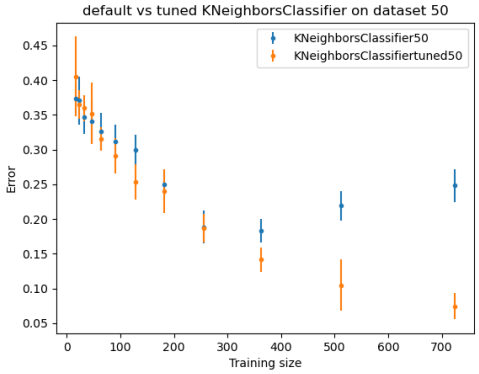

Figure 17: performance difference of tuned and default KNieghbors classifier

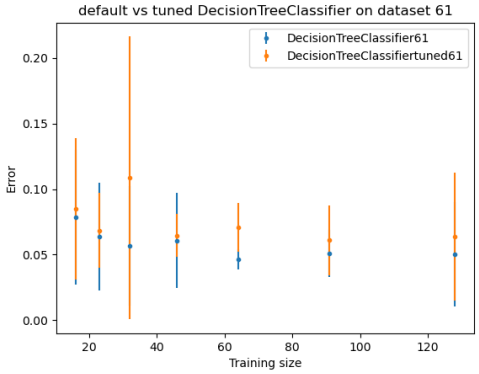

Figure 18: Learning curves of tuned and default Decision classifier with standard error

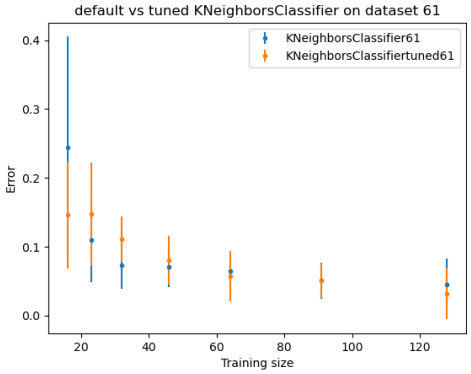

Figure 19: performance difference of tuned and default KNieghbors classifier

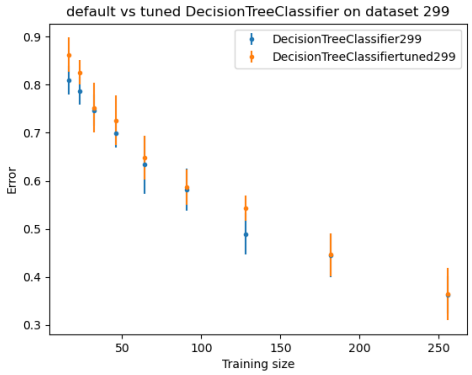

Figure 20: Learning curves of tuned and default Decision classifier with standard error

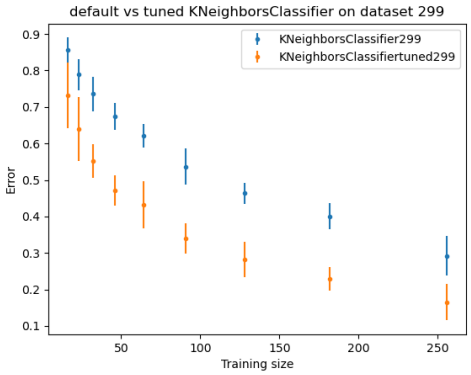

Figure 21: performance difference of tuned and default KNieghbors classifier

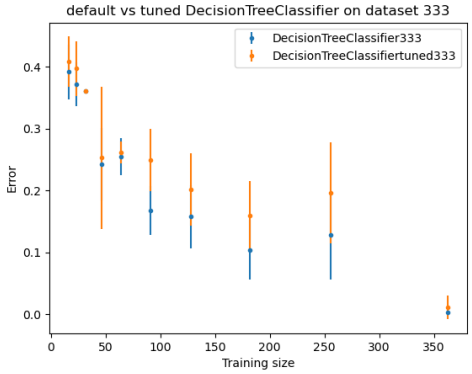

Figure 22: Learning curves of tuned and default Decision classifier with standard error

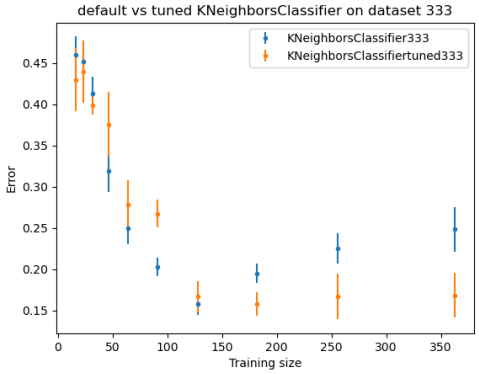

Figure 23: performance difference of tuned and default KNieghbors classifier

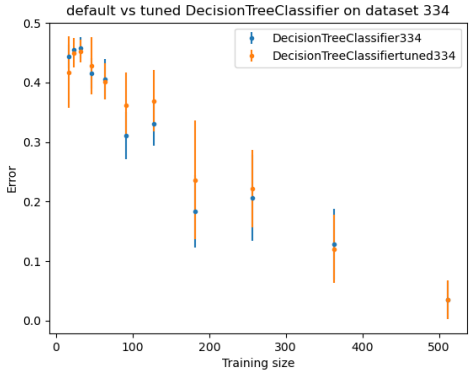

Figure 24: Learning curves of tuned and default Decision classifier with standard error

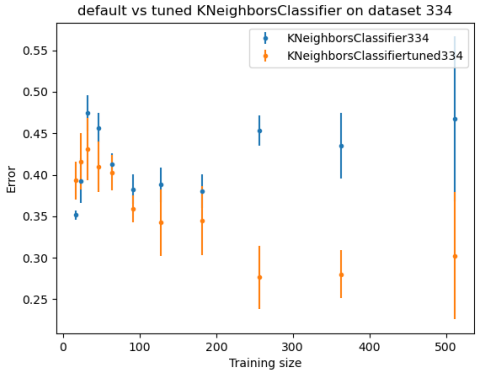

Figure 25: performance difference of tuned and default KNieghbors classifier

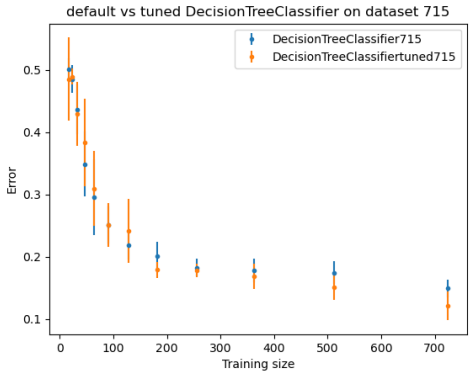

Figure 26: Learning curves of tuned and default Decision classifier with standard error

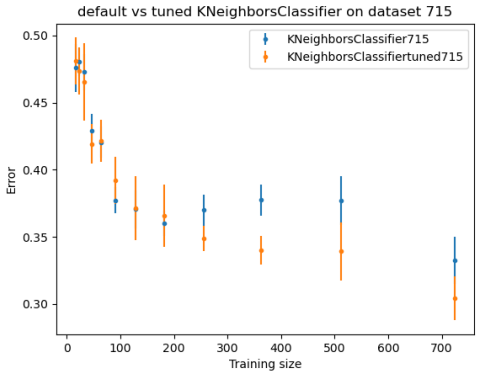

Figure 27: performance difference of tuned and default KNieghbors classifier

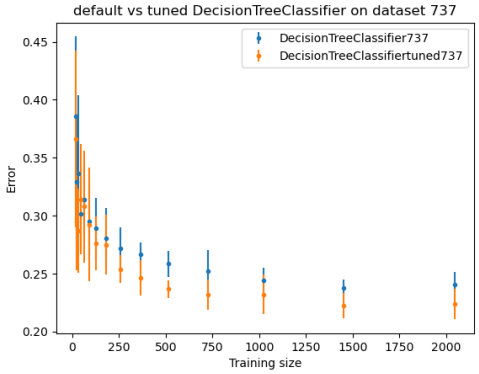

Figure 28: Learning curves of tuned and default Decision classifier with standard error

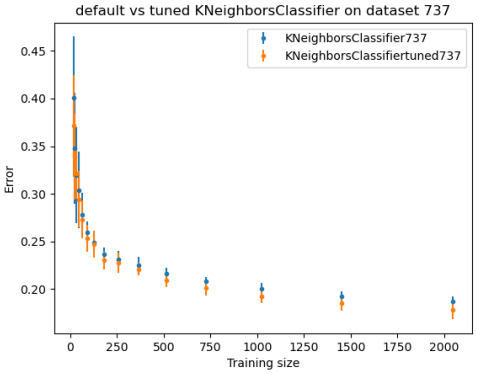

Figure 29: performance difference of tuned and default KNieghbors classifier

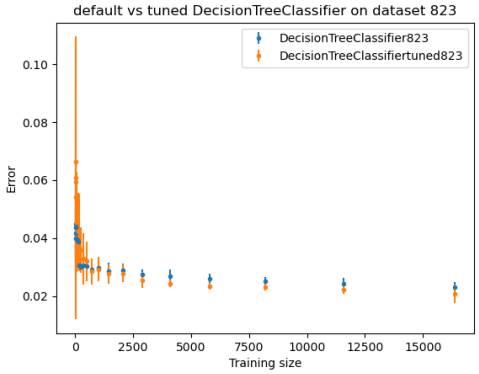

Figure 30: Learning curves of tuned and default Decision classifier with standard error

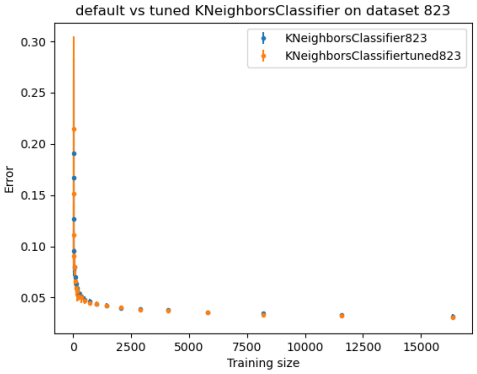

Figure 31: performance difference of tuned and default KNieghbors classifier

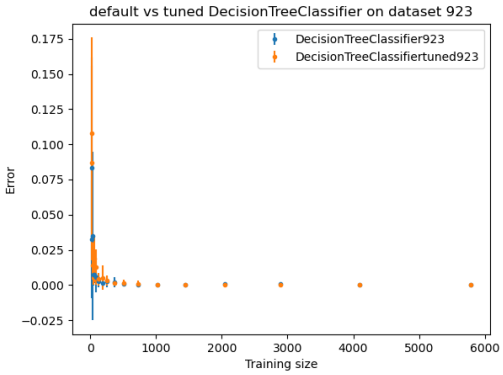

Figure 32: Learning curves of tuned and default Decision classifier with standard error

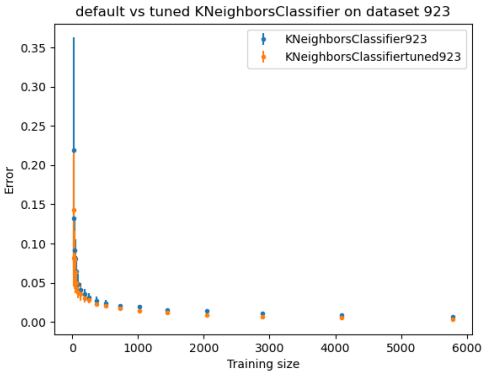

Figure 33: performance difference of tuned and default KNieghbors classifier

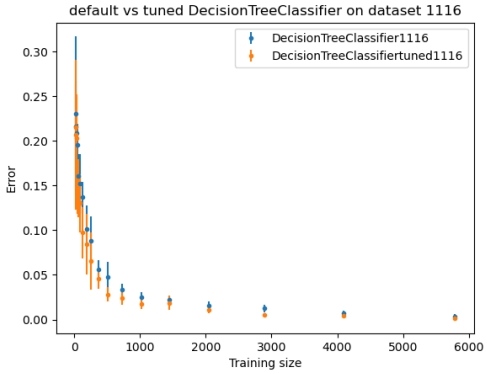

Figure 34: Learning curves of tuned and default Decision classifier with standard error

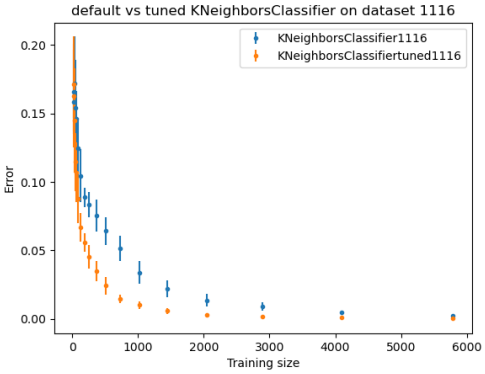

Figure 35: performance difference of tuned and default KNieghbors classifier

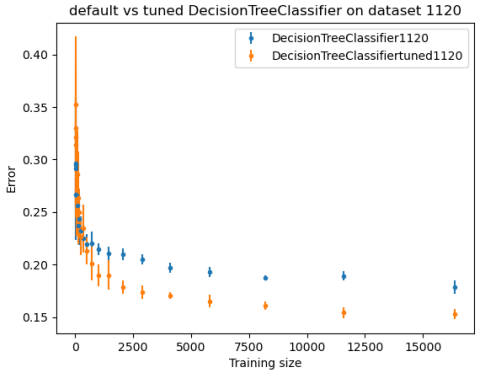

Figure 36: Learning curves of tuned and default Decision classifier with standard error

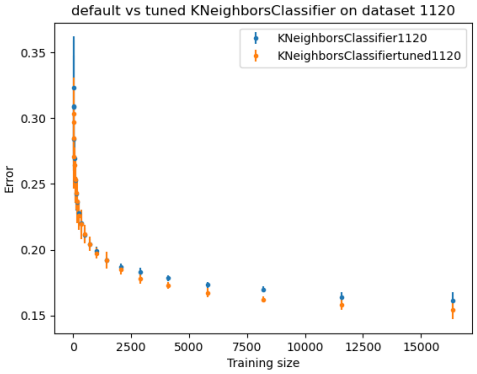

Figure 37: performance difference of tuned and default KNieghbors classifier

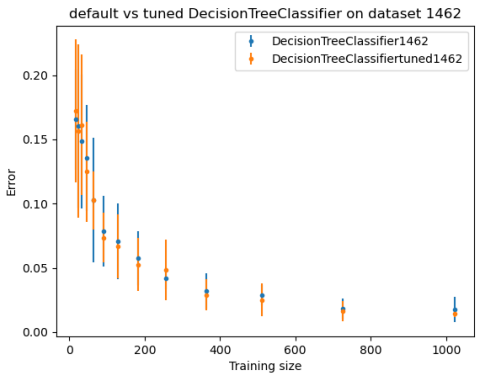

Figure 38: Learning curves of tuned and default Decision classifier with standard error

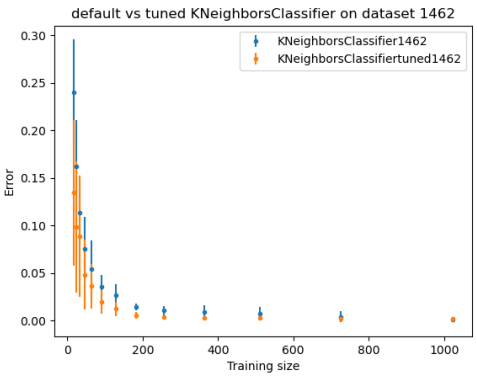

Figure 39: performance difference of tuned and default KNieghbors classifier

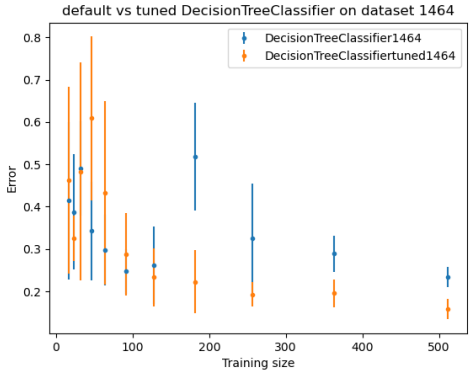

Figure 40: Learning curves of tuned and default Decision classifier with standard error

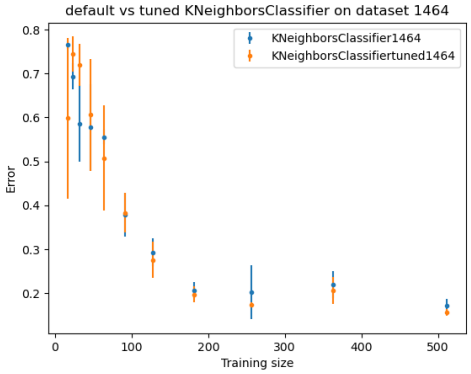

Figure 41: performance difference of tuned and default KNieghbors classifier

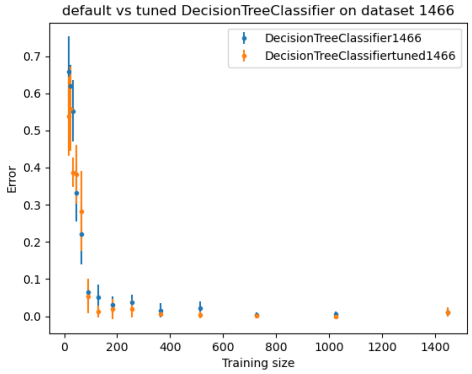

Figure 42: Learning curves of tuned and default Decision classifier with standard error

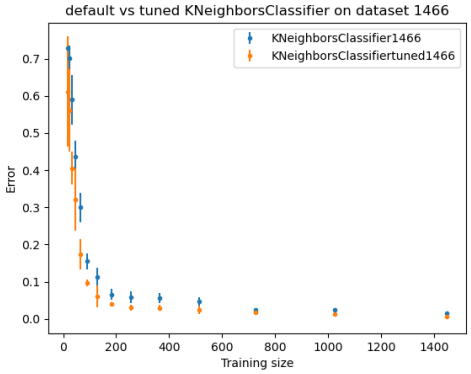

Figure 43: performance difference of tuned and default KNieghbors classifier

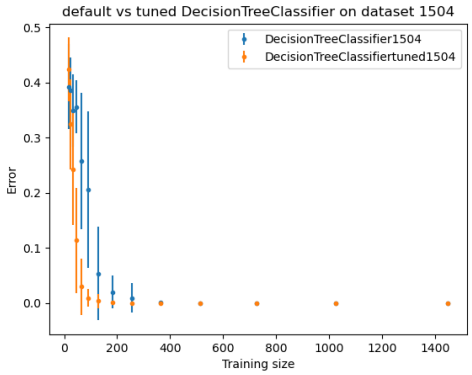

Figure 44: Learning curves of tuned and default Decision classifier with standard error

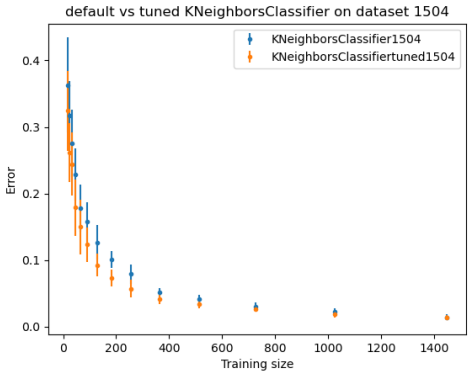

Figure 45: performance difference of tuned and default KNieghbors classifier

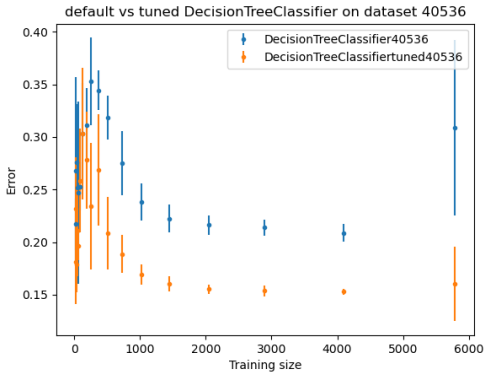

Figure 46: Learning curves of tuned and default Decision classifier with standard error

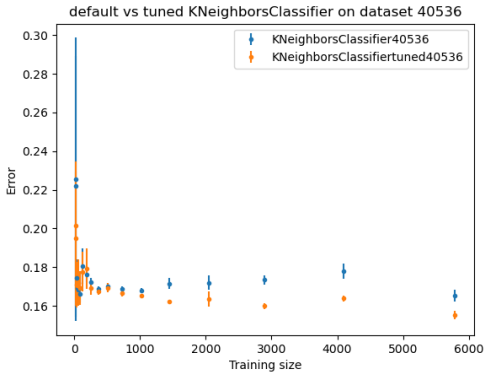

Figure 47: performance difference of tuned and default KNieghbors classifier

C QQ Plot

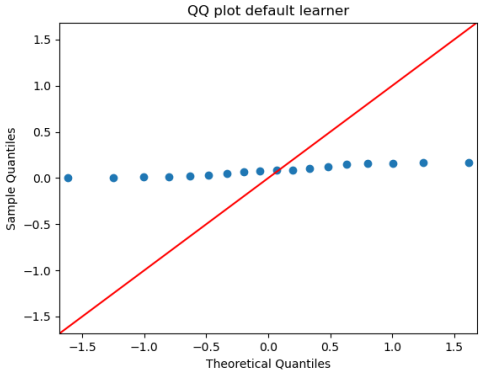

Figure 48: Learning curves of tuned and default Decision classifier with standard error

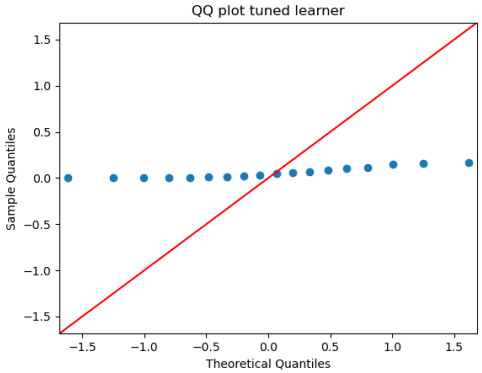

Figure 49: performance difference of tuned and default KNieghbors classifier