Different approaches to fitting and extrapolating the learning curve

Donghwi Kim
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
June 23, 2022

Abstract

Extrapolation of the learning curve provides an estimation of how much data is needed to achieve the desired performance. It can be beneficial when gathering data is complex, or computation resource is limited. One of the essential processes of learning curve extrapolation is curve fitting. This research first analyses the behaviour of existing curve fitting methods such as Newton, Levenberg-Marquardt and Evolutionary algorithms when fitting different function models on learning curves. Furthermore, it also illustrates a few techniques to improve the learning curve fitting and extrapolation procedure.

1 Introduction

Predicting how much data is needed to achieve the desired accuracy in machine learning can be helpful when computation resources are limited or data collection is difficult or expensive [14]. For example, when doing the hyper-parameter tuning, with the assistance of the learning curve extrapolation, turning that is predicted to be poor can be disregarded in the early stages and save the computation resource [5]. The learning curve plots the performance of the training set corresponding to the sample sizes. The extrapolation of the learning curve provides an estimation of the number of samples needed to reach expected performance [4]. In order to make the most of learning curve extrapolation, the method of curve fitting and extrapolation should be considered precisely.

Until now, several research studies have been done on curve fitting and learning curve extrapolation. Curve fitting is finding the best parameter values for some mathematical model function that best fits some data point series. So far, several studies have aimed to find the most suitable algorithm for curve fitting. Some researchers have evaluated different curve-fitting algorithms such as Newton, Gauss-Newton and Levenberg-Marquardt methods and concluded that Levenberg-Marquardt performs well in general cases regarding computation time and accuracy [9]. Also, some researchers have applied the genetic algorithm to fit curves [8]. It turned out that the genetic algorithm often performs better than Levenberg-Marquardt as it initiates with a population containing multiple candidate solutions making it less dependent on the quality of the initial parameter values than Levenberg-Marquardt, which starts from one initial parameter values [16]. Most research on learning curves used the Levenberg-Marquardt method to fit and extrapolate learning curves to perform their experiment [14; 3; 1]. Furthermore, some research has reduced the search space by limiting the bounds of parameter values by applying the idea that the error rate of the learning curve is between 0 and 1, and the error rate of a typical well-behaving learning curve decreases for larger training size, making the curve fitting and extrapolation process more efficient and accurate [1].

Even though several different curve-fitting algorithms exist, most studies on learning curves have experimented with only using the Levenberg-Marquardt method. Also, research on applying the genetic algorithm to curve fitting, in general, discusses only the accuracy, and it may be questionable whether it is still good regarding computational resources. The wrong selection of the initial parameter values of the model function or curve fitting algorithm may lead to an inaccurate curve fitting result [6]. Nevertheless, a study on a different curve-fitting method for fitting learning curves has not been done in-depth. Another concern on learning curve fitting is that extrapolation performance could be terrible even if the model function has fitted well on the given empirical learning curve with a limited training size [3]. Bad extrapolation result, for example, when the estimated error rate is not in the range between 0 and 1 while the error rate of a learning curve can never reach the value under 0 or above 1, does not help predict the data size needed to achieve specific accuracy.

This research will analyse the pros and cons of the different existing curve fitting algorithms for fitting the learning curve. It will also explore some ways of configuring the initial parameter values of the model function. Furthermore, it will examine some approaches to avoid terrible extrapolation results by modifying the objective function used in the curve fitting process.

The organisation of the sections is as follows. Section 2 illustrates the definition and details of each method experimented with in this research. After that, section 3 presents a few techniques to improve the learning curve-fitting procedure. Section 4 contains the setup configuration of curve fitting methods and information on the datasets used in the experiment. Section 5 presents experiment results, and section 6 discusses impactful or doubtful results. Section 7 summarises the research and discusses the future recommendations. Finally, section 8 discusses the reproducibility of the experiments.

2 Methodology

This research analyses the performance of the following different methodologies for the fitting and extrapolating learning curve.

2.1 Learning curve extrapolation

The learning curve extrapolation estimates how much more data is needed to achieve the desired accuracy by fitting some model function to the empirical learning curve generated with limited data size for training. The evaluation process of extrapolation is as follows. When given empirical learning curves generated from sufficient data size, the performance of extrapolation is measured using the hold-out method by dividing empirical learning curve into training anchors for fitting model functions and test anchors for evaluating extrapolation performance [3]. The figure 1 shows an example of the division of empirical learning into training anchor and test anchor, and extrapolation of the learning curve. After dividing

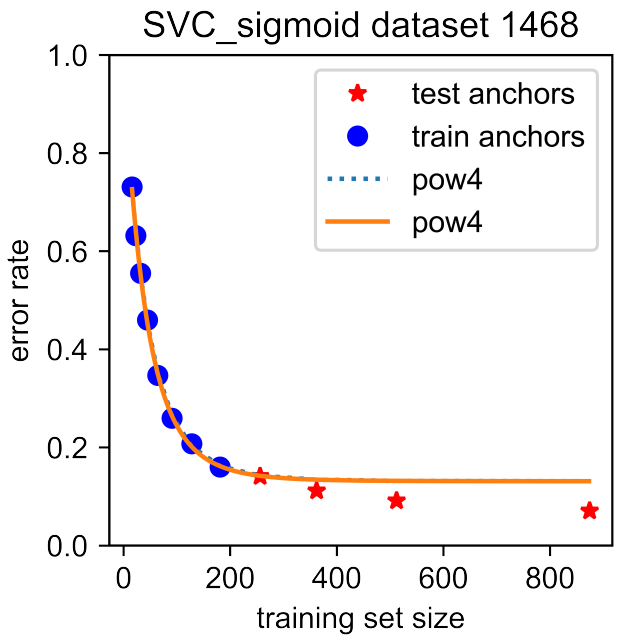


Figure 1: This figure shows an example of the division of the empirical learning curve into training anchor and test anchor. It is also possible to find extrapolation results on the test anchor generated by fitting the $pow4$ model function on the training anchor.

the empirical learning curve, the next task is to fit the model function on the empirical learning curve on the training anchor. In general, the main goal of the fitting curve is finding the best parameter values for a model function that minimises the mean squared error between the empirical learning curve and the model function on the training anchor. It is possible to achieve this goal with some curve fitting method described in section 2.2, which aims to find the parameter values for the model function that minimise the objective function (1). With optimal parameter values found for the model function through the curve fitting process on the test anchor, it is possible to extrapolate the learning curve on the test anchor. Finally, it is feasible to evaluate extrapolation performance by measuring the error between the extrapolated learning curve and the empirical learning curve on the test anchor.

2.2 Curve fitting methods

As mentioned in the previous subsection, the main goal of the fitting curve is finding the best parameter values for a model function that minimises the mean squared error between the empirical learning curve on the training anchor. The objective function that the curve fitting method aims find parameter values that minimise it is as follows:

$$f(\vec{x}) = \sum_{i=0}^{n} (e_i - m_i(\vec{x}))^2 \quad (1)$$

where $\vec{x}$ is the vector containing parameter values, 0 to n is the index of training sizes on the training anchor, e_i is the error rate from the empirical learning curve on ith training size, and $m_i(\vec{x})$ is the error rate estimated from the model function using parameter values $\vec{x}$ on ith training size. The example of model function can be found on table 1. Two different types of curve fitting methods are considered: the local optimisation method and the global optimisation method. The local optimisation method, such as Newton and Levenberg-Marquardt, starts from one initial point representing the potential best parameter values of the model function and finds the best solution in a limited search space near the initial point. On the other hand, for global optimisation algorithms, such evolutionary algorithms initiate with a population containing multiple candidate solutions that evolve throughout the iterative process. Unlike local optimisation algorithms, it aims to find the best solution in the entire search space. The following paragraphs will explain some curve-fitting algorithms in detail.

Newton method

The Newton method aims to find the parameter values, $\vec{x}$ such that $\nabla f(\vec{x}) = 0$ by iteratively updating $\vec{x}$ using the gradient and the hessian matrix of objective function, $f(\vec{x})$. Note that $f(\vec{x})$ has a local minimum at $\vec{x}$ where $\nabla f(\vec{x}) = 0$. The following equation illustrates the procedure of updating $\vec{x}$ in iterations.

$$\vec{x}_{k+1} = \vec{x}_k - \nabla^2 f(\vec{x}_k)^{-1} \nabla f(\vec{x}_k)$$

where $\vec{x}_k$ is the parameter values, $\nabla f(\vec{x}_k)$ is the gradient and $\nabla^2 f(\vec{x}_k)$ is the hessian matrix of the objective function, $f(\vec{x}_k)$ at iteration k. [9].

Levenberg-Marquardt method

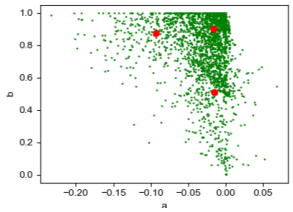
The Levenberg–Marquardt method (LM) is a combination of two optimisation methods: gradient descent and Gauss-Newton [6]. When parameter values are far from the optimal value, LM behaves like the gradient descent method, which finds optimal parameter values by updating parameter values in the steepest-descent direction [6; 10]. When parameter values are close to the optimal values, LM acts like Gauss-Newton. The Gauss-Newton method minimises the sum of the squared errors by assuming the objective function locally quadratic and finds the minimum of quadratic [9].

Differential evolution method

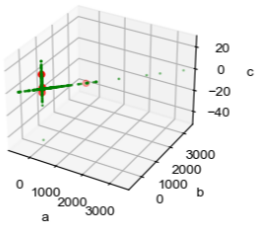
Differential evolution algorithm (DE) starts with the initialisation of population, which contains several candidate solutions. In each iteration, the population evolves through the crossover process on selected well-performing solutions from the population and mutation that keep solutions in the population diverse [12]. The algorithm terminates when it has found the optimal solution or reached the maximum iteration set beforehand. After termination of the algorithm, the local optimisation algorithm can be applied to the best solution in the population to enhance the performance slightly. It is called the polishing method.

3 Proposed further method

This research experimented with two techniques to further improve the learning curve-fitting procedure. The following subsections describe the details of each technique.



(a) Distribution of parameter values of $log2$



(b) Distribution of parameter values of $pow3$

Figure 2: The figures shows distribution of parameter values of the model function $log2$ and $pow3$ defined on section 4. The green scattered plot illustrates the distribution of the optimal parameter values for model functions of many different empirical learning curves, while the red plot represents the K-means centroids of the distribution. Note that datasets and learners from appendix A were used to generate empirical learning curves for this experiment.

3.1 Deriving initial parameter values with assist of k-means clustering algorithm

For local optimisation algorithms, which initiate with one initial point, a bad choice of initial parameters value may lead to poor curve fitting results [16]. Limiting the bounds of random initial parameter values can be one way to prevent poor curve fitting results [1]. However, the search space is still huge when there are many parameters in the model function, or the possible bounds of each parameter values are significant. Nevertheless, figure 2 shows that even though each parameter value lies on a large bound, the distribution of probable combinations of parameter values is dense. One way of finding the best initial parameter values using the distribution of parameter values of quantity of empirical learning curves is as follows, which is the method of initialising parameter values with assist of K-means clustering algorithm (KMI).

For KMI, the learning curve database is divided into two parts: one group of the dataset for determining the initial points and another for evaluating the curve fitting and extrapolation performance. The simplified process for each group of the datasets is shown in figure 3. First, for the group of datasets for determining the initial points, the model function is fitted on the entire empirical learning curves with the curve-fitting algorithm. Next, the K-means clustering algorithm derives centroids of optimal parameter values found from the previous step that fit the model function well on the empirical learning curve. The example of calculated centroids of parameter values from many empirical learning curves is shown in figure 2. For the datasets for evaluating the curve fitting and extrapolation performance, procedures remain the same as in section 2, except for using K-means centroids calculated in the previous step as the initial parameter values instead of random values.

3.2 Modification of objective function to avoid bad extrapolation

The problem of the fitting learning curve has characteristics that a typical curve-fitting problem does not possess. Unlike

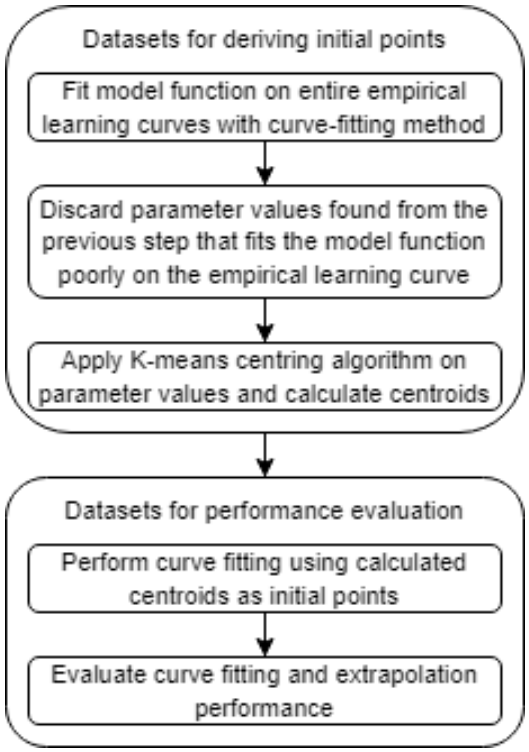


Figure 3: Process of driving initial parameter values and performance evaluation of KMI.

the general curve-fitting problem, the goal of the fitting learning curve is to extrapolate the learning curve on the test anchor to predict how much more data is needed to reach specific accuracy. Thus, if the extrapolation result is terrible, for example, when the predicted error rate is not between 0 and 1, it may be useless. Few different methods exist to avoid poor extrapolation results on the test anchor. One way could be running a curve-fitting algorithm repeatedly with a different initial point until it returns the values of parameters that have good extrapolation results on test anchors [3]. A further approach to this method can be limiting bounds to initial parameter values to avoid using initial parameter values that potentially result in bad extrapolation when using random initial parameter values [1]. The following paragraph illustrates the method to avoid bad extrapolation results throughout curve fitting algorithm execution via modification of the object function.

The following modification of the objective function is considered in the experiment:

$$f(\vec{x}) = \sum_{i=0}^{n} (e_i - m_i(\vec{x}))^2 + \sum_{i=n+1}^{k} ((\max(1, m_i(\vec{x})) - 1) + \min(0, m_i(\vec{x}))^2 \quad (2)$$

$$f(\vec{x}) = \sum_{i=0}^{n} (e_i - m_i(\vec{x}))^2 + \sum_{i=n+1}^{k} ((\max(e_n, m_i(\vec{x})) - e_n) + \min(0, m_i(\vec{x})))^2 \quad (3)$$

where $\vec{x}$ is the vector containing parameter values, 0 to n is the index of training sizes on the training anchor, $n+1$ to k is the index of training sizes on the test anchor, e_i is the error rate from the empirical learning curve on ith training size, and $m_i(\vec{x})$ is the error rate estimated from the model function using parameter values $\vec{x}$ on ith training size. The following paragraph explains each objective function in detail

The objective function of standard learning curve fitting considers how well the model function is fitted on the empirical learning curve on the training anchor, as shown in equation (1). The modified objective considers the performance of curve-fitting on the training anchor as well as examines how extrapolation behaves on the test anchor. The error range of the learning curve is always between 0 and 1. The first modification, shown in equation (2), adds an additional penalty to the objective function when the error rate of extrapolated learning curve goes below 0 or above 1 for test anchor size. Another characteristic of the most learning curve, the well-behaving learning curves, is that the error rate decreases for the greater training size. The second modification, shown in equation (3), adds an additional penalty to the objective function when the error rate estimated from extrapolation on the test anchor is larger than the error rate of the last training anchor from empirical learning curve.

4 Experimental Setup

The following subsections illustrate several important setup environments for the algorithm or method used in the experiment.

Data Collection and empirical curve generation

The datasets from OpenML [2] are used to create the empirical learning curves. Generation of the empirical learning curve for evaluation of the experiment result is done with five datasets using twelve different learners. The details of the datasets and learners used for the experiment can be found in appendix B. The information of datasets and leaners used to generate empirical learning curves for KMI to derive the initial parameter values is shown in appendix A. The empirical learning curve was generated via LCDB [3] using the stratified fold method.

Model function

This research examines the following model functions shown in the table 1 with parameters given bounds.

Algorithm implementation

The experiment on the different algorithms, Newton, Levenberg-Marquardt and differential evolution, was done using the SciPy optimisation library [15]. It is possible to find the important setting of the environment of the algorithm

		Bounds of the parameters							
Model	Function	a		b		c		d	
		min	max	min	max	min	max	min	max
$log2$	$-a\log x + b$	-1	0	0	1	-	-	-	-
$pow2$	$-ax^{-b}$	-1	0	-1	0	-	-	-	-
$exp3$	$a\exp -bx + c$	-10	100	-10	100	-100	10	-	-
$pow3$	$a - bx^{-c}$	0	150	0	150	0	10	-	-
$exp4$	$c - b\exp -ax^{d}$	-10	100	-10	100	-100	10	-5	15
$pow4$	$a - b*(x+d)^{-c}$	0	150	0	150	-5	50	-1	10

Table 1: The table illustrates model function and its bounds for the parameter. The bound first have been derived analytically and modified afterwards by examining shape of the actual existing empirical learning curves. Note that anchor sizes x has been scaled to be in range between 0 and 1.

in appendix C. Detailed description of each environment setting can be found on Scipy [15]. For KMI, the SciPy cluster library [15] was used to calculate the centroids of optimal parameter values of the different model functions for the number of empirical learning curves.

No further modification has been applied to the Newton and Levenberg-Marquardt method shown in section 2.2. For LM and Newton methods, two approaches have been used for initialising parameter values: random parameter values given bound defined in table 1 and parameter values derived by KMI explained in section 3.1. Furthermore, the hybrid approach of using two curve fitting methods, which is applying the Levenberg-Marquardt method to the one best result found by the Newton method, has also been experimented. For this approach, the optimal parameter values are computed with the Newton method with the trial on several initial points. After that, the LM is executed using the one best parameter values found by the Newton method as an initial point.

The DE has experimented on the different numbers of the initial population. The initial population was derived from the KMI. For the crossover and mutation methods, no modification was applied. It is possible to find further information on mutation and crossover methods on [12]. The Levenberg-Marquardt method was used as a polishing method, which is applied to the best solution of the population from last iteration to improve the performance slightly.

5 Results

This section illustrates the experiment result of the methodologies described in the previous sections.

Figure 4 illustrates how the method of initialising the parameter values affects the performance of the curve fitting for LM. Note that the performance of the curve fitting in this context refers to how the curve fitting of the model function is done successfully on the empirical learning curve on the training anchor. First, for the simple function model, such as $log2$ and $pow2$ with two parameters with small bounds, it was possible to reach its maximum curve fitting performance with trials on very few initial points. Also, the curve fitting performance was equivalent for both methods

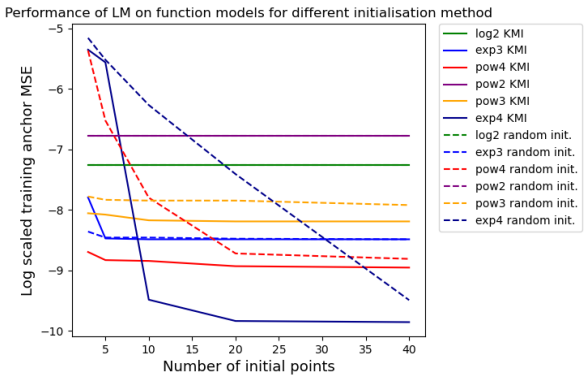


Figure 4: This figure compares the curve fitting performance of the different methods of initialising parameter values for different function models and the number of initial points used.

of initialising the parameter values, which are the random initialisation method and KMI. In other words, the quality of the initial parameter values chosen did not affect curve fitting performance for simple model functions. However, for the complicated models functions, such as $pow4$ and $exp4$ with four parameters with large bounds, trials on a sufficient amount of initial points were required to achieve its maximum performance with randomly initialised parameter values. It was possible to achieve maximum curve fitting performance of complicated function models with the trials on fewer initial points with KMI compared to using randomly initialised parameter values. Still, it was noticeable in figure 4 that even with the KMI, the complicated function model still needs more than ten initial points to reach its maximum curve fitting performance. In comparison, the simple function model achieved its maximum curve fitting performance with less than five initial points. Nevertheless, the figure 4 shows that when experimented on a sufficient number of the initial point, the more parameter the model function had, the better it fitted on the empirical learning curve on the training anchor. In other words, when a sufficient number of initial points are used, the complicated function model, such as $pow4$, fits better on the empirical learning curve on the training anchor than the simple function model, such as $log2$ and $pow2$.

Figure 5 shows a comparison regarding curve fitting performance and computation time on the Newton method and LM using different methods of initialising parameter values. As also explained in the previous paragraph, for LM, the method of initialising parameter values had an impact on curve fitting performance for only complicated function models, such as $pow4$. On the other hand, for the Newton method, the method of initialising parameter values impacted all model functions, and the curve fitting performance gap between KMI and random initialisation of parameter values was more significant than that of LM. It means that the performance of the Newton method was more dependent on the quality of initial parameter values when compared to the LM. Comparing the Newton method and LM shows that LM performs better on curve fitting regardless of model function and method of initialising parameter values. Considering computation time, complicated models, such as $pow4$, the Newton method finds its solution significantly faster than the LM. However, for the simple model function, $log2$, the computation time of the LM and Newton methods did not differ significantly. Furthermore, figure 5 also shows that for complicated model functions, applying the LM to the one best parameter values found by the Newton method using KMI performed almost good as only using LM while having considerably faster computation time.

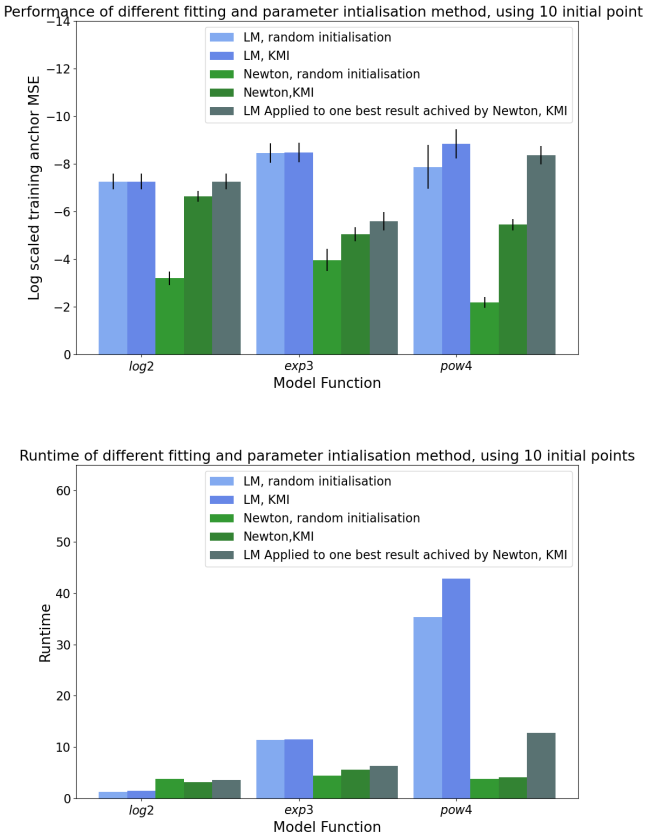
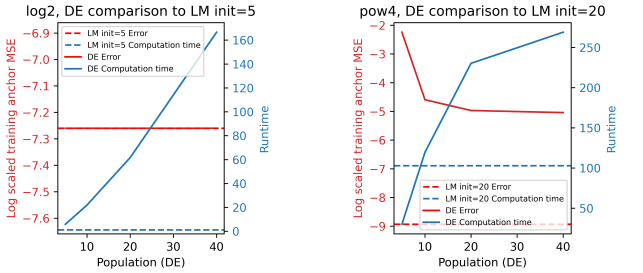


Figure 5: The first figure compares the curve fitting performance of different curve fitting algorithms and the method of initialising the parameter values. To measure the curve fitting performance, MSE error between empirical learning curve and extrapolated learning curve on training anchor is used. The second figure compares the computation time of different curve fitting algorithms and the method of initialising the parameter values. Note that 10 initial points were used for these experiment.

Figure 6 illustrates the performance of curve fitting and computation time of DE on different population sizes and compares DE to LM. For the simple function model, $log2$, with minimal initial population size, DE reached its maximum curve fitting performance equivalent to that achieved with LM. Figure 6a shows that from population size less



(a) $log2$ comparison of DE to LM with 5 initial points

(b) $pow4$ comparison of DE to LM with 20 initial points

Figure 6: The figure compares DE using different sizes of the initial population to the LM. The dashed lines illustrate the performance of LM regarding computation time and curve-fitting performance, while the solid line represents the performance of the differential evolution algorithm. Note that for both DE and Lm, KMI was used for the initial population and points, respectively.

than five, the performance of DE converged to performance equivalent to LM using five initial points. However, DE with population size five took a slightly longer computation time than LM with five initial points. On the other hand, for the complicated function model, $pow4$, larger population size for DE improved curve fitting performance. However, the increase in the size of population also leads to an increase in computation time. Figure 6b shows that even with a population size of 40, the curve fitting performance of DE was still below the performance of LM with only 20 initial points, and LM took much less computation time to achieve this result.

Figure 7 shows the extrapolation performance for different model functions using different buckets for training anchors for different curve fitting algorithms, methods of initialising parameter values and different objective functions. Note that the extrapolation performance in this context refers to how well the extrapolated learning curve predicts how much accuracy it can achieve with training sizes on the test anchor. For the objective function (1) that only considers curve fitting performance on training anchor, when comparing methods of initialising parameter values, KMI had equal or better extrapolation performance than randomly initialising the parameter values. Figure 7 shows that for object function (1), both the Newton method and LM, respectively, showed equal or better extrapolation performance when using KMI than randomly initialising the parameter values for all bucket sizes for training anchors for all model functions. It was also notable that the extrapolation performance gap between randomly initialising the parameter values and KMI was comparably larger for Newton than for LM. When comparing the Newton method and LM, the LM generally showed better performance. Figure 7 illustrates that except for the $log2$ model using the small bucket, using objective functions (1) and KMI, LM always showed better performance on extrapolation than the Newton method.

The detailed description of the objective functions (1), (2) and (3) are described in section 3.2. Compared to the object function (1), the (2) performed better on the extrapolation. Figure 7 shows that for both the Newton method and LM using KMI, the objective function (2) performed better on extrapolation than (1) on all the buckets sizes for training anchor for both $log2$ and $pow4$ model functions. When the objective function (3) is compared to the (2) for LM, there was very slight or no increase in the extrapolation performance. For the Newton method, a extrapolation performance comparison between objective functions (2) and (3) was inconsistent for different the buckets used for training anchor and model functions. Figure 7 shows that for the Newton method, objective function (3) performs better than (2) for some model functions for some bucket size used for training anchor and sometimes other way around.

Objective functions

(1)	Considers curve fitting performance of model function on empirical learning curve on training anchor
(2)	Addition to (1), adds extra penalty when error rate of extrapolated learning curve on test anchor is not in range between 0 and 1
(2)	Addition to (2), adds extra penalty when error rate estimated by extrapolated learning curve on test anchor goes above the error rate of last training anchor from empirical learning curve

Extrapolation performance

Init. method		Random		KMI					
Objective function		(1)				(2)		(3)	
Fitting method		Newton	LM	Newton	LM	Newton	LM	Newton	LM
Bucket	Model								
0.05 ~0.1	log2	0.18	0.22	0.32	0.22	0.63	0.69	0.48	0.69
	pow4	0.08	0.67	0.66	0.68	0.66	0.82	0.79	0.82
0.1~0.2	log2	0.2	0.33	0.38	0.33	0.63	0.76	0.55	0.76
	pow4	0.08	0.77	0.72	0.8	0.72	0.92	0.78	0.93
0.2~0.4	log2	0.29	0.43	0.43	0.43	0.65	0.8	0.68	0.81
	pow4	0.12	0.87	0.81	0.89	0.81	0.91	0.86	0.94
0.4~0.8	log2	0.09	0.68	0.54	0.68	0.58	0.89	0.59	0.9
	pow4	0.17	0.94	0.9	0.94	0.9	0.94	0.92	0.95

Figure 7: The first table briefly describes each objective function considered in the experiment, and it is possible to find the detailed information in section 3.2. The second table shows the percentile of good extrapolation of total extrapolation result. The good extrapolation result here refers to the case where the difference between the error rate of the last test anchor predicted by learning curve extrapolation and the empirical learning curve is less than 0.1. This experiment considers the different methods of initialising the parameter values, objective function and curve fitting for different bucket sizes of training anchor for $log2$ and $pow4$ model functions. Bucket size 0.1$\sim$0.2 means that the first 10$\sim$20% of the entire empirical learning curve is used as a training anchor and the rest as a test anchor. Note that 15 initial points were used for this experiment.

6 Discussion

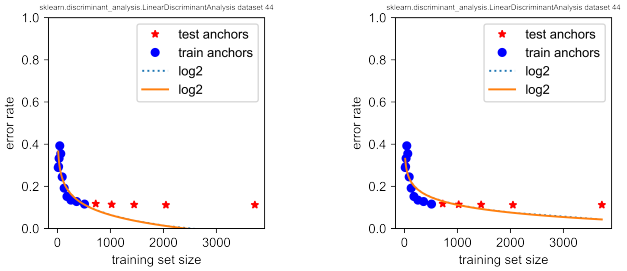
This section discusses some doubtful or interesting result found from the previous section.

Regarding initialisation of the parameter values, the result showed that KMI performed better on curve fitting than random initialisation of the parameter values, especially for the complicated function models. It is shown in figure 4. The advantage of the KMI illustrated in section 3.1 can be that

whenever it is possible to retrieve the number of datasets for deriving the initial parameter values, the KMI method can be easily applied to any function model disregarding how many parameters it has. Thus, KMI can be a suitable method for finding good initial parameters for model functions with too many parameters or possible bounds of each parameter are too large. Nevertheless, the setup cost for KMI, such as preparing the number of datasets for deriving the initial parameter values and computation time for retrieving the centroid through the K-means clustering algorithm, should be considered when choosing this method. For example, for simple models, $log2$, which showed good curve fitting performance on any method of initialising parameter values with the trial on a minimal number of initial points, KMI might not be the most efficient way.

The LM showed better curve fitting performance than the Newton method. This is shown in figure 5. Specifically for simple function models, its computation time was very slightly faster than the Newton method. On the other hand, for the complicated function models, LM performed better on curve fitting than the Newton method, but it took comparably more computation time. One notable result regarding initial parameter values was that when KMI was used instead of randomly initialising the parameter values, the increase in curve fitting performance of the Newton method was relatively higher than that of LM. The reason the curve fitting performance LM got less affected by the quality of the initial parameter could be that, as mentioned in section 2.2, LM uses the gradient descent method when the parameter values are far away from the optimal solution. Note that unlike the Newton method, which uses a gradient and the Hessian matrix of the objective function, the gradient descent method only uses a gradient to update parameter values in every iteration, making it slower but more accurate when the objective function is complicated [10]. Another noticeable result was that for complicated function models when using KMI, applying LM to only one best solution found by the Newton method showed curve fitting performance almost close to running LM on all initial points. But this approach took less computation time than running LM on all initial points. This approach might be a good alternative to using only LM for complicated function models when given limited computational resources, but curve fitting performance is also essential.

The result found that the DE did not perform better than LM and took more computation time. This is shown in the figure 6. Specifically, simple model functions only took a slightly longer computation time while showing the equivalent curve fitting performance to the LM. However, for the complicated model functions, even with large population size and spending longer computation time, DE could not reach the curve fitting performance of LM. One reason DE did not perform well could be that the crossover and mutation methods used in this experiment might not have been applicable for fitting curves. Improving crossover and mutation methods may improve the curve fitting performance of DE. Another reason DE did not show good curve fitting performance compared to LM could be too large possible bounds of each parameter. Figure 2 illustrates that even if the possible bounds of each parameter are big, the probable combination of each parameter may be limited. The characteristic of DE illustrated on 2.2 that explores the whole search space might not have been the most efficient way to find the solution in this case.



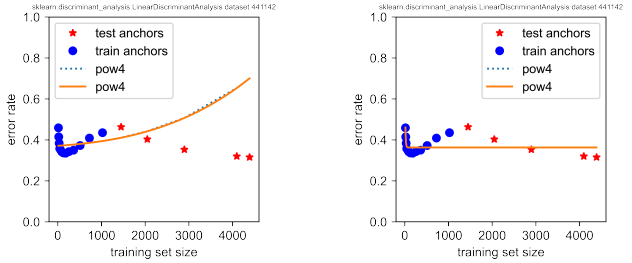
(a) LM using KMI with objective function (1)

(b) Newton method using KMI with objective function (1)

Figure 8: It is a sample of extrapolation results that compares LM and Newton methods using the objective function (1) and KMI for $log2$ with a small training anchor bucket size from the experiment illustrated in figure 7.

Regarding the method of initialising the parameter values, KMI resulted in having better performance on both curve fitting and extrapolation than randomly initialising the parameter values. Considering the curve fitting method, the LM always performed better than the Newton method on the curve fitting. Also, LM showed better performance on extrapolation in most of the cases. However, the Newton method showed better extrapolation than the LM for the $log2$ function model using the small bucket for training anchor with KMI and objective function (1). It is shown in figure 7. One cause may be that $log2$ was not a suitable function model for fitting some learning curves, having bad extrapolation even with the successful curve fitting on the training anchor. From figure 7, it was also notable that for LM, compared to the $pow4$ model function, the extrapolation performance of the $log2$ was always worse. For example, figure 8 shows that even though LM fitted the $log2$ model function better on empirical learning curve on the training anchor than the Newton method, the newton method performed better on the extrapolation. From this, it can be concluded that, in general, the better curve fitting performance results in better extrapolation results when an appropriate model function is chosen.

Modifying the objective function improved the extrapolation performance to some extent, which is shown in figure 7. Specifically, the objective function (2) that limits the bound of the error rate to be between 0 and 1 performed better on extrapolation when compared to the objective function (1). It is because, for all the learning curves, the error rate is always between 0 and 1. The objective function (3) explained in



(a) LM using KMI with objective function (2)

(b) LM using KMI with objective function (3)

Figure 9: It is a sample of extrapolation results that compares objective function 2 and 3 using LM from the experiment illustrated in figure 7.

section 3.2 assumes the learning curve typically behaves well, meaning that the error rate decreases for the larger training size. The result showed that the objective function (3) slightly performed better than the objective function (2) for LM. The factor that led objective function (3) to perform slightly better than (2) may be the characteristic of the datasets and learners that were used to generate the empirical learning curves for the experiment. For example, in figure 9, one sample from the experiment shows that objective function (3) showed better extrapolation than (2) when the empirical learning curve behaved weirdly on the training anchor but well behaved on the test anchor. However, suppose many empirical learning curves used in the experiment did not behave well on both training and test anchor. In that case, the result of comparing object functions (2) and (3) may have differed.

7 Conclusions and Future Work

It has shown that the simpler the function model is easier to fit on the empirical learning curve, taking less computation time, but the worse the extrapolation result is. In order to improve the curve fitting performance for the complicated function models that shows good extrapolation performance, this research has experimented with different approaches, and one founding was using KMI instead of randomly initialising the parameter values may be a solution to improve curve fitting performance. Nevertheless, instead of finding ways to fit complicated function models, solving the problem of finding the simple function model with good extrapolation performance could be an alternative way to achieve better extrapolation results in a short computation time.

This research has shown some modifications to the objective function used in the curve fitting process, which may enhance the extrapolation performance when used appropriately. The further approach can be modifying the objective function for each learning curve based on the characteristic of the dataset and labels, chosen learner or hyperparameter of the learner, which may affect the behaviour of the learning curve.

8 Responsible Research

This paragraph will briefly discuss the reproducibility of the experiment done in this research considering the recommendations of the Yale Law School Roundtable [11]. The first recommendation is to provide a link to the source and data used to generate the results [11]. For this research, the source code is accessible on the https://github.com/donghwikim7/lcf and datasets used to generate results are listed in the appendix. The second recommendation is to assign a unique id to each version of the released code [11]. However, there is no planned update on the implementation, and thus this recommendation is not considered. The third recommendation is to describe the computing environment and software version used in the publication [11]. The experiment has done on Ryzen 3600 with six core and twelve threads on Windows 11. It is possible to find the version of the software libraries used in the experiment on https://github.com/donghwikim7/lcf/blob/main/lcf.yaml. The fourth recommendation is to use open licensing for the code for reusability [11]. The implementation is MIT licenced so that anyone can reuse it. The fifth recommendation was to use an open-access contract for published papers [11]. This research will be available on the TU Delft repository so the public can access it. The sixth recommendation is to consider the readability and usability in the future [11]. The implementation has been done on Python [13] using Numpy [7] and SciPY [15] library, which will be readable in the future when possible.

References

[1] Best-fit learning curve model for the c4.5 algorithm. *Informatica*, 25(3):385–399, 2014.

[2] Giuseppe Casalicchio, Jakob Bossek, Michel Lang, Dominik Kirchhoff, Pascal Kerschke, Benjamin Hofner, Heidi Seibold, Joaquin Vanschoren, and Bernd Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, pages 1–15, 2017.

[3] Marco Loog Jan Van Rijn Felix Mohr, Tom Viering. Lcdb 1.0: An extensive learning curve ndatabase for classification tasks. "under review", .

[4] Lewis J. Frey and Douglas H. Fisher. Modeling decision tree performance with the power law. In David Heckerman and Joe Whittaker, editors, *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*, volume R2 of *Proceedings of Machine Learning Research*. PMLR, 03–06 Jan 1999. Reissued by PMLR on 20 August 2020.

[5] Matilde Gargiani, Aaron Klein, Stefan Falkner, and Frank Hutter. Probabilistic rollouts for learning curve extrapolation across hyperparameter settings, 2019.

[6] Henri P. Gavin. The levenberg-marquardt method for nonlinear least squares curve-fitting problems c ©. 2013.

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg,

Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[8] C.L. Karr, D.A. Stanley, B.J. Scheiner, and United States. Bureau of Mines. *Genetic Algorithm Applied to Least Squares Curve Fitting*. Number no. 9339 in Genetic Algorithm Applied to Least Squares Curve Fitting. U.S. Department of the Interior, Bureau of Mines, 1991.

[9] Farhad Morad. Non-linear curve fitting, 2019.

[10] Chafik Samir, P.-A Absil, Anuj Srivastava, and Eric Klassen. A gradient-descent method for curve fitting on riemannian manifolds. *Foundations of Computational Mathematics*, 12:49–73, 02 2012.

[11] V.C. Stodden. Reproducible research: Addressing the need for data and code sharing in computational science. *Computing in Science and Engineering*, 12:8–13, 01 2010.

[12] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 01 1997.

[13] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[14] Tom Viering and Marco Loog. The shape of learning curves: a review. *arXiv preprint arXiv:2103.10948*, 2021.

[15] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[16] Sinem ŞENTÜRK. Applied genetic algortihms approach to curve fitting problems, 2009.

Appendix A Datasets and learners used to generate empirical learning curve for deriving initial point for KMI

A.1 Learners

- SVC_linear
- SVC_poly
- SVC_rbf
- SVC_sigmoid
- sklearn.tree.DecisionTreeClassifier
- sklearn.tree.ExtraTreeClassifier
- sklearn.linear_model.LogisticRegression
- sklearn.linear_model.PassiveAggressiveClassifier
- sklearn.linear_model.Perceptron
- sklearn.linear_model.RidgeClassifier
- sklearn.linear_model.SGDClassifier
- sklearn.neural_network.MLPClassifier
- sklearn.discriminant_analysis.LinearDiscriminantAnalysis
- sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis
- sklearn.naive_bayes.BernoulliNB
- sklearn.naive_bayes.MultinomialNB
- sklearn.neighbors.KNeighborsClassifier
- sklearn.ensemble.ExtraTreesClassifier
- sklearn.ensemble.RandomForestClassifier
- sklearn.ensemble.GradientBoostingClassifier

A.2 Datasets

- 12 (mfeat-factors)
- 14 (mfeat-fourier)
- 16 (mfeat-karhunen)
- 18 (mfeat-morphological)
- 21 (car)
- 22 (mfeat-zernike)
- 23 (cmc)
- 24 (mushroom)
- 26 (nursery)
- 28 (optdigits)
- 30 (page-blocks)
- 31 (credit-g)
- 32 (pendigits)
- 36 (segment)
- 38 (sick)
- 46 (splice)
- 54 (vehicle)
- 57 (hypothyroid)
- 60 (waveform-5000)
- 179 (adult)
- 180 (covertype)
- 181 (yeast)
- 182 (satimage)

- 183 (abalone)
- 184 (kropt)
- 185 (baseball)
- 188 (eucalyptus)
- 273 (IMDB.drama)
- 293 (covertype)
- 300 (isolet)
- 351 (codrna)
- 354 (poker)
- 357 (vehicle_sensIT)
- 389 (fbis.wc)
- 390 (new3s.wc)
- 391 (re0.wc)
- 392 (oh0.wc)
- 393 (la2s.wc)
- 395 (re1.wc)
- 396 (la1s.wc)
- 398 (wap.wc)
- 399 (ohscal.wc)
- 401 (oh10.wc)
- 485 (analcatdata_vehicle)
- 554 (mnist_784)
- 679 (rmftsa_sleepdata)
- 715 (fri_c3_1000_25)
- 718 (fri_c4_1000_100)
- 720 (abalone)
- 722 (pol)
- 723 (fri_c4_1000_25)
- 727 (2dplanes)
- 728 (analcatdata_supreme)
- 734 (ailerons)
- 735 (cpu_small)
- 737 (space_ga)
- 740 (fri_c3_1000_10)
- 741 (rmftsa_sleepdata)
- 743 (fri_c1_1000_5)
- 751 (fri_c4_1000_10)
- 752 (puma32H)
- 761 (cpu_act)
- 772 (quake)
- 797 (fri_c4_1000_50)
- 799 (fri_c0_1000_5)
- 803 (delta_ailerons)
- 806 (fri_c3_1000_50)
- 807 (kin8nm)
- 813 (fri_c3_1000_5)
- 816 (puma8NH)
- 819 (delta_elevators)
- 821 (house_16H)
- 822 (cal_housing)
- 823 (houses)
- 833 (bank32nh)
- 837 (fri_c1_1000_50)
- 843 (house_8L)
- 845 (fri_c0_1000_10)
- 846 (elevators)
- 847 (wind)
- 849 (fri_c0_1000_25)
- 866 (fri_c2_1000_50)
- 871 (pollen)
- 881 (mv)
- 897 (colleges_aaup)
- 901 (fried)
- 903 (fri_c2_1000_25)
- 904 (fri_c0_1000_50)
- 910 (fri_c1_1000_10)
- 912 (fri_c2_1000_5)
- 913 (fri_c2_1000_10)
- 914 (balloon)
- 917 (fri_c1_1000_25)
- 923 (visualizing_soil)
- 930 (colleges_usnews)
- 934 (socmob)
- 953 (splice)
- 958 (segment)
- 959 (nursery)
- 962 (mfeat-morphological)
- 966 (analcatdata_halloffame)
- 971 (mfeat-fourier)
- 976 (JapaneseVowels)
- 977 (letter)
- 978 (mfeat-factors)
- 979 (waveform-5000)
- 980 (optdigits)
- 991 (car)
- 993 (kdd_ipums_la_97-small)
- 995 (mfeat-zernike)
- 1000 (hypothyroid)

- 1002 (ipums_la_98-small)
- 1018 (ipums_la_99-small)
- 1019 (pendigits)
- 1020 (mfeat-karhunen)
- 1021 (page-blocks)
- 1036 (sylva_agnostic)
- 1037 (ada_prior)
- 1039 (hiva_agnostic)
- 1040 (sylva_prior)
- 1041 (gina_prior2)
- 1042 (gina_prior)
- 1049 (pc4)
- 1050 (pc3)
- 1053 (jm1)
- 1059 (ar1)
- 1067 (kc1)
- 1068 (pc1)
- 1069 (pc2)
- 1111 (KDDCup09_appetency)
- 1116 (musk)
- 1119 (adult-census)
- 1120 (MagicTelescope)
- 1128 (OVA_Breast)
- 1130 (OVA_Lung)
- 1134 (OVA_Kidney)
- 1138 (OVA_Uterus)
- 1139 (OVA_Omentum)
- 1142 (OVA_Endometrium)
- 1146 (OVA_Prostate)
- 1161 (OVA_Colon)
- 1166 (OVA_Ovary)
- 1216 (Click_prediction_small)
- 1242 (vehicleNorm)
- 1457 (amazon-commerce-reviews)
- 1461 (bank-marketing)
- 1464 (blood-transfusion-service-center)
- 1475 (first-order-theorem-proving)
- 1485 (madelon)
- 1486 (nomao)
- 1487 (ozone-level-8hr)
- 1489 (phoneme)
- 1494 (qsar-biodeg)
- 1501 (semeion)
- 1515 (micro-mass)
- 1569 (poker-hand)
- 1590 (adult)
- 4134 (Bioresponse)
- 4135 (Amazon_employee_access)
- 4136 (Dexter)
- 4137 (Dorothea)
- 4534 (PhishingWebsites)
- 4538 (GesturePhaseSegmentationProcessed)
- 4541 (Diabetes130US)
- 4552 (BachChoralHarmony)
- 23380 (cjs)
- 23512 (higgs)
- 23517 (numerai28.6)
- 40497 (thyroid-ann)
- 40498 (wine-quality-white)
- 40668 (connect-4)
- 40670 (dna)
- 40685 (shuttle)
- 40691 (wine-quality-red)
- 40701 (churn)
- 40900 (Satellite)
- 40926 (CIFAR_10_small)
- 40971 (collins)
- 40975 (car)
- 40978 (Internet-Advertisements)
- 40981 (Australian)
- 40982 (steel-plates-fault)
- 40983 (wilt)
- 40984 (segment)
- 40996 (Fashion-MNIST)
- 41026 (gisette)
- 41027 (jungle_chess_2pcs_raw_endgame_complete)
- 41064 (convex)
- 41065 (mnist_rotation)
- 41066 (secom)
- 41138 (APSFailure)
- 41143 (jasmine)
- 41144 (madeline)
- 41145 (philippine)
- 41146 (sylvine)
- 41147 (albert)
- 41150 (MiniBooNE)
- 41156 (ada)
- 41157 (arcene)

- 41158 (gina)
- 41159 (guillermo)
- 41161 (riccardo)
- 41162 (kick)
- 41163 (dilbert)
- 41164 (fabert)
- 41165 (robert)
- 41166 (volkert)
- 41167 (dionis)
- 41168 (jannis)
- 41169 (helena)
- 41946 (Sick_numeric)
- 42732 (sf-police-incidents)
- 42733 (Click_prediction_small)
- 42734 (okcupid-stem)

Appendix B Datasets and learners used to generate empirical learning curve for experiment

B.1 Learners used for each dataset

- 3 (kr-vs-kp)
  - SVC_linear
  - SVC_poly
  - SVC_rbf
  - SVC_sigmoid
  - sklearn.ensemble.ExtraTreesClassifier
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.linear_model.LogisticRegression
  - sklearn.linear_model.PassiveAggressiveClassifier
  - sklearn.linear_model.Perceptron
  - sklearn.linear_model.RidgeClassifier
  - sklearn.linear_model.SGDClassifier
- 6 (letter)
  - SVC_linear
  - SVC_poly
  - SVC_rbf
  - SVC_sigmoid
  - sklearn.discriminant_analysis .LinearDiscriminantAnalysis
  - sklearn.discriminant_analysis .QuadraticDiscriminantAnalysis
  - sklearn.ensemble.ExtraTreesClassifier
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.linear_model.LogisticRegression
  - sklearn.linear_model.PassiveAggressiveClassifier
  - sklearn.linear_model.Perceptron
- 44 (spambase)
  - SVC_linear
  - SVC_poly
  - SVC_rbf
  - SVC_sigmoid
  - sklearn.discriminant_analysis .LinearDiscriminantAnalysis
  - sklearn.discriminant_analysis .QuadraticDiscriminantAnalysis
  - sklearn.ensemble.ExtraTreesClassifier
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.linear_model.LogisticRegression
  - sklearn.linear_model.PassiveAggressiveClassifier
  - sklearn.linear_model.Perceptron
- 1468 (cnae-9)
  - SVC_linear
  - SVC_poly
  - SVC_rbf
  - SVC_sigmoid
  - sklearn.discriminant_analysis .LinearDiscriminantAnalysis
  - sklearn.discriminant_analysis .QuadraticDiscriminantAnalysis
  - sklearn.ensemble.ExtraTreesClassifier
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.linear_model.LogisticRegression
  - sklearn.linear_model.PassiveAggressiveClassifier
  - sklearn.linear_model.Perceptron
- 41142 (christine)
  - SVC_linear
  - SVC_poly
  - SVC_rbf
  - SVC_sigmoid
  - sklearn.discriminant_analysis .LinearDiscriminantAnalysis
  - sklearn.discriminant_analysis .QuadraticDiscriminantAnalysis
  - sklearn.ensemble.ExtraTreesClassifier
  - sklearn.ensemble.GradientBoostingClassifier
  - sklearn.ensemble.RandomForestClassifier
  - sklearn.linear_model.LogisticRegression
  - sklearn.linear_model.PassiveAggressiveClassifier
  - sklearn.linear_model.Perceptron

B.2 Datasets infomation

Appendix C Curve fitting methods setting

C.1 Newton

- tolerance (objective function): $1.48e-08$
- maximum iteration: 100

id	name	instances	features	numerical	classes	missing
3	kr-vs-kp	3196	36	0	2	0%
6	letter	20000	16	16	26	0%
44	lespambase	4601	57	57	2	0%
1468	cnae-9	1080	856	856	9	0%
41142	christine	5418	1636	1599	2	0%

C.2 Levenberg–Marquardt

- tolerance (objective function): $1e-08$
- tolerance (parameter values): $1e-08$
- maximum iteration: $100 * n * (n+1)$
  where n is number of parameter model function has

Appendix D Differential evolution

- recombination: 0.7
- mutation: $(0.5, 1)$
- maximum iteration: 1000
- polishing method: Levenberg–Marquardt