machine learning & knowledge extraction

MDPI

Article

Do Randomized Algorithms Improve the Efficiency of Minimal Learning Machine?

Joakim Linja *, Joonas Hämäläinen, Paavo Nieminen and Tommi Kärkkäinen

Faculty of Information Technology, University of Jyväskylä, P.O. Box 35, University of Jyväskylä, FI-40014 Jyväskylä, Finland; joonas.k.hamalainen@jyu.fi (J.H.); paavo.j.nieminen@jyu.fi (P.N.); tommi.p.karkkainen@jyu.fi (T.K.)

* Correspondence: joakim.j.linja@jyu.fi

Received: 29 September 2020; Accepted: 11 November 2020; Published: 13 November 2020

Abstract: Minimal Learning Machine (MLM) is a recently popularized supervised learning method, which is composed of distance-regression and multilateration steps. The computational complexity of MLM is dominated by the solution of an ordinary least-squares problem. Several different solvers can be applied to the resulting linear problem. In this paper, a thorough comparison of possible and recently proposed, especially randomized, algorithms is carried out for this problem with a representative set of regression datasets. In addition, we compare MLM with shallow and deep feedforward neural network models and study the effects of the number of observations and the number of features with a special dataset. To our knowledge, this is the first time that both scalability and accuracy of such a distance-regression model are being compared to this extent. We expect our results to be useful on shedding light on the capabilities of MLM and in assessing what solution algorithms can improve the efficiency of MLM. We conclude that (i) randomized solvers are an attractive option when the computing time or resources are limited and (ii) MLM can be used as an out-of-the-box tool especially for high-dimensional problems.

Keywords: machine learning; supervised learning; distance–based regression; minimal learning machine; approximate algorithms; ordinary least–squares; singular value decomposition; random projection

1. Introduction

Minimal Learning Machine (MLM) [1,2] is a supervised learning method that is based on a linear multi-output regression model between the input and output space distance matrices. The distance matrices are computed with respect to a subset of data points referred to as reference points. Although the mapping between the distance matrices is conducted linearly, the kernel-based construction using pairwise distances to the reference points enables the MLM to learn nonlinear relations from data for classification and regression problems. Promising results were obtained with the MLM in several applications [3–5].

The original formulation of the MLM was proposed in [1] and fully formalized in [2]. However, closely related proximity-based machine learning methods were proposed already from the start of the 21st century in [6–10]. Although the proposed methods in [6,9] are also based on using distance kernel in supervised learning, they are very different from the MLM. In the MLM, distances are used directly as features, and the outputs of the distance regression model are only used to define the multilateration problem which can be solved efficiently to obtain the actual prediction. The distance kernel-based construction of the MLM provides advantages compared to the more popular supervised methods such as Neural Networks and Support Vector Machines: The MLM has only one hyperparameter—the number of reference points—and over-learning seems not to occur in multidimensional input spaces [3,11–14]. Moreover, compared to currently popular feedforward

deep learning techniques [15] with nonlinear optimization of multiple layers of weights whose dimensions depend on the data dimension, the number of unknowns in the linear problem for the MLM is independent of the number of features (see Section 2.1). Hence, the simple formulation and straightforward training can improve the efficiency and reliability of the machine learning framework where the MLM is being applied. Practitioners do not need to tune various metaparameters related to the structure of the model and the way in which it is trained, so that more efforts can be dedicated to the way in which a particular application is being presented for supervised learning (see, e.g., [3]).

Lately, it was shown that the MLM possesses basic theoretical guarantees [12]: interpolation and universal approximation capability. To reduce the complexity of the distance regression model, reference point selection methods for the MLM were proposed for classification [16,17] and regression [12]. In Ref. [13], use of a data independent regularization matrix in the MLM training was proposed to reduce the MLM model's complexity similarly to reference point selection. The prediction of the MLM is obtained by solving a multilateration optimization problem determined by the predicted distances of the distance matrix mapping. In general, methods such as Levenberg-Marquardt [1], Newton [14], or Localization Linear System [12] (LLS) can be used to solve this problem efficiently. However, in a single-output regression and in classification, iterative algorithms can be completely avoided by using analytic formulae and nearest neighbor classification [11,18]. Hence, computationally the most demanding step of the MLM is to recover the linear distance matrix mapping from the ordinary least-squares (OLS) problem during training. If cross-validation is used to optimize the MLM's only hyperparameter, the OLS problem has to be solved several times [12].

One possibility to reduce the computational burden in the linear distance regression is to use the iterative Singular Value Decomposition (SVD) update (pp. 101–102, [19]), which tries to reduce the total computational complexity by using the previously computed regression model. A more straightforward approach to speed up the computation is to use randomized SVD (pp. 49–50, [20]) or randomized LU decomposition (pp. 251–252, [21]) where the main idea is to compute the decomposition in a lower dimensional space approximately and then expand the result to the original dimensions. Comparisons between randomized and non-randomized solvers were reported previously, e.g., in [20], but to the best of our knowledge, such comparisons have not been performed when the overall quality of a machine learning framework, here the MLM, is being assessed. We also present a comparison of the MLM with shallow and deep feedforward neural network (FNN) models, experimenting on the effects of the number of features and the number of observations with both methods.

Let us briefly summarize our main findings here. Direct solvers, especially *Cholesky decomposition* hold their position as the go-to solver if accuracy of the MLM model is the main objective. They can be also extended to GPUs [22]. Randomized solvers are a valid option if time is an issue or the accuracy of the model can be slightly relaxed. In addition, differently from feedforward networks, the MLM remains both efficient and accurate in a high-dimensional situation where even 90% of the given features are unnecessary. Therefore, based on the comparison, the MLM should be generally preferred over FNN for high-dimensional datasets.

The contents of the paper are as follows: First, in Section 2, we introduce the basic MLM formulation and solvers for the MLM's distance matrix mapping. In Section 3, we show experimental results for the solver comparison with several real datasets. The expanded result tables are presented in Supplementary Material. Discussion of the results is given in Section 4. Finally, Section 5 concludes the paper.

2. Methods

Next, we summarize the basic MLM method [2,11], the reference point selection algorithm RS–maximin [11,12], and different choices of the linear system solvers.

2.1. MLM in a Nutshell

Given a set of inputs $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^M$, and the corresponding outputs $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^{N}$, where $\mathbf{y}_i \in \mathbb{R}^L$, the MLM originates from the distance matrices $\mathbf{D}_x \in \mathbb{R}^{N\times K}$ and $\mathbf{D}_y \in \mathbb{R}^{N\times K}$. These are computed with respect to sets of reference points $\mathbf{R} = \{\mathbf{r}_k\}_{k=1}^{K} \subseteq \mathbf{X}$ and $\mathbf{T} = \{\mathbf{t}_k\}_{k=1}^{K} \subseteq \mathbf{Y}$, where $\mathbf{D}_x(i,k) = \|\mathbf{x}_i - \mathbf{r}_k\|$, $\mathbf{D}_y(i,k) = \|\mathbf{y}_i - \mathbf{t}_k\|$, and $K \leq N$. Here $\|\cdot\|$ denotes the originally proposed euclidean distance [2], although the MLM method is applicable with any dissimilarity metric.

The linear mapping between the distance matrices can be recovered from the Ordinary Least-Squares (OLS) estimate (p. 36, [2])

$$\mathbf{B} = (\mathbf{D}_x{}^{\top}\mathbf{D}_x)^{-1}\mathbf{D}_x{}^{\top}\mathbf{D}_y. \tag{1}$$

For the special case when $N = K$ we have a single solution if the matrix $\mathbf{D}_x$ is of full-rank (p. 36, [2]). Then the mapping can be solved simply with

$$\mathbf{B} = \mathbf{D}_x{}^{-1}\mathbf{D}_y. \tag{2}$$

Equations (1) and (2) suggest that the inversion of the input distance matrix $\mathbf{D}_x$ has a central role in the computational efficiency of the MLM. It is a dense matrix with non-negative components because with distinct inputs only the distance from each reference point to itself ($\|\mathbf{x}_i - \mathbf{x}_i\|$) is zero.

The original suggestion in [2] was to select the reference points, i.e., locations of the distance-based basis functions, randomly. However, this does not ensure proper exploration and representation of the convex hull of the data. For this purpose, the RS–maximin algorithm was later suggested (p. 11, [12]), (pp. 36–37, [11]) for the reference point selection. RS–maximin is based on maximin clustering method by Gonzales [23]. It selects new reference points sequentially from the input space starting from the mean of data by maximizing the distance to the already selected reference points. This selection method makes the MLM approach fully deterministic and enforces possible duplicate observations in the input data to be the last ones to be selected.

The training phase of the MLM simply consists of the creation of distance matrices and the recovery of the coefficients $\mathbf{B}$ of the linear distance regression. To predict the output $\tilde{\mathbf{y}}$ of a new, unseen input $\tilde{\mathbf{x}}$ with the MLM, one needs to solve a multilateration problem [2]

$$\mathcal{J}(\tilde{\mathbf{y}}) = \sum_{i=1}^{K}\left(\|\tilde{\mathbf{y}} - \mathbf{t}_i\|^2 - \mathbf{ffi}_i^2\right)^2, \tag{3}$$

where $\mathbf{ffi} = [\|\tilde{\mathbf{x}} - \mathbf{r}_1\|, ..., \|\tilde{\mathbf{x}} - \mathbf{r}_K\|]\mathbf{B}$. As explained in Section 1, in single-output regression and multi-class classification, the second step can be solved efficiently by using direct methods [11,18].

As the training phase of the MLM is based on learning a multi-target regression model for the distance matrices, the computational costs and scalability of the method are solely determined by the number of observations also for the high-dimensional input and output data. More precisely, the number of features do not affect the dimensions of $\mathbf{B}$ as it is determined by the number of observations. It does affect the computation time of $\mathbf{B}$ as vectors need to be computed for $\mathbf{B}$. This is a significant difference, e.g., compared to the feedforward randomized [24] or deep [15] learning techniques.

2.2. OLS Calculation Methods

A total of nine different approaches for solving the matrix inverse in Equation (1) in the context of the MLM were chosen for the comparison. While matrix inversion is a thoroughly explored subject, the effect of a solver on the performance of the MLM, from both computational efficiency and prediction accuracy perspectives, has not been presented before. Iterative, inexact stochastic gradient-based methods such as Adam [25] are typically used in deep learning without rigorous control on the solution

accuracy of the nonlinear optimization problem during training, so here we test how randomized approximate solution affects the prediction capability of the MLM.

The methods were selected on the basis of covering different solution techniques and being available. The selected approaches are summarized in Table 1 along with their abbreviations used in the plots. Please note that the recently proposed randomized LU decomposition method in [21] was omitted from the comparison because it is based on a randomized SVD template which is readily covered by the other techniques.

Table 1. Methods compared.

Name	Abbreviation	Source
np.lstsq	Lstsq_X	[26]
np.inverse	np_inv_X	[26]
np.solve	np_sol_X	[26]
Cholesky decomposition	Cho.Dec	[26,27]
np.svd	np_svd	[26]
sp.svd	sp_svd	[27]
rank K random SVD	rKrSVD_XX	[20]
sk rank random SVD	sk_rrSVD_XX	[28]
SVD update	SVD–upd	[19]

More of the practical details of the methods compared are given in Section 3.1. It is also relevant to note that when using RS–maximin as the reference point selection algorithm for the MLM, the rank of matrix $\mathbf{D}_x$ increases as a function of the number of reference points as shown in Figure 1. We show the mean of the number of relevant singular values in Figure 1 as each simulation was repeated 40 times for each dataset and reference point percentage. This is explained further in Section 3.

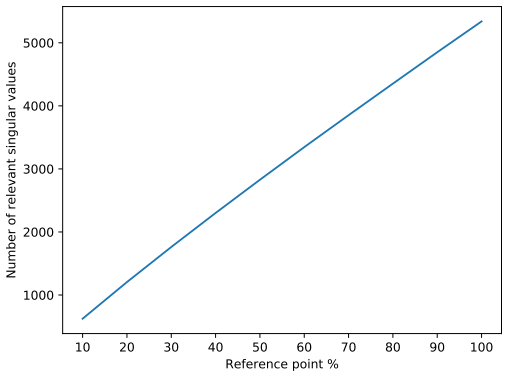


Figure 1. Mean of the number of relevant singular values in matrix $\mathbf{D}_x$ as the function of reference points with dataset Computer Activity.

The singular value decomposition [29] is a factorization method of a matrix $\mathbf{A} \in \mathbb{R}^{N \times K}$ in which it can be expressed as the product of three matrices: unitary $\mathbf{U} \in \mathbb{R}^{N \times N}$, diagonal $\mathbf{S} \in \mathbb{R}^{N \times K}$ and unitary $\mathbf{V} \in \mathbb{R}^{K \times K}$ in the way of $\mathbf{A} = \mathbf{USV}^{\top}$. The randomized SVD [20,28] methods are based on a random projection to a lower matrix rank by constructing a desired rank randomized orthogonal matrix and using it with the original matrix to construct the singular value decomposition. Their goal is to approximate the input matrix to a high enough degree to be useful in practical situations while providing speedup in computational times by momentarily moving the calculation to a lower dimension.

The use of randomized SVD's in solving $\mathbf{B}$ with Equation (1) is based on the singular value decomposition of the input space distance matrix $\mathbf{D}_x = \mathbf{USV}^\top$ which allows us to avoid computing the matrix inverse. By substituting $\mathbf{D}_x = \mathbf{USV}^\top$ into Equation (1) we get

$$\begin{aligned}\mathbf{B} &= \left(\mathbf{VS}^\top\mathbf{U}^\top\mathbf{USV}^\top\right)^{-1}\mathbf{VS}^\top\mathbf{U}^\top\mathbf{D}_y \\ &= \left(\mathbf{VS}^2\mathbf{V}^\top\right)^{-1}\mathbf{VS}^\top\mathbf{U}^\top\mathbf{D}_y \\ &= (\mathbf{V}^\top)^{-1}\mathbf{S}^{-2}\mathbf{V}^{-1}\mathbf{VS}^\top\mathbf{U}^\top\mathbf{D}_y \\ &= \mathbf{VS}^{-1}\mathbf{U}^\top\mathbf{D}_y.\end{aligned} \tag{4}$$

Equation (4) is useful since the SVD decomposition can be used as it is and there is no need to reconstruct $\mathbf{D}_x$ from the decomposition or to compute heavy matrix inversions. In this paper, we use Equation (4) in the place of Equation (1) when using SVD based methods.

3. Experiments and Results

In this section we describe the experimental setup and show the experimental results.

3.1. Details on the Methods

Of the nine tested unique methods, five were tested multiple times with small differences. Methods *Lstsq*, *np.inverse* and *np.solve* had two versions, one with Tikhonov regularization [30] and one without. Methods *rank K random SVD* and *sk rank random SVD* are based on random projection to a lower matrix rank and were tested with five different settings, one for each remaining rank percentage (20%, 40%, 60%, 80%, and 100%). Thus, the total number of tested methods and their versions was 20.

Of the methods, *np.lstsq*, *np.inverse*, *np.solve*, *Cholesky decomposition*, *np.svd*, *sp.svd* and *sk rank random SVD* were readily implemented in commonly available Python packages [26–28]. Of these, *np.inverse* and *np.solve* require the input matrix to be square and invertible. *Cholesky decomposition* requires the input matrix to be a Hermitian positive–definite matrix. Matrix $\mathbf{D}_x^\top\mathbf{D}_x$ in (1) is not guaranteed to satisfy these assumptions. As the readily implemented methods are readily available with documentation and source code, we will not go into detailed explanations here. As their names imply, *np.lstsq* uses a least squares solver to compute the approximate solution, *np.inverse* solves the inverse of the given matrix and *np.solve* computes the solution to a linear equation. Decomposition methods *Cholesky decomposition*, *np.svd*, *sp.svd* and *sk rank random SVD* compute their respective decomposition and SVD based ones make use of Equation (4). The *Cholesky decomposition* solves the inverse of a matrix by computing $\mathbf{L}$ and using a specialized solver made for triangular groups of equations.

Martinsson et al. [20] proposed a rank–*k* random SVD algorithm, for which we could not find any available implementation. Our implementation is based on algorithms 4.3 and 5.1 by (pp. 25–28, [31]). Moreover, the SVD update by (pp. 101–102, [19]) was readily available for Python 2.7 (link in source for original) and had to be modified to use Python 3.

Finally, the so-called industry standard least squares solver with Tikhonov regularization (*Lstsq*) was chosen as the baseline in the statistical analysis of the MLM model's generalization capability (i.e., cross-validation error) using the Kruskal–Wallis H test [32].

3.2. Datasets

A representative set of publically available machine learning datasets were chosen for the tests, as presented in Table 2. Of the public datasets, the first five are actual regression datasets and, therefore, primarily used in this study. They were previously used in the MLM research and they were chosen to enable us to probe the scalability of the OLS computation methods (p. 13, [12]). The largest dataset, *Mnist*, was created by combining the training set and the test set. Because of our special research interest in nanotechnology [3], the density functional theory (DFT) oriented nanostructures

as gathered by Juarez-Mosqueda et al. [33] were used to create nine different special datasets. These different feature representations of the Au_{38Q} hybrid nanostructures allow us to assess the effect of the number of features and the number of observations. The details on how these nine instances of data were formed are given in Appendix C.

Table 2. Datasets used.

Dataset name	Acronym	# Obser.	# Feat.	# Uniq. Targets	Source
Breast Cancer Progn. (Wisc.)	BC	194	32	94	[34]
Boston Housing	BH	506	13	229	[34]
Airplane Companies Stocks	AC	950	9	203	[35]
Computer Activity (CPU)	CA	8192	21	56	[36]
Census (House 8L)	CE	22,784	8	2045	[36]
Mnist	MN	70,000	784	10	[37]
Au_{38Q}	AuNx-yk	12,413	4000	12,413	[33]

Preprocessing each dataset included removal of incomplete observations and constant features, and normalization of the input and output ranges into $[0, 1]$ with minmax-scaling. Duplicate observations were not removed. This only affects the *Census* dataset due to it being the only one with duplicate observations (one duplicate). The duplicate was present in the simulations when the portion of the reference points increased but it did not affect the results because of the use of RS-maximin as the reference point selection algorithm (see Section 2.1).

The first two features (measurement identification number and class-based outcome variable) were removed from the Breast Cancer dataset and the feature "disease-free time or recurrence time" was used as the target output. For the other datasets, the last feature was used as the output feature. Table 2 presents the dimensions of the input matrices for the datasets. The sources of the datasets include a link to where we obtained the datasets.

3.3. Experimental Setup

Experiments were carried out using Python 3.7.3 with two Intel Xeon E5-2680 v3 CPUs (for a total of 24 threads at 2.50 GHz) and 188 GB RAM. The main packages were NumPy (version 1.16.4) [26], Scikit-learn (version 0.20.3) [28] and SciPy (version 1.3.0) [27]. Source code for the implemented *rKrSVD_XX* is provided in the supplementary files along with MLM and RS-maximin. Each dataset was randomly split into a training set (80%) and a test set (20%) 100 times. We measured the root mean square error (RMSE) value of the MLM test error and the process time taken to compute Equation (1) when the portion of the selected reference points increased according to $[10\%, 20\%, \ldots, 100\%]$. The measurements were performed on all methods for all reference points before a new randomly generated dataset split was created. This was to ensure that the results from each iteration would be comparable.

Exceptions had to be made with the *Census* dataset and *SVD update* due to time constraints. The dataset *Census* was randomly split 40 times instead of 100, a smaller set of methods were tested and with reference points $[10\%, 20\%, \ldots, 70\%]$. *SVD update* had to be measured differently, due to its iterative nature. The method was used to evaluate Equation (1) at the same reference points as the other methods. In the required intermediary update iterations, only the internal update was called to avoid reconstructing the SVD decomposition which the *SVD update* holds within itself. For *SVD update*, each provided process time value includes the time taken to evaluate Equation (1) and the time that was taken to get to that evaluation. The *SVD update* in total is the slowest of the selected methods and it was only experimented with the *Breast Cancer*, *Boston Housing* and *Airplane Companies Stocks* datasets.

Based on the results, we wished to study lower remaining rank percentages and chose to do this with *Census* and *Mnist*. The reasons are further expanded upon in Section 4. We selected *Cholesky Decomposition* as the baseline and compared it to *sk rank random SVD* with remaining rank percentages $[1\%, 10\%]$ at reference points $[10\%, 20\%, \ldots, 100\%]$. The measurements were repeated 30 times at

each reference point by randomly selecting the training and test sets. We repeated the process time and the model accuracy measurements 40 times. Each with randomly selected training and test set. The full MLM case of Equation (2) is not used in the experiments. The results as a function of different reference point percentages need to be comparable to each other, which removes the possibility of using Equation (2).

It is also interesting to compare the performance of MLM with a shallow and a deep feedforward neural network (FNN) models due to their popularity [38]. The experiments with FNNs were carried out using the popular Tensorflow_cpu 2.1 for Python 3 framework [39]. Both a shallow model with one hidden layer as well as a deep network architecture with three hidden layers, and four transformation layers altogether, were tested. We applied systematic, gradual decrease strategy to fix the sizes of the hidden layers. More precisely:

Deep feedforward neural network (FNN-4)

1. Input layer of size M,
2. Feedforward layer with sigmoid activation function of size $M/2$,
3. Feedforward layer with sigmoid activation function of size $M/4$,
4. Feedforward layer with sigmoid activation function of size $M/8$,
5. Feedforward layer with sigmoid activation function of size 1,

Shallow feedforward neural network (FNN-2)

1. Input layer of size M,
2. Feedforward layer with sigmoid activation function of size $M/2$,
3. Feedforward layer with sigmoid activation function of size 1,

FNNs were trained using maximum of 150 epochs using a batch size 10 with an early stopping criteria that followed the RMSE test error.I.E. the default settings for Adam-solver and sequential dense layer model given by Tensorflow 2.1 plus the early stopping criteria. Because of random initialization, the training process of FNN is not deterministic, so we repeated the training 10 times and selected the result based on the minimal training error. From the 40 results of the random splits into training/test datasets, we then calculated the mean and standard deviation for the test errors. The random splits into training/test datasets were the same as with the MLM, limiting to the first 40 in the case of smaller datasets (where total of 100 splits were used with the MLM). We measured the training time of the FNNs and compared them to Equation (1) computation time on the same hardware. The reported process times for the FNNs are the average of the sum of 10 trainings made for each training/test split.

Finally, we performed experimentation on the Au_{38Q} dataset. The dataset and its variants allowed us to study the effects of the number of observations and the number of features on the MLM by doing a method comparison between *Cholesky decomposition* and *random SVD* with remaining rank percentages $[1\%, 10\%]$. We also computed results with the FNNs for the Au_{38Q} dataset.

The MLM measurements were performed as previously, we had *Cholesky Decomposition* and variants of *sk_rrSVD_XX* (remaining rank percentages $[1, 10]$) as the solvers. Each dataset was randomly split into 40 different training and test datasets with 80/20 ratio and the reported normalized RMSE value is measured from the test set as well as the process time taken to solve Equation (1). FNN measurements were also carried out in the previous way with the 40 random 80/20 splits and 10 repeats at each random split.

3.4. Results

We were interested in measuring the MLM test error (RMSE) and process time (time taken to compute Equation (1)) and we measured them as a function of reference point percentages $[10\%, 100\%]$ for each dataset. We report the numerical values in tables and of these tables, we show the RMSE table for reference point percentage 70% in Table 3 here. Due to the number of tables, we opted to present the rest in Supplementary Material (Tables S1–S19). In Table 3, we report the mean $\overline{x}$ and

standard deviation σ of each measured method, dataset and reference point percentage combination. For rigorous statistical analysis, [40], we also show the result of Kruskal–Wallis H test (significance level 0.05) by bolding the text of the mean values that are considered to be different from the mean result. In the data tables, below the symbol of either mean $\overline{x}$ or standard deviation σ is the order of magnitude of the numbers in the same column.

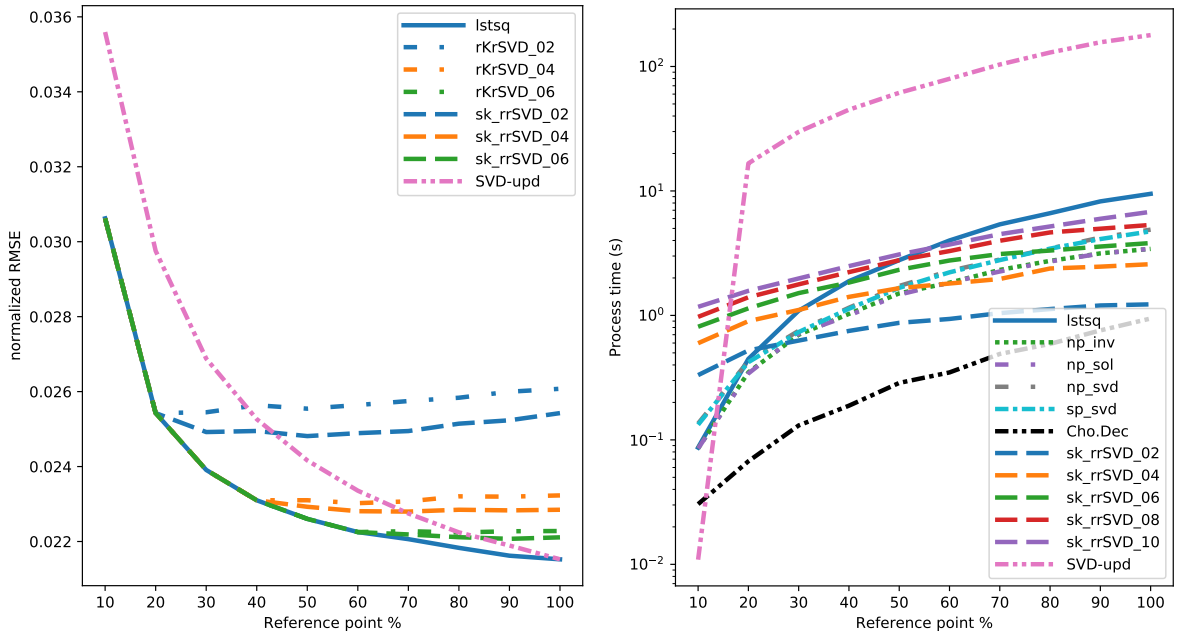
Table 3. RMSE for $K = 70$. Bolded numbers differ statistically from *Lstsq*.

Dataset	**BC**		**BH**		**AC**		**CA**		**CE**	
Method/coef.	$\overline{x}$ $1 \times \mathbf{10^{-1}}$	σ $1 \times \mathbf{10^{-2}}$	$\overline{x}$ $1 \times \mathbf{10^{-2}}$	σ $1 \times \mathbf{10^{-2}}$	$\overline{x}$ $1 \times \mathbf{10^{-2}}$	σ $1 \times \mathbf{10^{-3}}$	$\overline{x}$ $1 \times \mathbf{10^{-2}}$	σ $1 \times \mathbf{10^{-4}}$	$\overline{x}$ $1 \times \mathbf{10^{-2}}$	σ $1 \times \mathbf{10^{-3}}$
Lstsq	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	6.28	2.17
Lstsq (nr)	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
np.inv	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	6.28	2.17
np.inv (nr)	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
np.solve	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	6.28	2.17
np.solve (nr)	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
np.svd	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	6.28	2.17
sp.svd	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
Cho.Dec	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	6.28	2.17
rKrSVD02	**2.67**	2.33	**7.93**	1.38	**2.57**	1.47	**2.48**	8.17	6.21	2.12
rKrSVD04	2.72	2.27	7.1	1.36	**2.31**	1.48	2.45	8.27	6.24	2.18
rKrSVD06	2.75	2.37	6.86	1.36	2.23	1.46	2.44	8.39	6.27	2.18
rKrSVD08	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
rKrSVD10	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
skrSVD02	**2.65**	2.32	**7.8**	1.47	**2.49**	1.63	**2.46**	8.38	**6.18**	2.2
skrSVD04	2.72	2.35	6.98	1.38	**2.28**	1.46	2.44	8.3	6.24	2.2
skrSVD06	2.75	2.37	6.84	1.36	2.22	1.45	2.43	8.39	6.27	2.17
skrSVD08	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
skrSVD10	2.77	2.34	6.79	1.35	2.21	1.47	2.43	8.36	-	-
SVDU	2.74	2.36	7.07	1.47	**2.27**	1.62	-	-	-	-

We show plots of the RMSE and process time behaviors as a function of reference point percentages for *Airplane Companies Stocks* in Figure 2, for *Computer Activity* in Figure 3 and for *Census* in Figure 4. In the figures we show the mean value of either RMSE or process time and plot only the lines that are either from *Lstsq* or differ statistically from it using the Kruskal–Wallis H test with 0.05 significance level. As is shown in Table 3, we also calculated the standard deviation for each datapoint and as stated previously, we presented similar tables corresponding reference point percentages [10–60%, 80–100%] for RMSE and $[10\%, 100\%]$ for process time in Supplementary Material. Non-Tikhonov-regularized versions of methods are not shown in the figures in order to improve readability. Since the two random SVD methods have nearly identical performance, only the readily implemented version, *rank K random SVD* is shown in the figures.

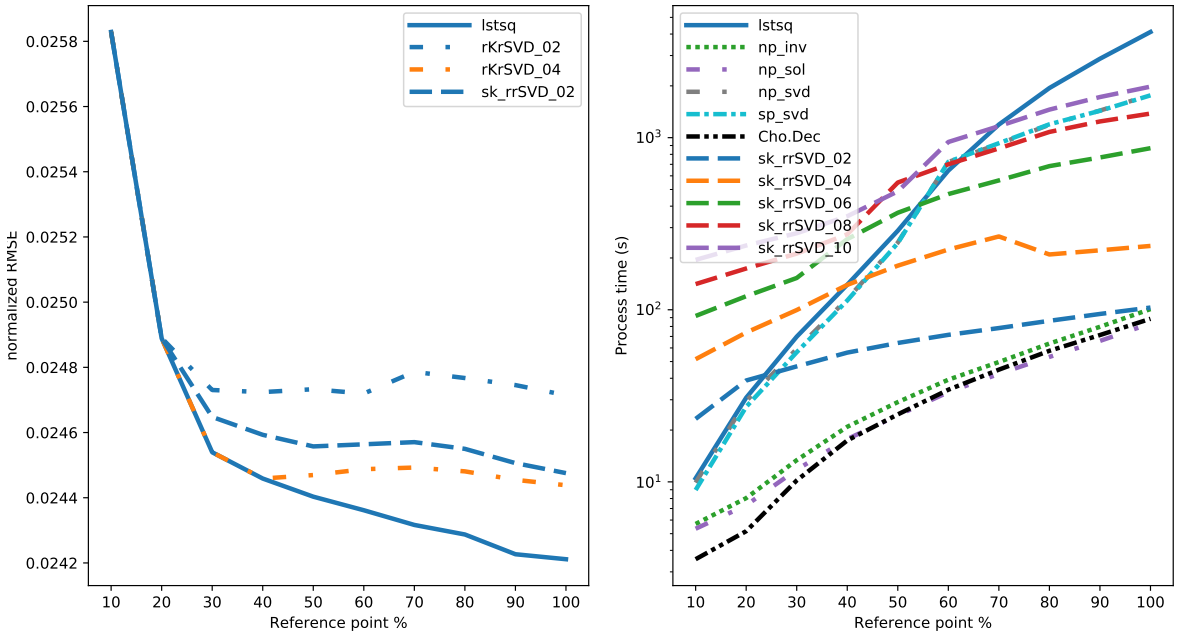
Please note that the process time in Figure 2b is not truly comparable between *SVD update* and the other methods because of the iterative nature of the *SVD update*. The process time values for *SVD update* include the iterative update steps that are necessary for the method. As the other methods do not need iterative steps, they naturally have faster computation times in the type of point testing that we performed.

After our initial simulations we wanted to see an even lower rank reduction percentage. To that end, we chose to use *Cholesky decomposition* as the baseline method due to its performance in the initial simulations. We compared *sk_rrSVD_XX* with remaining rank percentages $[1\%, 10\%]$ to the selected baseline with reference point percentages $[10\%, 100\%]$. As with previous simulations, we repeated the calculations 40 times in order to use statistical testing. The result for RMSE values and process times are shown in Figure 5 and the numerical values and standard deviations can be found in the Supplementary Material in Tables S20 and S21. Please note that unlike in the initial simulations, duplicates were removed from *Census* since the MLM requires all observations to be unique and *Census* as it is contains a duplicate observation.



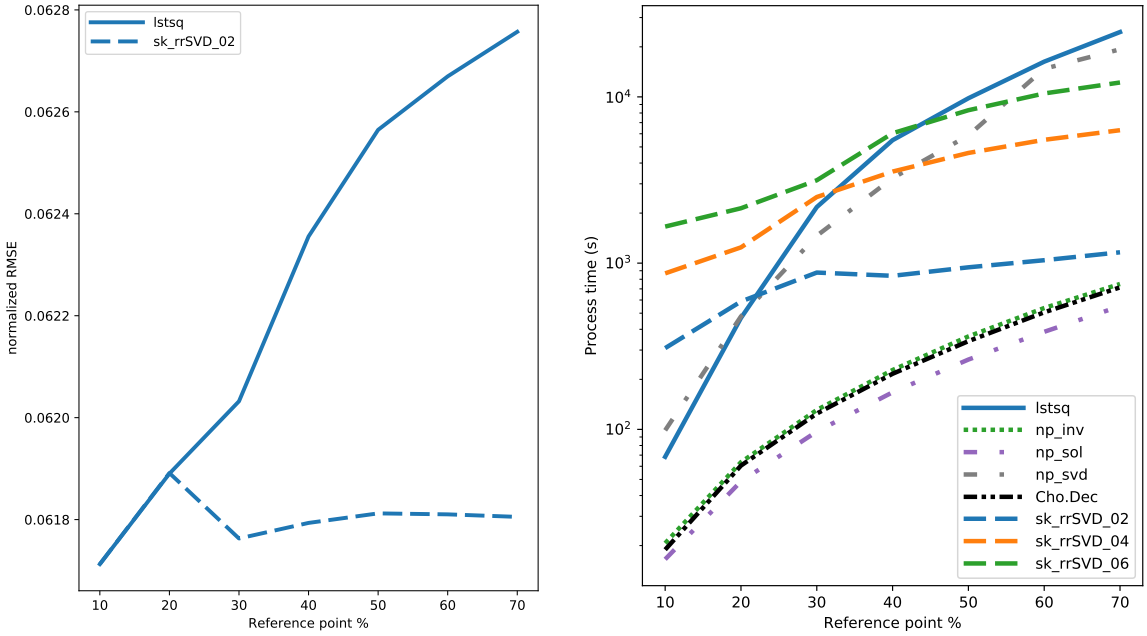
(**a**) Normalized RMSE (**b**) Process time

Figure 2. Normalized RMSE values (**a**) and process times (**b**) as a function of reference points for the *Airplane Companies Stocks* dataset. In (**a**), only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



(**a**) Normalized RMSE (**b**) Process time

Figure 3. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Computer Activity* dataset. In (**a**), only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



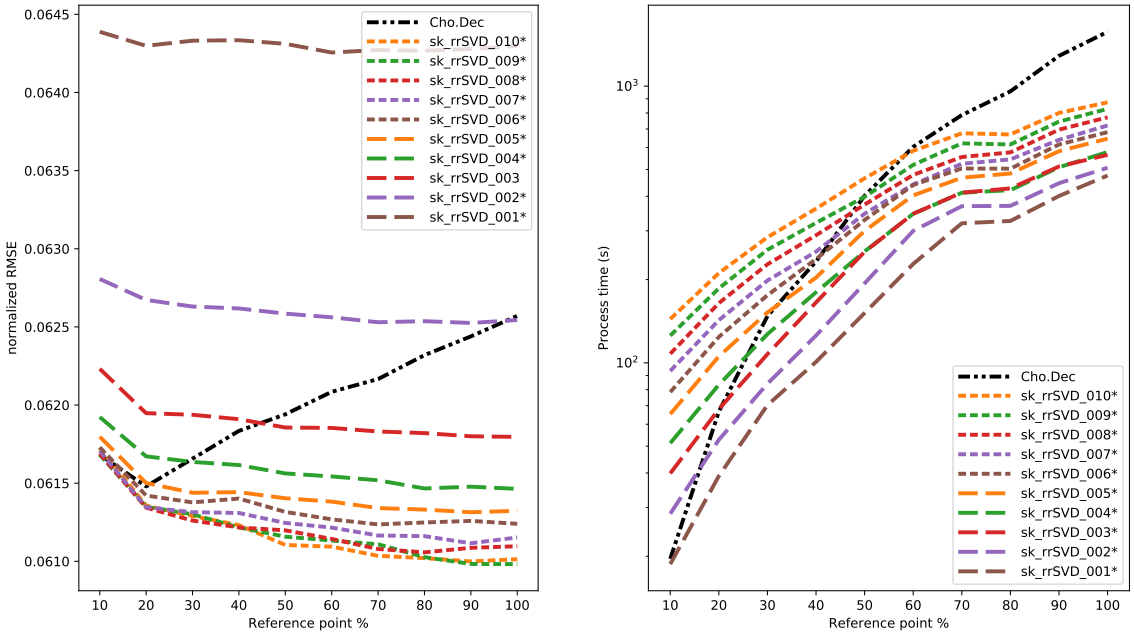
(**a**) Normalized RMSE (**b**) Process time

Figure 4. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Census* dataset. In (**a**), only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.

After the results shown in Figure 5, we wished to see the behavior on an even larger dataset. For this, we chose *Mnist* since it is readily available and sufficiently larger than *Census*. *Mnist* is a classification dataset but here it is considered to be an integer regression dataset. *Cholesky decomposition* continued to function as a baseline method to the compared *sk_rrSVD_XX* at reference point percentages $[10\%, 100\%]$. We used remaining rank percentages $[1\%, 10\%]$ with *sk_rrSVD_XX* and repeated each point calculation 40 times with randomized selection of training and testing sets. The result is shown in Figure 6 and the numerical values along with computed standard deviations are presented in Supplementary Material in Tables S22 and S23. Please note that *Mnist* had duplicate removal as a preprocessing step unlike the datasets in the initial simulations.

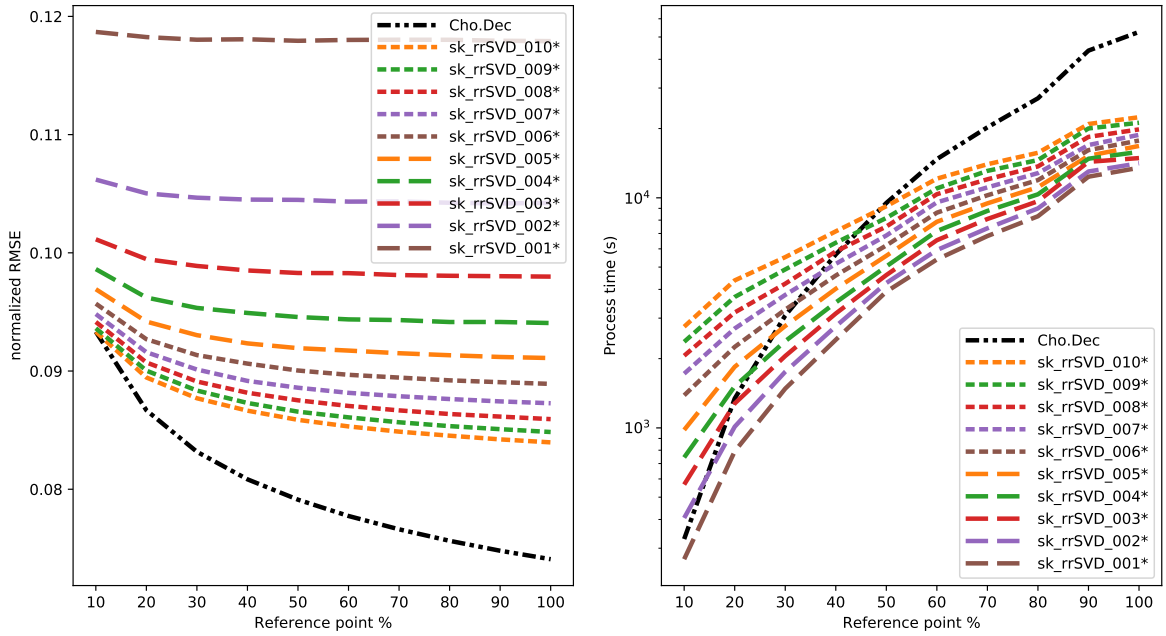
We opted to include result figures with the FNNs in Appendix A for datasets *Breast Cancer* (Figure A1), *Boston Housing* (Figure A2), *Airplane Companies Stocks* (Figure A3), *Census* (Figure A5) and *Mnist* (Figure A6). The raw data is provided in the supplementary as Tables S24 and S25. The figures in Appendix A contain the calculated results for the FNNs and the corresponding results for the MLM. Figures A1 and A2 are the normalized RMSE and process time used for the MLM and the FNNs for datasets *Breast Cancer* and *Boston Housing*. Figure 2 corresponds to Figure A3, where the difference is the addition of the FNN results. Similarily, Figures 3, 5 and 6 respectively correspond to Figures A4, A5 and A6. The figures presented in Appendix A have the item "FNN-4" representing the results relating to the deep neural network, or the neural network with four layers and "FNN-2" representing the one with two layers. For the figures with RMSE values, we also included the standard deviation and represented it as a see-through filled area.

Results for the dataset Au_{38Q} are shown in Appendix B in Figures A7 and A8. The raw data is provided in the supplementary as Tables S26–S45. It is important to note that in the name of readability, we included another version of Figure A7 in Figure A9. The difference being the removed plot of *FNN-2* due to its behaviour in the 4000 feature column. In addition, Figures A8 and A9 also have equalized Y-axes to better show the behaviour of MLM.



(**a**) Normalized RMSE (**b**) Process time

Figure 5. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Census* dataset. Variants of *sk_rrSVD_XX* that differ statistically from *Cholesky decomposition* are marked with *.



(**a**) Normalized RMSE (**b**) Process time

Figure 6. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Mnist* dataset. Variants of *sk_rrSVD_XX* that differ statistically from *Cholesky decomposition* are marked with *.

4. Discussion

Various methods to recover the weight matrix in the distance-regression and their effects to the accuracy of the whole MLM were studied with fifteen different instances of seven datasets. In addition to this, we also compared a smaller subset of MLM methods with two feedforward neural network

configurations. We will first provide discussion related to the direct and randomized solvers and then discuss the comparison between MLM and FNNs. We wished also to see the performance of the novel incremental SVD update algorithm *SVDU* but the computational resources it required forced us to stop using it on bigger datasets than *Airplane Companies Stocks* (see Figure 2b).

4.1. Comparison of Direct and Randomized Solvers for MLM

In the experiments, we wanted to see if the industry standard solver *Lstsq* could be improved upon by randomization. It was noted that the readily implemented solvers generally had the same accuracy with minor differences in progress times. Considering this, we do our comparisons of *random SVD* to either *Lstsq* due to its status as the industry standard or to *Cholesky decomposition* due to it generally outperforming all other readily implemented methods. We presented summary of the comparison in Table 4. In short, Table 4 tells us that randomized solvers beat the accuracy of *Cholesky decomposition* in two datasets and that the training time with randomized solvers is faster in 11 datasets. Expecially in the largest two and in the Au_{38Q} dataset. We can summarize Figures 2a, 3a and 4a by noting that readily implemented methods match each other in the accuracy they provide to the MLM and that randomized solvers start falling behind at 30% reference points and higher. It is to be expected that the introduction of randomness in the solver causes the accuracy of the MLM to change relative to the amount of introduced randomness. The differences between the methods are mostly found in the process time Figures 2b, 3b and 4b. Based on the process time behavior in the figures, it is clear that a specialized solver such as *Cholesky decomposition* is superior to *Lstsq* in the MLM context. This led us to leave *Lstsq* out of further experiments and use *Cholesky decomposition* as the default method in this comparison. The results also show the expected behaviour that the degree of randomness in the *random SVD* solver has a direct effect on the process time taken.

The process time figures gave us the idea to use even lower remaining rank percentages with the *random SVD* and to use it on a larger dataset. This led us to use remaining rank percentages $[1\%, 2\%, \ldots, 10\%]$ which were reported in Figures 5 and 6. Figures 5a and 6a, in conjunction with Figures 2a, 3a and 4a, show that the *random SVD* has a tendency to plateau around 30–50% reference point mark. This is to say that the *random SVD* does not provide the MLM with meaningful increases in accuracy at higher reference point percentages. The same 30–50% reference point mark is the area where low remaining rank percentage *random SVD* overtakes *Cholesky decomposition* in process time as is seen in Figures 5b and 6b. We do not know why in Figure 5a the *Cholesky decomposition* shows a non-improving accuracy score.

Table 4. Summary of comparison between *Cholesky decomposition* and randomized solvers for the MLM. Symbols + and − indicate that the randomized solver is better/worse than the *Cholesky decomposition*.

MLM (*Cholesky Decomposition*)	**Accuracy**	**Training Time**
Dataset	**rnd. Solver**	**rnd. Solver**
BreastCancer	+	−
BostonHousing	−	−
AirplaneCompanies	−	−
ComputerActivity	−	−
Census	+	+
Mnist	−	+
AuN2-4k	−	+
AuN2-8k	−	+
AuN2-12k	−	+
AuN10-4k	−	+
AuN10-8k	−	+
AuN10-12k	−	+
AuN100-4k	−	+
AuN100-8k	−	+
AuN100-12k	−	+

The results of the MLM in the case of the Au_{38Q} dataset are shown in Figures A7, A8 and A9. Figure A8 depicts the process times and Figure A9 the accuracy most clearly. An immediate observation from Figure A9 is that the subfigure columns for 400 features and 4000 features appear identical.

Figure A10 shows that the outlook of the dataset has not changed in any meaningful way when the number of features increases from 400 to 4000. The reason for that is found in the original nanocluster trajectory data for Au_{38Q} where the entire dataset is composed of nanocluster configurations where each adjacent nanocluster differs from each other in miniscule fashion. This is to say that the nanocluster configurations in Au_{38Q} are similar enough to each other that 4000 features do not provide additional information that could be used when compared to 400 features. Should there be more variance in the dataset (such as nanoclusters from another isomer) we would expect to see 4000 features provide information that 400 features cannot provide. This is at the same time a good argument for feature selection. Regarding *random SVD* and *Cholesky decomposition*, we are comfortable in noting that *Cholesky decomposition* is a clear winner in the accuracy it provides to the MLM with the Au_{38Q} dataset. Although we do suspect that the result would not be so cut an dry with a dataset of more variance. Considering the properties of the MLM in the case of Au_{38Q} in Figure A7 we can say the following: The increasing number of observations improves the accuracy of the MLM model and the increasing number of features do not improve the result if the increased number of features do not provide meaningful information. Also, the increasing number of features does not make the result worse either. Process time wise (Figure A8), the MLM behaved as expected. The process time taken increased as the number of observations increased and it changed only slightly by the changes in the number of features.

4.2. Comparison of MLM to Shallow and Deep FNNs

We begin the discussion about the FNN results with a short summary of them. In general, the deep FNN architecture, with higher computational costs, provided better accuracy than the shallow one. However, the variation of accuracy for both FNN models was rather extensive in the *Breast Cancer*, *Boston Housing*, *Computer Activity* and *Census* datasets. One could expect that the larger network *FNN-4* would have higher accuracy than the *FNN-2*, with longer training times as a drawback. One could also expect that the *FNN-2* models for datasets with low number of features could be competitive against the more complex *FNN-4*. Both assumptions are backed up by our results, although the process time for *Mnist* in Figure A6b was smaller for the *FNN-4* compared to *FNN-2*.

The results for the MLM and FNN comparison are summarized in Table 5. In terms of the overall performance, i.e., by considering both accuracy and training time, MLM with *Cholesky decomposition* was better than FNN in nine of fifteen (60%) of the cases. The training times were on a same level only with *Mnist*, otherwise MLM training was more efficient with a clear margin, altogether in 93% (14/15) of the cases. The FNN-4 model's accuracy was comparable to the MLM in five cases, and better than MLM only for the *Mnist* dataset. This outcome is probably due to the form of *Mnist* data, which is given by small 28×28 images whose processing through multiple higher-level feature generating layers is known to be advantageous, especially, compared to a direct distance computation between the gray scale values as with the MLM.

Regarding FNN and the Au_{38Q} dataset, one can immediately see that *FNN-2* is not competitive in a situation with 4000 features (Figure A7). This was fully expected, as *FNN-2* was not meant for datasets with high number of features (even if it adapts to the number of features in the dataset). In addition, *FNN-2* can at best only match *FNN-4* with the Au_{38Q} dataset. Observing *FNN-4* in Figure A9 reveals a trend. At first, the increasing number of features to 400 is beneficiary for both MLM and *FNN-4*. However, a further increase in the number of features to 4000 causes *FNN-4* to lose accuracy while MLM stays the same. In addition, the increase in the number of observations causes *FNN-4* to gain accuracy with 80 and 400 features whereas it loses accuracy in the case of 4000 features. This indicates that *FNN-4* cannot naturally handle an increased number of either observations or features that do not provide new information. Something that MLM shows itself to be capable of.

Table 5. Summary of comparison of MLM with *Cholesky decomposition* to a shallow (FNN-2) and a deep (FNN-4) neural network results. Symbols **+**, $\approx$, and $-$ indicate that the FNN results are better, no different, or worse than the MLM results, respectively. Symbol / and the symbols with it are used to signify that the FNN results are not properly in either category.

MLM (*Cholesky Decomposition*)	**Accuracy**		**Training Time**	
Dataset	**FNN-4**	**FNN-2**	**FNN-4**	**FNN-2**
BreastCancer	$\approx$	$\approx$	$-$	$-$
BostonHousing	$-$	$-$	$-$	$-$
AirplaneCompanies	$-$	$-$	$-$	$-$
ComputerActivity	$\approx$	$-$	$-$	$-$
Census	$-$	$-$	$-$	$-$
Mnist	$+$	$\approx$	$-$ / $\approx$	$-$/ $\approx$
AuN2-4k	$-$	$-$	$-$	$-$
AuN2-8k	$-$	$-$	$-$	$-$
AuN2-12k	$-$	$-$	$-$	$-$
AuN10-4k	$-$/ $\approx$	$-$ / $\approx$	$-$	$-$
AuN10-8k	$-$ / $\approx$	$-$ / $\approx$	$-$	$-$
AuN10-12k	$-$ / $\approx$	$-$	$-$	$-$
AuN100-4k	$-$	$-$	$-$	$-$
AuN100-8k	$-$	$-$	$-$	$-$
AuN100-12k	$-$	$-$	$-$	$-$

In order to improve the FNN models' accuracies, we would need to manually tailor the models for each dataset. This brings an important point in the comparison of the MLM to FNN models. The MLM has a clear advantage compared to the deep FNN models. Achieving a better result with the MLM can be conducted straightforwardly since MLM has a only one hyperparameter. Increasing the number of reference points increases the accuracy of the regression model with a cost of higher training time. In contrast, the FNN models are known to require careful hyperparameter selection and a well-designed structuring of the hidden layers. Zhou and Feng [41] argue that the training of deep neural network models resembles more an art than science or engineering.

5. Conclusions and Future Work

Summarizing our work, we showed results for the accuracy of the MLM and the process time taken to solve Equation (1) for randomized SVD algorithms with varying level of randomness and for direct solvers that are well-known to research community. In addition to this, we compared the accuracy of the MLM to deep and shallow FNN models by means of the number of observations and the number of features. To begin, we started with *Lstsq* as the standard and ended up switching it to *Cholesky decomposition* as it clearly outperformed *Lstsq* as well as other tested direct solvers. The only datasets where *Cholesky decomposition* was not the clear winner in accuracy provided to the MLM were *Breast Cancer* (see Tables 3, Tables S5–S9) and *Census* (see Figure 5a and Table S20). From there we can make the conclusion that if the accuracy of the MLM is the only objective, then randomized solvers do not provide an advantage if the dataset is preprocessed so that there are no duplicates. The randomized solvers come in to play if the training speed is an issue and one is handling a large number of observations. The *random SVD* follows the general principle that the higher the randomness the lower the accuracy and faster the training speed. However, in the context of the MLM the reduction of accuracy in the solver does not translate to as great of a loss in model accuracy. Meaning that one might expect to have results roughly somewhere in the ballpark if 99% of the information is replaced with randomness but in practice the results with *random SVD* at low remaining rank percentage are in the same order of magnitude. We can conclude that a researcher may choose to sacrifice some accuracy and gain speed in the training phase by choosing a randomized solver with a low remaining rank percentage.

As a whole, we demonstrated that:

1. If accuracy is the only objective in a prepared situation, one should choose to use a direct solver with the MLM.
2. If not, randomized solvers provide comparable performance to direct solvers while providing a speedup.
3. Utilization of linear training technique with a distance-based kernel is computational more efficient than nonlinear optimizers needed for the FNN models. In general, the MLM outperformed the shallow and deep FNNs with fixed architecture, with one exception. Especially, in cases of thousands of features, as with the experiments for the Au_{38Q} dataset here, the MLM is the recommended technique from both training time and model accuracy perspectives.

Therefore, the use of randomized algorithms appear to be a promising avenue for improving the efficiency of the MLM and other distance-based regression techniques. In addition to this, we believe that our comparison of the MLM with currently popular neural network techniques showed that the MLM is a viable option as the general method for nonlinear regression problems.

While this work focused on randomized algorithms, we find it an important topic of future work to expand the comparison also to iterative solution methods such as the Chebyshev method [42] and the block-iterative methods [43]. Especially the classical block Jacobi method with successive overrelaxation can be easily parallelized using data parallelism. It would also be interesting to see the performance of the MLM when it is set to use a GPU in the calculations.

Supplementary Materials: The following are available online at http://www.mdpi.com/2504-4990/2/4/29/s1 , Table S1: RMSE for $K = 10$, Table S2: RMSE for $K = 20$, Table S3: RMSE for $K = 30$, Table S4: RMSE for $K = 40$, Table S5: RMSE for $K = 50$, Table S6: RMSE for $K = 60$, Table S7: RMSE for $K = 80$, Table S8: RMSE for $K = 90$, Table S9: RMSE for $K = 100$, Table S10: Process time (s) for $K = 10$, Table S11: Process time (s) for $K = 20$, Table S12: Process time (s) for $K = 30$, Table S13: Process time (s) for $K = 40$, Table S14: Process time (s) for $K = 50$, Table S15: Process time (s) for $K = 60$, Table S16: Process time (s) for $K = 70$, Table S17: Process time (s) for $K = 80$, Table S18: Process time (s) for $K = 90$, Table S19: Process time (s) for $K = 100$, Table S20: RMSE for *Census* at reference point percentages $[10\%, 100\%]$, Table S21: Process time in seconds for *Census* at reference point percentages $[10\%, 100\%]$, Table S22: RMSE for *Mnist* at reference point percentages $[10\%, 100\%]$, Table S23: Process time in seconds for *Mnist* at reference point percentages $[10\%, 100\%]$, Table S24: RMSE for neural networks, Table S25: Process time (s) for neural networks, Table S26: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 10$, Table S27: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 20$, Table S28: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 30$, Table S29: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 40$, Table S30: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 50$, Table S31: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 60$, Table S32: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 70$, Table S33: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 80$, Table S34: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 90$, Table S35: Normalized RMSE mean, standard deviation and best result for Au_{38Q} $K = 100$, Table S36: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 10$, Table S37: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 20$, Table S38: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 30$, Table S39: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 40$, Table S40: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 50$, Table S41: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 60$, Table S42: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 70$, Table S43: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 80$, Table S44: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 90$, Table S45: Process time (s) mean, standard deviation and best result for Au_{38Q} $K = 100$, Source codes for the Equation (1), source codes for the FNN models and source code for the MLM. The Au_{38Q} dataset is available at 10.5281/zenodo.4268064.

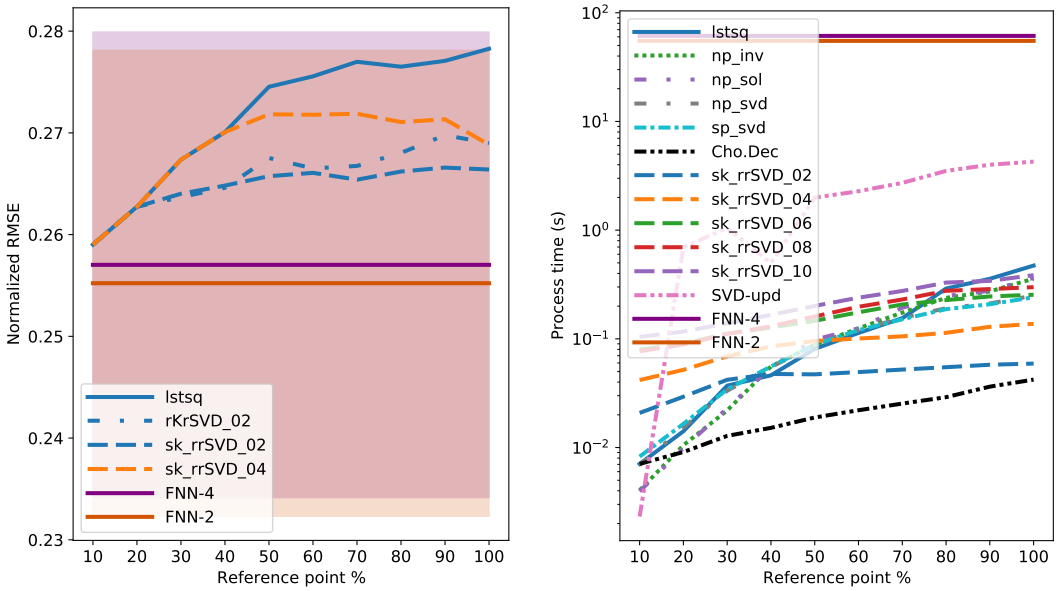
Author Contributions: Conceptualization, J.L., J.H., P.N., T.K.; methodology, J.L.; software, J.L. and J.H.; formal analysis, J.L.; investigation, J.L.; data curation, J.L.; writing—original draft preparation, J.L. and J.H.; writing—review and editing, J.L., J.H., P.N., T.K.; visualization, J.L., J.H., P.N., T.K.; supervision, P.N. and T.K.; project administration, T.K.; funding acquisition, T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Academy of Finland from the projects 311877 (Demo) and 315550 (HNP-AI).

Acknowledgments: We acknowledge grants of computer capacity from the Finnish Grid and Cloud Infrastructure (persistent identifier urn:nbn:fi:research-infras-2016072533).

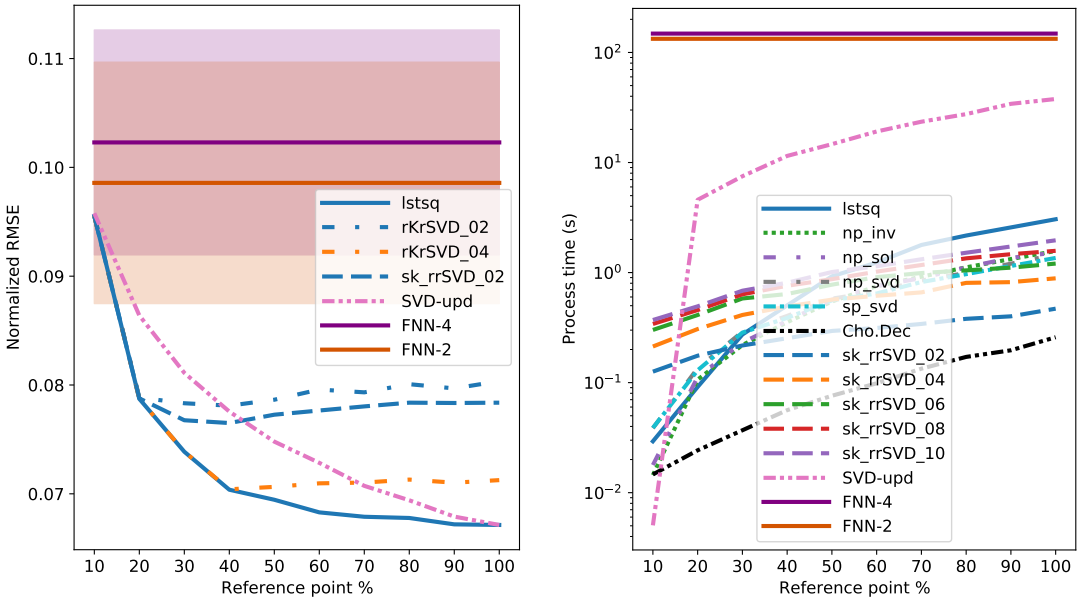
Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. MLM vs. FNN



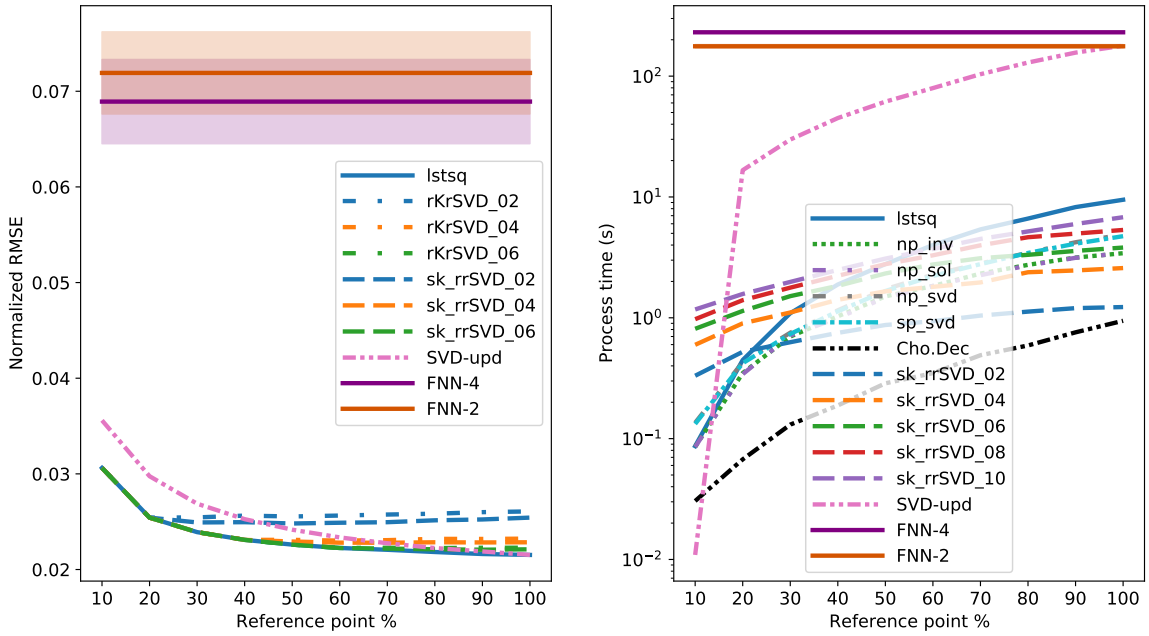
(**a**) Normalized RMSE (**b**) Process time

Figure A1. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Breast Cancer* dataset. In (**a**) for MLM, only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



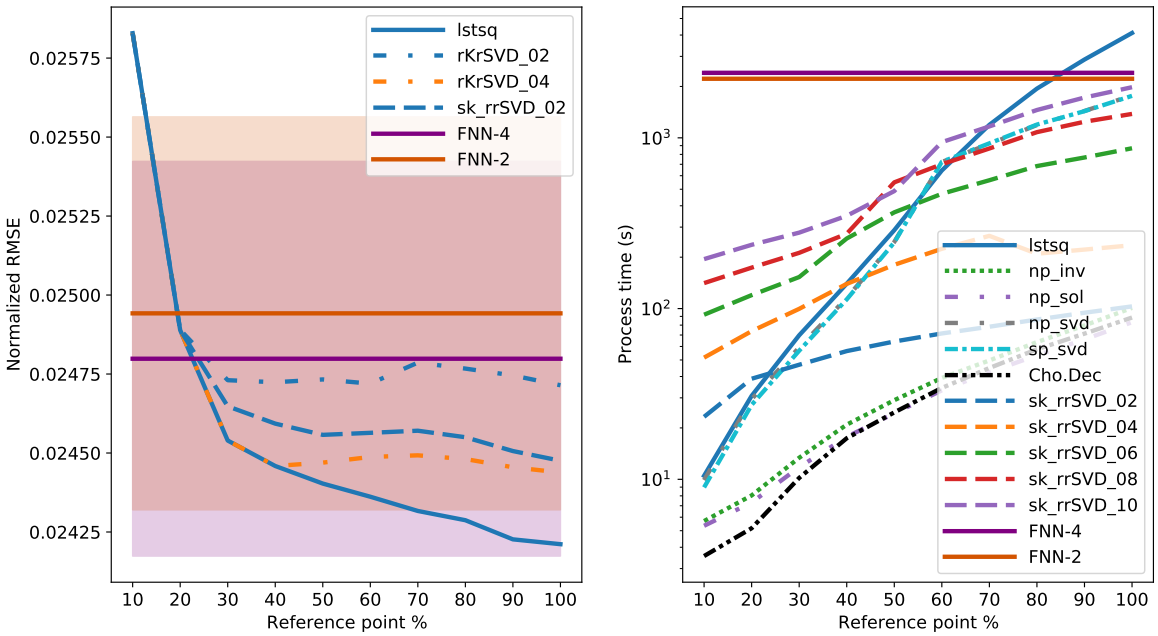
(**a**) Normalized RMSE (**b**) Process time

Figure A2. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Boston Housing* dataset. In (**a**) for MLM, only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



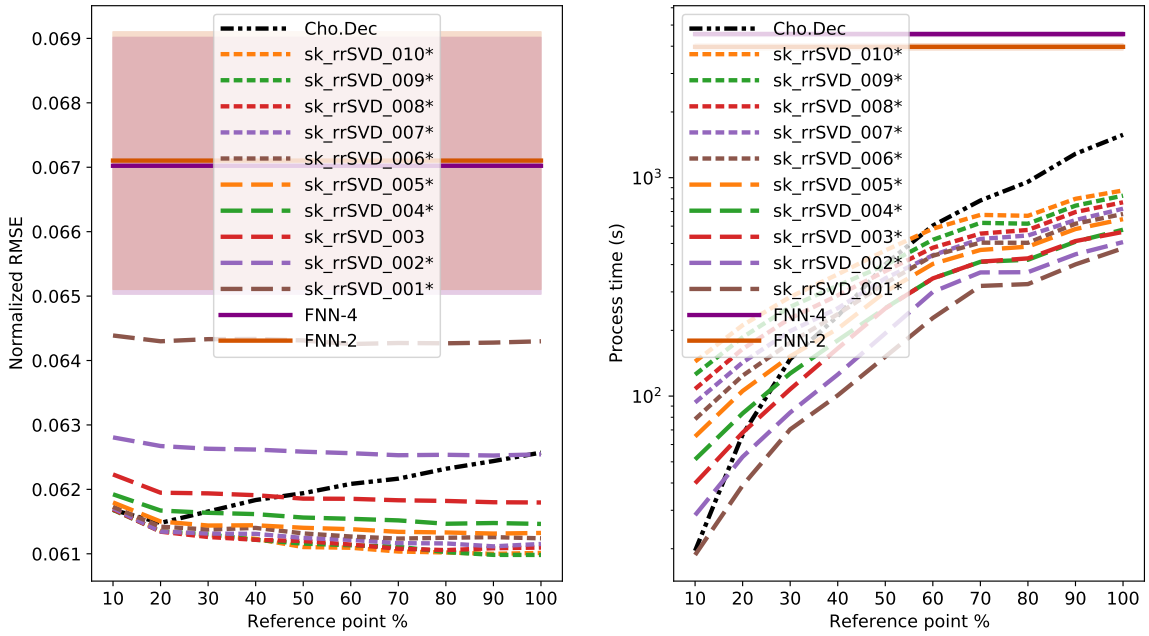
(**a**) Normalized RMSE (**b**) Process time

Figure A3. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Airplane Companies Stocks* dataset. In (**a**) for MLM, only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



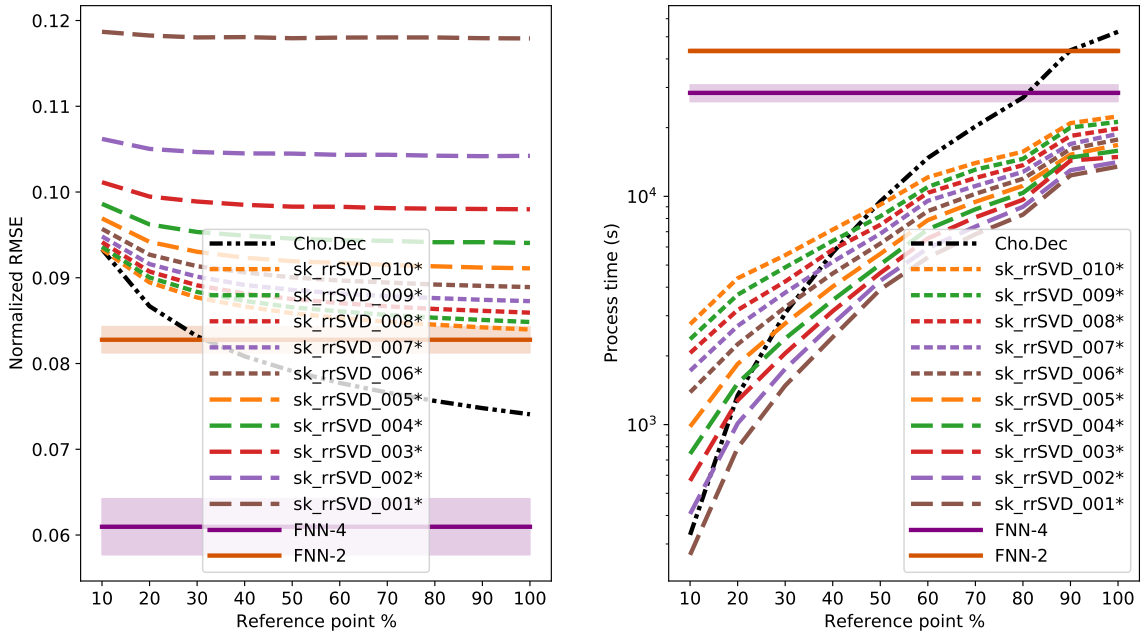
(**a**) Normalized RMSE (**b**) Process time

Figure A4. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Computer Activity* dataset. In (**a**) for MLM, only methods that at any reference point percentage differ from *Lstsq* with Kruskal–Wallis H test are shown along *Lstsq*.



(**a**) Normalized RMSE (**b**) Process time

Figure A5. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Census* dataset. For MLM, variants of *sk_rrSVD_XX* that differ statistically from *Cholesky decomposition* are marked with *.



(**a**) Normalized RMSE (**b**) Process time

Figure A6. Normalized RMSE values (**a**) and process time in seconds taken to compute Equation (1) (**b**) as a function of reference point percentage for the *Mnist* dataset. For MLM, variants of *sk_rrSVD_XX* that differ statistically from *Cholesky decomposition* are marked with *.

Appendix B. Au_{38Q}

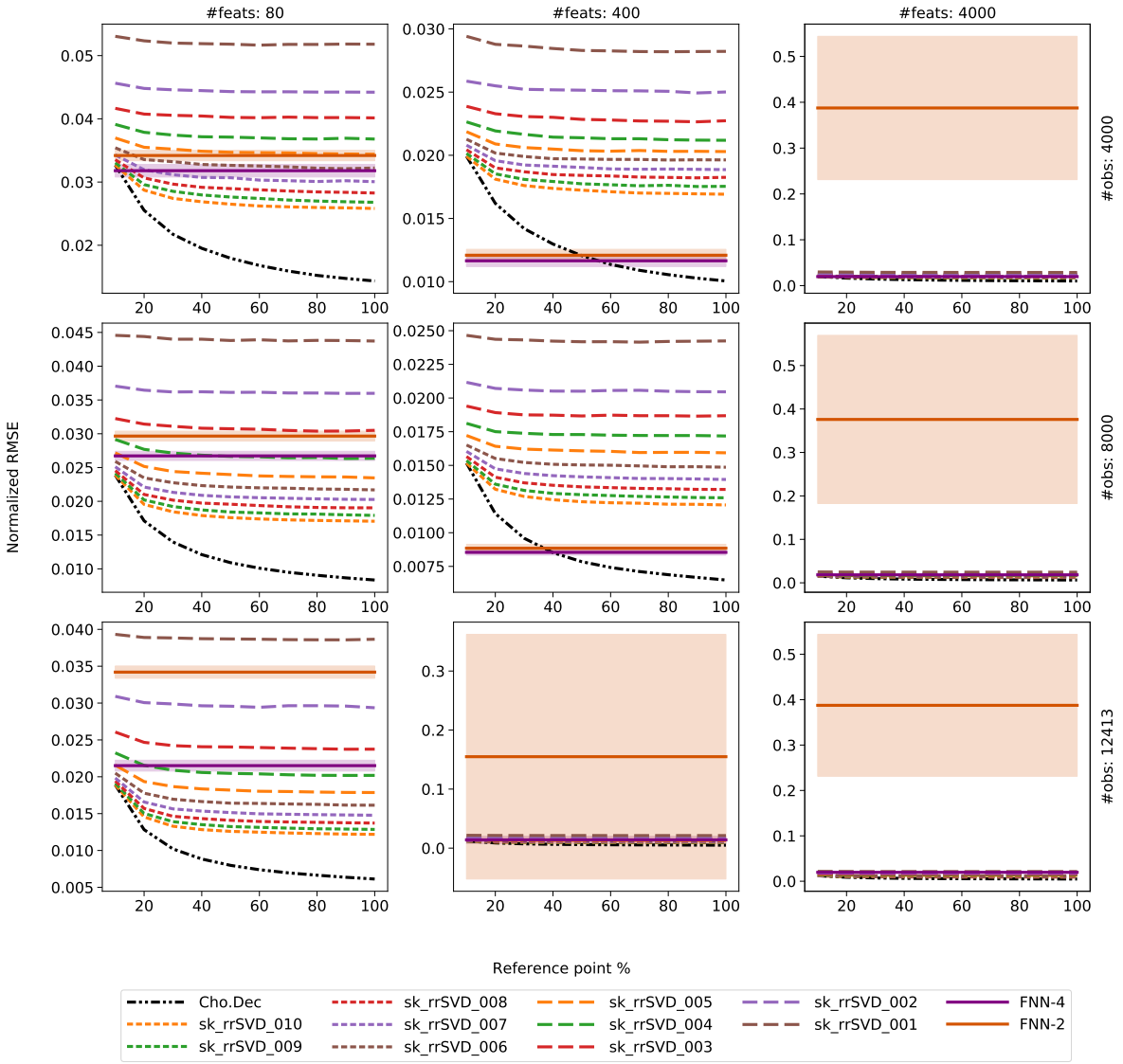


Figure A7. Normalized RMSE for Au_{38Q}. Each subfigure has reference point percentage on the horizontal axis and normalized RMSE on the vertical axis. The top row corresponds to dataset variants with 4000 observations, middle row represents variants with 8000 observations and the bottom row is for the full dataset. The left column represents dataset variants with 80 features, middle represents 400 features and the right column is for the full feature set. Please note that the Y–axis is different in each column.

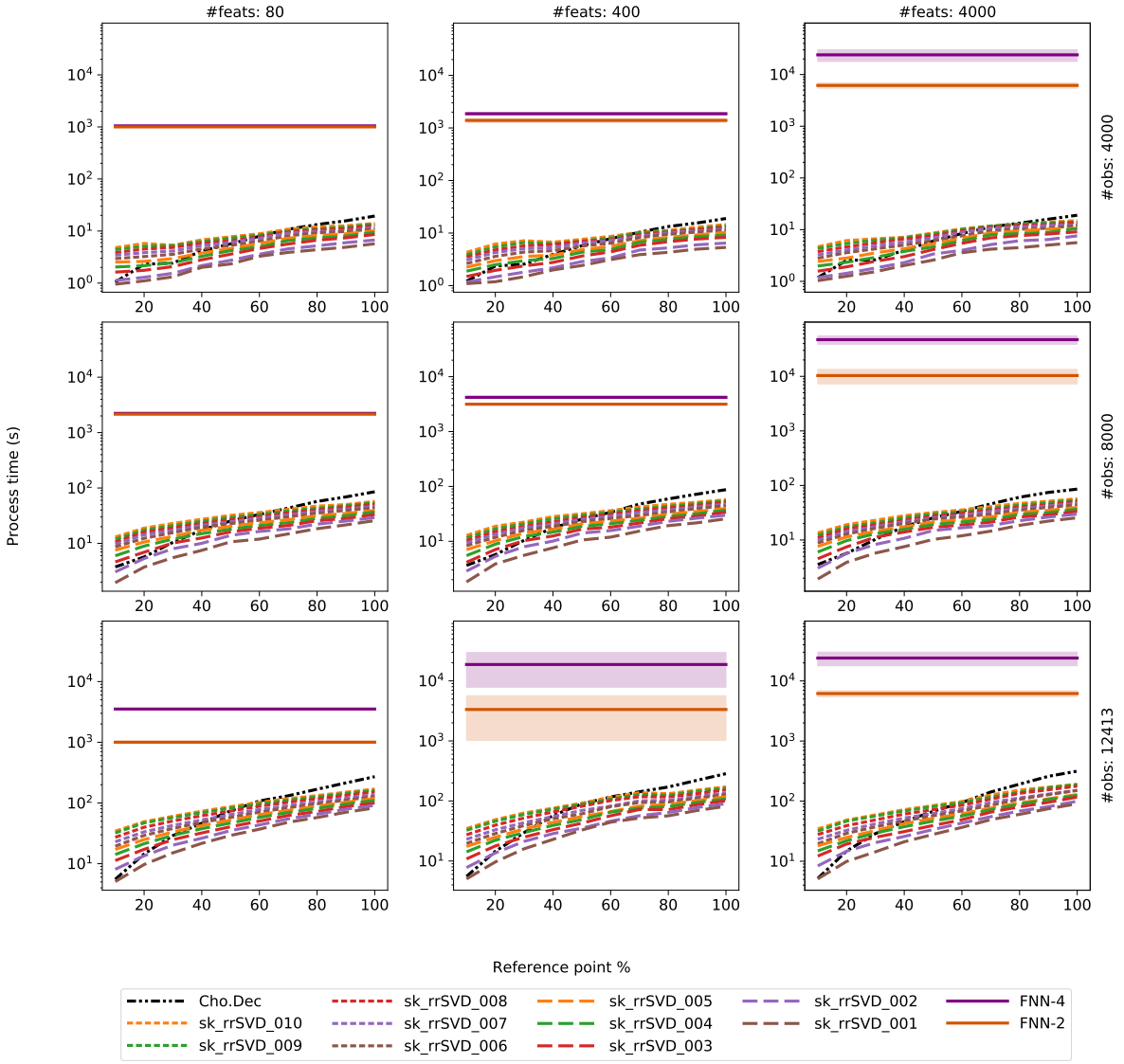


Figure A8. Process time for Au_{38Q}. Each subfigure has reference point percentage on the horizontal axis and normalized RMSE on the vertical axis. The top row corresponds to dataset variants with 4000 observations, middle row represents variants with 8000 observations and the bottom row is for the full dataset. The left column represents dataset variants with 80 features, middle represents 400 features and the right column is for the full feature set. Please note that the Y-axis is equalized in the subfigures.

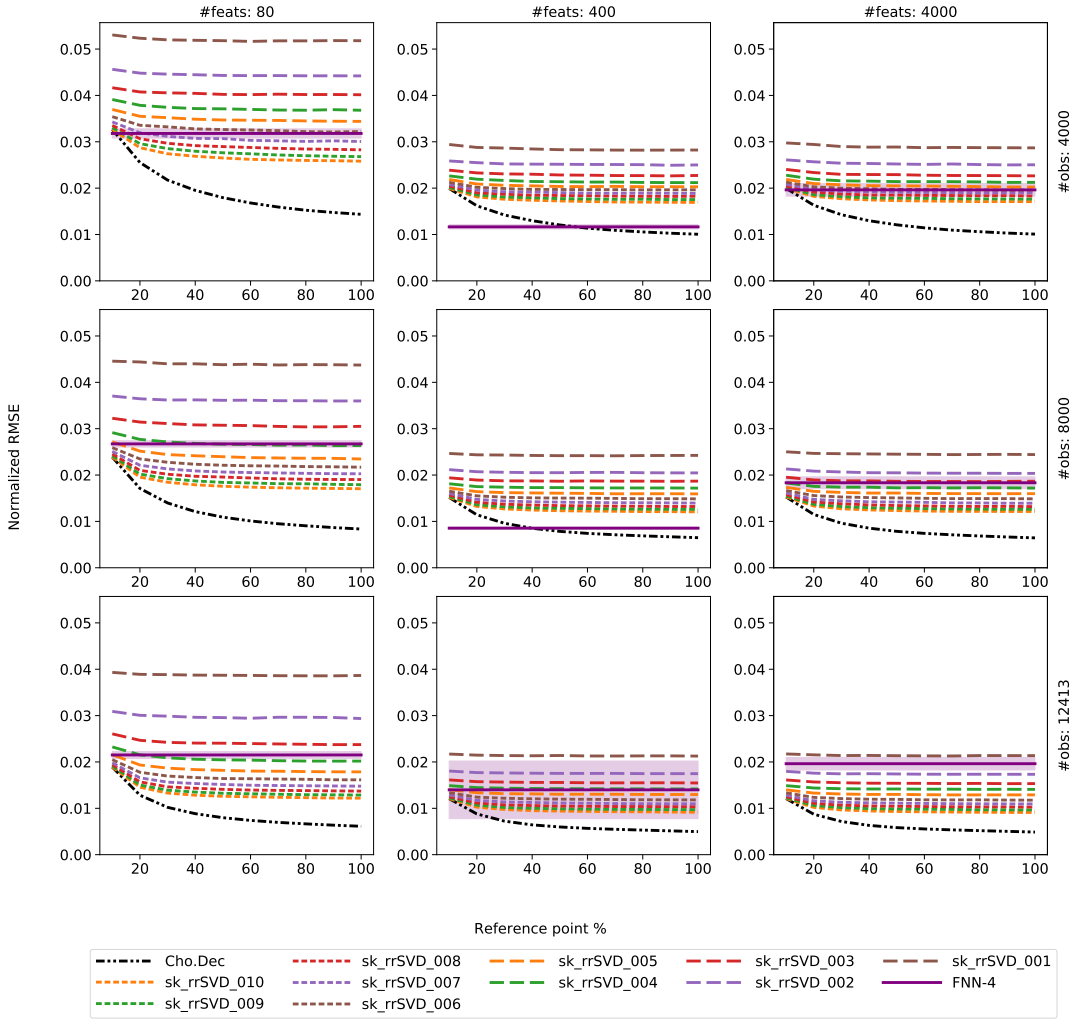


Figure A9. Same as Figure A7 but, for readability, without *FNN-2*. In addition, the Y–axis scale is same in each subfigure.

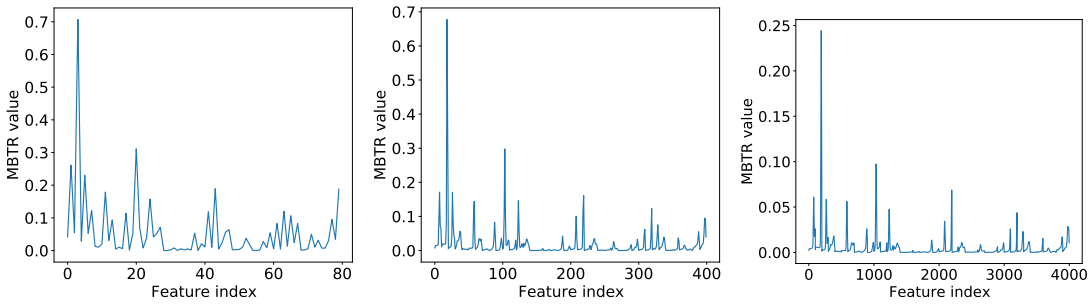
Appendix C. Au_{38Q} MBTR-K3 Dataset and Its Variants

The Au_{38Q} dataset has its roots in a hybrid nanostructure reported by Qian et al. [44]. The Au_{38Q} is a monolayer ligand protected nanoparticle, or a hybrid nanoparticle that is from here on referred to as a nanostructure (referring more to the physical structure than to the scientific concept, also to avoid confusion with the machine learning meaning of a cluster). The nanostructure was studied with molecular dynamics simulations under heating by Juarez-Mosqueda et al. [33] which produced a trajectory file (a data format containing one or more nanostructure configurations and metadata) for Au_{38Q} that contained 12,413 nanostructures formed during a molecular dynamics simulation from a start point to an end point. Each nanostructure in the trajectory is akin to a log of a time step during the molecular dynamics simulation. Each nanostructure is also paired with a value called potential energy, which can be used to describe the stability of a nanostructure. The potential energy was calculated by Juarez-Mosqueda et al. [33] using a density functional theory (DFT) calculator GPAW. However, a raw nanostructure configuration (the cartesian coordinates of the atoms) is problematic as it is for machine learning since there is no way to deterministically order the atoms so that the order in which they are given has no effect on the machine learning model. A set of cartesian coordinates are also not translationally and rotationally invariant, which is a major issue since the orientation and the exact location of a nanostructure are not allowed to have an effect. To solve this problem one has to use a concept called a descriptor. A descriptor is very literal in its function. We opted to use one called many

body tensor representation (MBTR) [45] which combines several descriptors to form its own. The use of MBTR in the context of nanostructures is also discussed in [3,45]. In our case, the description of a nanostructure is a vector, computed using a package known as Dscribe [46] (version 0.4.0). In short, the input values in our Au_{38Q} datasets are the description vectors and the output values are the potential energy values of each nanostructure.

We opted to use a setting denoted as $K = 3$ which computes a distribution of angles between groups of three elements and then concatenates those distributions to form a description vector. The Au_{38Q} is formed of four elements, hydrogen (H), carbon (C), sulfur (S) and gold (Au). The name Au_{38Q} refers to the nanostructure having 38 gold atoms as its core. For example, one of the distributions is formed from H-Au-C, meaning that the angle between vectors formed from H-Au and Au-C are computed. Repeated angles are of no interest, in our case there are 40 different three element angle groups [H-H-H, H-H-C, ..., Au-S-Au, Au-Au-Au] that are calculated to form the MBTR $K = 3$ description vector. We used mainly the default settings of the MBTR in Dscribe: l2 normalization (the length of a description vector was normalized to 1), cosine was used as the angle calculation function, measurement grid was set as $[-1, 1]$ and σ was 0.1. We used exponential weighting function with a scale of 0.5 and a cutoff of 1×10^{-3}. As we were describing a single nanostructure in a vacuum at a time, the descriptor is not periodic.

We mentioned that there are variants of Au_{38Q} dataset. In total we used nine variants by having two parameters with three values in each. One of the varied parameters was the number of features through the grid accuracy parameter N for the MBTR, which defines the number of points in a single angle distribution component. In other words, a grid accuracy parameter $N = 100$ results in 40 vectors of 100 length to form a description vector with 4000 features. The grid accuracy parameters that we used were $N \in [2, 10, 100]$. The number two is the lowest that can be used since a single number is not enough to describe a distribution. The other varied parameter was the number of observations. The full variant is denoted as 12 k, even though it uses the full 12,413 observations. Other two variants are 4 k and 8 k, which use 4000 observations and 8000 observations, respectively. The subsets of observations are taken from the full set with the use of the reference point selection algorithm RS-maximin. Since RS-maximin is deterministic, the first 4000 observations in 8 k are the same as all the observations in the 4 k subset. We denoted the variants of Au_{38Q} as AuNx-yk in the tables, where $x \in [2, 10, 100]$ and $y \in [4, 8, 12]$. As Au_{38Q} is not a commonly used dataset we included a figure showing the mean of all 12,413 MBTR description vectors with the three grid accuracy values in Figure A10 to aid the reader in understanding the way that a description vector might look like. The data files are available at 10.5281/zenodo.4268064.



(**a**) $N = 2$ (**b**) $N = 10$ (**c**) $N = 100$

Figure A10. Means of 12,413 MBTR-K3 descriptors for grid accuracy N values 2 (**a**), 10 (**b**) and 100 (**c**). Features represent N consecutive numbers representing the angle distribution measured between groups of three atoms. The nanostructure Au_{38Q} is composed of four elements, hydrogen (H), carbon (C), sulfur (S) and gold (Au). The first N features are H-H-H angles, second N are H-H-C and so on. Ending with Au-Au-Au angle. There are no repeated angles and so there are a total of 40 distributions laid consecutively to form the resulting MBTR-K3 vector.

References

1. De Souza Junior, A.H.; Corona, F.; Miche, Y.; Lendasse, A.; Barreto, G.A.; Simula, O. Minimal Learning Machine: A New Distance-Based Method for supervised Learning. In *International Work-Conference on Artificial Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 408–416.
2. De Souza Junior, A.H.; Corona, F.; Barreto, G.A.; Miche, Y.; Lendasse, A. Minimal Learning Machine: A novel supervised distance-based approach for regression and classification. *Neurocomputing* **2015**, *164*, 34–44. [CrossRef]
3. Pihlajamäki, A.; Hämäläinen, J.; Linja, J.; Nieminen, P.; Malola, S.; Kärkkäinen, T.; Häkkinen, H. Monte Carlo Simulations of Au38(SCH3)24 Nanocluster Using Distance-Based Machine Learning Methods. *J. Phys. Chem.* **2020**, *124*, 23. [CrossRef]
4. Marinho, L.B.; Almeida, J.S.; Souza, J.W.M.; Albuquerque, V.H.C.; Rebouças Filho, P.P. A novel mobile robot localization approach based on topological maps using classification with reject option in omnidirectional images. *Expert Syst. Appl.* **2017**, *72*, 1–17. [CrossRef]
5. Coelho, D.N.; Barreto, G.A.; Medeiros, C.M.S.; Santos, J.D.A. Performance comparison of classifiers in the detection of short circuit incipient fault in a three-phase induction motor. In Proceedings of the 2014 IEEE Symposium on Computational Intelligence for Engineering Solutions (CIES), Orlando, FL, USA, 9–12 December 2014; pp. 42–48.
6. Pekalska, E.; Duin, R.P. Automatic pattern recognition by similarity representations. *Electron. Lett.* **2001**, *37*, 159–160. [CrossRef]
7. Pekalska, E.; Paclik, P.; Duin, R.P. A generalized kernel approach to dissimilarity-based classification. *J. Mach. Learn. Res.* **2001**, *2*, 175–211.
8. Balcan, M.F.; Blum, A.; Srebro, N. A theory of learning with similarity functions. *Mach. Learn.* **2008**, *72*, 89–112. [CrossRef]
9. Zerzucha, P.; Daszykowski, M.; Walczak, B. Dissimilarity partial least squares applied to non-linear modeling problems. *Chemom. Intell. Lab. Syst.* **2012**, *110*, 156–162. [CrossRef]
10. Sanchez, J.D.; Rêgo, L.C.; Ospina, R. Prediction by Empirical Similarity via Categorical Regressors. *Mach. Learn. Knowl. Extr.* **2019**, *1*, 38. [CrossRef]
11. Kärkkäinen, T. Extreme minimal learning machine: Ridge regression with distance-based basis. *Neurocomputing* **2019**, *342*, 33–48. [CrossRef]
12. Hämäläinen, J.; Alencar, A.S.; Kärkkäinen, T.; Mattos, C.L.; Júnior, A.H.S.; Gomes, J.P. Minimal Learning Machine: Theoretical Results and Clustering-Based Reference Point Selection. *arXiv* **2019**, arXiv:1909.09978v2. To appear.
13. Florêncio, J.A.; Oliveira, S.A.; Gomes, J.P.; Neto, A.R.R. A new perspective for Minimal Learning Machines: A lightweight approach. *Neurocomputing* **2020**, *401*, 308–319. [CrossRef]
14. Hämäläinen, J.; Kärkkäinen, T. Newton's Method for Minimal Learning Machine. In *Computational Sciences and Artificial Intelligence in Industry—New Digital Technologies for Solving Future Societal and Economical Challenges*; Springer Nature: Cham, Switzerland, 2020.
15. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
16. Dias, M.L.D.; de Souza, L.S.; da Rocha Neto, A.R.; de Souza Junior, A.H. Opposite neighborhood: A new method to select reference points of minimal learning machines. In Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning—ESANN, Bruges, Belgium, 25–27 April 2018; pp. 201–206.
17. Florêncio, J.A.V.; Dias, M.L.D.; da Rocha Neto, A.R.; de Souza Júnior, A.H. *A Fuzzy C-Means-Based Approach for Selecting Reference Points in Minimal Learning Machines*; Barreto, G.A., Coelho, R., Eds.; Springer: Cham, Switzerland, 2018; pp. 398–407.
18. Mesquita, D.P.P.; Gomes, J.P.P.; Souza Junior, A.H. Ensemble of Efficient Minimal Learning Machines for Classification and Regression. *Neural Process. Lett.* **2017**, *46*, 751–766. [CrossRef]
19. Grigorievskiy, A.; Miche, Y.; Käpylä, M.; Lendasse, A. Singular value decomposition update and its application to (Inc)-OP-ELM. *Neurocomputing* **2016**, *174*, 99–108. [CrossRef]

20. Martinsson, P.G.; Rokhlin, V.; Tygert, M. A randomized algorithm for the decomposition of matrices. *Appl. Comput. Harmon. Anal.* **2011**, *30*, 47–68. [CrossRef]
21. Shabat, G.; Shmueli, Y.; Aizenbud, Y.; Averbuch, A. Randomized LU decomposition. *Appl. Comput. Harmon. Anal.* **2018**, *44*, 246–272. [CrossRef]
22. Abdelfattah, S.; Haidar, A.; Tomov, S.; Dongarra, J. Analysis and Design Techniques towards High-Performance and Energy-Efficient Dense Linear Solvers on GPUs. *IEEE Trans. Parallel Distrib. Syst.* **2018**, *29*, 2700–2712. [CrossRef]
23. Gonzalez, T.F. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.* **1985**, *38*, 293–306. [CrossRef]
24. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: A new learning scheme of feedforward neural networks. In Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 25–29 July 2004; pp. 985–990.
25. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
26. Oliphant, T.E. *A Guide to NumPy*; Trelgol Publishing: USA, 2006.
27. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv* **2019**, arXiv:1907.10121.
28. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
29. Stewart, G.W. On the Early History of the Singular Value Decomposition. *SIAM Rev.* **1993**, *35*, 551–566. [CrossRef]
30. Tikhonov, A.N.; Arsenin, V.J. *Solution of Ill-Posed Problems*; Winston&Sons: New York, NY, USA, 1977.
31. Halko, N.; Martinsson, P.G.; Tropp, J.A. Finding Structure with Randomness: Stochastic Algorithms for Constructing Approximate matrix Decompositions. *ACM Tech. Rep.* **2009**. [CrossRef]
32. Kruskal, W.H.; Wallis, W.A. Use of ranks in one-criterion variance analysis. *J. Am. Stat. Assoc.* **1952**, *47*, 583–621. [CrossRef]
33. Juarez-Mosqueda, R.; Sami, M.; Hannu, H. Ab initio molecular dynamics studies of Au38(SR)24 isomers under heating. *Eur. Phys. J.* **2019**, *73*. [CrossRef]
34. Dua, D.; Graff, C. UCI Machine Learning Repository. California, USA 2017. Available online: http://archive.ics.uci.edu/ml (accessed on 13 November 2020).
35. Torgo, L. *Airplane Companies Stocks*; Faculdade de Ciências da Universidade do Porto: Porto, Portugal, 1991.
36. University of Toronto. *Delve Datasets*; University of Toronto: Toronto, ON, Cananda, 1996.
37. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2323. [CrossRef]
38. Khamparia, A.; Singh, K.M. A systematic review on deep learning architectures and applications. *Expert Syst.* **2019**, *36*, e12400. [CrossRef]
39. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *arXiv* **2015**, arXiv:1603.04467.
40. Emmert-Streib, F.; Dehmer, M. Understanding Statistical Hypothesis Testing: The Logic of Statistical Inference. *Mach. Learn. Knowl. Extr.* **2019**, *1*, 54. [CrossRef]
41. Zhou, Z.H.; Feng, J. Deep forest. *Natl. Sci. Rev.* **2018**, *6*, 74–86. [CrossRef]
42. Li, H.B.; Huang, T.Z.; Zhang, Y.; Liu, X.P.; Gu, T.X. Chebyshev-type methods and preconditioning techniques. *Appl. Math. Comput.* **2011**, *218*, 260–270. [CrossRef]
43. Hadjidimos, A. Accelerated overrelaxation method. *Math. Comput.* **1978**, *32*, 149–157. [CrossRef]
44. Qian, H.; Eckenhoff, W.T.; Zhu, Y.; Pintauer, T.; Jin, R. Total Structure Determination of Thiolate-Protected Au38 Nanoparticles. *J. Am. Chem. Soc.* **2010**, *132*, 8280–8281. [CrossRef] [PubMed]

45. Huo, H.; Rupp, M. Unified Representation of Molecules and Crystals for Machine Learning. *arXiv* **2017**, arXiv:1704.06439.
46. Himanen, L.; Jäger, M.O.J.; Morooka, E.V.; Federici Canova, F.; Ranawat, Y.S.; Gao, D.Z.; Rinke, P.; Foster, A.S. DScribe: Library of descriptors for machine learning in materials science. *Comput. Phys. Commun.* **2020**, *247*, 106949. [CrossRef]