Spring 2016

Extreme Learning Machines: novel extensions and application to Big Data

Anton Akusok
University of Iowa

EXTREME LEARNING MACHINES: NOVEL EXTENSIONS AND APPLICATION TO BIG DATA

by

Anton Akusok

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Industrial Engineering
in the Graduate College of
The University of Iowa

May 2016

Thesis Supervisor: Associate Professor Amaury Lendasse

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Anton Akusok

has been approved by the Examining Committee for the thesis requirement for the Doctor of Philosophy degree in Industrial Engineering at the May 2016 graduation.

Thesis committee: _____
Amaury Lendasse, Thesis Supervisor

_____
Stephen Baek

_____
Andrew Kusiak

_____
Gregory R. Carmichael

_____
Ching-Long Lin

_____
Yong Chen

ACKNOWLEDGEMENTS

I would like to acknowledge Associate Prof. Amaury Lendasse for his patience and his invaluable support in all research, administrative and practical matters. I am happy to have such a nice professor, good supervisor and a great person by my side. With him, research is always fun and easy.

I also want to specifically acknowledge Dr. Yoan Miche with whom I wrote so many papers, and who always supported me with a good advice from my first day in the academia.

I am giving a special thanks for Dr. Francesco Corona for his Italian humour, and teaching me how to brew a good coffee of course!

I want to express my gratitude to D.Sc. Kaj-Mikael Björk and Magnus Westerlund for providing me with opportunities for inter-university collaboration. This is a great experience.

I want to acknowledge my co-authors: Alberto Guillén, Alexander Grigorievskiy, Alexandre Savio, Andrey Gritsenko, Bo He, Colin Swaney, David Veganzones, Dusan Sovilj, Emil Eirola, Eric Séverin, Erik Cambria, Francesco Corona, Guang-Bin Huang, Jozsef Hegedus, Juha Karhunen, Maarit Mantere, Maite Termenon, Manuel Graña, Mark van Heeswijk, Olli Simula, Paula Lauren, Philippe du Jardin, Rui Nian, Stephen Baek and Tatiana Chistiakova.

ABSTRACT

Extreme Learning Machine (ELM) is a recently discovered way of training Single Layer Feed-forward Neural Networks with an explicitly given solution, which exists because the input weights and biases are generated randomly and never change. The method in general achieves performance comparable to Error Back-Propagation, but the training time is up to 5 orders of magnitude smaller. Despite a random initialization, the regularization procedures explained in the thesis ensure consistently good results.

While the general methodology of ELMs is well developed, the sheer speed of the method enables its un-typical usage for state-of-the-art techniques based on repetitive model re-training and re-evaluation. Three of such techniques are explained in the third chapter: a way of visualizing high-dimensional data onto a provided fixed set of visualization points, an approach for detecting samples in a dataset with incorrect labels (mistakenly assigned, mistyped or a low confidence), and a way of computing confidence intervals for ELM predictions. All three methods prove useful, and allow even more applications in the future.

ELM method is a promising basis for dealing with Big Data, because it naturally deals with the problem of large data size. An adaptation of ELM to Big Data problems, and a corresponding toolbox (published and freely available) are described in chapter 4. An adaptation includes an iterative solution of ELM which satisfies a limited computer memory constraints and allows for a convenient parallelization.

Other tools are GPU-accelerated computations and support for a convenient huge data storage format. The chapter also provides two real-world examples of dealing with Big Data using ELMs, which present other problems of Big Data such as veracity and velocity, and solutions to them in the particular problem context.

PUBLIC ABSTRACT

Real world tasks can often be written mathematically as maximizing a number (like income) or minimizing another number (like expenses). Some of these tasks have exact mathematical solution. Exact solution of other tasks is unknown, but there are multiple correct examples. Machine Learning can learn approximate solution from examples, and this thesis is about state-of-the-art methods in Machine Learning.

Consider speech recognition problem: everyone of us can hear a word in a native language and easily scribe it with letters, but nobody can write down a mathematical algorithm of how he or she does that. Machine Learning has a method called Artificial Neural Network, which mimic the way how a brain works, and can learn from a set of prepared data (like sounds with the corresponding letters in the previous example) how to solve a problem without an exact algorithm – not perfectly, but good enough.

This thesis tells about Extreme Learning Machines, a kind of Artificial Neural Network that runs very fast on a computer. It presents two directions of research. First, the Extreme Learning Machine is trained not once (like Machine Learning methods normally do) but millions of times, each time a bit differently. That way helps learn new things about the data, like are there any mistakes in the labelling, or what is the best way of showing a complex data on a piece of paper.

Another direction of research is how to make Extreme Learning Machines even faster, or run with even more data, which is the same thing. The method is

improved by a better algorithm and a good program code. This allows running ELM on computers with small amount of memory, use graphics card to do the computations faster, read very large data piece-by-piece from a hard drive, and compute a large ELM in parts on many different computers at the same time to get results faster. An ELM method with all these improvements is published as a freely downloadable computer program, so anyone can use a fast ELM even if he/she is not an expert in Artificial Neural Networks. Also, this chapter presents two examples of solving large real-world problems with ELMs, where the main difficulty is not the ELM itself but the strange provided data and special requirements for a good solution.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS

N	Number of data samples
d	Number of input features
c	Number of output features or classes
L	Number of hidden neurons
M	Number of iterations (in MD-ELM)
L^1	$\|\|x\|\|_1$ norm
L^2	$\|\|x\|\|_2$ norm
$\mathbf{X}_{[N\times d]}$	Input data in matrix form
$\mathbf{T}_{[N\times c]}$	Target (true) outputs in matrix form
$\mathbf{Y}_{[N\times c]}$	Predicted outputs in matrix form
$\mathbf{W}_{[d\times L]}$	Input-to-hidden layer projection matrix in ELM
$\mathbf{b}_{[1\times L]}$	Bias vector in ELM
$\mathbf{H}_{[N\times L]}$	Hidden layer output of ELM in matrix form
$\boldsymbol{\beta}_{[L\times c]}$	Hidden-to-output layer projection matrix in ELM
$\mathbf{e}$	Vector of ones of a corresponding shape
$\phi()$	Activation function of a hidden neuron

LIST OF ABBREVIATIONS

BLAS . . . . Basic Linear Algebra Subprograms
BoVW . . . . Bag-or-Visual-Words
CV . . . . . Cross-Validation
ELM . . . . . Extreme Learning Machine
ELMVIS . . . Extreme Learning Machine-based Visualization
GP . . . . . Gaussian Processes
GPU . . . . . Graphics Processing Unit
HDF5 . . . . Fast and scalable file storage format
HP-ELM . . High Performance toolbox for Extreme Learning Machines
LARS . . . . Least Angle Regression
LOO . . . . . Leave-One-Out
MD-ELM . . Mislabeled samples Detection with Extreme Learning Machines
MDS . . . . . Multidimensional Scaling
MLP . . . . . Multilayer Perceptron
MNIST . . . Popular benchmark dataset of handwritten digits
MRSR . . . . Multi-Response Sparse Regression
MSE . . . . . Mean Squared Error
NeRV . . . . Neighbor Retrieval Visualizer
OP-ELM . . . Optimally Pruned Extreme Learning Machine
PCA . . . . . Principal Component Analysis
PRESS . . . . Predicted Residual Sum of Squares
RBF . . . . . Radial Basis Function
RMSE . . . . Root Mean Squared Error
SIFT . . . . . Scale Invariant Feature Transform
SLFN . . . . . Single Layer Feed-forward Neural network
SOM . . . . . Self-Organizing Maps
SVM . . . . . Support Vector Machine
SVR . . . . . Support Vector Regression
TROP-ELM Tikhonov Regularized and Optimally Pruned ELM
UCI . . . . . A popular online source of benchmark data

CHAPTER 1
INTRODUCTION

1.1 Aims and Scope

The purpose of this thesis is to advance the state-of-the-art knowledge in Extreme Learning Machines [Co13] (ELM), by improving the method itself and developing new technologies based on that method. The ELM is a recently discovered [HZS04] way of training Single Layer Feed-forward Neural networks [Hay98] (SLFN). Its performance in general is similar to that of a Multi-Layer Perceptron trained with Error Back-Propagation, but the training time is up to 5 orders of magnitude smaller.

The second chapter of the thesis gives an overview of the theory behind ELMs. It summarizes the standard methodology, including the essential tips and tricks which counter the negative effects of random initialization, and for example, make ELMs memory requirements invariant to the number of training samples.

The third chapter introduces extensions of ELMs to non-typical tasks in the field of neural networks. The basis of these extensions is an extremely fast training speed of the method. During the same time, say 5 minutes, it is possible to train a single Multilayer Perceptron or a Support Vector Machine, but more than 100,000 ELMs. While the training time measured in milliseconds is not very important for human beings who operate at much slower pace, it enables exciting new approaches that are based on extensive re-training of a model, or require training millions of different models. These approaches would be infeasible before with other nonlinear

models, or feasible with a linear model but provide less interesting results due to the linearity.

The last chapter of the thesis shows an extension of ELM for Big Data, and huge datasets in general. Its goal is to push the boundaries of neural networks as far as possible with the current hardware and software technology, increasing size of feasible models on a normal workstation hardware to tens of thousands of hidden neurons and billions of training samples; relaxing computer memory constraints, allowing an easy parallelization and utilization of accelerators for faster model training. Other examples of this part show applications of ELMs for addressing real-world Big Data tasks, which cannot be dealt with by only brute-force computation, but require building a whole methodology around the problem in question for finding a solution with the desired properties.

1.2 Author's Contributions

Author made the following contributions to the five journal articles, two conference papers, and the thesis itself:

Section 2.1 of the second chapter is based on the methodology from an author's paper [ABML15]. It summarizes the algorithm, solution and notations of an original ELM model, which are used throughout the thesis. Other sections of the second chapter are based on prior works, but cannot be omitted due to their importance in the field of ELMs.

The third chapter of the thesis presents four original author's works. The

first one applies ELM as a fast and robust nonlinear cost function data visualization, published in a joint paper [Co13]. The second one is a mathematical and algorithmic development of the first one, speeding the approach by four orders of magnitude and applying it to large data [AMB+16c, AMB+16a]. The third one uses ELM model as a sample quality indicator for finding originally mislabeled samples in a dataset [ADY+14, AVM+15]. The fourth one presents a very important addition to ELM predictions in a form of confidence intervals of such predictions [AGM+16, AMB+16b], demanded in many application areas.

The fourth chapter of the thesis is based on three papers. The first one is a high-performance ELM toolbox [ABML15], which presents the methodology, its freely available implementation, and comparison results with other articles in one paper. This work is one of the major contributions of an author to the field of Extreme Learning Machines, because it aims at popularizing the method among other researches and making advanced features of ELMs easily accessible by anyone for any application. Also it is tested on a dataset with half a billion training samples and 19,000 hidden neurons, showing that even such a huge model can be trained in a reasonable time on a single workstation. Second paper [AAAY13, AMK+15] presents an application of ELM to a real Big Data problem — a classification of websites based on their image content. The paper presents not only ELM but a whole methodology for dealing with the problem under specific constraints, which is one of a major differences between utilizing Big Data and normal large-scale datasets. The last paper presents a methodology for detecting malware files from a provided large set

of hashes, which is another application of ELMs to Big Data problems. The first two papers are fully author's work, while the third one is done in collaboration with Yoan Miche.

1.3 List of Publications

Included in the thesis (journal papers)

[ABML15] Anton Akusok, Kaj-Mikael Björk, Yoan Miche, and Amaury Lendasse, *High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications*, IEEE Access **3** (2015), 1011–1025.

[AGM+16] Anton Akusok, Andrey Gritsenko, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Paula Lauren, and Amaury Lendasse, *Adding Reliability to ELM Predictions by Confidence Intervals*, Neurocomputing (submitted 2016).

[AMB+16] Anton Akusok, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Paula Lauren, and Amaury Lendasse, *ELMVIS+: Fast Nonlinear Visualization Technique based on Cosine Distance and Extreme Learning Machines*, Neurocomputing (forthcoming 2016).

[AMH+14] Anton Akusok, Yoan Miche, Jozsef Hegedus, Rui Nian, and Amaury Lendasse, *A Two-Stage Methodology Using K-NN and False-Positive Minimizing ELM for Nominal Data Classification*, Cognitive Computation **6** (2014), no. 3, 432–445.

[AMK+15] Anton Akusok, Yoan Miche, Juha Karhunen, Kaj-Mikael Björk, Rui Nian, and Amaury Lendasse, *Arbitrary Category Classification of Websites Based on Image Content*, IEEE Computational Intelligence Magazine **10** (2015), no. 2, 30–41.

[AVM+15] Anton Akusok, David Veganzones, Yoan Miche, Kaj-Mikael Björk, Philippe du Jardin, Eric Séverin, and Amaury Lendasse, *MD-ELM: Originally Mislabeled Samples Detection using OP-ELM Model*, Neurocomputing **159** (2015), 242–250.

[Co13] Eric Cambria and others, *Extreme Learning Machines [Trends & Controversies]*, IEEE Intelligent Systems **28** (Nov.-Dec. 2013), no. 6, 30–59.

Included in the thesis (conference papers)

[AAAY13] Anton Akusok, Alexander Grigorievskiy, Amaury Lendasse, and Yoan Miche, *Image-based Classification of Websites*, Machine Learning Reports 02/2013 (Saarbrücken, Germany) (Thomas Villmann and Frank-Michael Schleif, eds.), Machine Learning Reports, vol. ISSN: 18, Workshop of the GI-Fachgruppe Neuronale Netze and the German Neural Networks Society in connection to GCPR 2013, September 2013, pp. 25–34.

[ADY+14] Anton Akusok, David Veganzones, Yoan Miche, Eric Séverin, and Amaury Lendasse, *Finding Originally Mislabels with MD-ELM*, Proceedings of ESANN2014: 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, i6doc, Bruges, Belgium, 23-25 April 2014, pp. 689–694.

[AMB+16a] Anton Akusok, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Paula Lauren, and Amaury Lendasse, *ELMVIS+: Improved Nonlinear Visualization Technique Using Cosine Distance and Extreme Learning Machines*, Proceedings of ELM-2015 Volume 2: Theory, Algorithms and Applications (II) (Jiuwen Cao, Kezhi Mao, Jonathan Wu, and Amaury Lendasse, eds.), Springer International Publishing, Cham, 2016, pp. 357–369.

[AMB+16b] ———, *Evaluating Confidence Intervals for ELM Predictions*, Proceedings of ELM-2015 Volume 2: Theory, Algorithms and Applications (II) (Jiuwen Cao, Kezhi Mao, Jonathan Wu, and Amaury Lendasse, eds.), Springer International Publishing, Cham, 2016, pp. 413–422.

Not included in the thesis

[CAK+15] Colin Swaney, Anton Akusok, Kaj-Mikael Björk, Yoan Miche, and Amaury Lendasse, *Efficient Skin Segmentation via Neural Networks: HP-ELM and BD-SOM*, INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015 **53** (2015), 400–409.

[EGA+15] Emil Eirola, Andrey Gritsenko, Anton Akusok, Kaj-Mikael Björk, Yoan Miche, Dusan Sovilj, Rui Nian, Bo He, and Amaury Lendasse, *Extreme Learning Machines for Multiclass Classification: Refining Predictions with Gaussian Mixture Models*, Advances in Computational Intelligence (Ignacio Rojas, Gonzalo Joya, and Andreu Catala, eds.), Lecture Notes in Computer Science, vol. 9095, Springer International Publishing, 2015, pp. 153–164 (English).

[GMA+14] Alexander Grigorievskiy, Maarit Mantere, Anton Akusok, Eirola Eirola, and Amaury Lendasse, *Forecasting the Outbursts of the Photometry Light Curve of Star V363 Lyr*, Proceedings of International Work Conference on Time Series Analysis (Ignacio Rojas Ruiz and Gonzalo Ruiz Garcia, eds.), vol. 1, June 2014, pp. 520–531.

[LAS+13] Amaury Lendasse, Anton Akusok, Olli Simula, Francesco Corona, Mark van Heeswijk, Emil Eirola, and Yoan Miche, *Extreme Learning Machine: A Robust Modeling Technique? Yes!*, Advances in Computational Intelligence (Ignacio Rojas, Gonzalo Joya, and Joan Gabestany, eds.), Lecture Notes in Computer Science, vol. 7902, Springer Berlin Heidelberg, 2013, pp. 17–35 (English).

[MAV+15] Yoan Miche, Anton Akusok, David Veganzones, Kaj-Mikael Björk, Eric Séverin, Philippe du Jardin, Maite Termenon, and Amaury Lendasse, *SOM-ELM—Self-Organized Clustering using ELM*, Neurocomputing **165** (2015), 238 – 254.

[MCA+12] Yoan Miche, Tatiana Chistiakova, Anton Akusok, Amaury Lendasse, Rui Nian, and Alberto Guillén, *Fast Variable Selection for Memetracker Phrases Time Series Prediction*, Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments (New York, NY, USA), PETRA '12, ACM, 2012, pp. 47:1–47:6.

[MOH+16] Yoan Miche, Ian Oliver, Silke Holtmanns, Anton Akusok, Amaury Lendasse, and Kaj-Mikael Björk, *On Mutual Information over Non-Euclidean Spaces, Data Mining and Data Privacy Levels*, Proceedings of ELM-2015 Volume 2: Theory, Algorithms and Applications (II) (Jiuwen Cao, Kezhi Mao, Jonathan Wu, and Amaury Lendasse, eds.), Springer International Publishing, Cham, 2016, pp. 371–383.

[SEM+16] Dusan Sovilj, Emil Eirola, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Anton Akusok, and Amaury Lendasse, *Extreme learning machine for missing data using multiple imputations*, Neurocomputing **174, Part A** (2016), 220 – 231.

CHAPTER 2
THEORY OF EXTREME LEARNING MACHINES

Extreme Learning Machines [HZS04, HZS06, HZDZ12, Co13] (ELM) as an important emergent Machine Learning techniques, are proposed for both "generalized" Single-Layer Feed-forward Networks (SLFNs) [HZS04, HZDZ12, HCS06, Hua15, Hua14] and multi layered feedforward networks [Hua15]. Unlike traditional learning theories and learning algorithms, ELM theories show that hidden neurons need not be tuned in learning and their parameters can be independent of the training data, but nevertheless ELMs have universal approximation and classification properties [HCS06, Hua15, Hua14]. In most cases, the ELM hidden neurons can be randomly generated, which means that all the parameters of the hidden neurons (e.g., the input weights and biases of additive neurons, the centres and the impact factors of RBF nodes, frequencies and the shift of Fourier series, etc) can be randomly generated and therefore also independent of the training data. Some related efforts had been attempted before [Whi89, Whi06, PPS94] with parts of SLFN generated randomly or taken from a subset of data samples [WMR92], however, they either lack proof of the universal approximation capability for fully randomized hidden neurons, or can be considered as specific cases of ELM [IP95]. ELM, consisting of a wide type of feed forward neural networks, is the first method [Hua15, Hua14] which can universally approximate any continuous function with almost any nonlinear and piecewise continuous hidden neurons.

A distinct property of ELM is the non-iterative linear solution for the output

weights, which is possible because there is no dependence between the input and output weights like in the Back-propagation [Hay98] training procedure. A non-iterative solution of ELMs provides a speedup of 5 orders of magnitude compared to Multilayer Perceptron [Ros58] (MLP) or 6 orders of magnitude compared to Support Vector Machines [CV95] (SVM), as shown in section 2.3.4.

ELM originally belongs to the set of regression methods [HZS04, HMZ+06]. The universal approximation property implies that an ELM can solve any regression problem with a desired accuracy, given that it has enough hidden neurons and training data to learn the parameters for all the hidden neurons. ELMs are also easily adapted for classification problems [HZDZ12]. For multiclass classification, the index of the output node with the highest output indicates the predicted label of input. Then the predicted class is assigned by the maximum output of an ELM. Multi-label classification [TK07] is handled similarly, but the predicted classes are assigned by all outputs, which are greater than some threshold value.

Extreme Learning Machines are well suited for dealing with Big Data [ZOT14] problems because their solution is so rapidly obtained and memory requirements do not grow with data size. Indeed, they are used for analyzing Big Data [AMH+14, AAAY13, AMK+15, HBKV15]. A GPU acceleration [vML+09, vMOL11] significantly speeds up the computations.

Extreme Learning Machines also benefit greatly from model structure selection and regularization, which reduces possible negative effects of random initialization and over-fitting. The methods include L^1 [MBJ+08, MSB+10] and L^2 [MvB+11]

regularization, as well as other methods [YME+13] like handling imbalance classification [ZHC13]. The problem is again the absence of ready to use toolboxes, which are focused on particular existing methods [MSB+10].

2.1 Basic Extreme Learning Machine

An ELM is a fast training method for SLFN networks (Figure 2.1). A SLFN has three layers of neurons, but the name *Single* comes from the only layer of nonlinear neurons in the model: the hidden layer. Input layer provides data features and performs no computations, while an output layer is linear without a transformation function or bias.

In the ELM method, input layer weights $\mathbf{W}$ and biases $\mathbf{b}$ are set randomly and never adjusted (random distribution of the weights is discussed in section 4.2.1). Because the input weights are fixed, the output weights $\boldsymbol{\beta}$ are independent of them (unlike in Back-propagation [Hay98] training method) and have a direct solution without iteration. For a linear output layer, such solution is also linear and very fast to compute.

Random input layer weights improve the generalization properties of the solution of a linear output layer, because they produce almost orthogonal (weakly correlated) hidden layer features. The solution of a linear system is always in a span of inputs. If the range of solution weights is limited, orthogonal inputs provide a larger solution space volume with these constrained weights. Small norms of the weights tend to make the system more stable and noise resistant as errors in input will not be amplified in the output of the linear system with smaller coefficients. Thus random hidden layer generates weakly correlated hidden layer features, which allow for a solution with a small norm and a good generalization performance.

A formal description of an ELM is the following. Consider a set of N distinct

training samples $(\mathbf{x}_i, \mathbf{t}_i),\ i \in [\![1, N]\!]$ with $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{t}_i \in \mathbb{R}^c$. Then a SLFN with L hidden neurons has the following output equation:

$$\sum_{j=1}^{L} \boldsymbol{\beta}_j \phi(\mathbf{w}_j \mathbf{x}_i + b_j),\ i \in [\![1, N]\!], \tag{2.1}$$

with ϕ being the activation function (a sigmoid function is a common choice, but other activation functions are possible including linear) [HZDZ12, Hua15, Hua14], $\mathbf{w}_i$ the input weights, b_i the biases and $\boldsymbol{\beta}_i$ the output weights.

The relation between inputs $\mathbf{x}_i$ of the network, target outputs $\mathbf{t}_i$ and estimated outputs $\mathbf{y}_i$ is:

$$\mathbf{y}_i = \sum_{j=1}^{L} \boldsymbol{\beta}_j \phi(\mathbf{w}_j \mathbf{x}_i + b_j) = \mathbf{t}_i + \epsilon_i,\ i \in [\![1, N]\!], \tag{2.2}$$

where ϵ is noise. Here the *noise* includes both random noise and dependency on variables not presented in the inputs $\mathbf{X}$.

Hidden Neurons

Hidden neurons transform the input data into a different representation. The transformation is done in two steps. First, the data is projected into the hidden layer using the input layer weights and biases. Second, the projected data is transformed. A nonlinear transformation function greatly increases the learning capabilities of an ELM, because it is the only place where a nonlinear part can be added in ELM method. After transformation, the data in the hidden layer representation $\mathbf{h}$ (see Figure 2.1) is used for finding output layer weights.

The hidden layer is not constrained to have only one type of transformation

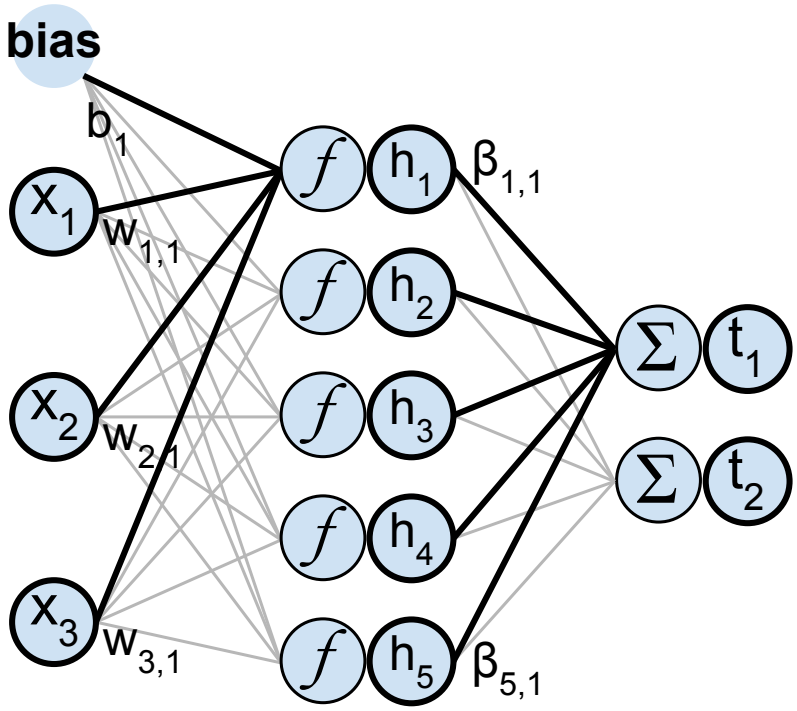


Figure 2.1: Computing the output of an SLFN (ELM) model.

function in neurons. Different functions can be used (sigmoid, hyperbolic tangent, threshold, etc.) [HZDZ12, Hua15, Hua14]. Some neurons may have no transformation function at all. They are linear neurons, and learn linear dependencies between data features and targets directly, without approximating them by a nonlinear function. Usually the number of linear neurons equals the number of data features, and each of these neurons copy the corresponding feature (by using an identity $\mathbf{W}$ and zero $\mathbf{b}$).

Another type of neurons commonly present in ELMs is the Radial Basis Function (RBF) neurons [CCG91]. They use distances to centroids as inputs to the hidden layer, instead of a linear projections. The nonlinear projection function is applied as usual. ELMs with RBF neurons compute predictions based on similar training data

samples, which helps solving tasks with a complex dependency between data features and targets. Any function (norm) of distances between samples and centroids can be used, for instance L^2, L^1 or L^∞ norms.

Matrix Form of ELMs

Practically, ELMs are often solved in a matrix form with a closed form solution. An implementation with matrices is easy to write and fast to run on computers. An ELM is written in a matrix form by gathering outputs of all hidden neurons into a matrix $\mathbf{H}$ as in equation 2.3. A graphical representation is shown in Figure 2.2. The matrix form of ELMs is used hereafter.

$$\mathbf{H} = \begin{bmatrix} \phi(\mathbf{w}_1\mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_L\mathbf{x}_1 + b_L) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{w}_1\mathbf{x}_N + b_1) & \cdots & \phi(\mathbf{w}_L\mathbf{x}_N + b_L) \end{bmatrix}, \tag{2.3}$$

$$\boldsymbol{\beta} = \left(\boldsymbol{\beta}_1^T \cdots \boldsymbol{\beta}_L^T\right)^T, \qquad \mathbf{T} = \left(\mathbf{t}_1^T \cdots \mathbf{t}_N^T\right)^T. \tag{2.4}$$

Although the ELM procedure include a training aspect, like other neural networks, the network structure itself is not noticeable in practice. Mathematically, there are only matrices describing the projection between the two linear spaces. Thus an ELM is viewed as two projections: input $\mathbf{XW}$ and output $\mathbf{H}\boldsymbol{\beta}$, with a (nonlinear) transformation between them $\mathbf{H} = \phi(\mathbf{XW} + \mathbf{e}^T\mathbf{b})$. The number of hidden neurons regulates the size of matrices $\mathbf{W}$, $\mathbf{H}$ and $\boldsymbol{\beta}$; but the network neurons are never treated separately.

With different types of hidden neurons, the first projection and transformation

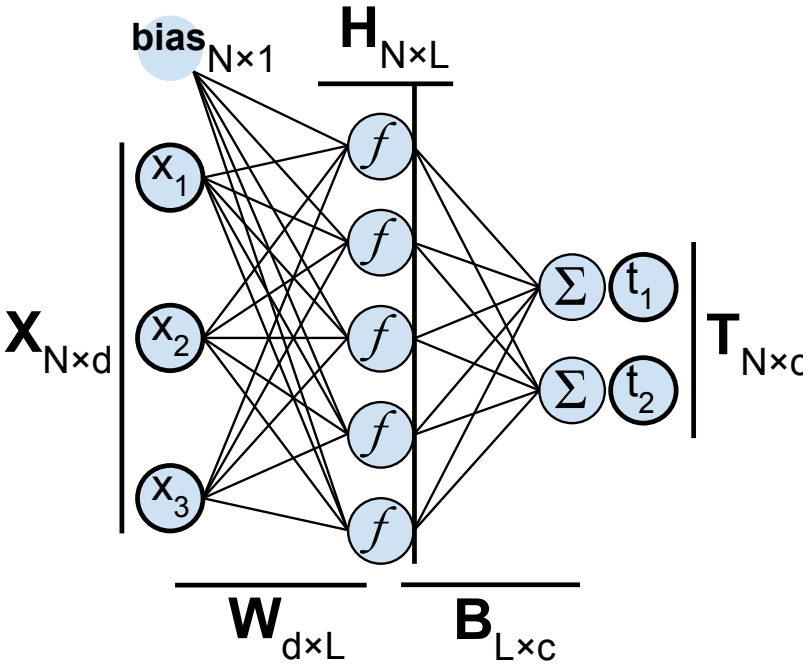


Figure 2.2: A matrix form of an ELM.

are performed independently for each type of neurons. Then the resulted sub-matrices $\mathbf{H}_1$ are concatenated along the second dimension. For two types of hidden neurons:

$$\mathbf{H} = [\mathbf{H}_1 \mid \mathbf{H}_2] = [\phi_1(\mathbf{X}\mathbf{W}_1 + \mathbf{e}^T\mathbf{b}_1) \mid \phi_2(\mathbf{X}\mathbf{W}_2 + \mathbf{e}^T\mathbf{b}_2)]. \tag{2.5}$$

Linear neurons are added into ELM by simply copying inputs into the hidden layer outputs:

$$\mathbf{H} = [\mathbf{H}_1 \mid \mathbf{H}_2 \mid \mathbf{X}] = [\phi_1(\mathbf{X}\mathbf{W}_1 + \mathbf{e}^T\mathbf{b}_1) \mid \phi_2(\mathbf{X}\mathbf{W}_2 + \mathbf{e}^T\mathbf{b}_2) \mid \mathbf{X}]. \tag{2.6}$$

Theoretical Guarantees of Extreme Learning Machines

With the previous notations, the following theorem 2.1 is proposed in [HZS06], which is the pillar of the ELM idea. The theorem states that with randomly initialized input weights and biases for the SLFN, and under the condition that the activation function is infinitely differentiable, then the hidden layer output matrix can be determined and will provide an approximation of the target values with arbitrary small noise ϵ.

Theorem 2.1. *Given any* $\epsilon > 0$ *and an activation function* $\phi : \mathbb{R} \mapsto \mathbb{R}$ *infinitely differentiable in any interval, there exists* $L < N$ *such that for* N *distinct samples* $(\mathbf{x}_i, \mathbf{t}_i), \mathbf{x}_i \in \mathbb{R}^d, \mathbf{t}_i \in \mathbb{R}^c$*, for any* $\mathbf{w}_j \in \mathbb{R}^d$ *and* $b_j \in \mathbb{R}$*,* $||\mathbf{H}_{[N\times L]}\boldsymbol{\beta}_{[L\times c]} - \mathbf{T}_{[N\times c]}|| < \epsilon$.

The way to calculate the output weights $\boldsymbol{\beta}$ from the knowledge of the hidden layer output matrix $\mathbf{H}$ and target values $\mathbf{T}$, is proposed with the use of a Moore-Penrose generalized inverse of the matrix $\mathbf{H}$, denoted as $\mathbf{H}^\dagger$ [RM72]. This proposed solution has three main properties making it an appealing solution:

1. It is one of the least-squares solutions of the equation $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, hence the minimum training error can be reached with this solution.
2. It is the solution with the smallest norm among the least-squares solutions.
3. The smallest norm solution among the least-squares solutions is unique and it is $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$.

Theoretical proofs and a more thorough presentation of the ELM algorithm are detailed in the original paper [HZS06]. In Huang *et al.*'s later work it has been

proved that the ELM is able to perform universal function approximation [HCS06].

ELM Solution with Pseudo-inverse

Most often, an ELM problem is over-determined ($N > L$), with the number of training data samples larger than the number of hidden neurons. For determined ($N = L$) and under-determined ($N < L$) instances, ELM should use regularization [HZDZ12]. Otherwise it overfits and has a poor generalization performance.

A unique solution for an over-determined system is given by a minimum L^2 norm of the training error. It may be found using the Moore-Penrose generalized inverse [RM72] (pseudoinverse) of the matrix $\mathbf{H}$, denoted as $\mathbf{H}^\dagger$. As the matrix $\mathbf{H}$ has a full column rank, the pseudoinverse is computed as in equation (2.9).

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{2.7}$$

$$\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T} \tag{2.8}$$

$$\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T, \tag{2.9}$$

The pseudoinverse is prone to numerical instabilities if the matrix $\mathbf{H}^T\mathbf{H}$ is close to singular. Practically (in Matlab® and Python), the implementations of the pseudoinverse include a small regularization term $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H}+\alpha\mathbf{I})\mathbf{H}^T$ where $\alpha = 50\epsilon$ and ϵ is the machine precision for a used type of floating point numbers. Adding a regularization term makes matrix $\mathbf{H}^T\mathbf{H}$ non-singular, and the same solution applicable also for determined and under-determined systems.

Classification with ELMs

An ELM is a regression model, but it is easily adapted for classification. To classify a dataset with ELM, data targets need to be set in a special encoding manner.

If the classes are categorical and independent, then one target feature is created for each class. Targets for the correct classes are set to one, and targets for irrelevant classes are set to zero. This encoding creates a unit length vector for each class, which is orthogonal to vectors of all other classes. Distances between target vectors of different classes are the same, so the class independence is kept. The predicted class is assigned according to the target with the largest ELM output.

If the classes are ordinal and have a ranking, then they are translated into real numbers. Only one target feature is used for all the classes, and a predicted class is the one with the closest number to an ELM output.

In a multi-label problem, a sample can have multiple correct classes. The targets are created similarly as in the independent classes problem formulation. The predicted classes are assigned for all ELM outputs greater than a threshold value.

Using ELM for classification with independent classes changes the way how the prediction error is calculated. The classification error does not penalize (or encourage) small changes in the ELM output values, which do not lead to a different classification. This makes a difference in the model structure selection (described in section 2.1), where an optimization w.r.t. the MSE regression error finds an incorrect optimal number of hidden neurons, and creates a model with a sub-optimal classification prediction performance.

Model Structure Selection in ELMs

Model structure selection prevents ELM from learning noise from data and over-fitting. It does so by artificially limiting the learning ability of an ELM. A training dataset has multiple instances of inputs, and the corresponding targets, which are generated by the projected data and an added noise. The *noise* term includes both random noise and projection from features not present in the inputs. Learning particular data samples together with the associated noise is called over-fitting. An over-fitted ELM model has worse generalization performance (prediction performance on new data), which can be measured using a validation set of data. A model structure selection process finds an optimal generalization performance by changing the amount of model parameters or applying regularization to the model.

A hyper-parameter of ELMs, which governs the amount of effective parameters, is the number of hidden neurons L. The optimum number of neurons is found with a validation set, a cross-validation procedure or a Leave-One-Out validation procedure [SLWV03] (which has an efficient solution in ELMs). Hidden neurons can be added and removed randomly, or they can be ranked by their relevance to the problem. This ranking is called "Optimal Pruning" [MSB+10] and it achieves better performance with a trade-off of a longer runtime. Neuron pruning methods correspond to L^1-regularization.

Another model structure selection technique available in ELMs is the Tikhonov regularization [Tik63]. It reduces an effective number of model parameters by reducing the influence of neuron outputs without removing neurons by themselves.

Tikhonov regularization is efficient for achieving numerical stability in near-singular ELMs (and linear problems in general). This regularization corresponds to L^2-regularization, and can be combined with L^1 to achieve the best results [MvB+11].

Model structure selection is less important in Big Data tasks, because with a large number of samples an ELM model learns to ignore noise. Large tasks are often complex enough not to overfit even at the limits of the hardware. Also, most model structure selection methods significantly increase runtime, which is a limiting factor for training large ELM models.

2.2 L^1-regularized ELM

An ELM algorithm can have some issues when encountering irrelevant or correlated data. For this reason, an L^1 regularization is applied on the hidden layer neurons, possibly pruning out some of them. The resulted L^1-regularized ELM is widely known as an Optimally-Pruned ELM [MBJ+08, MSL08] (OP-ELM). The OP-ELM extends the original ELM algorithm and wraps this extended algorithm within a methodology using a pruning of the neurons, leading to a more robust overall algorithm. Pruning of neurons in a network built using ELM has been proposed in [ROTZ08] for classification purposes, and using statistical tests as a measure of relevance of the neurons regarding the output. The OP-ELM presented here applies to both classification and regression problems and uses a Leave-One-Out criterion for the selection of an appropriate number of neurons.

The Problem of ELM with Irrelevant Variables

An ELM models tend to have problems when irrelevant or correlated variables are present in the training data set. As an illustration of this, a toy example with two cases, without and with an irrelevant variable, are tested and compared. Figure 2.3 shows the ELM model obtained by training on the sum of sines example. In this case, the ELM model fits very well to the training data, with no apparent perturbation or distortion.

On Figure 2.4, an additional variable containing a pure Gaussian noise, totally unrelated to the actual data, is also used as an input. The additional noise variable is

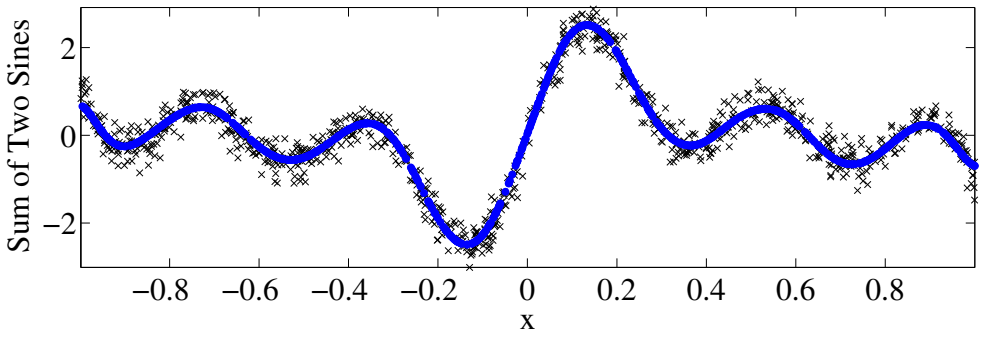


Figure 2.3: An example of training result using ELM on a sum of two sines. Blue dots represent the ELM model fitting the data points (gray crosses).

not shown in the figure. The ELM model on top of the data is much more spread and approximate than in the previous case. Overall, the global fitting of the ELM model to the actual data is not as good as before. For this reason, it is proposed in the OP-ELM methodology to perform a pruning of the irrelevant variables, via pruning of the related neurons of the SLFN built by the ELM.

2.2.1 Optimally-Pruned ELM (OP-ELM)

The OP-ELM is made of three main steps summarized in Figure 2.5. The very first step of the OP-ELM methodology is the actual construction of the SLFN using the original ELM algorithm with a lot of neurons.

Second and third steps are presented in more details in the next two subsections and are meant for an effective pruning of the possibly useless neurons of the SLFN: Multi-Response Sparse Regression [ST05] (MRSR) algorithm allows obtaining a ranking of the neurons according to their usefulness, while the actual pruning is

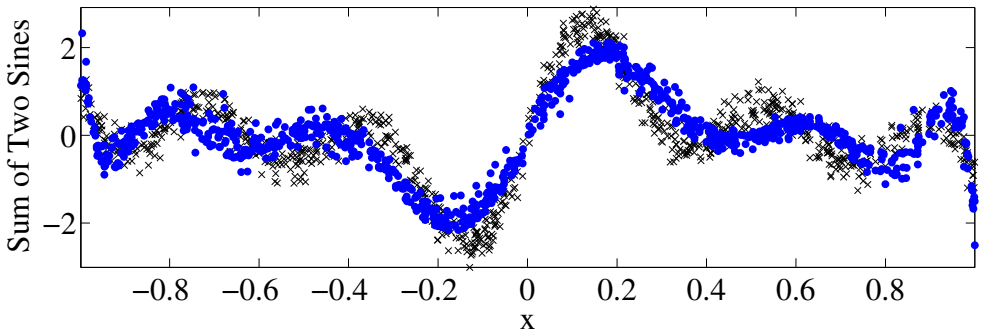


Figure 2.4: An example using the same sum of sine as in Figure 2.3 and an additional noisy variable (not represented here) for training. The obtained ELM model is much more spread and approximate, due to the irrelevant variable included.

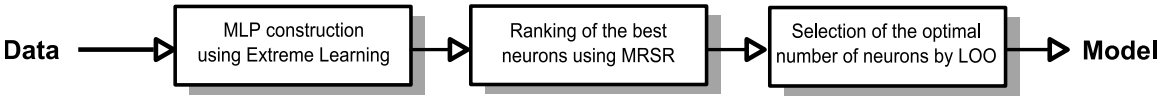


Figure 2.5: The three steps of the OP-ELM algorithm.

performed using the results of the Leave-One-Out validation.

The OP-ELM algorithm uses a combination of three different types of kernels, for robustness and more generality, where the original ELM proposed to use only sigmoid kernels. The used types are linear, sigmoid and Gaussian (RBF) kernels. Having a linear kernel included in the network helps when the problem is linear or nearly linear. The Gaussian kernels have their centers taken randomly from the data points, similarly as in [PG90], and widths randomly drawn between percentile 20 percent and percentile 80 percent of the distance distribution of the input space, as

suggested in [CSFS02]. The sigmoid weights are drawn randomly from a uniform distribution in the interval $[-5, 5]$ in order to cover the whole zero mean and unit variance data range.

The OP-ELM methodology can also handle multi-output and multi-class problems in both regression and classification using multiple inputs.

2.2.2 Multiresponse Sparse Regression: MRSR

In order to get rid of the useless neurons of the hidden layer, the Multi-Response Sparse Regression, proposed by Timo Similä and Jarkko Tikka in [ST05], is used.

The main idea of the algorithm is the following: Denote by $\mathbf{H} = [\mathbf{h}_1 \dots \mathbf{h}_L]$ the $N \times L$ regressor matrix. MRSR adds each column of the regressor matrix one by one to the model $\mathbf{Y}^k = \mathbf{H}\boldsymbol{\beta}^k$, where $\mathbf{Y}^k = [\mathbf{y}_1^k \dots \mathbf{y}_c^k]$ is the model approximation of the targets. The $\mathbf{W}^k$ weight matrix has k nonzero rows at k^{th} step of the MRSR. With each new step a new nonzero row, and a new column of the regressor matrix is added to the model. More specific details of the MRSR algorithm can be found from the original paper [ST05].

It can be noted that the MRSR is mainly an extension of the Least Angle Regression (LARS) algorithm [EHJT04] and hence, it is actually a variable ranking technique, rather than a selection one. An important detail shared by the MRSR and the LARS is that the ranking obtained is exact if the problem is linear. In fact, this is the case with the OP-ELM, since the neural network built in the previous step is

linear between the hidden layer and the output. Therefore, the MRSR provides an exact ranking of the neurons for our problem. Because of the exact ranking provided by the MRSR, it is used in the output layer of an ELM to rank its hidden neurons.

2.2.3 Leave-One-Out (LOO) or PRESS Statistics in ELM

Since the MRSR only provides a ranking of the hidden neurons, the decision over the actual best number of neurons for the model is taken using a Leave-One-Out validation method.

One problem with the LOO error is that it can be very time consuming, if the data set has a high number of samples. Fortunately, the PRESS (Predicted REsidual Sum of Squares) statistics provide a direct and exact formula for the calculation of the LOO error for linear models. See [Mye90] and [BBB98] for details of this formula and its implementations:

$$\varepsilon^{\text{PRESS}} = \frac{\mathbf{t}_i - \mathbf{h}_i \boldsymbol{\beta}_i}{1 - \mathbf{h}_i \mathbf{P} \mathbf{h}_i^T}, \tag{2.10}$$

where $\mathbf{P}$ is defined as $\mathbf{P} = (\mathbf{H}^T\mathbf{H})^{-1}$ and $\mathbf{H}$ is the hidden layer output matrix.

The original PRESS formula from eq. (2.10) can be expressed in terms of ELM output layer variables as

$$\text{MSE}^{\text{PRESS}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{t_i - \mathbf{h}_i \left(\mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{h}_i^T t_i}{1 - \mathbf{h}_i \left(\mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{h}_i^T} \right)^2, \tag{2.11}$$

which means that each observation is "predicted" using the other $N-1$ observations and the residuals are finally squared and summed up. Algorithm 2.1 proposes to implement this formula in an efficient way, by matrix computations. Note that on

step 4, only diagonal elements of a matrix produce $\mathbf{PH}^T$ need to be computed.

Algorithm 2.1 Allen's PRESS algorithm, in a fast matrix form.

1: Compute the utility matrix $\mathbf{C} = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}$

2: And $\mathbf{P} = \mathbf{HC}$;

3: Compute the pseudo-inverse $\boldsymbol{\beta} = \mathbf{CH}^T\mathbf{t}$;

4: Compute the denominator of the PRESS $\mathbf{D} = \mathbf{1} - \text{diag}\left(\mathbf{PH}^T\right)$;

5: And finally the PRESS error $\varepsilon = \frac{\mathbf{t}-\mathbf{H}\boldsymbol{\beta}}{\mathbf{D}}$;

6: Reduced to a MSE, $\text{MSE}^{\text{PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$.

For a multi-output ELM with c output neurons the PRESS formula is expressed in a matrix notation as:

$$\text{MSE}^{\text{PRESS}} = \frac{1}{Nc}\sum_{n=1}^{N}\sum_{k=1}^{c}\left(\frac{\mathbf{T} - \mathbf{H}\left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}}{\left[\mathbf{1}_N - \text{diag}\left(\mathbf{H}\left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\right)\right]\mathbf{1}_c^T}\right)_{ik}^2, \tag{2.12}$$

A fast matrix computation is similar to Algorithm 2.1, with a two-dimensional PRESS error $\varepsilon_{[N\times c]}$ on step 5.

The final decision over the appropriate number of neurons for the model can then be taken by evaluating the LOO error versus the number of neurons used. Here, the neurons are already ranked by the MRSR.

In order to give an overview of the usefulness of the ranking step performed by the MRSR algorithm, the final model structure selection for the OP-ELM model using the Ailerons data set (see Section 2.2.4) is shown in Figure 2.6.

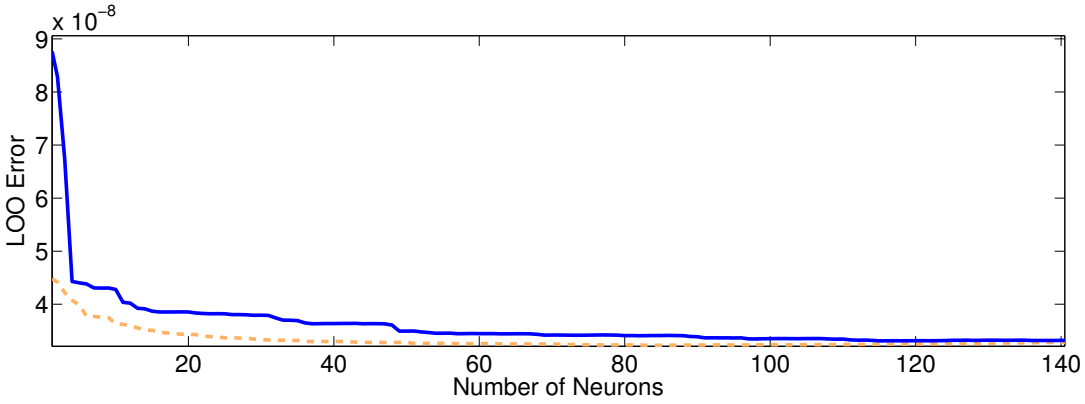


Figure 2.6: Comparison of LOO error with and without the MRSR ranking. The solid blue line represents the LOO error without and the dashed orange one with the MRSR ranking.

It can be noted from Figure 2.6 that the OP-ELM benefits greatly from the MRSR ranking step. The convergence is faster, because the LOO error gets to the minimum faster when the MRSR is used than when it is not. Also, the number of neurons is far less in the LOO error minimum point when using the MRSR ranking, thus leading to a smaller network with the same performance.

In the end, a SLFN possibly using a mix of linear, sigmoid and Gaussian kernels is obtained, with a highly reduced number of neurons, all within a small computational time.

2.2.4 Experiments

In the following, five methodologies are compared using several regression and classification tasks. The compared methods are Gaussian Processes [Ras04] (GP),

Support Vector Machines [CV95] (SVM), Multi-Layer Perceptron [Bis96] network (MLP) trained with error back-propagation algorithm, the original Extreme Learning Machine algorithm (ELM) and the proposed Optimally-Pruned ELM (OP-ELM).

Data Sets

Fifteen different data sets have been chosen for the experiments, eleven for regression and four for classification problems. The data sets are collected from the UCI Machine Learning Repository [Lic13] and they have been chosen by the overall heterogeneity in terms of number of samples, variables and classes for classification problems.

Table 2.1 summarizes the different attributes for the fifteen data sets. All data sets have been pre-processed in the same way. 10 different random permutations of the whole data set are taken without replacement, and two thirds are used to create the training set and the remaining third is used for the test set. Then, the training set is normalized, zero mean and unit variance, and the test set is also normalized using the same mean and variance used for the training set. Because the test set is normalized using the same normalization parameters than for the training, it is most likely not exactly zero mean and unit variance.

It should also be noted that the proportions of the classes, for the classifications cases, have been kept balanced: each class is represented in an equal proportion, in both training and test sets. This is important in order to have relevant test results.

Table 2.1: Information about the selected data sets. Number of variables and number of samples for both training and testing, two-thirds of the whole set for training and one third for test. For classification problems, the variables column also includes the number of classes in the data set.

		Samples	
Regression	# of Variables	Train	Test
Abalone	8	2784	1393
Ailerons	5	4752	2377
Elevators	6	6344	3173
Computer	12	5461	2731
Auto price	15	106	53
CPU	6	139	70
Servo	4	111	56
Breast Cancer	32	129	65
Bank	8	2999	1500
Stocks	9	633	317
Boston	13	337	169
Classification			
Iris	4/3	100	50
Wisconsin Breast Cancer	30/2	379	190
Pima Indians Diabetes	8/2	512	256
Wine	13/3	118	60

Experiments

Experiments have been conducted using the online versions of the methodologies, unaltered. All experiments have been run on the same `x86_64` Linux machine with at least 4GB of memory (no swapping for any of the experiments) and 2+GHz processor. It should be noted that even though some methodologies are using parallelization of the tasks, the computational times are reported considering single-threaded execution on one single core, for the sake of comparisons. The hyper-parameters for the SVM and the MLP are selected using a 10-fold Cross-Validation.

The SVM is performed using the SVM toolbox [CL11] with the default settings for the hyper-parameters and the grid search: the grid is logarithmic between 2^{-2} and 2^{10} for each hyper-parameter; nu-SVC has been used for classification and epsilon-SVR for regression, with radial basis function kernel. The original grid search has been replaced by a parallelization process, which distributes parts of the grid over different machines.

The MLP [Bis96] is performed using a Neural Network toolbox, which is part of the Matlab© software from the Mathworks. The training of the MLP is performed using the Levenberg-Marquardt backpropagation. In order to decrease the possibility of local minima with the MLP, the training is repeated 10 times for each fold and the best network according to the training error is selected for validation. For example, in order to validate the MLP network using 12 hidden neurons, we have to train a total of 100 MLP networks with 12 hidden neurons to evaluate the validation error. This procedure is done for each number of hidden nodes from 1 to 20 and the appropriate

number according to the validation MSE is selected.

The Gaussian Processes is performed using a gpml toolbox for Matlab© from Rasmussen and Williams [RW06]. The GP is performed using the default settings taken from the examples of usage of the toolbox.

Finally, the OP-ELM was used with all possible kernels, linear, sigmoid and gaussian, using a maximum number of 100 neurons.

Computational Times

Computational times are first reviewed for all five methodologies. Tables 2.2 and 2.3 give the computational times for training and test steps (sum of both), for each methodology. It can be noted that for all five methodologies, the computational times for the test steps are negligible compared to the training times; this is especially clear for large training times, like the SVM or MLP ones.

According to Tables 2.3 and 2.2, the ELM is the fastest algorithm by several orders of magnitude compared for example to the SVM. This is in line with the claims of the ELM authors. The proposed OP-ELM is between one and three orders of magnitude slower than the original ELM, but still much faster than the rest of the compared methods in all data sets.

However, the ranking of the SVM, MLP and GP regarding the computational times is not exactly the same in all data sets, but in every case they are clearly slower than the ELM and OP-ELM.

The main reason, why the OP-ELM has been designed in the first place, is

Table 2.2: Computational times (in seconds) for all five methodologies on the regression data sets. Algorithms have been sorted by computational time. "Auto P." stands for Auto Price data set and "Breast C." for Breast Cancer data set.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
SVM	6.6e+4	4.2e+2	5.8e+2	3.2e+5	2.6e+2	3.2e+2
MLP	2.1e+3	3.5e+3	3.5e+3	8.2e+3	7.3e+2	5.8e+2
GP	9.5e+2	2.9e+3	6.5e+3	6.3e+3	2.9	3.2
OPELM	5.7	16.8	29.8	26.2	2.7e-1	2.0e-1
ELM	4.0e-1	9.0e-1	1.6	1.2	3.8e-2	4.2e-2

	Servo	Breast C.	Bank	Stocks	Boston	
SVM	1.3e+2	3.2e+2	1.6e+3	2.3e+3	8.5e+2	
MLP	5.2e+2	8.0e+2	2.7e+3	1.2e+3	8.2e+2	
GP	2.2	8.8	1.7e+3	4.1e+1	8.5	
OPELM	2.1e-1	4.2e-1	8.03	1.54	7.0e-1	
ELM	3.9e-2	4.8e-2	4.7e-1	1.1e-1	7.4e-2	

Table 2.3: Computational times (in seconds) compared for all five methodologies for classification data sets. "Wisc. B.C." for Wisconsin Breast Cancer data set and "Pima I.D." for Pima Indians Diabetes data set.

	Iris	Wisc. B.C.	Pima I.D.	Wine
SVM	2.3e+2	2.9e+3	3.3e+3	3.8e+2
MLP	7.6e+2	1.7e+3	4.1e+2	1.2e+3
GP	7.6e-1	6.1	5.8	1.9
OPELM	7.4e-2	1.1	9.6e-1	4.4e-1
ELM	2.4e-2	4.3e-2	4.8e-2	2.7e-2

to add more robustness to the very simple and fast ELM algorithm. Experimental results for this robustness are presented in the next subsection through test results.

Test Errors

Because the validation results, while providing a good measure of the model fit to the data, do not measure the actual interpolation properties of the model, only the test results for the five models are presented in Tables 2.4 and 2.5.

According to the test results, the SVM is very reliable on average. Meanwhile, as mentioned earlier, the ELM can have good results with respect to its computational speed. But also, it can have very high Mean Square Errors on some test sets, for example in Auto price and CPU data sets.

In this regard, the OP-ELM manages to keep a good MSE, when comparing to other algorithms, and even rather close to the performance of the SVM (and of the GP) on many data sets used in the experiments. This comforts the earlier claims that the OP-ELM keeps a part of the speed of the ELM and, therefore, is much faster than most common algorithms, while remaining robust and accurate and providing good interpolation models.

Finally, in order to give an overview of the pruning result for the OP-ELM, Table 2.6 lists the selected neurons for two data sets, one for regression and one for classification, namely Ailerons and Iris.

One can see that the total number of kept neurons is fairly stable, and so is

Table 2.4: Mean Square Error results in boldface (and standard deviations in regular) for all five methodologies for the regression data sets. "Auto P." stands for Auto Price data set and "Breast C." for Breast Cancer data set.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
SVM	**4.5**	**1.3e-7**	**6.2e-6**	**1.2e+2**	**2.8e+7**	**6.5e+3**
	2.7e-1	2.6e-8	6.8e-7	8.1e+1	8.4e+7	5.1e+3
OPELM	**4.9**	**2.8e-7**	**2.0e-6**	**3.1e+1**	**9.5e+7**	**5.3e+3**
	6.6e-1	1.5e-9	5.4e-8	7.4	4.0e+6	5.2e+3
ELM	**8.3**	**3.3e-8**	**2.2e-6**	**4.9e+2**	**7.9e+9**	**4.7e+4**
	7.5e-1	2.5e-9	7.0e-8	6.2e+1	7.2e+9	2.5e+4
GP	**4.5**	**2.7e-8**	**2.0e-6**	**7.7**	**2.0e+7**	**6.7e+3**
	2.4e-1	1.9e-9	5.0e-8	2.9e-1	1.0e+7	6.6e+3
MLP	**4.6**	**2.7e-7**	**2.6e-6**	**9.8**	**2.2e+7**	**1.4e+4**
	5.8e-1	4.4e-9	9.0e-8	1.1	9.8e+6	1.8e+4

	Servo	Breast C.	Bank	Stocks	Boston	
SVM	**6.9e-1**	**1.2e+3**	**2.7e-2**	**5.1e-1**	**3.4e+1**	
	3.3e-1	7.2e-1	8.0e-4	9.0e-2	3.1e+1	
OPELM	**8.0e-1**	**1.4e+3**	**1.1e-3**	**9.8e-1**	**1.9e+1**	
	3.3e-1	3.6e+2	1.0e-6	1.1e-1	2.9	
ELM	**7.1**	**7.7e+3**	**6.7e-3**	**3.4e+1**	**1.2e+2**	
	5.5	2.0e+3	7e-4	9.35	2.1e+1	
GP	**4.8e-1**	**1.3e+3**	**8.7e-4**	**4.4e-1**	**1.1e+1**	
	3.5e-1	1.9e+2	5.1e-5	5.0e-2	3.5	
MLP	**2.2e-1**	**1.5e+3**	**9.1e-4**	**8.8e-1**	**2.2e+1**	
	8.1e-2	4.4e+2	4.2e-5	2.1e-1	8.8	

Table 2.5: Correct classification rates in boldface (and standard deviations in regular) for all five methodologies for classification data sets. "Wisc. B.C." for Wisconsin Breast Cancer data set and "Pima I.D." for Pima Indians Diabetes data set.

	Iris	Wisconsin B.C.	Pima I.D.	Wine
SVM	**95.4**	**91.6**	**72.7**	**95.83**
	1.9	1.7	1.5	2.9
OPELM	**95.0**	**95.6**	**74.9**	**90.7**
	2.1	1.3	2.4	4.9
ELM	**72.2**	**95.6**	**72.2**	**81.8**
	1.01	1.2	1.9	6.2
GP	**95.6**	**97.3**	**76.3**	**96.1**
	2.3	0.9	1.8	2.1
MLP	**94.8**	**95.6**	**75.2**	**96.0**
	3.8	1.9	1.9	2.4

Table 2.6: Details of numbers of selected neurons in OP-ELM for the Delta Ailerons and Iris data sets. L stands for linear neurons, S for sigmoid ones and G for Gaussian.

	Ailerons				Iris			
Run #	L	S	G	Total	L	S	G	Total
1	5	50	25	80	2	16	6	24
2	5	50	30	85	3	17	4	24
3	5	49	21	75	2	16	6	24
4	5	50	45	100	2	8	4	14
5	5	50	40	95	2	13	4	19
6	4	43	13	60	2	4	3	9
7	5	48	17	70	2	7	5	14
8	4	36	10	50	2	5	2	9
9	5	50	45	100	2	10	2	14
10	3	27	5	35	2	13	4	19

the number of linear neurons. It is interesting to note that the amount of neurons for each type is more stable for classification data sets than for regression one. On

average, the situation depicted here is globally similar for other data sets.

2.3 L^1 and L^2-regularized ELM

This section deals with a problem encountered in the original OP-ELM. The Leave-One-Out criterion originally used in the OP-ELM for the pruning is very fast, but raises numerical problems which possibly "disturb" the pruning strategy.

The proposed solution to this situation is by the use of L^2 regularization in the OP-ELM. The concept of regularization for regression problems using L^1, L^2 or other norms-based penalties on the regression weights has been studied extensively (see for example [EHJT04, GHW79, HK70, Owe06, ST05, Thi76, Tib96, Tik63, ZRY09, ZH05]) and some of the most widely used methods are presented in section 2.3.1: Lasso [Tib96], Tikhonov regularization [Tik63, HK70], but also hybrid penalties such as the Elastic Net [ZH05] and the Composite Absolute Penalties [ZRY09].

While these penalties are either of only one kind (traditionally L^1 or L^2) or used simultaneously (see Owen's hybrid [Owe06] for example), an iterative use of both regularizations is applied to ELM. An L^1 penalty is first used to rank the neurons, followed sequentially by an L^2 penalty to prune the network accordingly. Section 2.3.3 details the approach used, by a modification of Allen's PRESS statistic [All74].

2.3.1 The L^2 penalty: Tikhonov Regularization

One possible approach to find a solution which deems a lower MSE than the Ordinary Least Squares one is to use regularization in the form of Tikhonov regularization proposed in [Tik63] (a.k.a. Ridge Regression [HK70]). This time, the minimization problem involves a penalty using the square of the regression coeffi-

cients

$$\min_{\lambda,\mathbf{w}} \left[\sum_{i=1}^{N} \left(t_i - \mathbf{x}_i^T \mathbf{w} \right)^2 + \lambda \sum_{j=1}^{L} w_j^2 \right]. \tag{2.13}$$

Thanks to a bias–variance tradeoff, the Tikhonov regularization achieves better prediction performance than the traditional OLS solution, and it outperforms the Lasso solution in cases were the variables are correlated. One famous advantage of the Tikhonov regularization is that it tends to identify/isolate groups of variables, enabling further interpretability (this grouping can be very desirable for some data sets, as mentioned in [ZH05]).

The major drawback of this regularization method is similar to one mentioned for the OLS: it does not give any parsimonious solution since all variables are retained due to the L^2 penalty. Therefore the Tikhonov regularization does not select variables directly, contrary to the Lasso which actually performs variable selection "internally" (given that λ is large enough to set some coefficients to zero).

2.3.2 Hybrid Penalties

In an attempt to overcome the drawbacks of each of the two approaches, hybrid solutions have been developed which use both the L^1 and the L^2 penalties in the same minimization problem. Below are proposed three approaches that tackle this problem: the Elastic Net [ZH05], the Composite Absolute Penalties [ZRY09], and finally an original approach by Owen [Owe06] using an "inverted" Huber loss function.

The Elastic Net

Zhou and Hastie in [ZH05] propose to alleviate the problems encountered by the Tikhonov regularization (lack of sparsity) while keeping its good performance thanks to the L^2 penalty. This is done using a composite of the Lasso and Tikhonov regularization, by combining the two penalties L^1 and L^2 in the form of a weighted penalty

$$\lambda_1 \sum |\mathbf{w}_j| + \lambda_2 \sum \mathbf{w}_j^2, \tag{2.14}$$

with λ_1 and λ_2 positive (controlling the sparsity of the model). This version of the penalty term is denoted as the "naïve" elastic net by the authors, which admits an easily computed solution, provided that λ_1 and λ_2 are defined and already optimal. As the authors mention in [ZH05], this naïve version of the algorithm is fast to obtain and rather efficient, but creates a greater shrinkage effect (on the regression coefficients) than the original Lasso, which adds bias to the solution, while not reducing significantly the variance of it. In the end, the naïve version only seems to work well when it is close enough to the Tikhonov or Lasso case (i.e. λ_1 very small or λ_2 very small).

The "normal" version of the elastic net is then a scaled naïve one: defining $\mathbf{T}^* = (\mathbf{T};\ \mathbf{0}_{L\times 1})^T$, $\mathbf{X}^* = \frac{1}{\sqrt{1+\lambda_2}} \left(\mathbf{X};\ \sqrt{\lambda_2}\, \mathbf{I}_{L\times L}\right)^T$ and $\mathbf{w}^* = \sqrt{1+\lambda_2}\mathbf{w}$, the minimization problem for the elastic net is then

$$\min_{w^*} \left[\sum_{i=1}^{N+L} \left(t_i^* - \mathbf{x}_i^{*T}\mathbf{w}^*\right)^2 + \frac{\lambda_1}{\sqrt{1+\lambda_2}} \sum_{j=1}^{L} |w_j| \right]. \tag{2.15}$$

The scaling performed allows to reduce the problem of shrinkage present in the naïve

version of the elastic net, while retaining the advantages of the original naïve approach (e.g. the automatic variable selection).

In practice, the algorithm is implemented as a modification of the LARS algorithm (the LARS-EN) since once λ_2 is fixed, the computations are similar to that of a Lasso.

While the LARS-EN is a very efficient way of implementing the elastic net approach, it remains that two parameters need optimizing: λ_1 and λ_2. Usually, this is done by the use of classical Cross-Validation (CV) which is unfortunately costly for it requires a two-dimensional search, which is hardly feasible if one wants to keep the ELM speed property.

Composite Absolute Penalty (CAP)

In [ZRY09], Zhao *et al.* propose to use a more generalized version of the penalty term, by using a vector of penalties on which is computed a norm. Denoting by

$$\|\mathbf{a}\|_\gamma = \left(\frac{1}{N} \sum_{i=1}^{N} |a_i|^\gamma \right)^{1/\gamma}, \tag{2.16}$$

the γ-norm of a vector $\mathbf{a} = (a_1, \ldots, a_N)^T$ for $\gamma \in \mathbb{N}^*$, the method of Composite Absolute Penalties (CAP) generalizes the concept used in the Elastic Net penalty to

$$\left\| \left(\|\mathbf{w}_{G_1}\|_{\gamma_1}, \|\mathbf{w}_{G_2}\|_{\gamma_2}, \ldots, \|\mathbf{w}_{G_k}\|_{\gamma_k} \right) \right\|_{\gamma_0}, \tag{2.17}$$

where G_j is a subset of $\{1, \ldots, L\}$ and $\mathbf{w}_{G_j}$ is obtained by extracting the components denoted in G_j from $\mathbf{w}$.

It can be seen that the penalty term is therefore a γ_0-norm on a vector of

penalties $\|\mathbf{w}_{G_i}\|_{\gamma_i}$. This general formulation of the penalty for example comes down to the Lasso when $\gamma_j = 1, \forall j$.

While the generalization capability of the CAP approach is clear, the determination of the groups G_j and of the γ_j is time-consuming and prone to heuristics/*a priori* information on the variables. Again, cross-validation is typically used for the determination of the γ_j, leading to important computational times, again not "compatible" with the ELM speed.

Owen's Hybrid

A slightly different approach is proposed by Owen in [Owe06], by the use of an original loss function for the penalty. The problem is formulated as

$$\min \left[\sum_{i=1}^{N} L\left(t_i - \mathbf{x}_i^T \mathbf{w}\right) + \sum_{j=1}^{L} P\left(w_j\right) \right], \tag{2.18}$$

where the $L(\cdot)$ function can be assumed to be a 2-norm $\|\cdot\|_2^2$ in this case. The emphasis is here put on the P function, which is chosen (or more "designed") to behave like an absolute value function for small w_j for sparse solutions to arise, and like a quadratic function on large w_j to retain the properties of the Tikhonov regularization.

The author proposes an "inverted" Huber loss function for that purpose. While the Huber function is such that

$$\mathcal{H}(z) = \begin{cases} z^2 & \text{for } |z| \leq 1 \\ 2|z| - 1 & \text{for } |z| \geq 1 \end{cases}, \tag{2.19}$$

the "inverted" version (also scaled to accommodate thresholding) is given by

$$\mathcal{B}_M(z) = \begin{cases} |z| & \text{for } |z| \leq M \\ \frac{z^2+M^2}{2} & \text{for } |z| \geq M \end{cases}. \tag{2.20}$$

The M value permits to choose where the transition between the absolute value function and the quadratic one takes place.

The minimization problem ends up as a convex one, with a large number of constraints (see [Owe06] for more details), which is unfortunately computationally very expensive for significant data sets.

In the end, it can be noted that all the variants of the minimization problem presented here are convex problems and have hence an optimal solution that is reachable by standard convex optimization techniques. Unfortunately, the large number of parameters or constraints on the minimization problem makes it more difficult to solve, and cross-validation is often used for the determination of the parameters. This in turn implies large computational times.

While the properties of both the Lasso and the Tikhonov regularization are desirable, the penalties combining both lead to complex minimization problems which take too long for the application to OP-ELM. The following section 2.3.3 proposes to use the two approaches in turn, instead of together, along with fast matrix computations.

2.3.3 Tikhonov Regularized OP-ELM (TROP-ELM)

Recently, Deng *et al.* in [DZC09] proposed a Regularized Extreme Learning Machine algorithm, which is essentially a L^2 penalized ELM, with a possibility to weight the sum of squares in order to address outliers interference. Using the notations from the previous section, the minimization problem is here

$$\min_{\lambda,\mathbf{d},\mathbf{w}} \left[\lambda \sum_{i=1}^{N} \left(d_i \left(t_i - \mathbf{x}_i^T \mathbf{w} \right) \right)^2 + \sum_{j=1}^{L} w_j^2 \right], \tag{2.21}$$

where the d_i are the weights meant to address the outliers.

This extension of the ELM clearly (from the results in [DZC09]) brings a very good robustness to outliers to the original ELM. Unfortunately, it suffers from the problems related to L^2 penalties, that is the lack of sparsity for example.

As described before, the original OP-ELM already implements a L^1 penalty on the output weights, by performing a LARS between the hidden and output layer. It is here proposed to modify the original PRESS LOO criterion for the selection of the optimal number of neurons by adding a Tikhonov regularization factor in the PRESS, therefore making the modified PRESS LOO a L^2 penalty applied on the L^1 penalized result from the LARS.

In the following are used matrix operations such as $\frac{\mathbf{A}}{\mathbf{B}}$ to refer to the matrix $\mathbf{C}$ such that $(c_{i,j}) = \frac{a_{i,j}}{b_{i,j}}$. Also the $\mathrm{diag}(\cdot)$ operator is used to extract the diagonal of a matrix, $\mathrm{diag}(\mathbf{A}) = (a_{1,1}, \ldots, a_{n,n})^T$.

Tikhonov-Regularized PRESS (TR-PRESS)

The main drawback of the original PRESS statistics (section 2.2.3) lies in the use of a pseudo-inverse in the calculation, which can lead to numerical instabilities if the ELM hidden layer representation $\mathbf{H}$ is not full rank. This is unfortunately very often the case, with real-world data sets. The following approach proposes two improvements on the computation of the original PRESS: regularization and its fast matrix calculation.

In [GHW79], Golub *et al.* note that the Singular Value Decomposition (SVD) approach to compute the PRESS statistic is preferable to the traditional pseudo-inverse mentioned above, for numerical reasons. In this very same paper is proposed a generalization of Allen's PRESS, as the Generalized Cross-Validation (GCV) method, which is technically superior to the original PRESS, for it can handle cases were the data is extremely badly defined —for example if all $\mathbf{H}$ entries are 0 except the diagonal ones.

In practice, from our experiments, while the GCV is supposedly superior, it leads to identical solutions with an increased computational time, compared to the original PRESS and the Tikhonov-regularized version of PRESS presented below.

Algorithm 2.2 gives the computational steps used, in matrix form, to determine the $\mathrm{MSE}^{\text{TR-PRESS}}(\lambda)$ from

$$\mathrm{MSE}^{\text{TR-PRESS}}(\lambda) = \sum_{i=1}^{N} \left(\frac{t_i - \mathbf{h}_i \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1} \mathbf{h}_i^T t_i}{1 - \mathbf{h}_i \left(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I}\right)^{-1} \mathbf{h}_i^T} \right)^2, \tag{2.22}$$

which is the regularized version of Equation (2.11).

Globally, the algorithm uses the SVD of $\mathbf{H}$ to avoid computational issues, and

Algorithm 2.2 Tikhonov-Regularized PRESS. In practice, the REPEAT part of this algorithm (convergence for λ) is solved by a Nelder-Mead approach [NM65], a.k.a. downhill simplex.

1: Decompose $\mathbf{H}$ by SVD: $\mathbf{H} = \mathbf{USV}^T$;

2: Compute the products (used later): $\mathbf{A} = \mathbf{XV}$ and $\mathbf{B} = \mathbf{U}^T\mathbf{t}$;

3: **repeat**

4: Using the SVD of $\mathbf{H}$, compute the $\mathbf{C}$ matrix by:
$$\mathbf{C} = \mathbf{A} \times \begin{pmatrix} \frac{S_{11}}{S_{11}^2+\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{S_{NN}}{S_{NN}^2+\lambda} \end{pmatrix};$$

5: Compute the $\mathbf{P}$ matrix by: $\mathbf{P} = \mathbf{CB}$;

6: Compute $\mathbf{D}$ by: $\mathbf{D} = \mathbf{1} - \text{diag}\left(\mathbf{CU}^T\right)$;

7: Evaluate $\varepsilon = \frac{\mathbf{t}-\mathbf{P}}{\mathbf{D}}$ and the actual MSE by $\text{MSE}^{\text{TR-PRESS}} = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$;

8: **until** convergence on λ is achieved

9: Keep the best $\text{MSE}^{\text{TR-PRESS}}$ and the λ value associated.

introduces the Tikhonov regularization parameter in the calculation of the pseudo-inverse by the SVD. This specific implementation happens to run very quickly, thanks to the pre-calculation of utility matrices ($\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$) before the optimization of λ.

In practice, the optimization of λ in this algorithm is performed by a Nelder-Mead [NM65] minimization approach, which happens to converge very quickly on this problem (`fminsearch` function in Matlab©).

Through the use of this modified version of PRESS, the OP-ELM has an L^2 penalty on the regression weights (regression between the hidden and output layer),

for which the neurons have already been ranked using an L^1 penalty. Figure 2.7 is a modified version of Figure 2.5 illustrating the TROP-ELM approach.

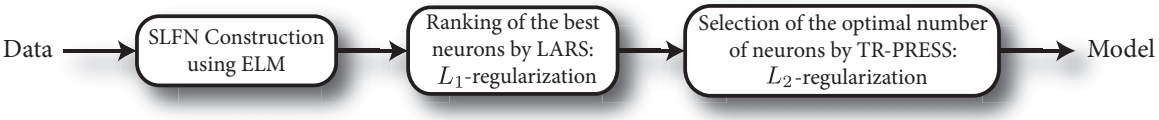


Figure 2.7: The proposed regularized OP-ELM (TROP-ELM) as a modification of Figure 2.5.

Figure 2.8 illustrates the effect of the regularization factor introduced in the TR-PRESS: the Mean Square Error is more stable regarding the increase of the number of neurons following the ranking provided by LARS (L^1 penalty). The introduction of the L^2 penalty has a very visible regularization effect here (the situation is similar for the other datasets), avoiding numerical instabilities, for example.

The following section provides a comparison of the modified OP-ELM (denoted TROP-ELM for Tikhonov-Regularized OP-ELM) with the original OP-ELM on 11 different regression data sets from the UCI Machine Learning Repository [Lic13], along with other classical Machine Learning methods.

2.3.4 Experiments

In order to compare the proposed TROP-ELM with the original OP-ELM and other typical Machine Learning algorithms, eleven data sets from UCI Machine

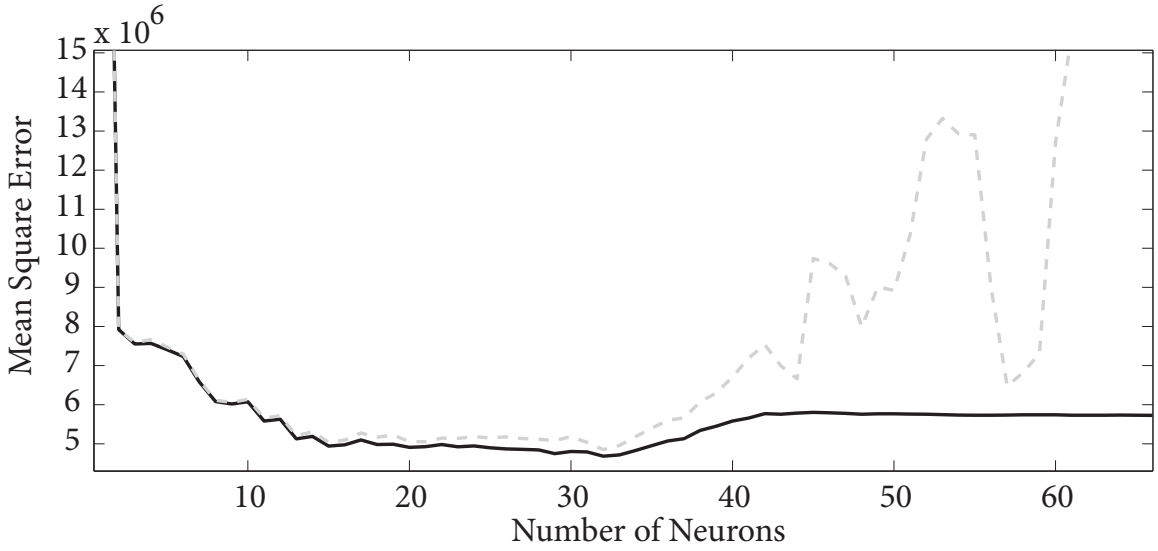


Figure 2.8: Comparison of the MSE for the original OP-ELM and the proposed TROP-ELM for Auto Price dataset for a varying amount of neurons. OP-ELM is shown as grey dashed line, TROP-ELM as solid black line, and the neurons are ranked by the LARS. The regularization enables here to have a more stable MSE along the increase of the number of neurons.

Learning Repository [Lic13] have been used. They are chosen similarly to those in section 2.2.4 for comparison purposes. Table 2.7 summarizes the details of each data set.

The data sets have all been processed in the same way: for each data set, ten different random permutations are taken without replacement; for each permutation, two thirds are taken for the training set, and the remaining third for the test set (see Table 2.7). Training sets are then normalized (zero-mean and unit variance) and test sets are also normalized using the very same normalization factors than

Table 2.7: Details of the data sets used, along with the number of variables.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
# of Variables	8	5	6	12	15	6
Training	2784	4752	6344	5461	106	139
Test	1393	2377	3173	2731	53	70

	Servo	Breast C.	Bank	Stocks	Boston
# of Variables	4	32	8	9	13
Training	111	129	2999	633	337
Test	56	65	1500	317	169

for the corresponding training set. The results presented in the following are hence the average of the ten repetitions for each data set. This also enables to obtain an estimate of the standard deviation of the results presented (see Table 2.8).

Experiments are performed using the available versions of the methodologies, unaltered. All experiments have been run on the same $x86\_64$ Linux machine with at least 4 GB of memory (no swapping for any of the experiments) and 2+ GHz processor. Also, even though some methodologies implementations are taking advantage of parallelization, computational times are reported considering single-threaded execution on one single core, for the sake of comparisons.

The SVM is performed using the SVM toolbox [CL11]; MLP [Bis96] is using a neural network toolbox, part of the Matlab© software from the MathWorks, Inc; the GPML toolbox for Matlab© from Rasmussen and Williams [RW06] is used for the GP; finally, the OP-ELM was used with all possible kernels, linear, sigmoid, and Gaussian, using a maximum number of 100 neurons and similarly for the TROP-ELM.

For more details on the parameters used for each toolbox, please refer to [MSB+10].

First are reported the Mean Square Errors (and standard deviations) for the six algorithms tested. It can be seen that the proposed TROP-ELM is always at least as good as the original OP-ELM, with an improvement on the standard deviation of the results, over the ten repetitions for each data set (only for the Boston Housing case is the standard deviation larger for the TROP-ELM than the OP-ELM): over the eleven data sets, the TROP-ELM performs on average 27% better than the original OP-ELM and gives a standard deviation of the results 52% lower than that of the OP-ELM (also on average over the eleven data sets).

Also, the TROP-ELM is clearly as good (or better) as the GP in six out of the eleven data sets —Ailerons, Elevators, Auto Price, Breast Cancer, Bank and Boston— in which cases it has a similar (or lower) standard deviation of the results. This with a computational time usually two or three orders of magnitude lower than the GP.

Table 2.9 gives the computational times for each algorithm and each data set (average of the ten repetitions). It can be seen that the TROP-ELM keeps computational times of the same order as that of the OP-ELM (although higher on average), and remains several orders of magnitudes faster than the GP, MLP or SVM. Of course, as for the OP-ELM, the computational times remain one to two orders of magnitude above the original ELM.

Finally, Table 2.10 reports the average number of neurons (average over the ten repetitions for each data set) selected for the final model structure of the OP-

Table 2.8: Mean Square Error results (*boldface*) and standard deviations (*regular*) for all six methodologies for regression data sets. "Auto P." stands for Auto Price and "Breast C." for Breast Cancer data sets.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
SVM	**4.5**	**1.3e-7**	**6.2e-6**	**1.2e+2**	**2.8e+7**	**6.5e+3**
	2.7e-1	2.6e-8	6.8e-7	8.1e+1	8.4e+7	5.1e+3
MLP	**4.6**	**2.7e-7**	**2.6e-6**	**9.8**	**2.2e+7**	**1.4e+4**
	5.8e-1	4.4e-9	9.0e-8	1.1	9.8e+6	1.8e+4
GP	**4.5**	**2.7e-8**	**2.0e-6**	**7.7**	**2.0e+7**	**6.7e+3**
	2.4e-1	1.9e-9	5.0e-8	2.9e-1	1.0e+7	6.6e+3
ELM	**8.3**	**3.3e-8**	**2.2e-6**	**4.9e+2**	**7.9e+9**	**4.7e+4**
	7.5e-1	2.5e-9	7.0e-8	6.2e+1	7.2e+9	2.5e+4
OP-ELM	**4.9**	**2.8e-7**	**2.0e-6**	**3.1e+1**	**9.5e+7**	**5.3e+3**
	6.6e-1	1.5e-9	5.4e-8	7.4	4.0e+6	5.2e+3
TROP-ELM	**4.8**	**2.7e-8**	**2.0e-6**	**2.4e+1**	**7.0e+6**	**4.1e+3**
	4.2e-1	1.5e-9	5.2e-8	6.2	2.2e+6	2.9e+3

	Servo	Breast C.	Bank	Stocks	Boston	
SVM	**6.9e-1**	**1.2e+3**	**2.7e-2**	**5.1e-1**	**3.4e+1**	
	3.3e-1	7.2e-1	8.0e-4	9.0e-2	3.1e+1	
MLP	**2.2e-1**	**1.5e+3**	**9.1e-4**	**8.8e-1**	**2.2e+1**	
	8.1e-2	4.4e+2	4.2e-5	2.1e-1	8.8	
GP	**4.8e-1**	**1.3e+3**	**8.7e-4**	**4.4e-1**	**1.1e+1**	
	3.5e-1	1.9e+2	5.1e-5	5.0e-2	3.5	
ELM	**7.1**	**7.7e+3**	**6.7e-3**	**3.4e+1**	**1.2e+2**	
	5.5	2.0e+3	7.0e-4	9.35	2.1e+1	
OP-ELM	**8.0e-1**	**1.4e+3**	**1.1e-3**	**9.8e-1**	**1.9e+1**	
	3.3e-1	3.6e+2	1.0e-6	1.1e-1	2.9	
TROP-ELM	**6.1e-1**	**1.1e+3**	**1.1e-3**	**8.4e-1**	**1.9e+1**	
	2.2e-1	1.7e+2	3.4e-5	5.8e-2	4.4	

Table 2.9: Computational times (in seconds) for all five methodologies on the regression data sets. "Auto P." stands for Auto Price and "Breast C." for Breast Cancer data sets.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
SVM	6.6e+4	4.2e+2	5.8e+2	3.2e+5	2.6e+2	3.2e+2
MLP	2.1e+3	3.5e+3	3.5e+3	8.2e+3	7.3e+2	5.8e+2
GP	9.5e+2	2.9e+3	6.5e+3	6.3e+3	2.9	3.2
ELM	4.0e-1	9.0e-1	1.6	1.2	3.8e-2	4.2e-2
OP-ELM	5.7	16.8	29.8	26.2	2.7e-1	2.0e-1
TROP-ELM	12.2	14.6	44.3	13.9	4.8e-1	1.2

	Servo	Breast C.	Bank	Stocks	Boston	
SVM	1.3e+2	3.2e+2	1.6e+3	2.3e+3	8.5e+2	
MLP	5.2e+2	8.0e+2	2.7e+3	1.2e+3	8.2e+2	
GP	2.2	8.8	1.7e+3	4.1e+1	8.5	
ELM	3.9e-2	4.8e-2	4.7e-1	1.1e-1	7.4e-2	
OP-ELM	2.1e-1	4.2e-1	8.03	1.54	7.0e-1	
TROP-ELM	8.4e-1	7.8e-1	4.4	1.1	1.5	

ELM and TROP-ELM. It can be seen that only in the cases of Computer Activity and Stocks data sets are all the neurons selected for the final model (which suggests that a larger number of neurons given initially to the model might lead to better results). Otherwise, the selected amount varies largely over the data sets and slightly between the OP-ELM and TROP-ELM.

The effect of the L^1 penalty is here obvious, on the number of neurons retained in the final model structure (compared to the ELM or Regularized ELM, for example, which are less parsimonious), while the L^2 penalty introduced enables to regularize the weights chosen and improve the performances of the final model.

Table 2.10: Average number of neurons selected for the final model for both OP-ELM and TROP-ELM. Average computed over 10 repetitions with 100 initial neurons.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
OP-ELM	36	75	74	100	14	33
TROP-ELM	42	80	53	100	15	28

	Servo	Breast C.	Bank	Stocks	Boston	
OP-ELM	36	12	98	100	66	
TROP-ELM	42	14	93	100	59	

CHAPTER 3
NOVEL EXTENSIONS OF EXTREME LEARNING MACHINES

Extreme Learning Machine is an extremely fast method. Once the hyper-parameters are found with some way of regularization, training a model takes milliseconds for small datasets. Even for large datasets, finding the output weights is a linear problem — and it allows to apply all sorts of tricks from linear algebra.

In general, Machine Learning methods are designed to be trained $\mathcal{O}(1)$ times. A notable exception is linear model, but it cannot capture nonlinear dependencies by its own nature. An ELM has both nonlinear power and a training speed approaching that of a linear model. In practice, in a reasonable time (minutes to hours on a single computer workstation, depending on a data size) for N data samples an ELM may be fully re-trained $\mathcal{O}(N)$ times, or partially re-trained using some approximation even an $\mathcal{O}(N^2)$ times.

The ability of obtaining multiple trained models (and multiple prediction errors, for example) provides possibilities to implement various algorithms that are considered infeasible with other Machine Learning techniques. This chapter presents two examples of such approaches, and lists further directions of research in its concluding section.

3.1 Data Visualization with ELMs

Data visualization is an old problem in Machine Learning [LV07]. High-dimensional data is ubiquitous in the modern world, but it stays virtually impenetrable for human analysis, except for images or video. For the exploratory data analysis of an arbitrary high dimensional data, a suitable visualization should be created. It is commonly restricted to two or three dimensions, which are easier to show, but for the visualization to be useful it must be representative of the original data. As plotting two- or three-dimensional data in preferred way is an easy task with modern programming packages, the dimensionality reduction is the object of investigation of this chapter.

The proposed ELM visualisation method, denoted *ELMVIS* for convenience, maps the data points to some fixed points - prototypes, in the visualization space. Their exact position is weakly relevant to data, and may be chosen arbitrarily, for example as a grid or normally distributed points. Then the prototypes are randomly assigned to data points, and an ELM is used to estimate the reconstruction error. To find the visualization, several points are chosen, then their assignment is permuted, and the error is re-estimated. Any better solution found is kept, otherwise the permutation is abandoned. While the exact solution requires a factorial number of trials (all possible permutations of N points), experiments show acceptable convergence rate with up to several hundred points due to a fast reconstruction error estimation with ELM. Benefits of the method are its generality and only one parameter being the number of neurons in ELM which does not require exact tuning.

3.1.1 State of the Art

Various methods can be utilized for a data visualization task. A common assumption in dimensionality reduction, and especially in data visualization, is that the original data points lie on a low-dimensional manifold. If the assumption holds, then the points of a manifold may be mapped onto a low-dimensional visualization space with small information loss.

The naive dimensionality reduction method is variable (feature) selection, but a few selected variables could present only a part of the data structure, if any. Other dimensionality reduction methods optimize a selected criterion, with different criteria resulting in two different algorithms.

Linear dimensionality reduction methods such as Principal Components Analysis (PCA) [Pea01] and linear Multidimensional Scaling (MDS) [Kru64] yield the same results, as proven in [LV07]. Their criterion is variance maximization which works for datasets with linear dependencies, but the general performance may be poor.

If the variables are relevant but correlated (which is often the case), the dimensionality of data is higher than necessary. Then the same data could be explained by a smaller set of transformed variables, and is said to lie on a manifold [LV07]. As an example, one can imagine a camera rotating around an object at a fixed distance, then the pictures of that camera would lie on a 2-dimensional manifold (sphere), while their actual dimensions would be much higher. Many nonlinear dimensionality reduction methods, including those listed in the next section, aim to find and unfold such a

manifold using various cost functions and training algorithms. Even PCA would find a manifold in the data, if the data is linear. Manifolds are commonly found by preserving the neighborhood in original and reduced spaces. Topology-preserving methods that use graph distances, like Curvilinear Distance Analysis (CDA) [LLDV00, LLV04], normally provide excellent results for un-foldable manifolds.

Without evident manifold structure, or if the dimensionality of manifold is still higher than the one of a reduced space, topology-preserving methods lose their point. And in a very high dimensional space, neighbourhood rank is a weak metric [BGRS99]. This is caused by an empty space phenomenon [ST83] and the curse of dimensionality, studied thoroughly in [BGRS99]. The problem comes from the change of the distribution of distances between points in space as the dimensionality goes up. Distances between points in a dataset are typically normally distributed. With the increase of a space dimensionality, the mean of that normal distribution increases whereas the variance stays the same. It causes the distribution to concentrate around some value, and reduces the distance differences between various ranked neighbors, making the nearest neighbors unstable already at 10-20 dimensions [BGRS99]. These cases require a nonlinear dimensionality reduction method with general cost function without other assumptions. The proposed Extreme Learning Machine (ELM)-based visualization method uses natural reconstruction error, while the nonlinearity of ELM provides the desired nonlinear projection.

The visualization methods may be divided into two major groups, separated by whether they try to keep distances of topology structure. Distance-preserving meth-

ods include Multidimensional Scaling (MDS) [Kru64], which gives the same solution as PCA; Sammon's mapping [Sam69]; Curvilinear Component Analysis (CCA) [DH97]; Isomap [Ten98, TdL00]; Curvilinear Distance Analysis (CDA) [LLV04], and Kernel PCA [SSSM98]. Topology preserving methods are Self-Organizing Maps (SOM) [Koh82]; Generative Topographic mapping (GTM) [BSW98]; Locally Linear Embedding (LLE) [RS00]; Laplacian Eigenmaps [BN01, BN03], Isotop [LAV03] and Neighbor Retrieval Visualizer (NeRV) [VPN+10]. Out of these, the three benchmark methods selected are PCA, SOM and NeRV.

Visualization Quality Measures

There are different ways to measure and compare the quality of a visualization. The Mean Squared Error (MSE) of reconstruction came from the dimensionality reduction, and is a universal measure of quality. However, it requires a reversed projection from visualization to the original data space, which not all the methods can provide. So other quality measures are often used.

One of the common measures is precision and recall of a projection. It comes from the classification task, where the definitions are:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.1}$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.2}$$

The visualization task has no classes, but they are created manually by setting all points within a certain neighborhood as $+1$ class, and the others as -1

class [KP11], as shown on Figure 3.1. As in visualization both precision and recall depend on the size of a neighborhood used for their calculation, which is not the case in classification, other similar measures are used: continuity is similar to precision, and trustworthiness to recall [VK01].

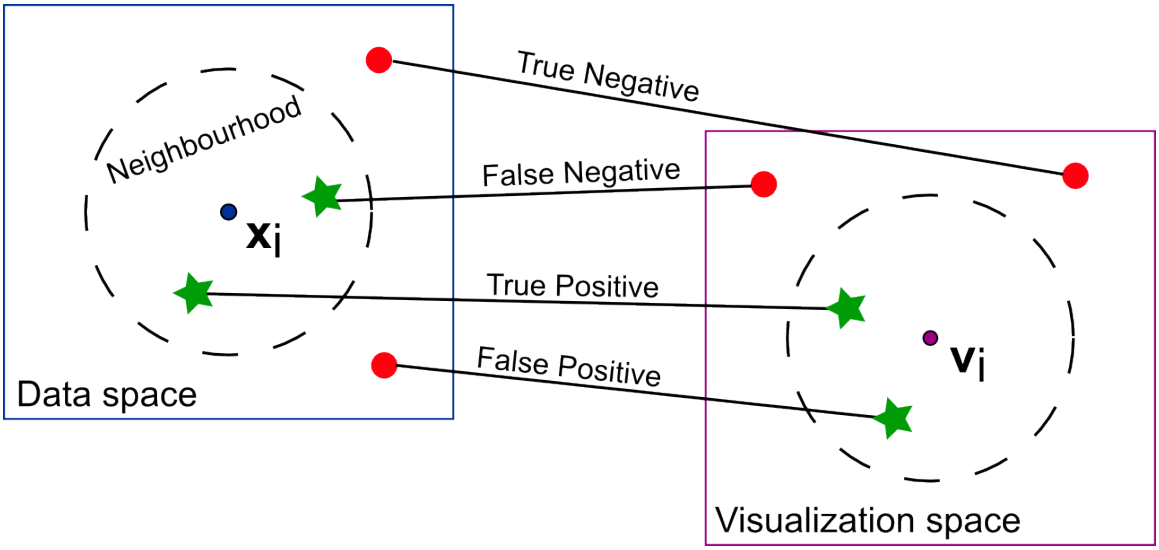


Figure 3.1: Four types of samples considered in calculating precision and recall, defined for visualization. Stars are samples in the neighbourhood, and circles are the ones outside.

Another method, called Mean Relative Rank Error (MRRE), is a neighborhood preservation ratio. Based on the ideas from, among others, [GS96, BP92, BHV99], and refined by [LV07], this measure displays the average normalized error in ranking within k nearest neighbors. The normalization puts the measure in range between 0 and 1, where 0 corresponds to the perfect match of the first k neighbors, and 1 to

the replacement of the first k neighbors by the most distant k points. Depending in which space the closest k neighbors are chosen for calculation, two $MRRE's$ exist: $MRRE_{\mathbf{X}\rightarrow\mathbf{V}}(k)$ can be compared to continuity, and $MRRE_{\mathbf{V}\rightarrow\mathbf{X}}(k)$ to trustworthiness.

And the last but not the least, a plot of visualized points may be used as a measure of goodness [LV07]. This is especially true if data points can be observed directly such as with images, then the user can estimate the quality of clustering by simply browsing the visualized data.

Principal Components Analysis

Principal Components Analysis (PCA) is a linear method, which has an exact and relatively fast solution. Given the dataset $\mathbf{X}$ with N samples as rows of $\mathbf{X}$ and d features as columns of $\mathbf{X}$, PCA decomposes the covariance matrix $\mathbf{C}_{xx}$ into eigenvectors $\mathbf{U}$ and eigenvalues $\mathbf{\Lambda}$. Eigenvalues of $\mathbf{\Lambda}$ are ranked from largest to smallest, and the corresponding eigenvectors in $\mathbf{U}$ are placed accordingly.

$$\mathbf{C}_{xx} = \mathbf{X}^T\mathbf{X} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U} \tag{3.3}$$

$$\mathbf{V} = \mathbf{X}\mathbf{U}_{:,1:k} \tag{3.4}$$

where $\mathbf{V}$ are points in the visualization space, and $\mathbf{U}$ has only the first k columns.

PCA projects data points to the dimensions of the largest variance. Its advantages are simplicity, robustness and lack of parameters. The main drawback of PCA is its linearity which captures a linear manifold, but nonlinear manifolds would

be squashed using PCA. This leads to poor data mapping, and large reconstruction errors. Also the obtained mapping have orthogonal dimensions, which is not always desired. Furthermore, PCA has perfect recall but precision may be poor.

Self Organizing Maps

Self-Organizing Maps (SOM) [Koh82] is a powerful method for visualization, trained iteratively. It includes a mandatory vector quantization [MAV+15], which is useful especially with large number of data samples, as it reduces noise and can reveal the shape of a manifold. SOM has a very intuitive and easy to understand algorithm: it starts with a low-dimensional grid, conforming to the data in the same way as wrapping a ball with a sheet of paper. When one point of a SOM's grid moves, its neighbors in the lattice move too, in the same direction. This keeps the grid coherent in the data space.

SOM is initialized by setting a lattice in the visualization space (usually two dimensions with square of hexagon grid), and randomly initializing the lattice points, or prototypes, in the data space. A training algorithm runs through the entire data matrix $\mathbf{X}$ several times. For each data point $\mathbf{x}_i$, an index r of the closest prototype is found, and data space coordinates $\mathbf{c}$ for all prototypes are updated using the following equations:

$$r = \arg\min_s d(\mathbf{x}_i, \mathbf{c}_s) \tag{3.5}$$

$$\mathbf{c}_s \leftarrow \mathbf{c}_s + \alpha v_\lambda(r, s)(\mathbf{x}_i - \mathbf{c}_s) \tag{3.6}$$

where d is a distance function (typically Euclidean), and α is a learning rate between 0 and 1. v_λ is called the neighborhood function, and it returns zeros for non-neighbors, and ones or other non-zero values for valid neighbors.

For visualization of results, prototypes of the original lattice are denoted accordingly to the points in data space which they represent. Common notation is color, though words or even images may be used as well. SOM is good in approximating well shaped manifolds [LV07]. One of the drawbacks is that points in the lattice are typically equally spaced, and distances between them give no idea about real distances between clusters in data space.

Neighbor Retrieval Visualizer

Neighbor Retrieval Visualizer (NeRV) [VPN+10] is created from the most general point of view on data visualization, with a specifically tailored cost function. NeRV defines visualization as an information retrieval task. Given any visualization space (projection space) sample $\mathbf{v}_i$ as a query, the low-dimensional visualization of data is used to retrieve its neighbors. Lets take the probabilistic approach, and assume that any point j may be retrieved as a neighbor, with probability $q_{j|i}$. The form of $q_{j|i}$ is unknown, but it follows several rules. First, it must be nonnegative and sum to one over j; therefore a suitable probability function can be:

$$q_{j|i} = \frac{exp(-f_{i,j})}{\sum_{k \neq i} exp(-f_{i,k})}, \; f_{i,j} \in \mathbb{R} \tag{3.7}$$

Then $f_{i,j}$ from (3.7) should be an increasing function of distance between $\mathbf{y}_i$ and $\mathbf{y}_j$, and be independent of other points $\mathbf{y}_k$, or a choice of particular point j.

However it may depend on i, which helps fit a distance measure to the particular areas of data space (sparsely and densely populated). Then the suitable form of $f_{i,j}$ would be $f_{i,j} = ||\mathbf{v}_i - \mathbf{v}_j||^2/\boldsymbol{\sigma}_i^2$ where the multiplier $1/\boldsymbol{\sigma}_i$ allows the function to grow at an individual rate for each point i; and gives the definition of the probabilistic model of retrieval:

$$q_{j|i} = \frac{exp(-\frac{||\mathbf{v}_i - \mathbf{v}_j||^2}{\boldsymbol{\sigma}_i^2})}{\sum_{k \neq i} exp(-\frac{||\mathbf{v}_i - \mathbf{v}_j||^2}{\boldsymbol{\sigma}_i^2})} \tag{3.8}$$

The probabilistic model of retrieval works in the visualization space only. To create a NeRV cost function, the counterpart model should be defined in the original data space. Lets call it $p_{j|i}$, a probabilistic model of relevance for the original data, defined as follows:

$$p_{j|i} = \frac{exp(-\frac{d(\mathbf{x}_i, \mathbf{x}_j)^2}{\boldsymbol{\sigma}_i^2})}{\sum_{k \neq i} exp(-\frac{d(\mathbf{x}_i, \mathbf{x}_j)^2}{\boldsymbol{\sigma}_i^2})}, \tag{3.9}$$

where $d(\cdot, \cdot)$ is any suitable distance measure in the original data. That form of relevance function allows usage of either data points themselves, or just a distance matrix of the data, if available.

Obviously, for a good visualization the neighborhood models of the original and the projected data should be as close as possible. Because they are defined as a distribution, a natural dissimilarity measure between distributions is the Kullback-Leibler divergence [KL51], defined as:

$$D(a_i, b_i) = \sum_{i \neq j} a_{j|i} log \frac{a_{j|i}}{b_{j|i}}, \tag{3.10}$$

where a and b are probability distributions.

Using the previous notations, $D(p_i, q_i)$ would become a *smoothed recall*, and $D(q_i, p_i)$ a *smoothed precision*. For a cost function of NeRV, the corresponding expectations are used instead of raw values, and a hyper-parameter λ to set a trade-off between precision and recall. The final cost function is presented in equation 3.11. Further details, as well as practical optimization matters, available in the original paper [VPN+10].

$$E_{NeRV} = \lambda \mathbb{E}_i[D(p_i, q_i)] + (1 - \lambda)\mathbb{E}_i[D(q_i, p_i)] \tag{3.11}$$

3.1.2 Methodology

The ELM in ELMVIS method is utilized as a nonlinear metric for reconstruction error, which stands for recall, or continuity [KP03] in data visualization field. Visualization is trained by random shuffling of some indexes and evaluating the reconstruction error using ELM. By replacing permutations in input (visualization) space with permutations in the order of the output data points, as described in subsection 3.1.2, the hidden layer of ELM is set constant. Thus it must be calculated only once, and the further training of ELM remains a fast linear task of one matrix multiplication. The following subsections describe each step in more details.

Data Visualization with ELM

The goal of ELMVIS method is to maximize the recall by the minimization of an MSE of a nonlinear reconstruction provided by an ELM. Given the N data points $\mathbf{x}_i \in \mathbb{R}^D$, compactly written as a matrix $\mathbf{X} = (\mathbf{x}_1^T \dots \mathbf{x}_N^T)^T$, the goal is to

find such points $\mathbf{v}_i \in \mathbb{R}^d$ (schematically shown on Figure 3.2), denoted as $\mathbf{V} = (\mathbf{v}_1^T \dots \mathbf{v}_N^T)^T$, which minimize the recall using the reconstruction error of ELM as a nonlinear metric. Typically d equals 2 or 3, while D could be large. Note that an ELM in the methodology performs an inverse projection $\mathbb{R}^D \leftarrow \mathbb{R}^d$ from low-dimensional visualization space to a high-dimensional original data space to estimate a reconstruction error; whereas other dimensionality reduction methods mostly use direct projection $\mathbb{R}^D \rightarrow \mathbb{R}^d$.

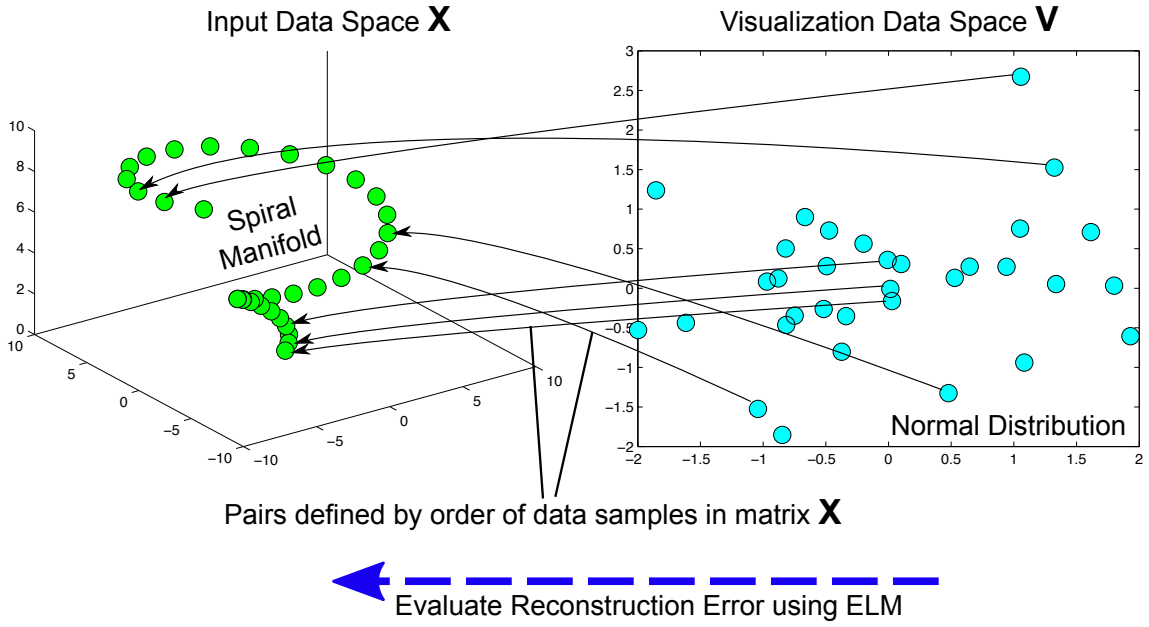


Figure 3.2: Projecting a high-dimensional spiral manifold data to a lower-dimensional visualization space. Visualization points are fixed, and only the pairing of the original and visualization data samples is changed.

ELM needs both input and output samples to be able to train. Data points $\mathbf{X}$ are already known, so one must set the visualization points $\mathbf{V}$. Because the manifold structure of a high dimensional data $\mathbf{X}$, if any, is unlikely to project well onto a 2- or 3-dimensional planes (except in artificially created datasets), the exact positioning of points $\mathbf{V}$ is not of a great importance. This allows fixing the positions of $\mathbf{V}$ at the beginning. Knowing $\mathbf{V}$ and $\mathbf{X}$, the only thing left to find is which point $\mathbf{v}_i$ corresponds to which point $\mathbf{x}_i$. The correspondence may be expressed as an ordering matrix $\mathbf{O}$. At initialization $\mathbf{O}^0$ is an identity matrix of size $N \times N$. Then some of its ones exchange indexes, like $(1_{i,i}\, 1_{j,j}) \rightarrow (1_{i,j}, 1_{j,i})$, which swaps samples $\mathbf{v}_i$ and $\mathbf{v}_j$ after application:

$$\mathbf{V}^{iter} \leftarrow \mathbf{V}^{iter-1}\mathbf{O}^{iter} \tag{3.12}$$

A general overview of the methodology is presented on a Figure 3.3. The ELMVIS starts by initializing N visualization space points $\mathbf{V}$, taken either from a Gaussian distribution or from a regular grid. Then an ELM is initialized, and the ordering matrix $\mathbf{O}$ is set to an identity matrix. An initial reconstruction MSE is calculated then. After that, an iteration starts by choosing a random number of samples out of N, a permuting the corresponding rows of $\mathbf{O}$. The ordering matrix $\mathbf{O}$ is applied to visualization points by multiplication, which permutes the samples in $\mathbf{V}$ the same way. Then the reconstruction error is re-calculated, but if it increases, the permutation of rows of $\mathbf{o}$ is rolled back; and a new iteration begins by again choosing a number of samples and permuting the corresponding rows in $\mathbf{O}$. Iterations keep repeating until the error decreases to the desired value, or reaching the iteration limit.

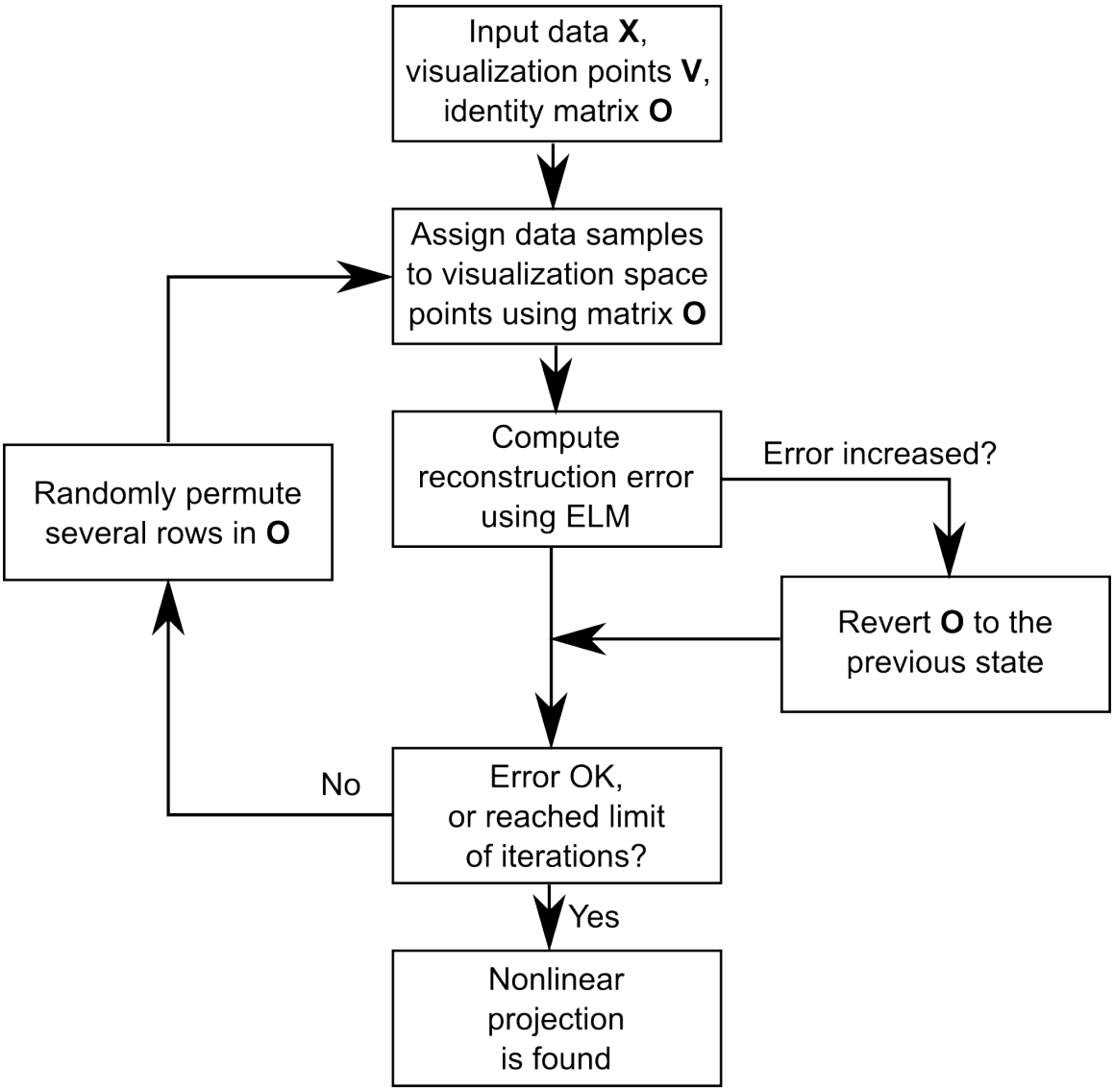


Figure 3.3: A general diagram of the ELMVIS training algorithm. An iteration includes permuting the order of several random points, and continues until convergence to required value of error, or a given number of times.

Adapting ELM for Data Visualization

The direct data visualization algorithm requires recalculation of the whole ELM. The most computationally costly part is a re-calculation of matrix $\mathbf{H}$, and its pseudo-inverse $\mathbf{H}^{\dagger}$. However, lets check again the structure of an ELM on Figure 3.4. It is easy to notice that for changes in $\mathbf{V}$, the whole ELM needs recalculating; while for changes in $\mathbf{X}$ the points $\mathbf{V}$ and a hidden layer representation $\mathbf{H}$ may remain constant, and only the output weight matrix needs to be updated.

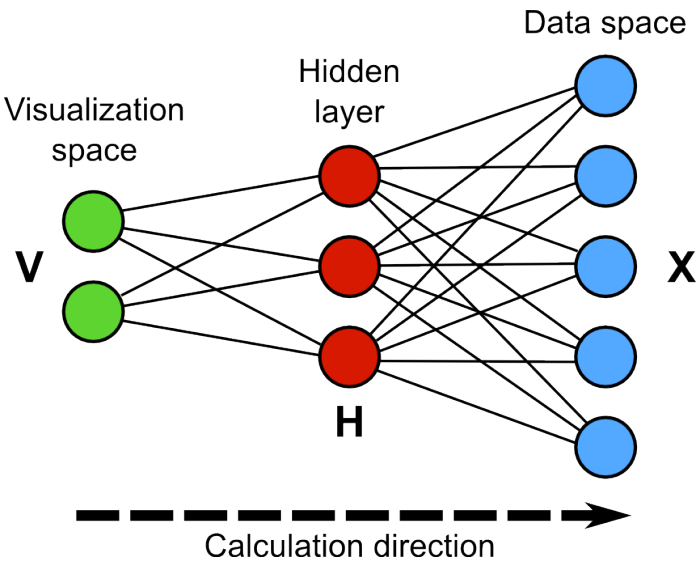


Figure 3.4: A schematic representation of ELM in ELMVIS. Changes in $\mathbf{V}$ require recalculation of the hidden matrix $\mathbf{H}$ and its pseudo-inverse, while for changes in $\mathbf{X}$ matrix $\mathbf{H}$ stays constant.

The reconstruction mean squared error MSE_{rec}

$$MSE_{rec} = \frac{1}{ND}\sum_{i=1}^{N}\sum_{j=1}^{D}(\hat{x}_{ij} - x_{ij})^2 \tag{3.13}$$

depends on the $\hat{\mathbf{x}}_i$, which is a prediction of an ELM, trained using data pairs $(\mathbf{v}_i, \mathbf{x}_i)$. But the solution of ELM is a linear system of equations, and the nonlinear part of ELM is applied to each transformed input vector separately of others. So the nonlinear mapping of an ELM is independent of the order of training pairs $(\mathbf{v}_i, \mathbf{x}_i)$, and so does the MSE_{rec}.

That fact allows to adapt the ELM in ELMVIS to cut the computational load. Multiplying an ordering matrix $\mathbf{O}$ with either $\mathbf{V}$ or $\mathbf{X}$ yield exactly the same new pairs $(\mathbf{v}'_i, \mathbf{x}'_i)$, though the order of pairs will differ. But because the reconstruction error does not depend on particular ordering of the pairs, these operations are interchangeable. So the proposed adaptation of ELM consists of replacing changes in $\mathbf{V}$ by changes in $\mathbf{X}$, as in (3.14).

$$(\mathbf{X}^{iter} \leftarrow \mathbf{X}^{iter-1}\mathbf{O}^{iter}) \Longrightarrow (\mathbf{V}^{iter} \leftarrow \mathbf{V}^{iter-1}\mathbf{O}^{iter}) \tag{3.14}$$

In the ELM structure of Figure 3.4, replacing changes in $\mathbf{V}$ by changes in $\mathbf{X}$ will keep the matrices $\mathbf{H}$ and $\mathbf{H}^{\dagger}$ constant. They need to be calculated only once on initialization; during iterations the reconstruction of $\mathbf{X}$ is obtained using the following rule:

$$\hat{\mathbf{X}} = \mathbf{H}\boldsymbol{\beta} = \mathbf{H}(\mathbf{H}^{\dagger}\mathbf{X}) = (\mathbf{H}\mathbf{H}^{\dagger})\mathbf{X} \tag{3.15}$$

Denoting a new matrix $\mathbf{H}_2 = \mathbf{H}\mathbf{H}^\dagger$ and calculating it at the initialization, the training of ELM on each iteration is reduced to a single matrix multiplication. That gives the necessary speed to run hundreds of thousands or even millions of iterations within a few minutes.

ELMVIS Algorithm

A summary of the ELMVIS data visualization method is presented on the algorithm 3.1. The output is the ordering $\mathbf{O}$, which is used to map the data points $\mathbf{X}$ on coordinates given by $\mathbf{V}$. The next section presents the results of an application of the method to several datasets, including a common spiral benchmark, as well as two high-dimensional faces datasets.

3.1.3 Experiments

The ELMVIS visualization methodology was tested on three datasets. The selected reference methods are PCA as the baseline, Self-Organizing Maps (SOM) [Koh82] as another method which uses fixed visualization points, and NeRV [VPN$^+$10] as a state-of-the-art nonlinear visualization method.

The primary comparison uses reconstruction error, an MSE of a reconstruction of the original data. A visualization method is assumed to have good performance, if its visualization have a low MSE_{rec}. Reverse projection of visualized data to the original space is required to obtain the error; for NeRV, which is the only method that does not provide such projection, it was learned from the representation in the similar way as in [LV07], using a separate ELM. The errors for all methods are gathered in

Algorithm 3.1 ELMVIS Algorithm

input data $\mathbf{X} \in \mathbb{R}^D$ with N samples

target value of MSE_{min}; or iteration limit $Iter_{max}$

select N visualization space points $\mathbf{V}\{Gaussian\ distributed,\ or\ on\ a\ grid\}$

select number of neurons nn of ELM to roughly match the complexity of dataset

calculate $\mathbf{H}$ using $\mathbf{V}$ $\{in\ training,\ \mathbf{V}\ only\ needed\ for\ this\ step\}$

calculate $\mathbf{H}_2 = \mathbf{H}\mathbf{H}^\dagger$

$MSE_{best} = 1$, $iter = 0$

while $MSE_{best} > MSE_{min}$ or $iter < Iter_{max}$ **do**

$iter \leftarrow iter + 1$

update $\mathbf{O}^{iter} \leftarrow \mathbf{O}^{iter-1}$ by randomly permuting $k < N$ of its rows

apply the ordering $\mathbf{X}' \leftarrow \mathbf{X}\mathbf{O}^{iter}$

obtain reverse projection $\hat{\mathbf{X}} = \mathbf{H}_2\mathbf{X}'$

estimate reconstruction error $MSE_{rec} = \frac{1}{ND}\sum_{i=1}^{N}\sum_{j=1}^{D}(\hat{x}_{i,j} - x'_{i,j})$

if $MSE_{rec} < MSE_{best}$ **then**

$MSE_{best} = MSE_{rec}$ $\{update\ the\ best\ error\}$

else

$\mathbf{O}^{iter} \leftarrow \mathbf{O}^{iter-1}$ $\{revert\ the\ permutation\}$

end if

end while

return ordering $\mathbf{O}^{iter}$ $\{best\ mapping\ of\ \mathbf{X}\ to\ visualization\ points\ \mathbf{V}\}$

table 3.1.

Table 3.1: MSE of reconstruction on all datasets in comparison. The best error of 100 restarts is shown for all methods except PCA, due to a random initialization procedure. ELMVIS initialization method shown in parentheses.

Dataset	PCA	SOM	NeRV	ELMVIS (Gaussian)	ELMVIS (PCA)
Spiral	0.482	0.054	0.011	0.049	0.060
Sculpture faces	0.980	0.916	0.769	0.718	0.724
Real faces	0.724	0.511	0.501	0.462	0.449

Spiral Dataset

The first dataset for testing is a spiral toy dataset, which is a common and relatively hard benchmark. The spiral is drawn in a two-dimensional space, and the goal is to project it into one dimension. It consists of $N = 100$ points, distributed evenly along its line by including a squared root term into the input data $\mathbf{X}$ equation:

$$\mathbf{X} = \begin{pmatrix} 2\sqrt{\alpha}\cos\left(\pi K\sqrt{\alpha}\right) \\ 2\sqrt{\alpha}\sin\left(\pi K\sqrt{\alpha}\right) \end{pmatrix}, \tag{3.16}$$

where α is distributed evenly between 0 and 1; K determines the amount of swings the spiral makes and is set to 3 in the experiment. The visualization points $\mathbf{V}$ are evenly distributed on a line; both $\mathbf{X}$ and $\mathbf{V}$ are normalized to have zero mean and unit variance. In this experiment, the amount of neurons of ELM and SOM is set to 5. Results on a spiral dataset for all the methods are presented in table 3.1. ELMVIS

model and data mapping is shown on Figure 3.5, and a reconstruction learned from NeRV results - on Figure 3.6.

The projection obtained by PCA is poor as expected, as PCA projecting from 2D into 1D would just squash the second dimension of a spiral along the direction of the largest variance. NeRV succeeded in finding a manifold, which it meant to do, thus expectedly showing great results even after estimating its mapping by a separate ELM, so it remains state-of-the-art on a spiral dataset. SOM showed good results as well. ELMVIS partially unfolds the spiral, but some parts remain torn and misplaced. Also eventual outliers appera (green dot on Figure 3.5) because the random permutation algorithm have not found the best solution in a given range of iterations. Still the results of ELMVIS on a spiral dataset are acceptable, far better than the naive PCA.

Convergence of ELMVIS

The experimental convergence speed of ELMVIS is tested, to get average values in an experimental setup instead of the exact solution (which is equivalent to the worst case scenario). Spiral test is the fastest of the three due to a smaller number of neurons and lower original data dimensionality, while convergence speed is independent of these values and only relies on the amount of test points. Note that the graphs here represent averages over many runs; while other results of ELM runs show the best outcome, corresponding to the best random initialization of a hidden layer of that ELM.

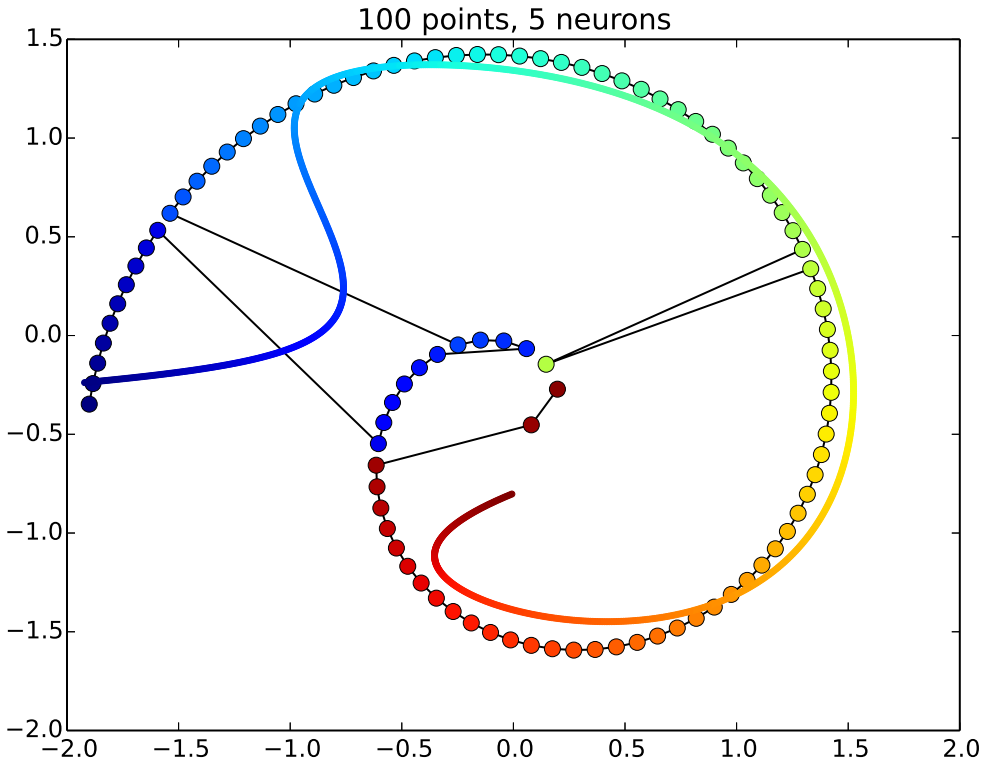


Figure 3.5: An example of ELMVIS fitting the spiral data. Thinner colour line is a back projection of ELM, black lines and colour denote the ordering of points. Some points are mapped wrongly because the solution is not exact.

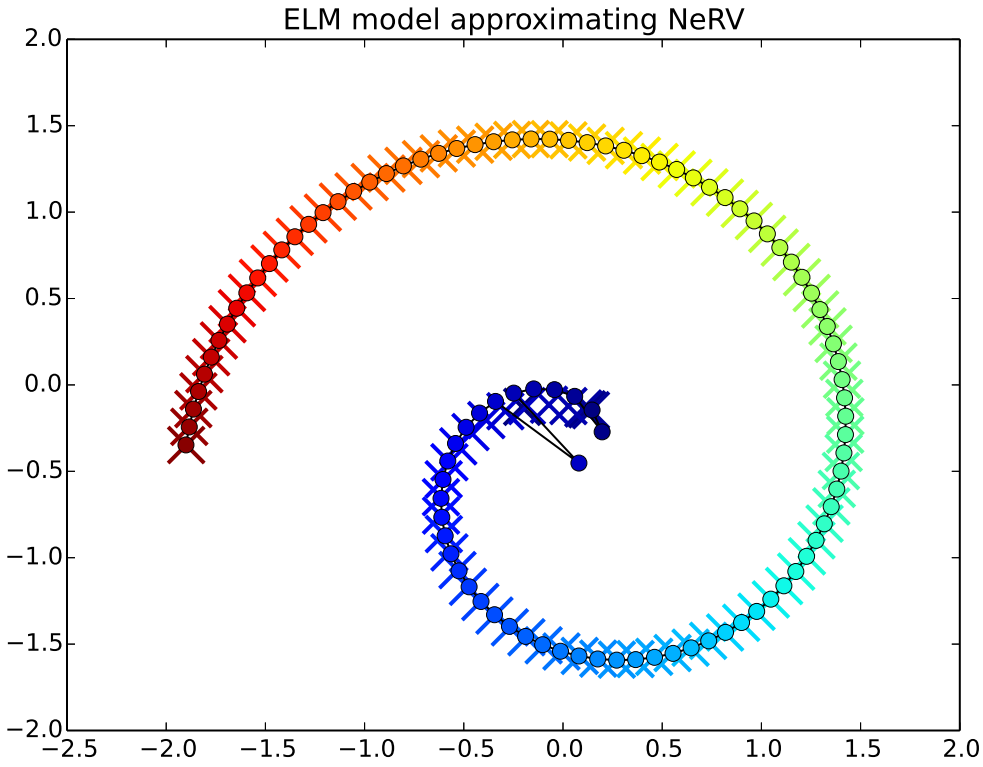


Figure 3.6: ELM reconstruction, learned from NeRV results. Only one point deviates from the perfect approximation. ELM model printed with crosses for visibility, as it mostly coincides with the data.

As it is stated in methodology section, complexity of exact solution of ELMVIS is factorial in the number of points, due to the random permutations training algorithm. The real speed of convergence was estimated on different sized subsets of the spiral data, ranging from 20 to 100 points. For each separate amount of points, 100000 training steps were performed, and the experiments and restarted 100 times with different initial pairings. The obtained convergence plot with average values and some standard deviations is presented on Figure 3.7.

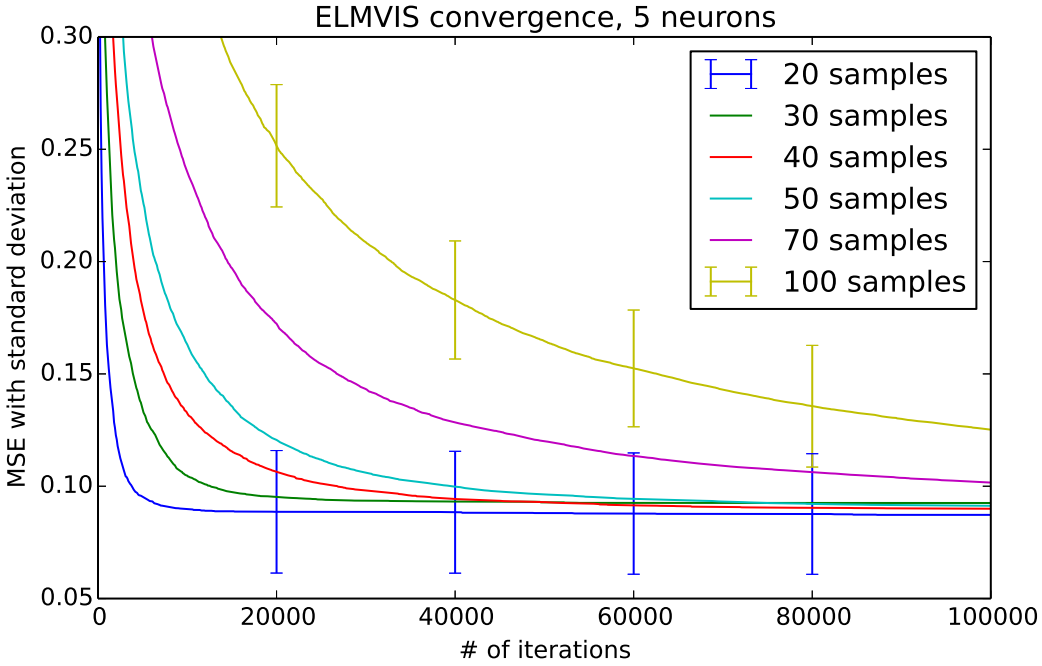


Figure 3.7: Convergence of the ELM visualization algorithm on the spiral dataset, with 100000 training steps and 100 restarts. Plots are ordered from 20 samples the lowest to the 100 the highest. Only some standard deviations are shown to avoid clutter.

As can be seen from a graph, in all settings the ELMVIS tends to converge roughly to the same reconstruction error. For 50 points the convergence already reached at iteration 60000, which is far less than the factorial of 50. The results show that the real convergence speed remains feasible for applications with low to medium amount of data samples.

Artificial Faces Dataset

A set of 698 face images is proposed in [TdL00], and then widely used for benchmark purposes, for instance in [LV07, VPN$^+$10]. These images are artificially generated faces rendered from a 3D sculpture head under different poses and lighting directions. Examples of faces are shown on a figure 3.8.

Figure 3.8: Some examples from the 698 sculpture face pictures from [TdL00].

Each image consists of an array of 64 by 64 brightness values of pixels, which gives the input data dimensionality of 4096. A preprocessing step is applied to reduce dimensionality with PCA; the first 240 principal components keep over 99%

of the global variance [LV07]. Such preprocessing step provides almost lossless data compression, and thus is applied to the sculptural faces dataset.

The resulting 240-dimensional data with 698 samples is projected to a two dimensional visualization space. ELMVIS uses 100 randomly selected samples at a time, and is repeated 100 times to get a true estimate. ELM and SOM use 20 neurons for visualization. The results are presented in table 3.1. The PCA totally fails to compress further the data, which is already compressed with another PCA. Two first principal components keep only 2% of variance. SOM gives bad results also, probably due to a lack of low-dimensional manifold in preprocessed data. So does NeRV, although it may be explained by a bad estimation of a reverse projection MSE obtained with ELM. ELMVIS gives the best results out of the three methods, but still the error is too high for the results to be good in absolute numbers.

An example visualization is shown on Figures 3.9,3.10. In visualization, NeRV mapping worked very well in grouping similar faces together, though the clusters are unnecessarily torn apart. ELMVIS also put some similar faces close to each other, but it seems not to care much about mirrored neighbours, and not all of the same neighbours are grouped in one cluster.

Real Faces Dataset

The data set of 1965 real faces [RS00] is gathered from frames of a video. Images of the dataset are grey-scale and 28x20 pixels if size, which gives 560 dimensional samples which are utilized as is. Figure 3.11 shows some random samples from the

Figure 3.9: Sculpture face images mapped to a grid using the same ELMVIS with 20 neurons. Small clusters of similar faces can be observed, for instance in the top left corner.

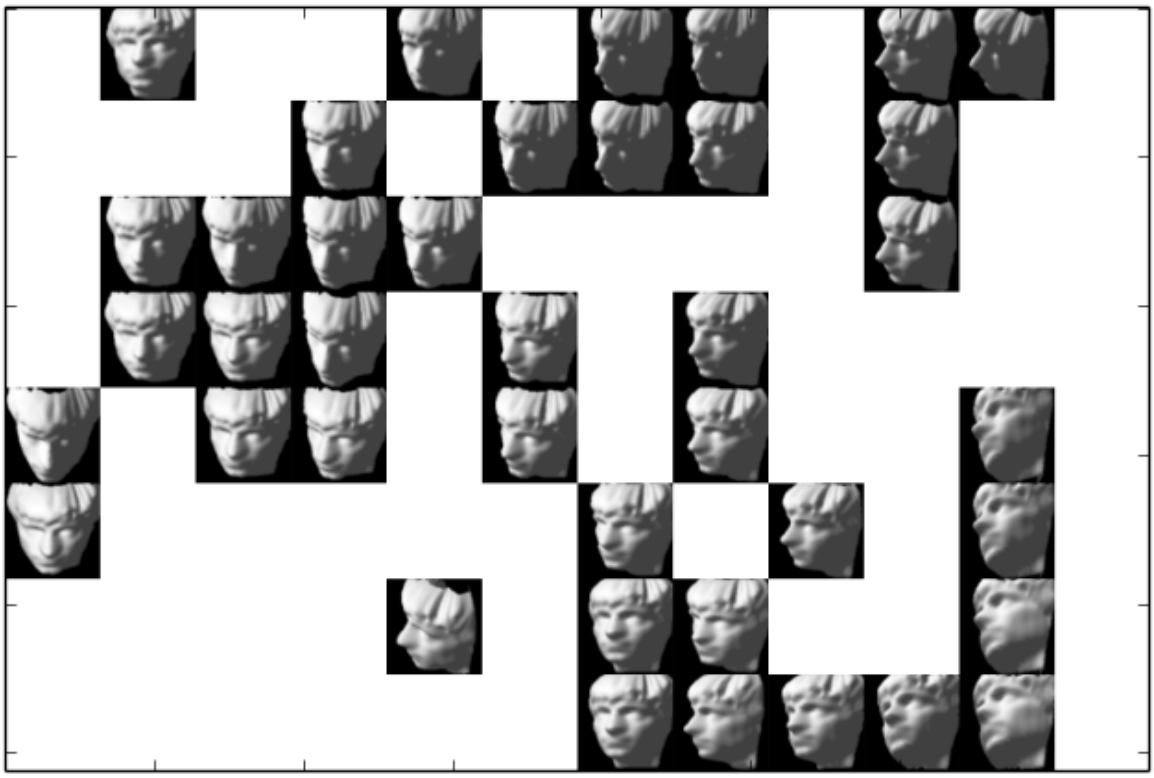


Figure 3.10: Sculpture face images mapped to a grid using the NeRV results. If several faces correspond to the same grid cell, a random one is displayed.

real faces image set.

Figure 3.11: Random examples from the 1965 real faces proposed in [RS00].

As images have 560 features, number of neurons in ELMVIS and SOM was raised to 40. The ELM visualization method was run with visualization points taken from a Gaussian distribution, or two first principal components of PCA. Reconstruction MSE for all the methods are given in table 3.1.

Here PCA gives the worst results, as expected. SOM performed similarly to NeRV in terms of a reconstruction MSE. ELMVIS showed the best results with both Gaussian and PCA initializations. A visualization of ELMVIS with 400 data samples to a 20x20 regular grid is presented on Figure 3.12; mapping of the real faces data obtained by NeRV and split into the same grid is shown on Figure 3.13.

Computational Time

Short experiments were run on a laptop with 2,2GHz Inter Core i7 CPU; longer jobs presented above were performed using computer resources within the Aalto University School of Science "Science-IT" project. Computational speed of ELMVIS

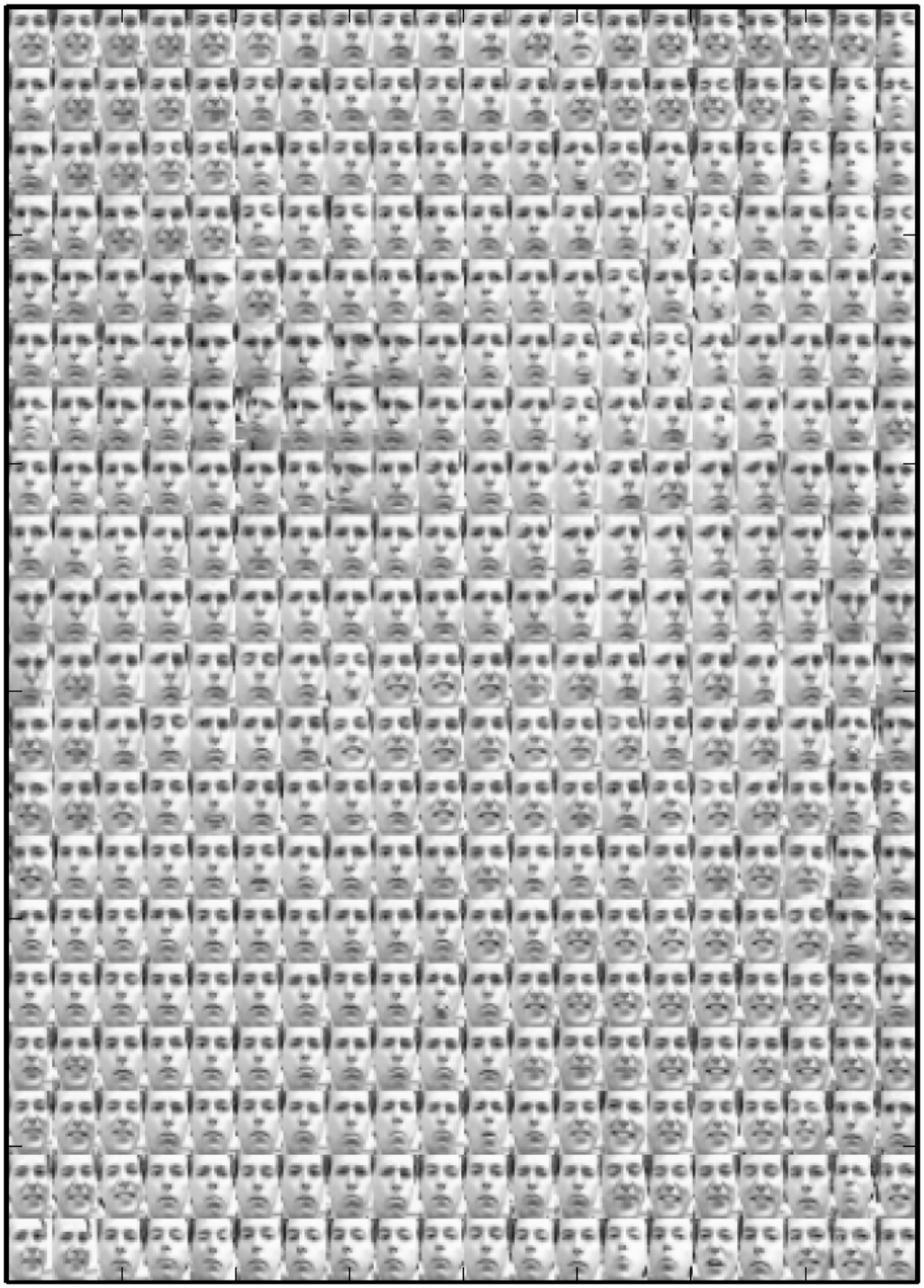


Figure 3.12: Subset of 400 real faces mapped to a 20x20 regular grid by ELMVIS. Clustering of the data is easily seen, although clusters are not unique due to a high nonlinearity of ELM.

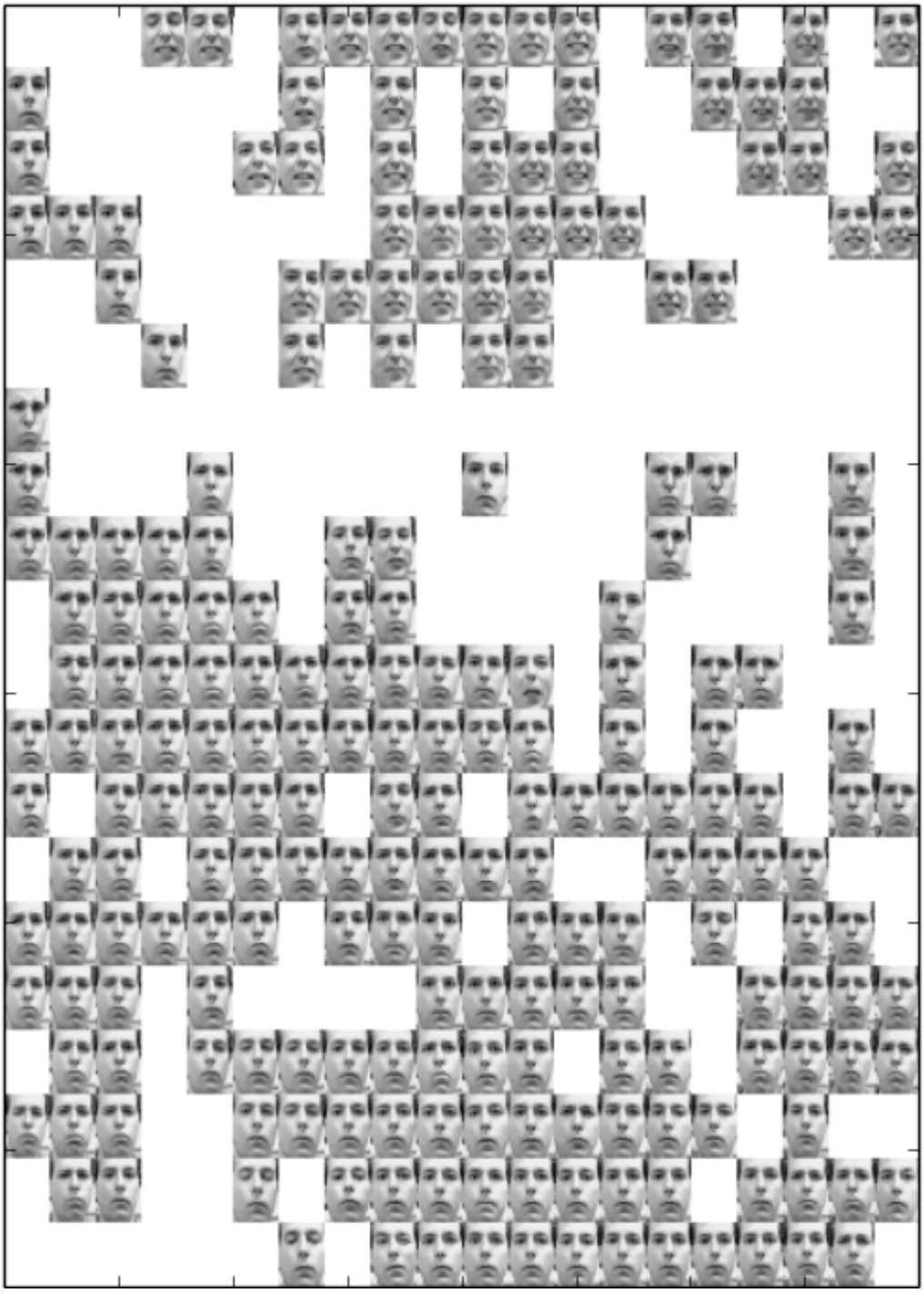


Figure 3.13: Whole set of real faces visualized by NeRV. Results are displayed on a 20x20 grid, if several images occupy a cell a random one is shown. Clustering of data is observed, though clusters are not solid.

depends heavily on the number of neurons and the output data dimensionality, and ranges from 300,000 iterations per minute on a laptop for easy tasks to 10,000 iterations per minute on the Aalto University computational cluster for the largest visualizations. Other methods were tested on a laptop, and took less that 10 minutes in all setups.

3.2 Fast Visualization Method ELMVIS+

The ELMVIS+ method formulates data visualization as an assignment problem [BDM12] of the data samples to the same number of given visualization points, which are fixed. An ELM model learns the de-projection of visualization points back into the original data space, where a cost function is calculated. The ELM solution is expressed as a helper matrix **A** that captures nonlinear dependencies between all visualization space points. Multiplying the matrix of data samples by **A** provides the optimal ELM approximation of these data samples, irrespective of their order. The fixed visualization points cause the matrix **A** to be fixed, and computed only once.

An original assignment problem is a challenging NP-hard [BDM12] optimization task, similar to an open loop travelling salesman problem [GP02]. The optimization process uses the fact that matrix **A** presents the optimal ELM solution for any order of data samples. It computes delta of the error for swapping two random data samples, which has a closed form equation with the proposed cosine similarity-based error and can be evaluated millions of times per second. Swaps that reduce the error are applied to the data, resulting in a greedy optimization with complexity $\mathcal{O}(N^2)$.

ELMVIS+ learns similar visualization to the original ELMVIS, but the observed speed-up exceeds forty thousand times. The updated method provides a fast and useful way of data visualization onto arbitrary fixed set of points in the visualization space. It has only one hyper-parameter, that is the number of neurons in the ELM model. The local optimum problem may be solved by batch update, or multiple re-runs of the method. The new cost function works for very high-dimensional data.

3.2.1 Methodology

The ELMVIS+ is a visualization method based on Extreme Learning Machines (ELM). It uses predefined (and fixed) visualization points, and assigns data samples to them. ELM learns a reverse projection (de-projection) of visualization points onto the original data samples, and an error is computed in the original data space. Thus, ELM builds a nonlinear reconstruction of the data, and the method cost function is a reconstruction error which stands for recall, or continuity [KP03] in the data visualization field. The idea of ELMVIS+ is presented in Figure 3.14.

While the visualization points and their order is fixed, the order of data samples is not defined. ELMVIS+ method computes an error (a cost function of visualization) with the current order of data samples. The cost function is optimized by swapping two random data samples to change that order, and computing the error; a swap which lowers an error is kept and the ELM projection is updated accordingly. Swaps which increase an error are reverted. Swap and update are the two main steps of ELMVIS+, repeated until a good projection is found.

Optimization in ELMVIS+

An ELM in ELMVIS+ is trained on a dataset consisting of inputs $\mathbf{V}$ and targets $\mathbf{X}$. The data is arranged in pairs $(\mathbf{v}_i, \mathbf{x}_i),\ i \in [\![1, N]\!]$. The indexes i in $\mathbf{V}$ and $\mathbf{X}$ are given implicitly by the position of row i in the corresponding data matrix.

The optimization starts by selecting two random indexes $a, b : a \neq b,\ a, b \in [\![1, N]\!]$. Then two samples $\mathbf{x}_a, \mathbf{x}_b$ in $\mathbf{X}$ are swapped (exchanged) while $\mathbf{V}$ remains

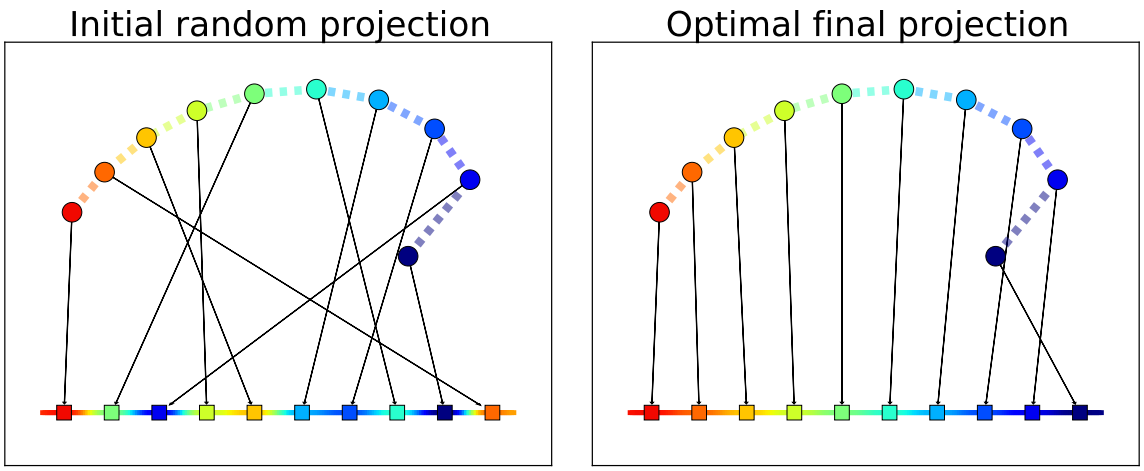


Figure 3.14: The idea of ELMVIS+: ELMVIS+ searches for an optimal assignment of data points $\mathbf{x}_i$ (circles) in a high-dimensional space (here $d = 2$) to fixed visualization points $\mathbf{v}_j$ (squares) in a lower-dimensional space (here $d = 1$). An ELM learns a projection $\mathbf{V} \rightarrow \mathbf{X}$ and estimates $\hat{\mathbf{X}} = f(\mathbf{V})$, used for error computation $E(\mathbf{X}, \mathbf{V}) = -\sum_{i=1}^{|\mathbf{X}|} cos(\hat{\mathbf{x}}_i, \mathbf{x}_i)$. An effect of **swapping** a pair of rows $(\mathbf{x}_a, \mathbf{x}_b)$ in $\mathbf{X}$ for random a, b is evaluated with a closed form equation for constant $\mathbf{V}$. If a swap decreases the error, the rows a, b are actually swapped in $\mathbf{X}$, and $\hat{\mathbf{X}}$ is **updated**. This changes an assignment of data points $\mathbf{X}$ to $\mathbf{V}$. Any visualization points can be used, and the initial assignment is random.

constant. This creates a new dataset, with a different mapping between $\mathbf{V}$ and the new $\mathbf{X}_{ab}$. The same ELM is trained on this new dataset, and outputs a new estimate $\hat{\mathbf{X}}_{ab}$. If the new estimate $\hat{\mathbf{X}}_{ab}$ is closer to the original data in $\mathbf{X}_{ab}$ than the previous one $\hat{\mathbf{X}}$ is to its original $\mathbf{X}$, then the swap is kept. Otherwise the swap is reverted by exchanging $\mathbf{x}_a$ and $\mathbf{x}_b$ again.

In practice, the change in error for swapping samples $\mathbf{x}_a$ and $\mathbf{x}_b$ is computed without modifying matrix $\mathbf{X}$ or explicitly re-training full ELM. Computing the change of error is called a swap step. If an error decreases for a particular pair of (a, b), then the rows in $\mathbf{X}$ are actually swapped and $\hat{\mathbf{X}}$ is updated to $\hat{\mathbf{X}}_{ab}$. This is when an update step takes place. When ELMVIS+ terminates, matrix $\mathbf{X}$ has an optimal order of samples $\mathbf{x}_i$ for the given visualization points $\mathbf{v}_i$.

The effect of updates on $\hat{\mathbf{X}}$ is presented on Figure 3.15. Initially, an order of points $\mathbf{x}_i$ is random and projected points $\hat{\mathbf{X}}$ are far from the true samples $\mathbf{X}$. But even a single update moves $\hat{\mathbf{X}}$ noticeably closer to $\mathbf{X}$. After a few updates, points $\hat{\mathbf{X}}$ and $\mathbf{X}$ are reasonably close — but no pair of indexes (a, b) can further decrease an error, so the solution is locally optimal. It is possible to get a better solution by swapping five points on the right with five points on the left, but the method is limited to swapping only two points at a time. Swapping k points gives a space of possible swaps of size $\binom{N}{k}$, which is prohibitively large for searching if $k > 2$. So swaps with only two points are considered in ELMVIS+. Multiple different initializations help find a solution closer to the global optimum.

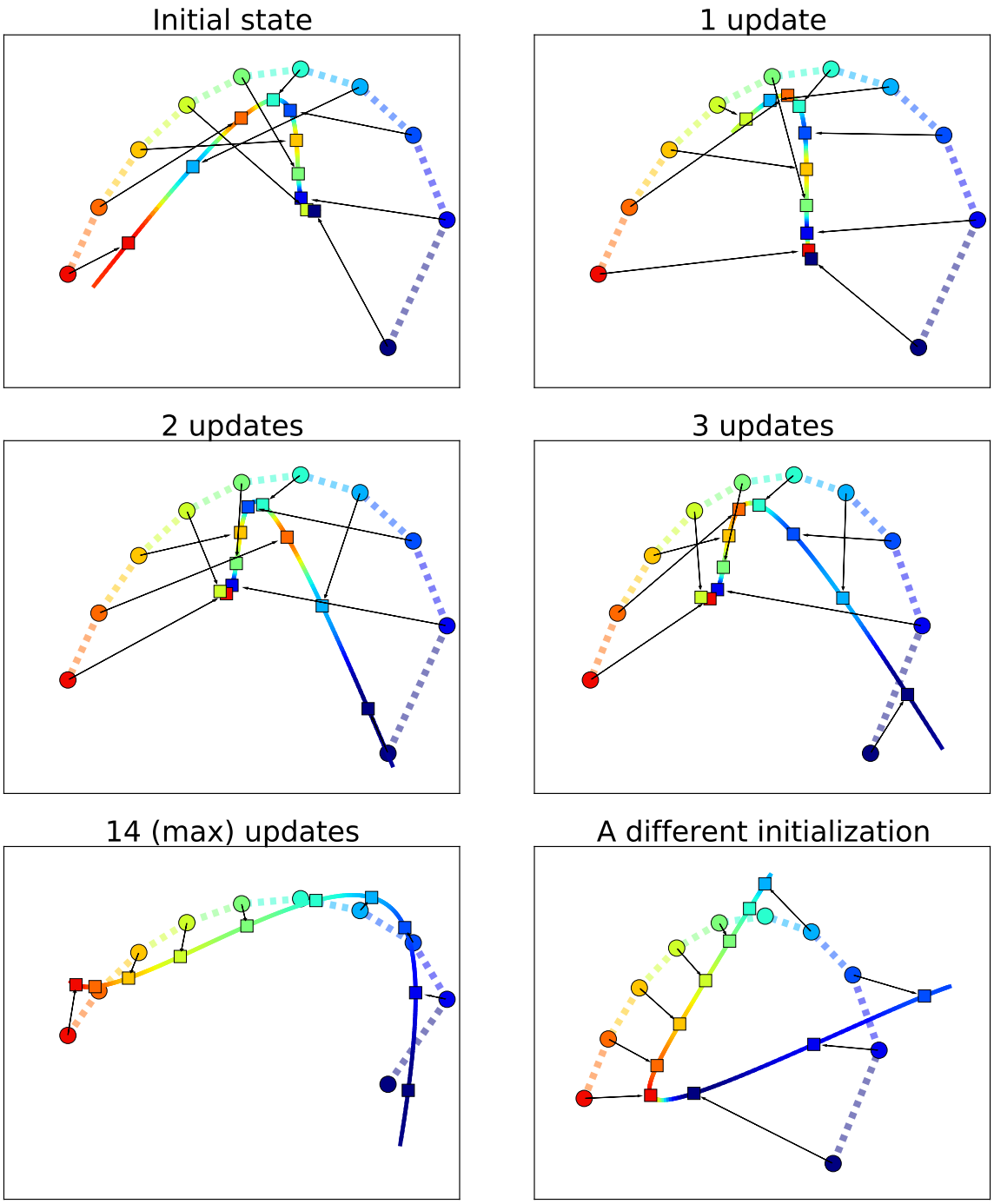


Figure 3.15: An example of ELMVIS+ optimization process. Two-dimensional data (dash line with circles) is visualized onto one-dimensional space. ELM de-projection of the visualization points into the original data space is shows by a solid line.

Fast Error Estimation from ELM

The error of ELMVIS+ method is the negative cosine similarity between samples $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$. The cosine similarity can be used because the absolute value of an error is irrelevant for the optimization, and the cosine similarity provides a convenient formula with good speedup over MSE.

The negative cosine similarity gives a fast and convenient way of evaluating error in the reverse-projecting ELM framework. A dot product between two vectors is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\| \cos\theta \tag{3.17}$$

Assume the input data is normalized to $\|\mathbf{x}\| = 1$, then $\|\hat{\mathbf{x}}\| \approx 1$ and

$$\text{similarity} = \cos\theta = \frac{\mathbf{x} \cdot \hat{\mathbf{x}}}{\|\mathbf{x}\|\|\hat{\mathbf{x}}\|} = \mathbf{x} \cdot \hat{\mathbf{x}} = \mathbf{x}^T\hat{\mathbf{x}} \tag{3.18}$$

For the whole data matrices $\mathbf{X}$, $\hat{\mathbf{X}}$ the error E, which is a negative cosine similarity, is:

$$E = -\text{trace}(\mathbf{X}^T\hat{\mathbf{X}}) \tag{3.19}$$

Here $\hat{\mathbf{X}}$ is an ELM prediction. However because the visualization points $\mathbf{V}$ are fixed, the output of an ELM hidden layer $\mathbf{H}$ never changes and has to be computed only once. A formula based on $\mathbf{H}$ is derived from the ELM solution:

$$\hat{\mathbf{X}} = \mathbf{H}\beta \tag{3.20}$$

$$\beta = \mathbf{H}^{\dagger}\mathbf{X} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X} \tag{3.21}$$

$$\hat{\mathbf{X}} = \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X} \tag{3.22}$$

$$\mathbf{X}^T\hat{\mathbf{X}} = \mathbf{X}^T\mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X} \tag{3.23}$$

$$\text{define } \mathbf{A} := \mathbf{H}(\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T \tag{3.24}$$

$$E = -\text{trace}(\mathbf{X}^T\mathbf{A}\mathbf{X}) \tag{3.25}$$

The matrix $\mathbf{A}$ in equation (3.25) is constant; it depends only on $\mathbf{H}$ and is computed once after an ELM model is built. It is re-used to find a change in error E for every swap of samples in $\mathbf{X}$. Minimizing equation (3.25) by changing the order of samples in $\mathbf{X}$ is similar to a quadratic assignment problem [BDM12], except the cost matrix $\mathbf{A}$ has size $(N \times N)$ instead of $(d \times d)$.

In fact, there is a closed form equation for the change of error Δ_E if a row $\mathbf{x}_a$ in matrix $\mathbf{X}$ is changed by an amount (vector) δ. The formula is given below. If only Δ_E is required, two last update steps are omitted.

$$\Delta_E = \sum_{j=1}^{d} \left(\mathbf{A}_{a,a}\delta_j^2 + 2\hat{\mathbf{x}}_{a,j}\delta_j \right) \tag{3.26}$$

$$\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} - \mathbf{A}_{:,a} \times \delta \tag{3.27}$$

$$\mathbf{X}_{:,a} \leftarrow \mathbf{X}_{:,a} + \delta \tag{3.28}$$

Swapping two samples in $\mathbf{X}$ is interpreted as changing two different rows in $\mathbf{X}$. But then the update of $\hat{\mathbf{X}}$ from equation (3.27) must take place between the changes,

and it is heavy on computation and memory operations. A modified Δ_E expression for swapping samples $\mathbf{x}_a$ and $\mathbf{x}_b$ without step (3.27) between them is:

$$\delta = \mathbf{x}_b - \mathbf{x}_a \tag{3.29}$$

$$\omega = \mathbf{x}_a - \mathbf{x}_b \tag{3.30}$$

$$\Delta_E = \sum_{j=1}^{d} \left(\mathbf{A}_{a,a}\delta_j^2 + 2\hat{\mathbf{x}}_{a,j}\delta_j + \mathbf{A}_{b,b}\omega_j^2 + 2\hat{\mathbf{x}}_{b,j}\omega_j + 2\mathbf{A}_{b,a}\delta_j\omega_j \right) \tag{3.31}$$

$$\hat{\mathbf{X}} \leftarrow \hat{\mathbf{X}} - \begin{pmatrix} \mathbf{A}_{:,a} & \mathbf{A}_{:,b} \end{pmatrix} \begin{pmatrix} \delta \\ \omega \end{pmatrix} \tag{3.32}$$

$$\mathbf{X}_{:,a} \leftarrow \mathbf{X}_{:,a} + \delta \tag{3.33}$$

$$\mathbf{X}_{:,b} \leftarrow \mathbf{X}_{:,b} + \omega \tag{3.34}$$

Equation (3.31) provides a difference in error if samples $\mathbf{x}_a$ and $\mathbf{x}_b$ are swapped. If this difference is negative, then the method proceeds to actual swap and update steps, given by the next three assignments. Otherwise, different random indexes (a, b) are chosen and Δ_E is evaluated again.

ELMVIS+ Algorithm

The ELMVIS+ method starts by taking a random permutation of $\mathbf{X}$ and training an ELM on $(\mathbf{V}, \mathbf{X})$ dataset. Then a fixed matrix $\mathbf{A}$ is computed from equation (3.25), and an initial data reconstruction $\hat{\mathbf{X}} = \mathbf{AX}$ is obtained. This is done only once when the ELMVIS+ method starts.

Then two random indexes $a, b \in [\![1, N]\!]$ are taken, and the change in error Δ_E is computed by formula (3.31). This is a swap step. If the change of error is negative

$\Delta_E < 0$, then an update step is performed which exchanges rows $\mathbf{x}_a, \mathbf{x}_b$ in $\mathbf{X}$, and updates $\hat{\mathbf{X}}$ to be the optimal reconstruction for the new $\mathbf{X}$. If the change of error is non-negative $\Delta_E \geq 0$, then swaps continue.

Swap and update steps run until some stopping criterion is met, such as a maximum number of swap steps without a single update, a maximum number of updates, or a runtime limit. When ELMVIS+ terminates, data samples in matrix $\mathbf{X}$ are in the optimal order for the given visualization points in $\mathbf{V}$. A visual scheme of ELMVIS+ method is shown on Figure 3.16.

The only parameter of the ELMVIS+ method is the number of hidden neurons L in an ELM. The optimal L depends on a task, and controls variation across the visualization space. Too few neurons fail a visualization, while too many neurons create an overly variative visualization pattern. A good number is found by trial.

3.2.2 Experiments

The ELMVIS+ visualization method is compared to the original ELMVIS method on the same three datasets. Reference methods are PCA as the baseline, SOM as another method which uses fixed visualization points, and NeRV as the state-of-the-art nonlinear visualization method. A fast ELM model is provided by a toolbox from [ABML15].

The overview of performance of all methods is given by the MSE of a data reconstruction from visualization. The reconstruction (a reverse projection) already exists in ELMVIS+ and ELMVIS; for other methods it is learned by a separate model

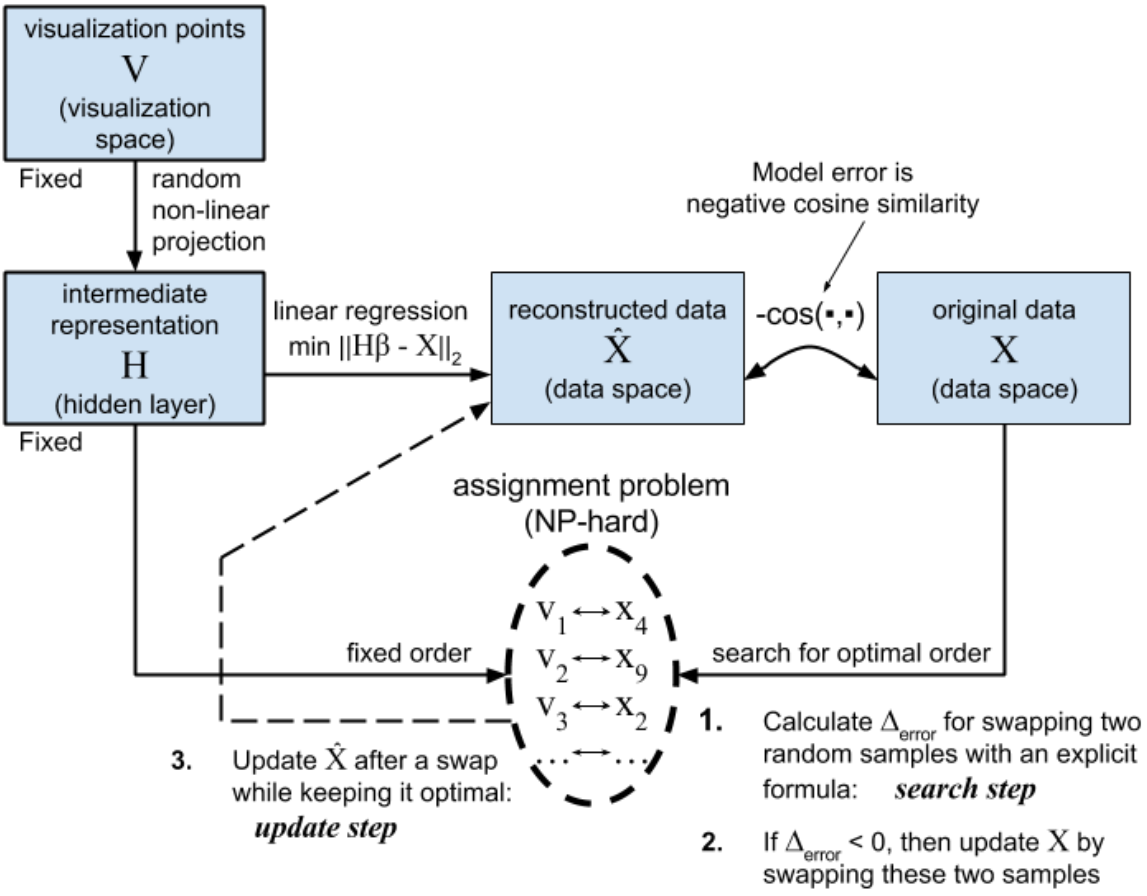


Figure 3.16: Schematic representation of ELMVIS+ algorithm. It starts by randomly permuting rows of $\mathbf{X}$ and training an ELM on $(\mathbf{V}, \mathbf{X})$ dataset. A fixed matrix $\mathbf{A}$ is computed as in eq. (3.25), and initial data reconstruction $\hat{\mathbf{X}} = \mathbf{AX}$ is created. A swap step looks for a new order of samples in $\mathbf{X}$ which gives $\Delta_E < 0$, after that an update step changes $\mathbf{X}$ and a new optimal $\hat{\mathbf{X}}$. Swap and update steps continue until a stopping criterion is reached.

as in [LV07]. This separate model is another ELM; the lowest error over 100 retrains is presented. The errors for all methods are gathered in Table 3.2.

Table 3.2: Reconstruction MSE for all methods; the lowest error of 100 initializations is shown. ELMVIS+ on *Sculpture faces* dataset is run twice: with the compressed ($d = 240$) and an original ($d = 4096$) image representations.

Dataset	PCA	SOM	NeRV	ELMVIS	**ELMVIS+**
Spiral	0.482	0.054	0.011	0.049	0.017
Sculpture faces	0.980	0.916	0.769	0.718	0.712 (compressed) 0.292 (original)
Real faces	0.724	0.511	0.501	0.449	0.156

ELMVIS+ method always performs better and runs faster than the original ELMVIS. It is much better than any other method in *Sculpture faces* because it is able to process the huge original dimensionality of the data, and in *Real faces* because it achieves a better optimization with millions of swaps evaluated in an hour with the new cost function. On *Spiral* dataset it is second only to NeRV, but the errors are similarly small.

Implementation and Computational Time

The ELMVIS+ method is implemented in different versions of Python-based code, available online[5]. The versions of code are: pure Python (with Numpy library

[5] `https://github.com/akusok/elmvis`

for fast matrix operations) denoted as "python", Python code compiled with Cython (which generates C code from Python script) denoted as "C", Python code compiled with Cython and having a fine-tuned parallel swap function written in pure C denoted as "C optimized", and a GPU-based version with data residing in GPU memory denoted as "GPU". All computations are done in double precision, because tests with single-precision code have shown inaccuracies in estimating Δ_E significant for the ELMVIS+ method and poor convergence. All tests and experiments hereafter are performed on a desktop with 4-core 4.6GHz CPU and Nvidia Titan Black GPU, which is similar to a common server accelerator Nvidia Tesla K20.

ELMVIS+ method has two types of steps: swaps and updates. Swap speed is extremely fast, see Figure 3.17. It is increased greatly compared to the original ELMVIS by introducing cosine similarity-based error and replacing full data matrix operations by simple vector operations. The speedup reaches 43,000 times with a fine-tuned parallel code in C (evaluated on 10,000 test samples of MNIST). Update speed involves updating a large $N \times d$ matrix $\hat{\mathbf{X}}$ and is limited by memory bandwidth, that benefits GPU implementation. A workaround is a batch update of several successful swaps at once (Figure 3.17, *C opt, batch*), but a large batch size may cause the method to diverge. A batch size of 5% of N caused divergence in experiments, while the batch size of 1% of N worked fine.

In practical computation, initial optimization has *swap:update* ratio of 10:1 and is limited by the update speed. It is run most efficiently on GPU or with batch updates. Later optimization and fine-tuning has *swap:update* ratio increased

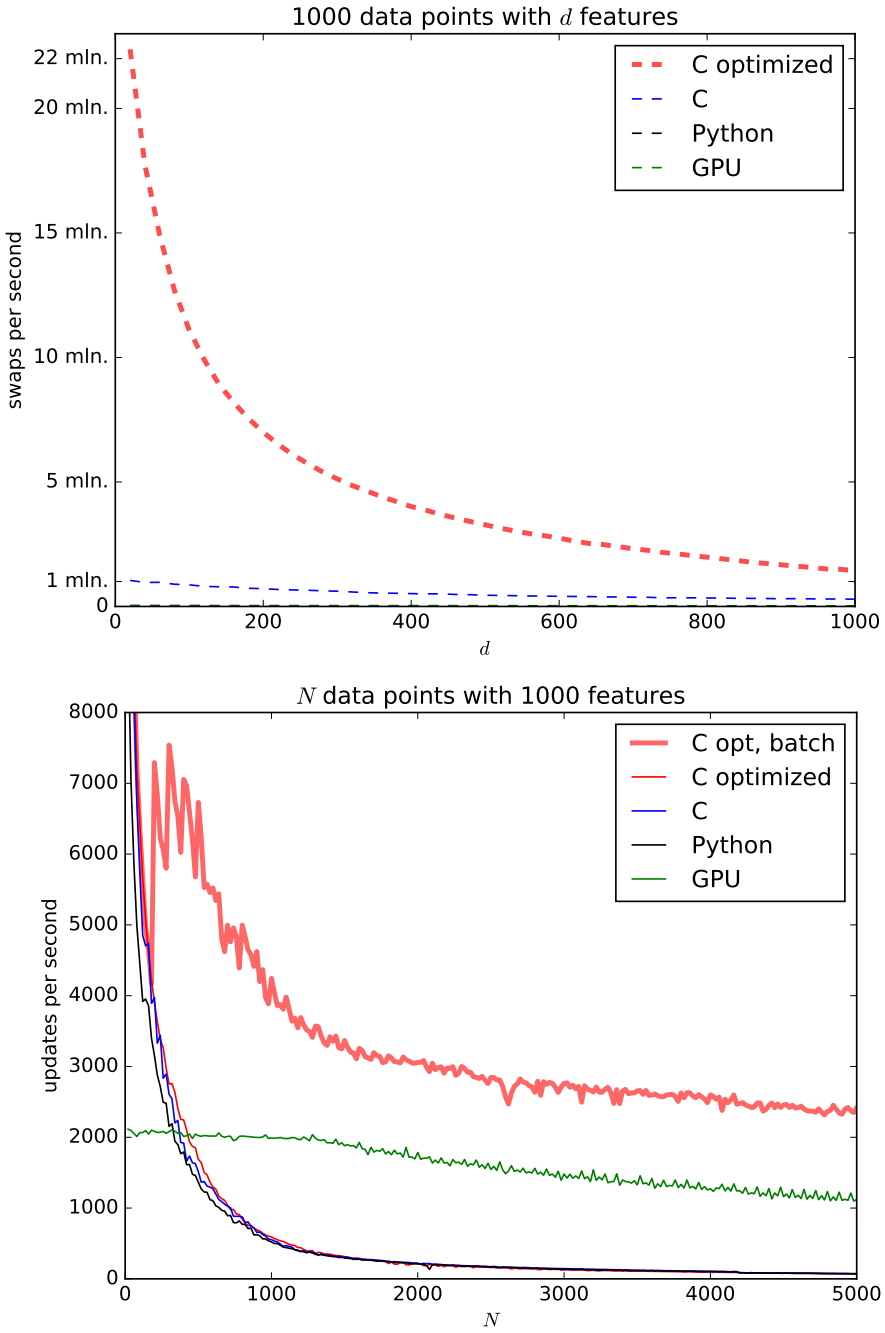


Figure 3.17: ELMVIS+ runtime speed summary. Swap speed is fast with parallel implementation in C (*top*). Update speed in much slower (*bottom*) and is limited by memory bandwidth, benefiting a GPU implementation. A workaround for the bandwidth limitation is to perform a batch update (size $0.01 * N$).

to 1,000...1,000,000:1 and is limited by swap speed. Here an optimized C method performs the best. As the optimization changes rows of data matrix $\mathbf{X}$ in-place, different implementations of ELMVIS+ may be run repeatedly on the same data matrix $\mathbf{X}$, improving it every time. For example, a GPU implementation does initial optimization and an optimized C implementation follows for fine-tuning.

Visualization of MNIST Test Set

The ELMVIS+ method is compared with other methods in performance and speed on a test set of MNIST handwritten digits dataset with $N = 10000$. The original representation of numbers as 28×28 grayscale images had $d_{\text{original}} = 784$ features. Only $d = 594$ features with at least 10 non-zero values are selected, because features with only zeros or little non-zero values cause deficit rank and numerical problems with some of the methods, for instance with PCA.

The methods used in visualization include ELMVIS+, original ELMVIS, SOM, NeRV and PCA. Their runtimes are given in Table 3.3, and visualizations in Figures 3.18-3.23. Methods do not have access to class information, this information is only used for creating plots.

Table 3.3: Runtimes of different visualization methods for MNIST test set.

	ELMVIS+	ELMVIS	SOM	NeRV	PCA
Time	1m 37s	> 43h	56m	1h 58m	0.25s

Visualization of ELMVIS+ (Figure 3.18) finishes in 1.6 minutes, and further fine-tuning does not noticeably improve the result (Figure 3.19). Original ELMVIS fails to converge with 10,000 samples even after 43 hours (Figure 3.20). It converged in 15 minutes with 300 samples, but even 1000 samples is too much for the old visualization method. SOM finishes in one hour with results similar to ELMVIS+ (Figure 3.21) but only 1000 neurons are used, because in SOM a number of neurons is typically less than the number of points. This makes sense in the SOM framework but the runtime may not be directly comparable. NeRV took two hours to produce a visualization (Figure 3.22) with interesting results. NeRV gives more insight in the original data distribution, for instance all digits one are very different to others, but not all classes are separated in the visualization. PCA is extremely fast to run at a quarter of a second, but the visualization (Figure 3.23) has different digits mostly mixed together, as is expected from a linear model applied to nonlinear data. Overall, ELMVIS+ performance is comparable to the best visualization methods, with faster runtime.

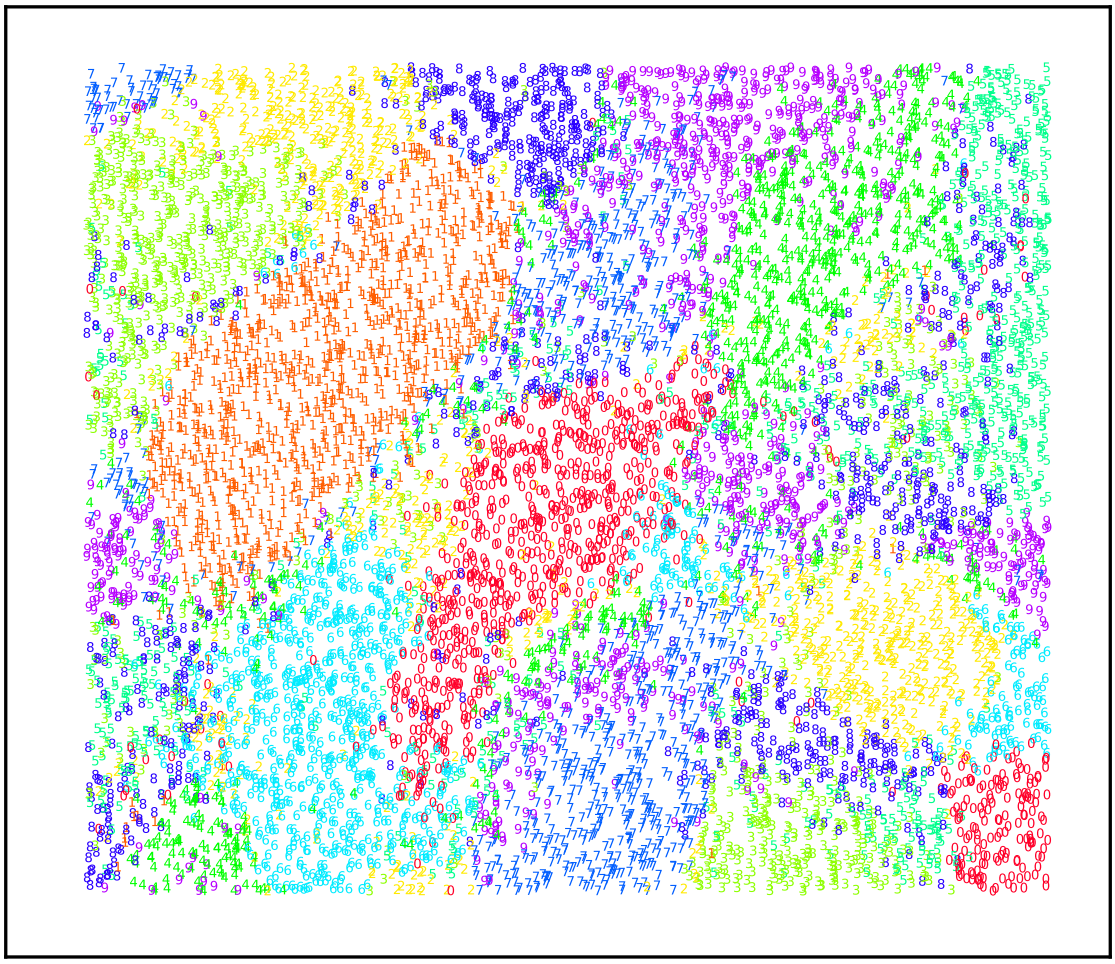


Figure 3.18: ELMVIS+ visualization of MNIST handwritten digits test set with 10,000 samples, using ELM with 20 neurons. Optimization finishes in one minute and 37 seconds.

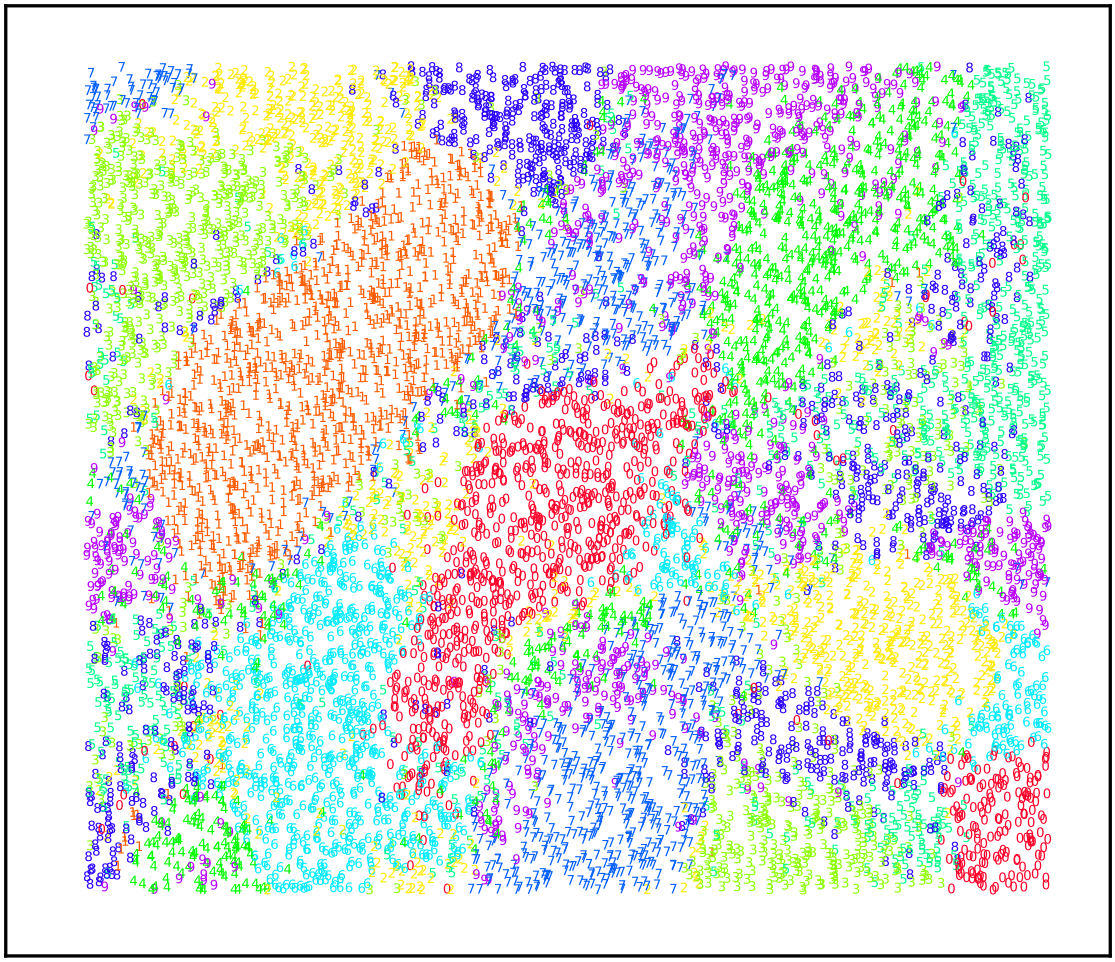


Figure 3.19: ELMVIS+ visualization of MNIST handwritten digits, further optimized from the previous figure for a total runtime of four minutes. Convergence almost reached at this point, with cosine similarity only improved from 0.344 to 0.346 compared to the previous figure in 180 million swaps, and the difference barely noticeable to a human viewer.

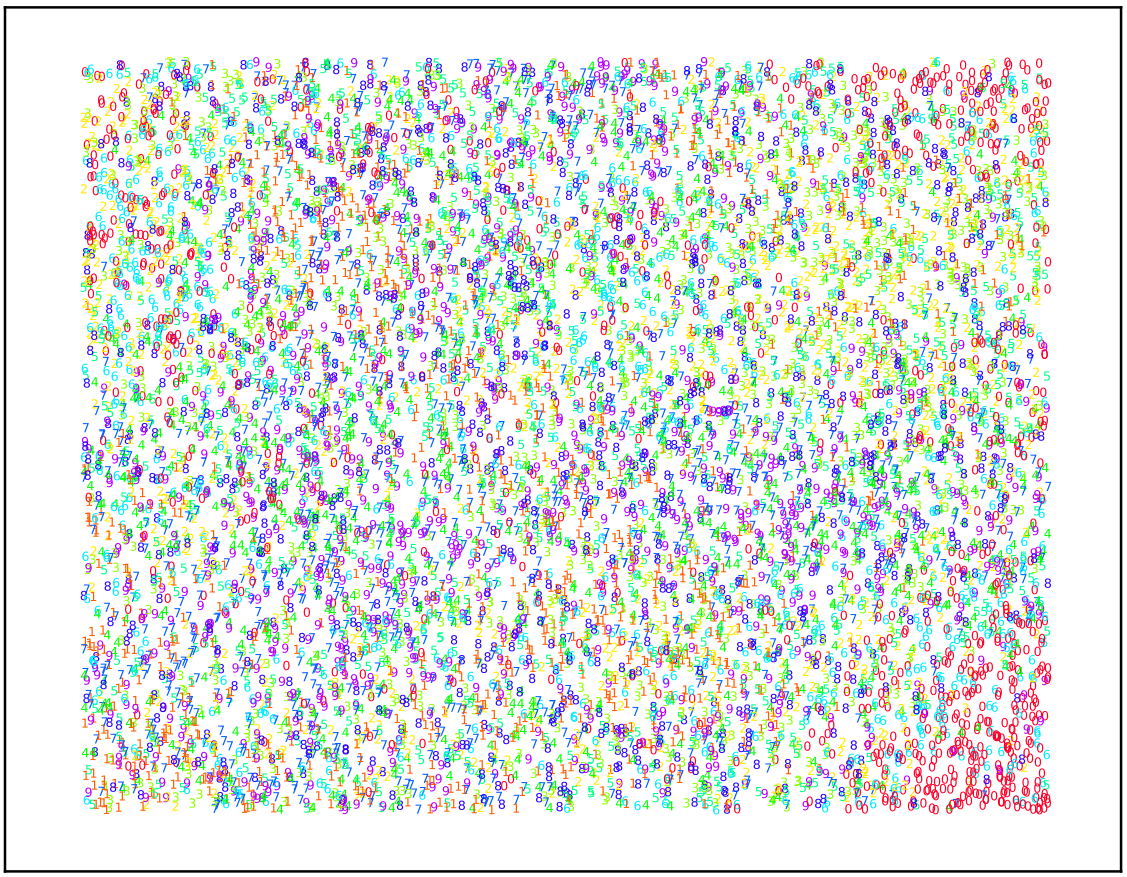


Figure 3.20: Original ELMVIS visualization of MNIST handwritten digits test set with 10,000 samples, using ELM with 20 neurons. Optimization runs for 43 hours but is nowhere close to finish. The original ELMVIS is uncomparably slower than ELMVIS+.

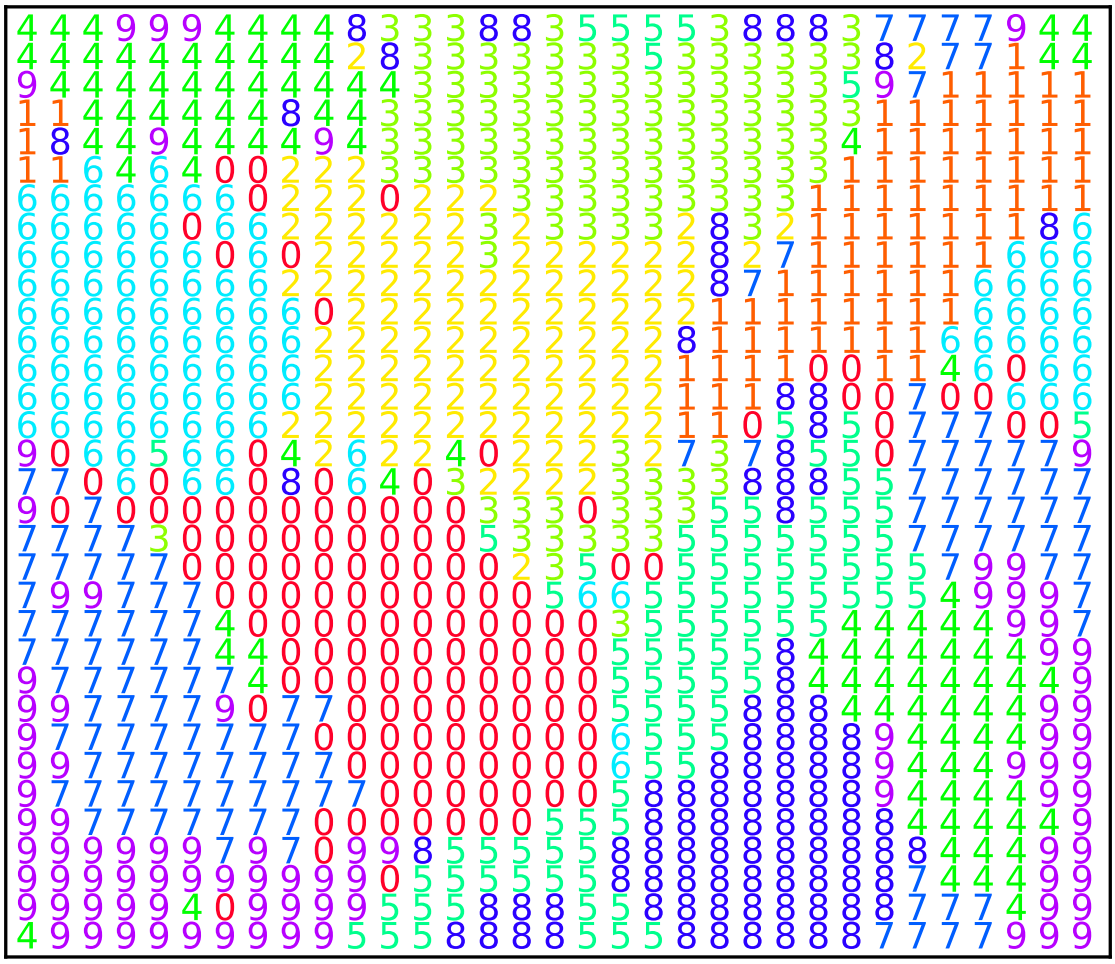


Figure 3.21: SOM visualization of MNIST handwritten digits test set with 10,000 samples, using 1000 nodes. Majority class is shown on each node. Optimization finishes in about one hour.

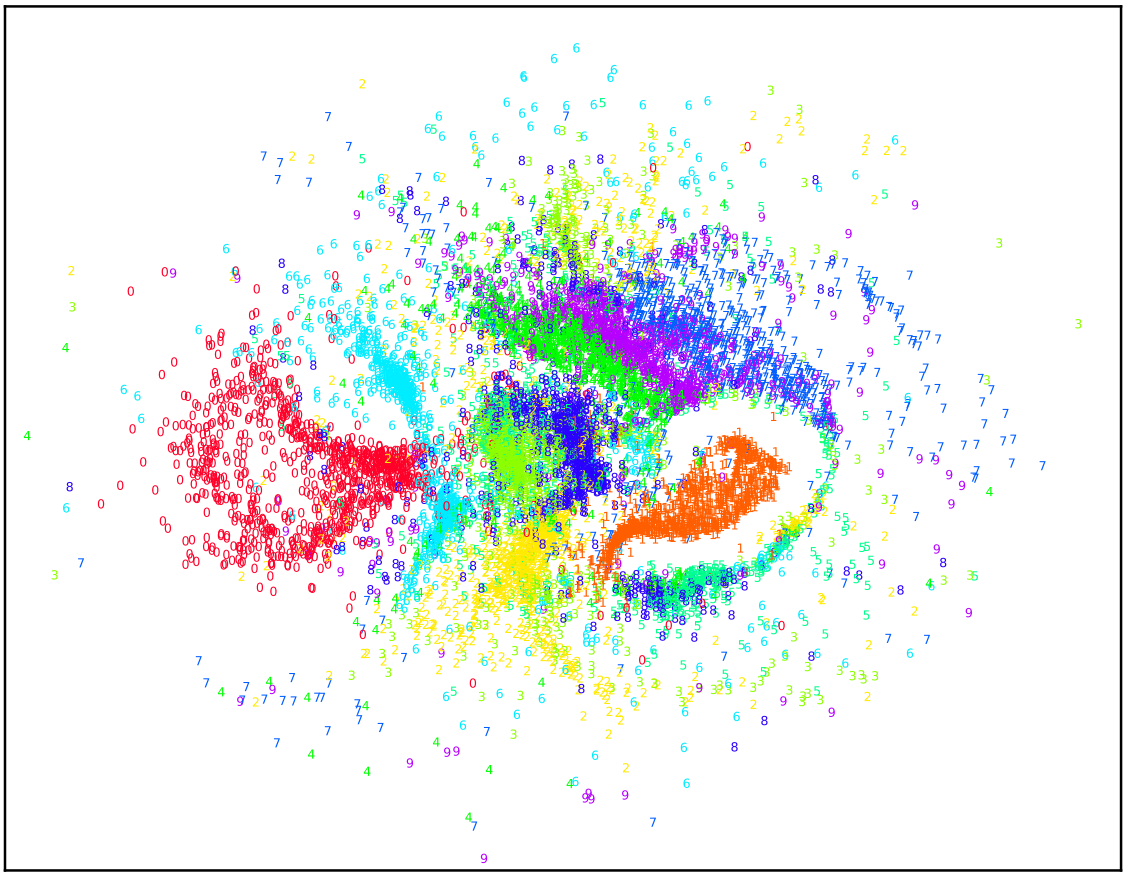


Figure 3.22: NeRV visualization of MNIST handwritten digits test set with 10,000 samples. Optimization finishes in about two hours.

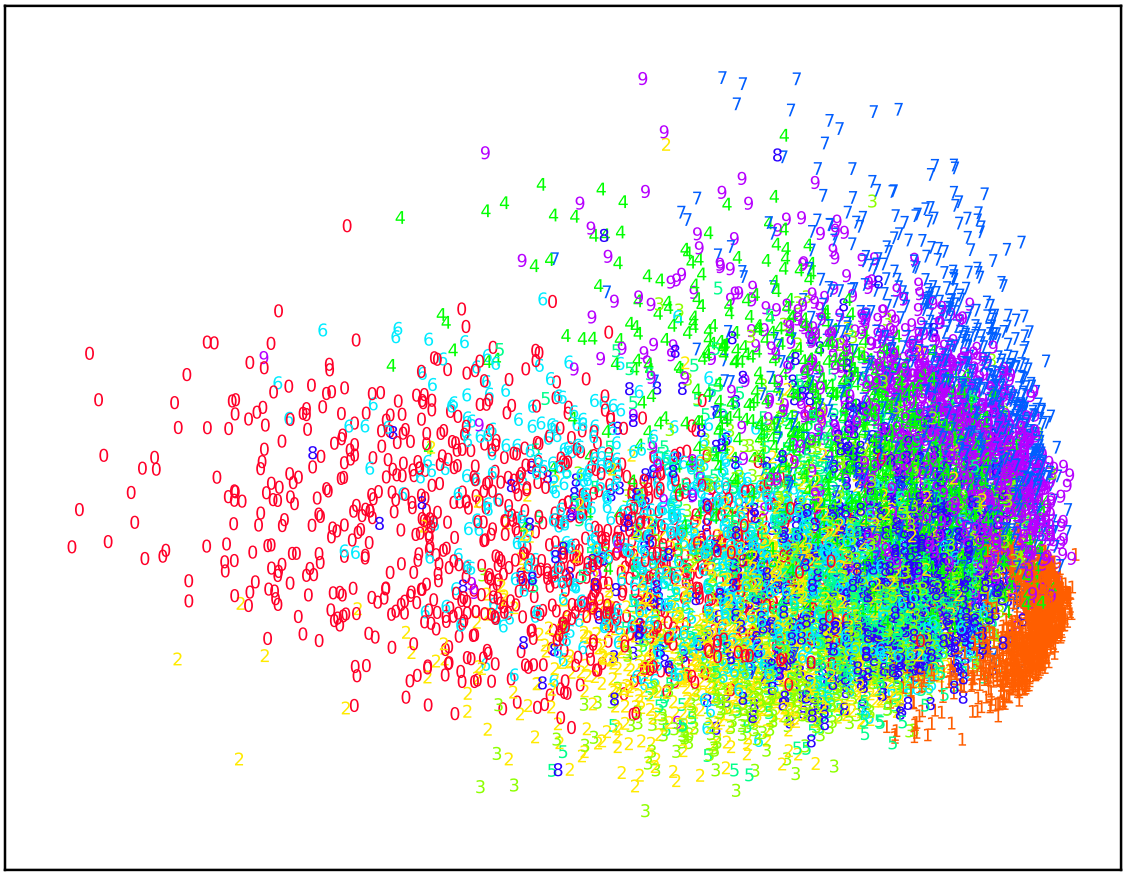


Figure 3.23: PCA visualization of MNIST handwritten digits test set with 10,000 samples. Computations with singular value decomposition take only 0.25 second, but PCA fails to show inner data structure and separate different digits.

Visualization of MNIST Training Set

Another experiment is done with full training set of MNIST with $N = 60000$ samples, see Figure 3.24. The visualization without batch updates took a few hours on the desktop, because GPU cannot be used due to insufficient memory for storing matrix $A_{60000\times60000}$, and a larger possible swap space which is quadratic in the number of samples N. ELMVIS+ optimization is run by multiple consecutive steps with a limit of 500 updates for each step, having the intermediate results saved. The output is visualized as separate images and a video. Visualization points are selected uniformly at random on a plane, and the initial assignment is random. The result of ELMVIS+ is visualized by printing the original pictures of MNIST digits at coordinates of their corresponding visualization points, using different colors for different digits and making the black background transparent.

A clear self-organization is observed during the convergence stage (Figure 3.25 presents visualization in the middle of convergence), which is supported by a large enough number of samples such that every cluster of data is large and clearly visible. Some samples of different classes (like several writings of numbers 4 and 9) never cluster separately, because they are very similar — but other style of writing of the same numbers are never confused. This experiment shows that Big Data scale adds more value to visualization than simply processing more data, and shows new useful insights.

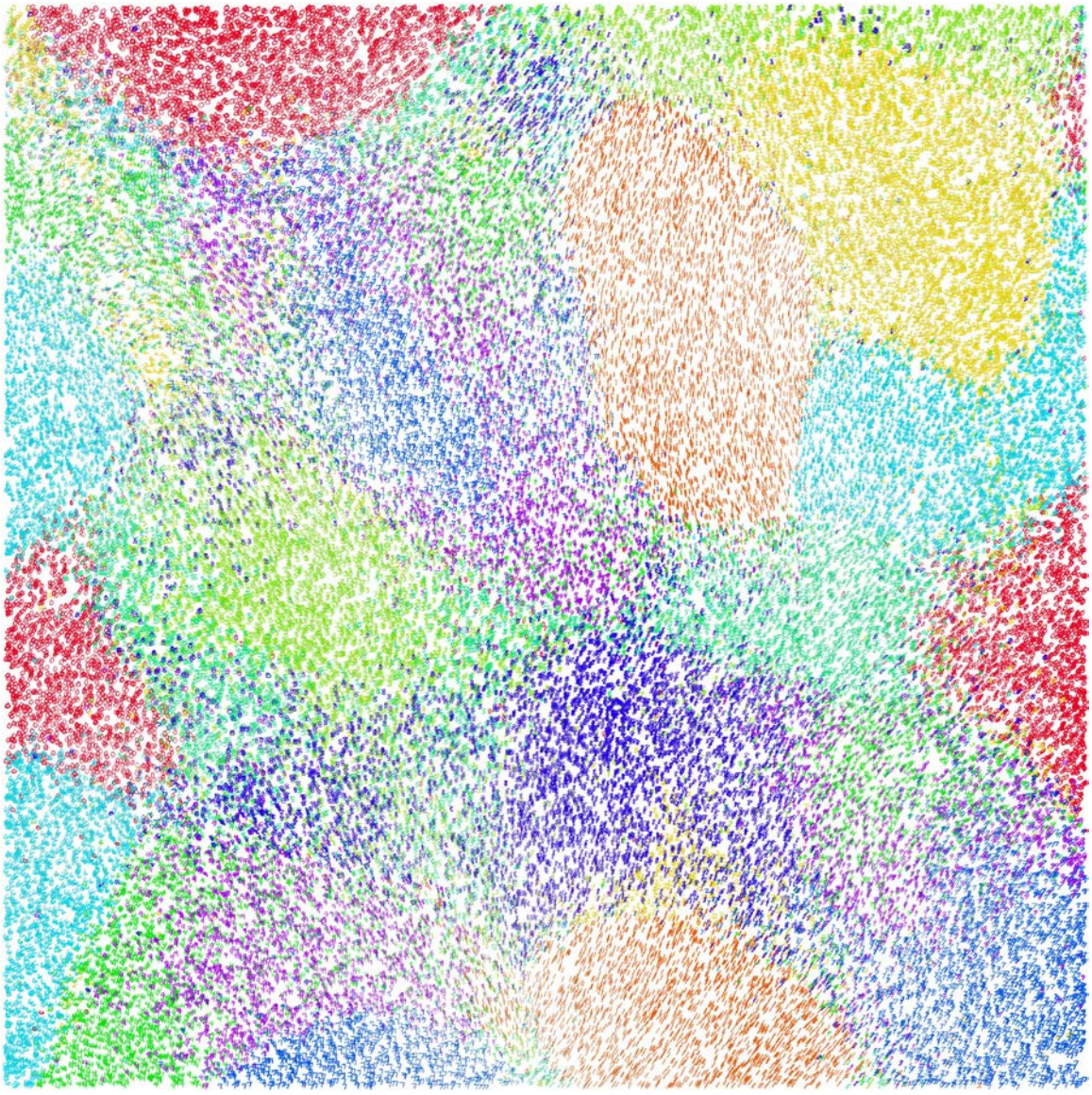


Figure 3.24: Visualization of MNIST handwritten digits training set with 60,000 samples after 500,000 updates, using ELM with 15 neurons. Original image are 4000×4000 pixels.

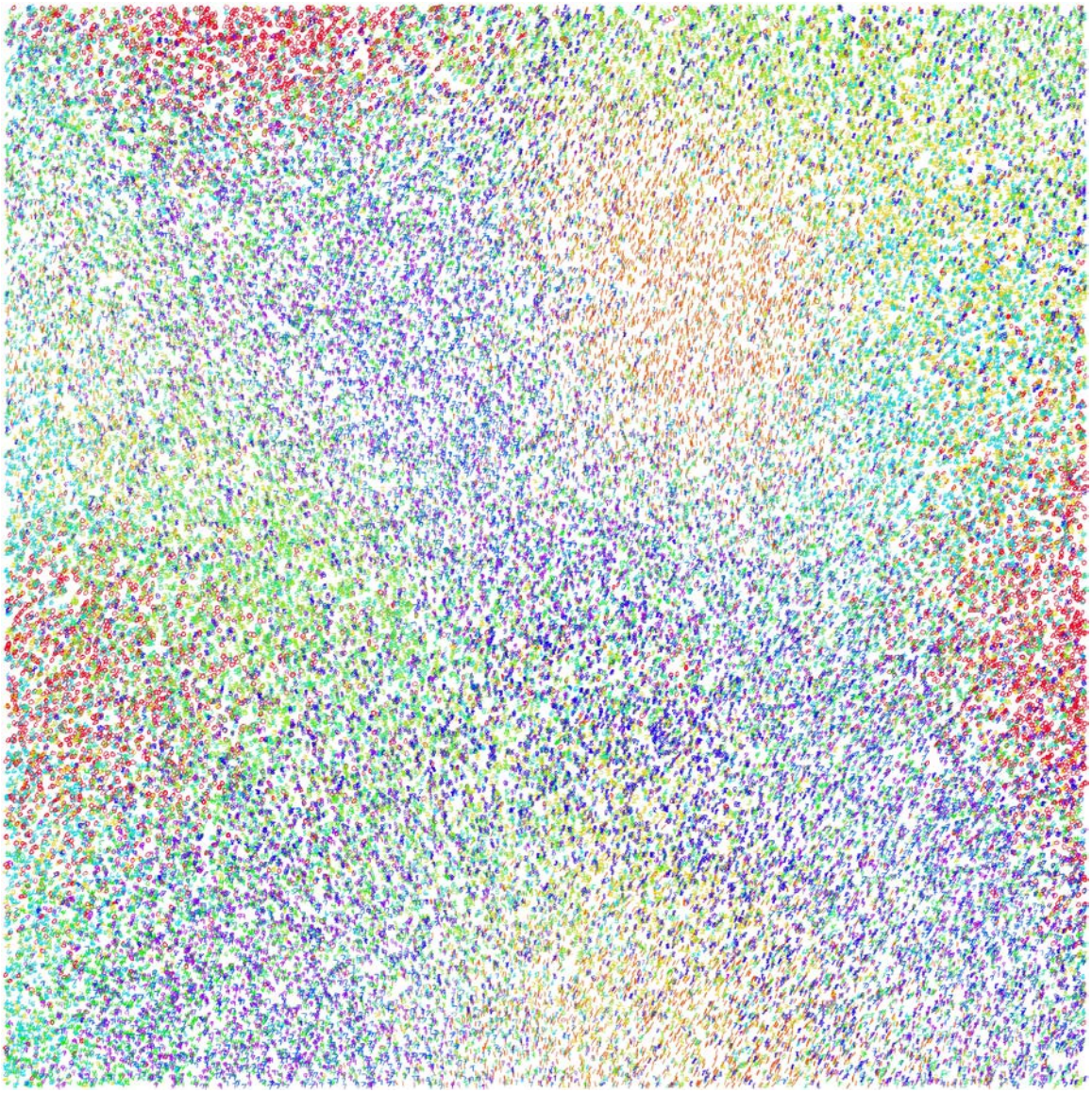


Figure 3.25: Visualization of MNIST handwritten digits training set with 60,000 samples after 100,000 updates. Convergence is in process, some clusters have formed.

Visualization on Arbitrary Spaces

The ELMVIS+ takes any visualization points, which don't have to be generated by a particular probability distribution. Arbitrary data can be used as visualization input.

In this example, the two-dimensional visualization points are taken in a shape of a football team emblem of our university. MNIST digits are mapped onto the visualization points, and an ELM model de-projects the two-dimensional inputs back into the original digits data space. Visualization on Figure 3.26 is shown using separate five different digits classes to show different digits.

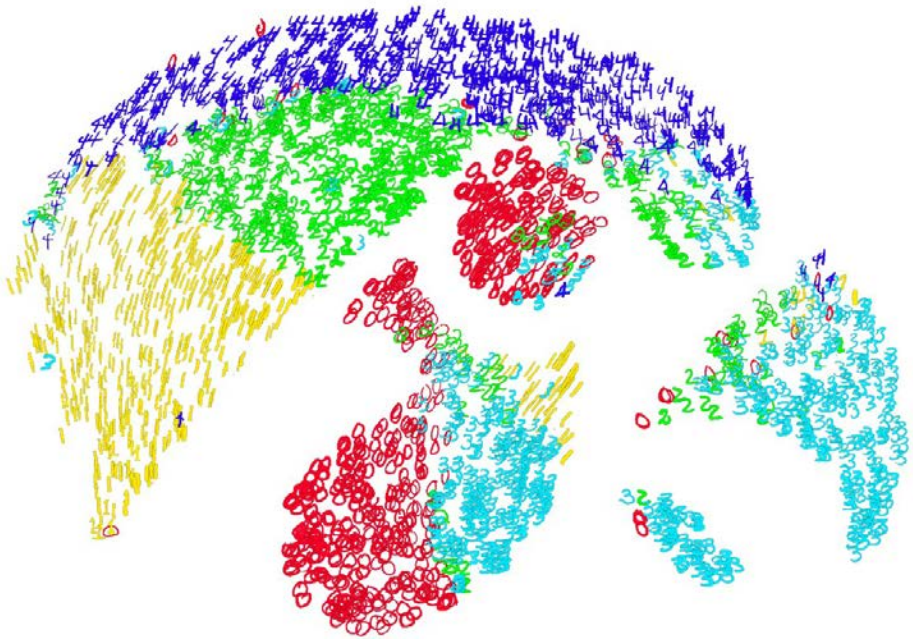


Figure 3.26: Visualization of MNIST digits using an Iowa Hawkeyes football team emblem shape. It uses 10,000 samples from only five classes of MNIST digits for clarity.

3.3 Detection of Mislabeled Samples with ELMs

This section focuses on finding data samples with incorrect labels in a given dataset. Such samples create "label noise", which is generally considered more harmful than feature noise [ZW04]. The work is motivated by studies on a financial dataset [du 07] where each sample corresponds to a company labeled as either healthy or bankrupt. In this dataset, incorrectly labeled samples are important both by themselves (i.e. as companies eligible for a loan but mislabeled by "bankrupt"), and for the whole dataset to allow building more precise Machine Learning models with a limited amount of data (because each sample is expensive and slow to obtain). There are other areas where detection of particular mislabeled samples is important, like medical applications [GLG99].

The idea of detecting mislabeled samples is to utilize their effect of increasing the model complexity [BF99, LCLo09]. In SLFN, a more complex model requires more hidden neurons to learn [Hay98], or equally a more complex model will result in a lower accuracy with the same amount of hidden neurons. Correcting an incorrect sample label will decrease a model error of a fixed SLFN. Note a difference between mislabeled samples and outliers — an outlier is not a typical sample of any class, so changing its label will not lead to a decrease in model error; although outlier detector methods are also used for dealing with mislabeled data [LND09].

3.3.1 Methodology

There exist multiple sources of label noise. First, noise can be generated by simple mistakes in data gathering and processing, like people mistyping or sensor malfunction [BF99, ZW04]. For real datasets, such noise is estimated to be roughly 5% not including other factors [Red98]. Second, experts who label the data can make mistakes. This happens especially in cases where labelling quality is traded for lower labeling price, for instance with crowdsourcing [YKL11] like Amazon Mechanical Turk [SOJN08] framework. Third, labeling criterion may be vague, then different experts will produce different labels. For example, in EEG segmentation exact beginnings and ends of signals are not formally defined, and different doctors give slightly different signal boundaries [HRT04]. At last, the existing information may be insufficient for reliable labelling of data [BF99].

Recent methods of Machine Learning in the presence of mislabeled data can be aggregated in three categories [FV14]. Data cleansing (or filtering) methods [BF99] pre-process dataset and fix incorrect labels or remove such samples [ZWC03]. The resulting clean dataset is used with general Machine Learning methods. Noise-robust methods [JWF10, CKW12] like k-nearest neighbors [ON97] are tuned to perform well despite the presence of label noise. It is even possible to achieve same theoretical performance with label noise as without one, although in simple cases [Lac74]. Noise-tolerant methods include label noise in their model; an extensive overview of such methods is presented in [FV14], section VII. A good survey is given by Frénay in his PhD thesis [Ben13].

Overview of the Methodology

Let's take an original dataset $\{\mathbf{X}, \mathbf{T}\}$ of N data samples with d features stored as a matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$, and the associated output labels $\mathbf{T} \in \{0, 1\}^{N \times c}$ with c different classes, only one of which is non-zero for each sample. This work distinguishes between three types of data samples: correctly labeled, originally mislabeled (unknown samples with incorrect labels), and artificially mislabeled (known samples with artificially corrupted labels). Each sample in a dataset has an associated *sample score* $\mathbf{S}$, which shows how likely for that sample is being a mislabeled. Artificially mislabeled samples have higher than average score, and they work as a baseline for finding originally mislabeled ones.

The method starts by creating some artificially mislabeled samples in a dataset; denote outputs with artificially mislabeled samples as $\mathbf{T}^*$. Then the baseline estimation of the generalization mean squared error $\underline{\text{MSE}_{\text{LOO}}}$ is calculated for the dataset with artificial mislabels $\{\mathbf{X}, \mathbf{T}^*\}$, using the OP-ELM classification model. After that, a *flip* is generated by making k more artificial mislabeled samples randomly (where k is the size of a flip). The new model error MSE_{LOO} is estimated for the flip $\{\mathbf{X}, \mathbf{T}^*_F\}$, and if it is lower than the baseline error, scores of the k added artificial mislabeled samples are increased by 1. After the flip is analyzed, data labels are reverted back to $\mathbf{T}^*$.

Detecting Unknown Mislabeled Samples

Sample scores after a large number of flips are shown on Figure 3.27. They are randomly distributed, with different distributions for artificially mislabeled and correctly labeled samples, but the same distribution for artificial and original mislabels.

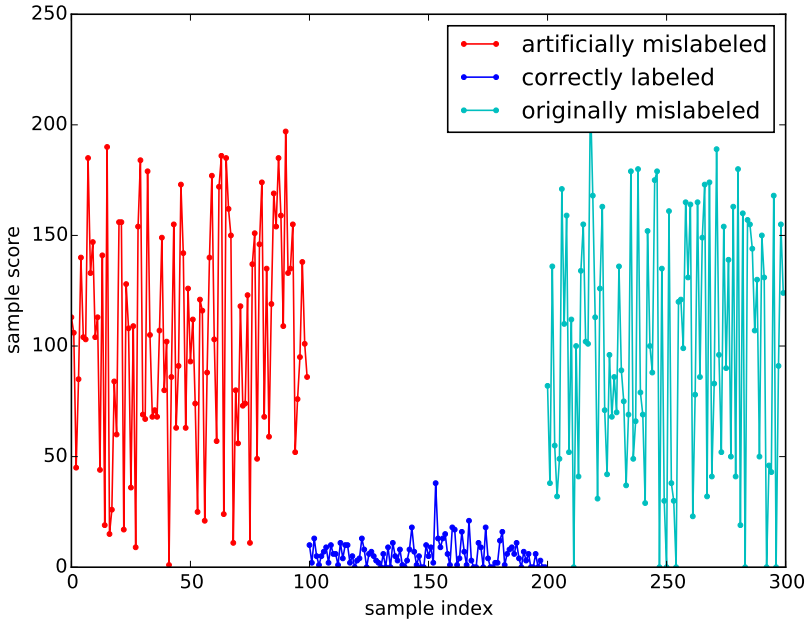


Figure 3.27: Sample scores of 300 samples after a large number of flips, for a XOR toy dataset. First 100 samples are artificial mislabels, second 100 samples are correctly classified, and the last 100 samples are original mislabels.

Sample scores can be formally analyzed by fitting Gaussian distributions to them - one for the artificially mislabeled samples, and one for the others (correctly

labeled and a small proportion of originally mislabeled). Initially all scores are zero, so the two Gaussian distributions have the same zero mean and zero variance. But as the scores increase, artificially mislabeled distribution will have a significantly higher mean value. The hypothesis that these two distributions have different means is checked with a Welch's t-test [Wel47], an adaptation of an independent Student's t-test to distributions with different variances (the variances on Figure 3.27 are obviously different).

Flips are repeated until a null hypothesis of the same distributions' means can be rejected with a high confidence (0.1%). Then non-artificially mislabeled samples with scores outside of the 3 sigma area of their Gaussian distribution are marked as detected original mislabels.

One issue with that method is the quantization of scores, as shown on Figure 3.28. Gaussian approximation works poorly with a low number of quantization levels. The second MD-ELM parameter for stopping the flips is introduce to account for the score quantization effect. This is a quantization threshold q, equal to the average score of artificially mislabeled samples. The effect of the quantization threshold and its optimal value is analyzed in the experimental results section.

The last parameter to take into account is a number of models m to build. Originally mislabeled samples are unknown and artificially mislabeled samples are ignored in the detection stage. With only one model, it is possible to select an original mislabel as an artificial mislabel, and fail to detect it. This scenario is avoided by having several models with different artificial mislabels, and averaging their scores

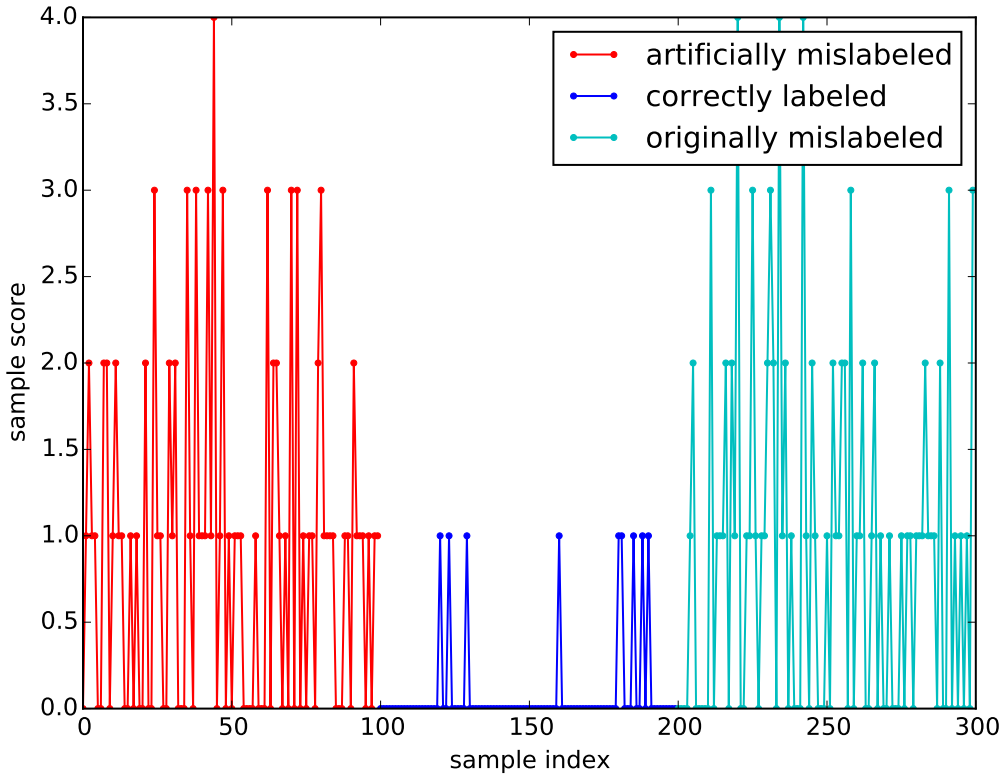


Figure 3.28: Sample scores of 300 samples after a low number of flips, for a XOR toy dataset. A strong quantization effect makes Gaussian approximation of score values erroneous. This plot corresponds to a score quantization threshold $q = 1$.

for non-artificially mislabeled samples before the detection. In that case, there will be scores for all samples, and all originally mislabeled ones are possible to detect.

MD-ELM Algorithm

The full MD-ELM algorithm is presented below in Algorithm 3.2. There α is a proportion of artificially mislabeled samples created in a dataset; it is set to 5%

in the experiments. Stopping criteria are computationally expensive to evaluate, so an evaluation occurs after every M flips of each model. A reasonable value of M is 1000. Each model keeps track of indexes of artificially mislabeled samples, and returns scores for artificially mislabeled $\mathbf{S}^{\text{art}}$ and original samples $\mathbf{S}^{\text{orig}}$ separately. When the final scores are calculated, each sample is processed separately because only non-artificially mislabeled scores are used, so different samples have different amount of scores (not always m scores per sample).

Algorithm 3.2 Algorithm of the proposed MD-ELM.

1: Original inputs $\mathbf{X} \in \mathbb{R}^{N \times d}$ and outputs $\mathbf{T} \in \{0,1\}^{N \times c}$

2: Parameters: number of models m, quantization threshold q^*, size of a flip k

3: **for** i = 1..m **do**

4: Generate $\mathbf{T}_i^*$ with αN random artificial mislabels

5: Build OP-ELM model with $\{\mathbf{X}, \mathbf{T}_i^*\}$ and calculate $\underline{\text{MSE}}_{\text{LOO}}^i$ for that model

6: Initialize sample scores $\mathbf{S}_i \in \mathbb{N}^N = \mathbf{0}$ for that model

7: **end for**{*continues at next page*}

3.3.2 Experiments

Datasets

The methodology is tested on a three real world datasets for experiments. Two of them are are Nursery and Breast Cancer UCI datasets [Lic13], which are

9: **repeat**

10: **for** i = 1..m **do**

11: **for** l = 1..M **do**

12: Generate flip targets $\mathbf{T}^*_{i,Fl}$ by creating k artificial mislabels

13: Evaluate $\text{MSE}^i_{\text{LOO}}$ using $\{\mathbf{X}, \mathbf{T}^*_{i,Fl}\}$

14: **if** $\text{MSE}^i_{\text{LOO}} < \underline{\text{MSE}^i_{\text{LOO}}}$ **then**

15: Increase score $\mathbf{S}_i$ by one for those k samples

16: **end if**

17: Discard flip targets $\mathbf{T}^*_{i,Fl}$

18: **end for**

19: **end for**

20: Obtain scores $\mathbf{S}^{\text{art}}$ for all artificial mislabels from all m models

21: Obtain scores $\mathbf{S}^{\text{orig}}$ for all non-artificial mislabels from all m models

22: Get t statistic using Welch's t-test on $\{\mathbf{S}^{\text{art}}, \mathbf{S}^{\text{orig}}\}$

23: Compute quantization level $q = \frac{m}{|\mathbf{S}^{\text{art}}|} \sum \mathbf{S}^{\text{art}}$

24: **until** $t < 10^{-4}$ and $q > q^*$

25: Get final scores $\mathbf{S}_i = \frac{1}{|\mathbf{S}^{\text{orig}}_i|} \sum \mathbf{S}^{\text{orig}}_i,\ i = 1..N$

26: Calculate mean μ and standard deviation σ of final scores $\mathbf{S}$

27: Report indexes of original mislabels $i \mid \mathbf{S}_i > \mu + 3\sigma$

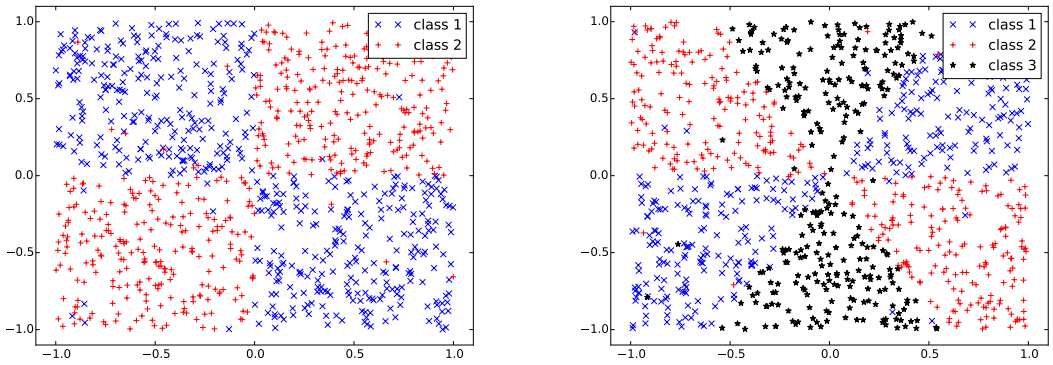
used for classification performance analysis after application of the MD-ELM method. Only the average classification performance can be evaluated because exact original mislabeled samples are unavailable for those datasets. They are available for the last dataset of 500 companies from the field of Corporate Finance [du 07], with 50% healthy ones and 50% bankrupt. The samples detected with the MD-ELM method are checked with the experts in the field, and a summary of the method's performance is given.

Two toy datasets are used for illustration of MD-ELM method, as well as for parameter selection and validation (Figure 3.29). One is a XOR dataset for binary classification (denoted as "XOR"). The other one is a 3-class dataset for multi-class tasks, denoted as "PIE". Both datasets have inputs $\mathbf{X} \in \mathbb{R}^{N \times 2}$ generated i.i.d. within the range $\{-1, 1\}$. Outputs are $\mathbf{T} \in \{0, 1\}^{N \times 2}$ for XOR and $\mathbf{T} \in \{0, 1\}^{N \times 3}$ for PIE. The datasets have $N = 1000$ samples, 20 of which are made originally mislabeled.

Number of Models and Quantization Threshold

MD-ELM uses artificially created mislabeled samples in its algorithm, which are ignored in the detection stage. Thus one model can miss originally mislabeled samples, if they are selected as artificially mislabeled ones. This problem is solved by having multiple data models; in MD-ELM this is multiple OP-ELM models.

The quantization threshold is the major parameter setting the required number of iterations, and the runtime. It is the mean score of artificially mislabeled samples; in case of multiple models scores are summed over all the models. Higher



(a) XOR dataset (b) PIE dataset

Figure 3.29: Examples of two- (a) and three-class (b) toy datasets used for parameter selection. Different classes are depicted by different colors and marker types.

quantization threshold produces larger score values (with more discrete levels) which are approximated by the Gaussian distribution more precisely, and provide better accuracy. The effects of different models and different quantization scores are shown on Figure 3.30.

The plots show that the quantization threshold q has a large impact on the performance - it should be large to achieve good results. But the runtime is directly proportional to the quantization threshold in the experiments. The minimum value which leads to good results lie between 32 and 64; it is taken as $q = 50$ in further experiments. The number of models m influence the results much less. There should be more than 1 model; 5 models are used through the experiments. The amount of models has almost no effect on runtime of flips (iterations) and provide a very minor increase of runtime of the method by initializing more distinct OP-ELM

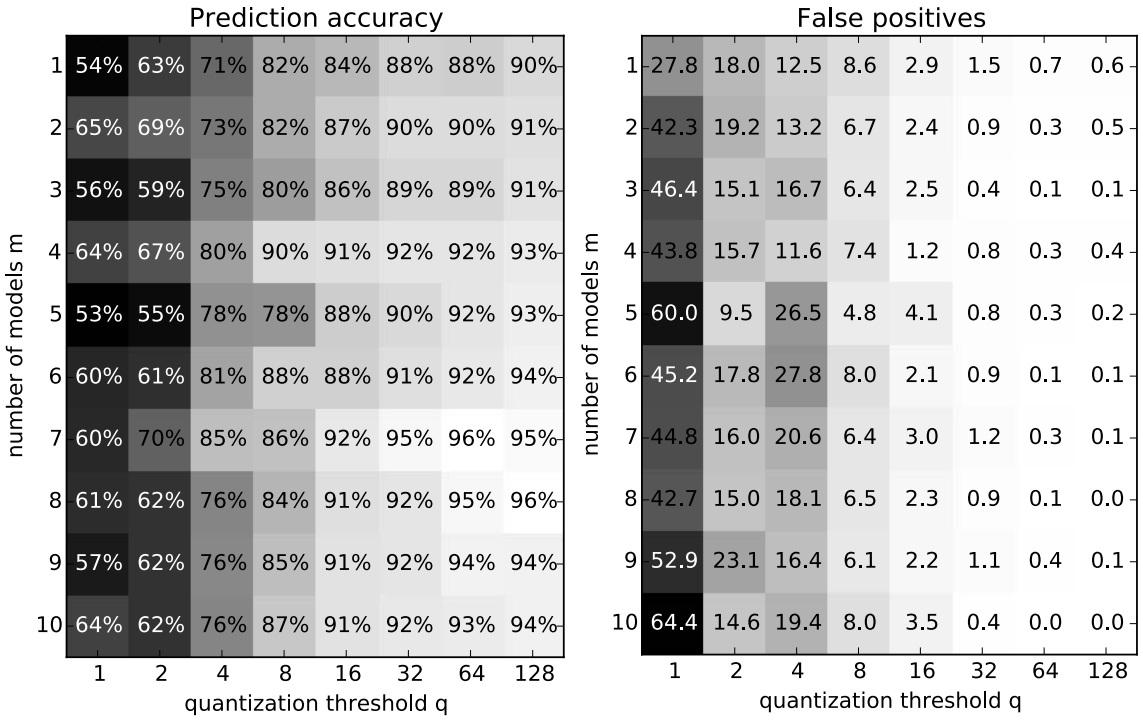


Figure 3.30: Effect of different amount of models and quantization thresholds on detection accuracy and the amount of false positives. Experiments are for a XOR dataset, averaged over 10 repetitions.

Optimal Number of Samples to Flip

An important parameter of MD-ELM is an optimal number k of samples to artificially mislabel at each flip. It can be 1,2,3 or more samples at once. With $k > 1$, reaching the required quantization threshold q will take more flips, because the decrease of $\mathrm{MSE}_{\mathrm{LOO}}$ from fixing a label of originally mislabeled sample might be countered by its increase from corrupting another correct label. On the other hand, re-labeling several samples at once allows for detecting several mislabeled samples close together.

The effect of relabeling only one sample ($k = 1$) in each flip is shown on Figure 3.31. The method finds all originally mislabeled samples with a high quantization threshold, but the amount of False Positives is very high. An advantage of the MD-ELM method is a low number of False Positives, so $k = 1$ should not be chosen.

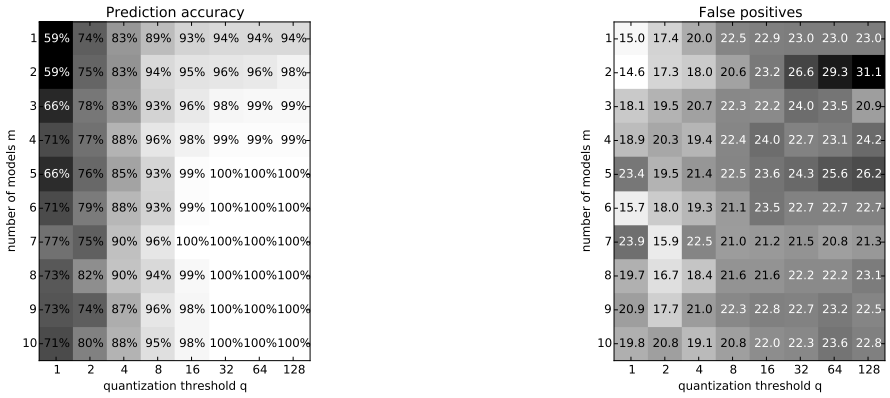


Figure 3.31: Experimental results for a XOR dataset with size of a flip $k = 1$, averaged over 10 repetitions.

Experimental results with $k = 3$ are shown on Figure 3.32. Surprisingly, the amount of False Positives is also high, so $k = 3$ is a bad choice for MD-ELM. A possible explanation is that if classes of 3 samples are changed on a border, the ELM model can learns that change and achieve a lower $\mathrm{MSE}_{\mathrm{LOO}}$, increasing sample scores as if they are originally mislabeled. The position of False Positives with the best experimental setting for XOR dataset are shown on Figure 3.33.

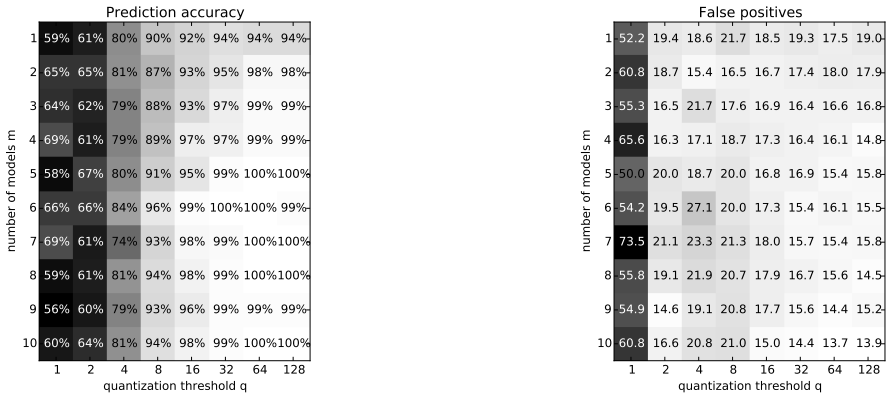


Figure 3.32: Experimental results for a XOR dataset with size of a flip $k = 3$, averaged over 10 repetitions.

Considering the runtime, $k = 1$ and $k = 2$ require similar number of flips, while $k = 3$ needs more flips. Graph of runtime for XOR and PIE datasets is shown on Figure 3.34. Runtime is directly proportional to the number of flips, and for small tasks like XOR or PIE the program runs at approximately 300,000 flips per minute on a fast 4-core i7 CPU.

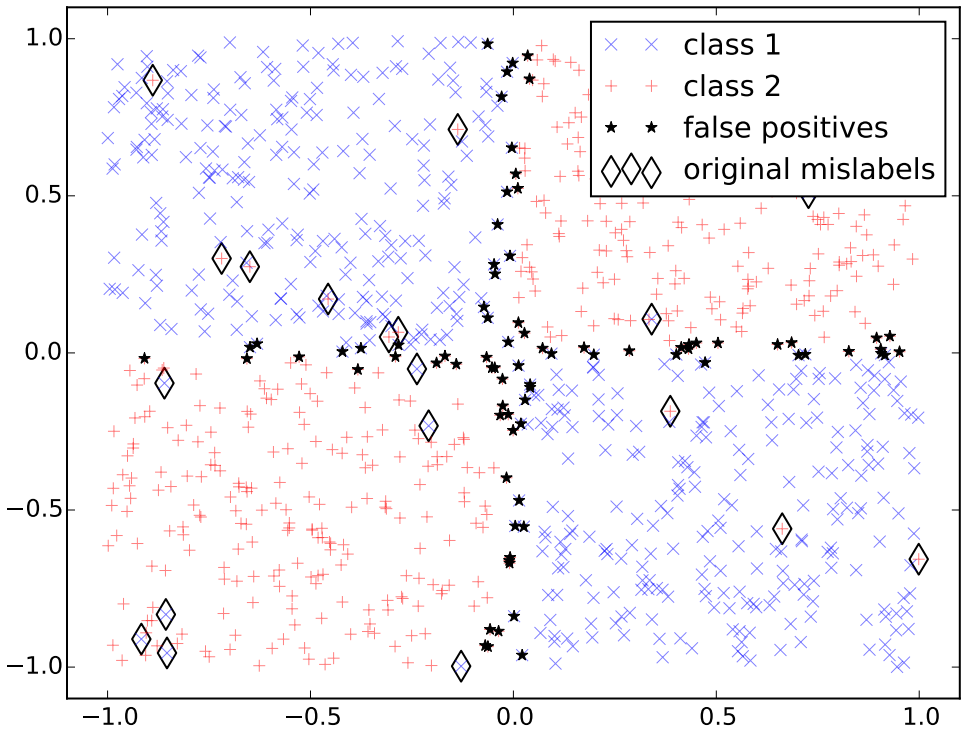


Figure 3.33: False Positives in detected originally mislabeled samples for a XOR dataset with $k = 3$ and $q = 128$. All False Positives are located at the classification borders.

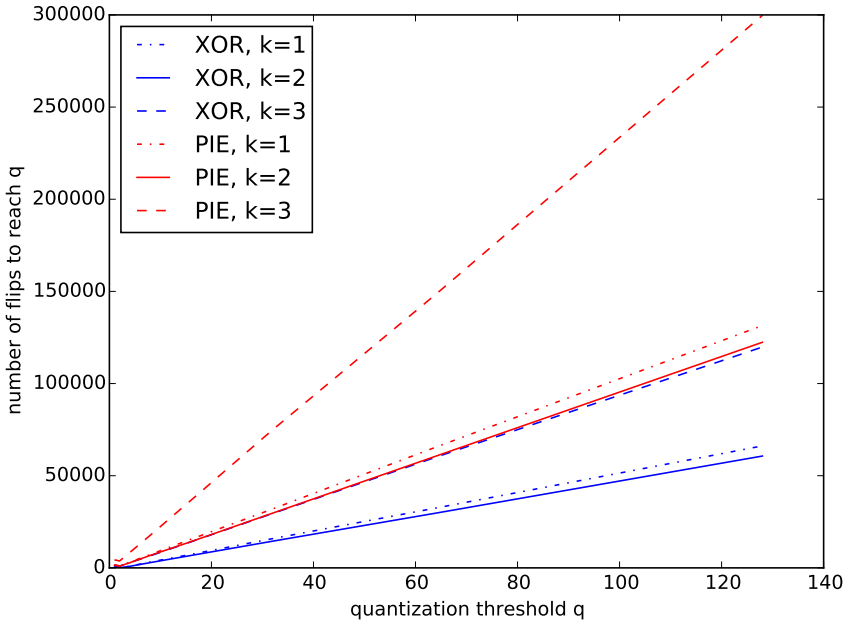


Figure 3.34: Number of flips required to achieve the desired quantization threshold q for XOR and PIE datasets. For $k = 3$, the method required much more flips than for $k = 1$ and $k = 2$. The Welch's t-test was never a limiting factor for those two toy datasets.

As a summary of parameter selection, the best parameter values are $m = 5$ models, $k = 2$ samples to re-label in each flip, and $q = 50$ quantization threshold. These values are used in the next experiments.

Results on XOR Dataset

The XOR dataset has been presented before for the analysis of the parameters of the methodology, and is used here with 1000 samples. Of these samples, a set of 20 samples are mislabeled intentionally, to act here as *original mislabels*. Detection results for the 2-class XOR dataset are presented on Figure 3.35.

The figure shows that 17 out of 20 original mislabels are found in all 10 runs, one is found in 90% cases, one in 50% and only one sample is almost never detected (10% cases). Samples which are rarely detected lie in a central region, which is the most hard to predict by OP-ELM. The reason they are not detected is probably that in a region with low classification confidence the change of a sample label might not decrease an overall MSE_{LOO}, and such sample is not detected as a mislabeled one.

Results on PIE Dataset

As for the XOR dataset, this toy dataset is the same as the one used for the parameters analysis, with 1000 samples and 3 classes. 20 samples are mislabeled intentionally to act as original mislabels. Detection results for the 3-class PIE dataset are presented on Figure 3.36.

Again, 16 out of 20 original mislabels are detected in all runs, 1 in 9 out of 10 runs, 2 samples detected half of the time, and only 1 sample is almost never detected

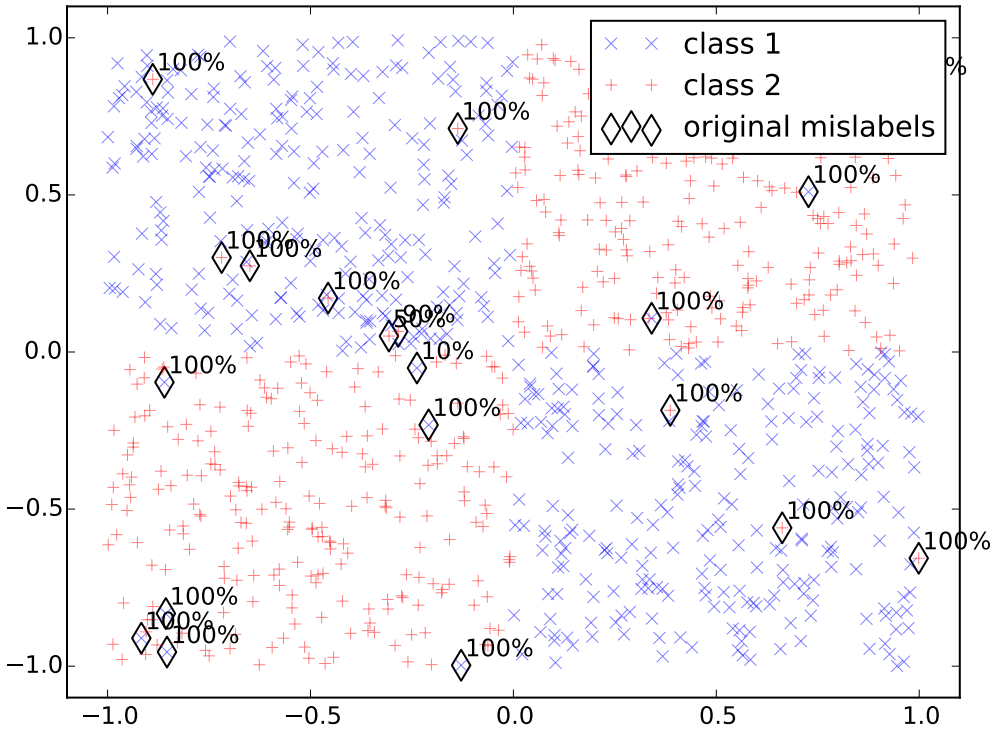


Figure 3.35: Plot of XOR data with marked mislabeled samples and their detection percentages. Experiments are performed 10 times with $m = 5$, $k = 2$ and $q = 50$.

(1 out of 10 times). This last sample lies on a border near the center of the plot, in an area which is the hardest for the classifier to work on.

Results on Nursery Dataset

In this dataset from the UCI Machine Learning Repository [Lic13] there are 12960 samples, lying in 8 dimensions and with an output consisting in 5 classes. The MD-ELM method is used to detect originally mislabeled samples — 146 samples are

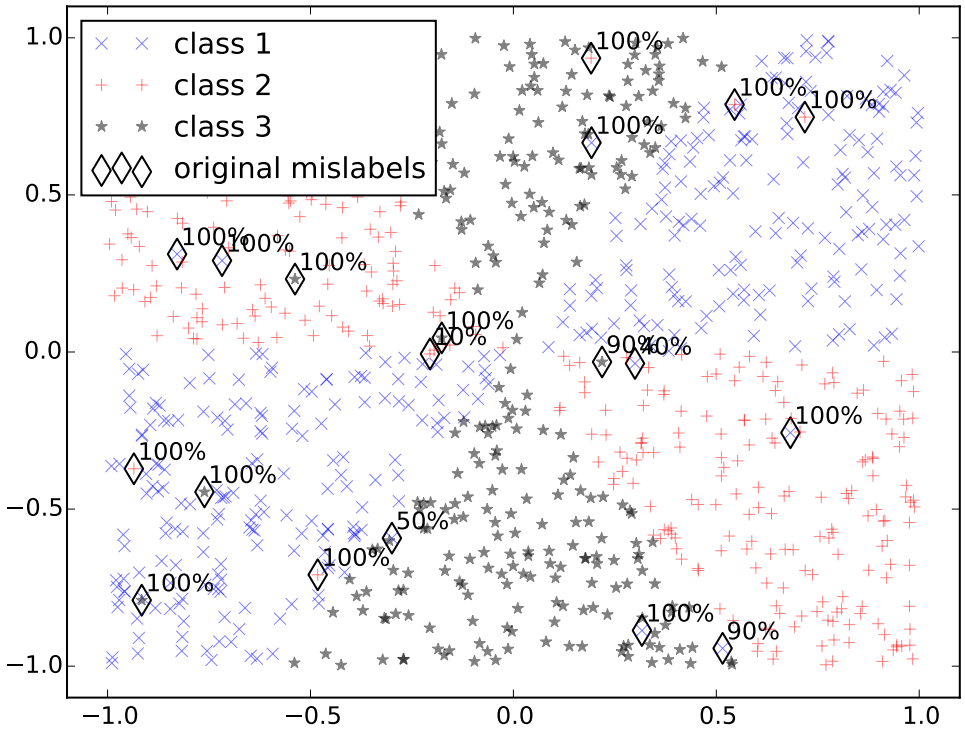


Figure 3.36: Plot of PIE data with marked mislabeled samples and their detection percentages. Experiments are performed 10 times with $m = 5$, $k = 2$ and $q = 50$.

detected; and then the classification accuracy is checked with original classes and classes with fixed originally mislabeled. The results are shown on Figure 3.37.

The results show that the classification accuracy with modified class labels is higher than with the original ones, in a wide range of model sizes. The maximum accuracy improvement is 0.5% and the average over different number of neurons is 0.25%. The computational time of MD-ELM method for the Nursery dataset is 15 minutes on a 4-core i7 CPU.

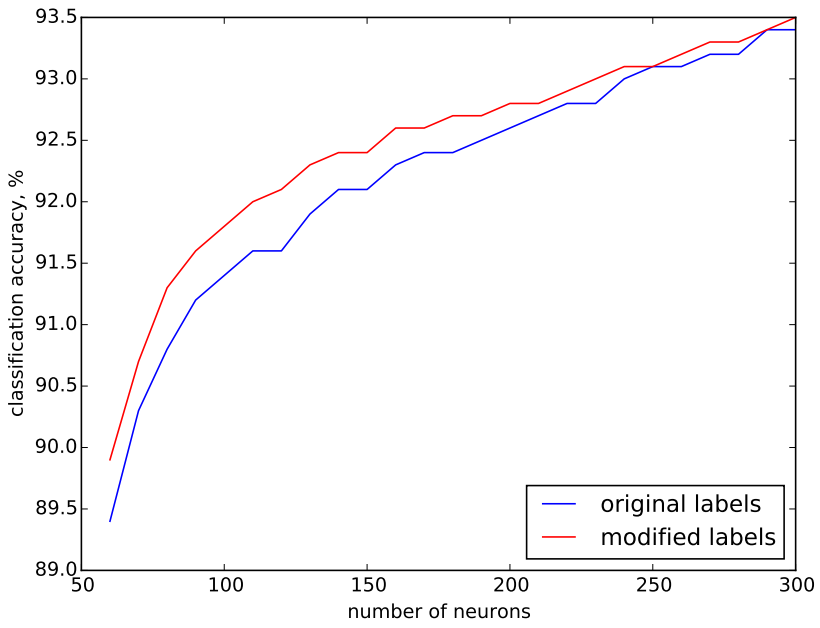


Figure 3.37: Classification accuracy of ELM for Nursery dataset with original labels, and with labels with fixed detected originally mislabeled. Results are average over 100 runs.

Results on Breast Cancer Dataset

This dataset from the UCI Machine Learning Repository [Lic13] has 689 samples with 9 features, and 2 classes. The MD-ELM has found only 6 original mislabels in that dataset. The results with those mislabels fixed are shown on Figure 3.38.

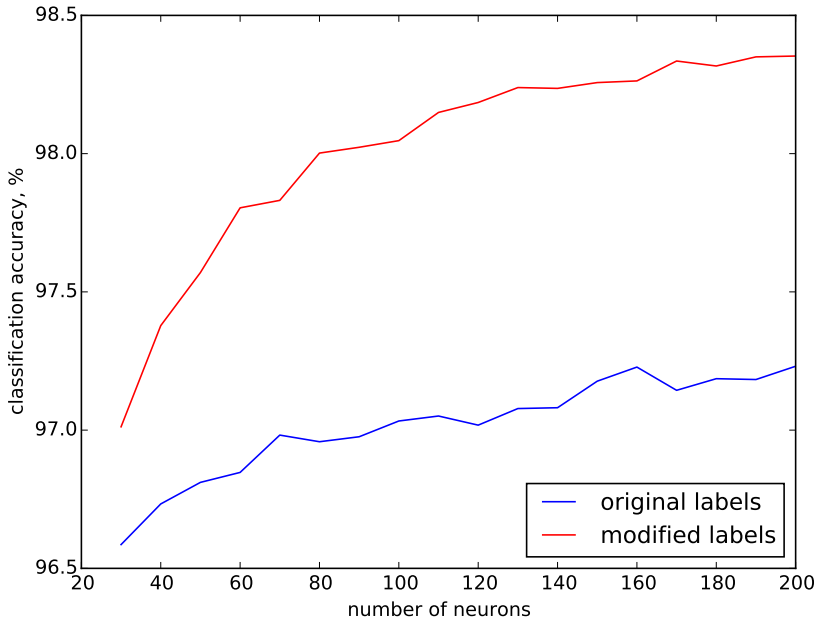


Figure 3.38: Classification accuracy of OP-ELM for Breast Cancer with original labels, and with labels with fixed detected originally mislabeled. Results are average over 100 runs.

The results show a large improvement in classification with only 6 samples fixed. Those samples are probably the original mislabels of the dataset. The computational time of the MD-ELM method for the Breast Cancer dataset takes only a few

seconds.

3.3.3 Results of Real World Financial Dataset

The ability to predict bankruptcy of a firm is crucial for an investor or a creditor (bank) who wishes to ensure that he will be reimbursed later on. This experiment adopts binary classification to label the firms. An healthy company means that it is able to reimburse its debt and it has continuity and future. However, a bankrupted company is one that is unable to meet its financial obligations. In other words, it cannot pay back its debtors and begin a liquidation process that stands for sale or cessation of the company. The data set was built by du Jardin [du 07], and includes 500 firms from year 2002. In the data set, the proportion of healthy and bankrupted firms is 50 : 50 and the firms are all from the trade sector. Before acquiring the anomalies of each sample, variable selection was applied [KME+11] with 7 variables selected for the training. The histogram of the calculated sample scores $\mathbf{S}^{\text{orig}}$ is shown on Figure 3.39.

The method shows 20 samples with high anomaly that may be mislabeled. Those samples have been analyzed by two independent financial experts. For the selected samples #41, #212, #437 and #448, both experts consider that the samples are surely mislabeled in the first place. For the most of samples, one out of two experts considers that they are mislabeled. This is the case for the selected samples: #160, #168, #301, #427, #458, #465, #474, #482, #483, #485, #490 and #494. For #288, #452, #454 and #486, the experts do not consider these selected samples as

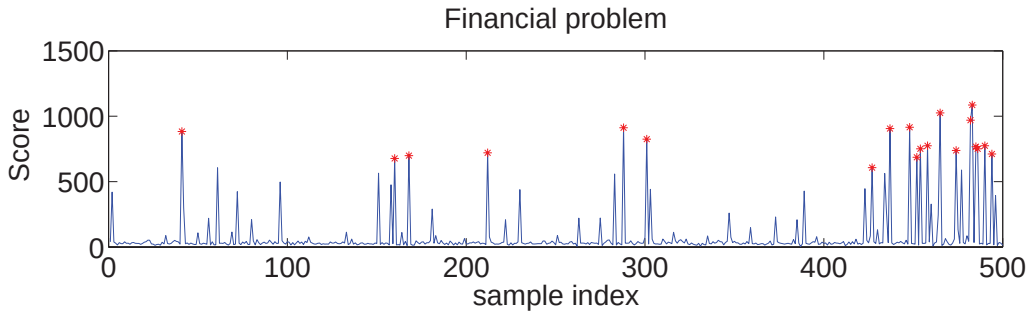


Figure 3.39: Sample scores for the bankruptcy prediction dataset. Expert-identified originally mislabeled samples are denoted with red stars.

mislabeled. Taking into account the experts classification, the proposed method seems to be successful in 16 mislabeled samples out of 20. These selected mislabel samples will be investigated in detail in the future by other financial experts using more information about the selected companies. It can be considered that the proposed methodology is successful since only 20 companies have to been analyzed furthermore instead of the initial 500 companies, to identify actual mislabels.

3.4 Confidence Intervals for ELM Predictions

ELMs are powerful nonlinear methods, but they share one common drawback of nonlinear methods in practical applications, which is a non transparency of results (predictions). A prediction made by a linear model from input data is easily explained and interpreted by observing the coefficients of the input data features. Results with an explanation are easier to trust and apply for people outside a Machine Learning field. nonlinear models lack such transparency, so their results are less trusted, and thus nonlinear methods (including ELM) are sometimes rejected despite a supreme performance compared to linear methods.

This paper proposes a way of providing transparent and interpretable results for ELM models by adding confidence intervals [SH15, LJRV05, PLB10] to predictions. Unlike the usual statistical approach with Mean Squared Error (MSE) [Bis06] that evaluates an average performance of an ELM model over the whole dataset, the proposed method computes particular confidence intervals for each data sample. These intervals are small for samples on which a model is accurate, and large for samples where a model is unstable and inaccurate. A confidence for each particular sample makes ELM predictions more intuitive to interpret, and an ELM model better applicable in practice under task-specific requirements to precision and recall of predictions.

3.4.1 Methodology

Intuition

Confidence of a predicted output for a data sample, and the confidence intervals, are properties of a particular dataset. A training sample alone is an input-output pair $(\mathbf{x}, \mathbf{y})$, where an output $\mathbf{y}$ is a fixed number (or vector) without *confidence* notion.

When many training samples $(\mathbf{x}_i, \mathbf{y}_i),\ i \in [\![1, N]\!]$ are gathered in a dataset $(\mathbf{X}, \mathbf{Y})$, that dataset represents an implicit projection function $f : \mathbf{X} \to \mathbf{Y}$. An ELM approximates function f by a smooth function $\hat{f} : \mathbf{X} \to \hat{\mathbf{Y}}$. The smoothness (enforced by regularization) improves generalization performance, but it prohibits the function $\hat{f}$ to pass exactly through points $(\mathbf{x}_i, \mathbf{y}_i)$. Instead, $\hat{f}$ passes through $(\mathbf{x}_i, \hat{\mathbf{y}}_i)$, and a true output $\mathbf{y}_i$ is assumed to be randomly drawn from a normal distribution centered at a predicted output $\hat{\mathbf{y}}_i$. Here random values and confidence intervals come into the ELM prediction model. A confidence interval is simply a scaled standard deviation of an output for sample $(\mathbf{x}_i, \mathbf{y}_i)$, which tells how reliable a corresponding predicted output $\hat{\mathbf{y}}_i$ is.

Confidence Intervals for Predictions

Confidence interval is a property of a particular dataset, not a sample by itself. For a test sample a, if training samples with inputs like a have similar outputs, then an output of a is predicted accurately. It is still predicted accurately if some of these training samples are missing. For another test sample b, if training samples with inputs like b have very different outputs, then an output of b is not known

precisely. Also if some of these training samples are missing, prediction for b can change drastically. See Figure 3.40 for a visual example.

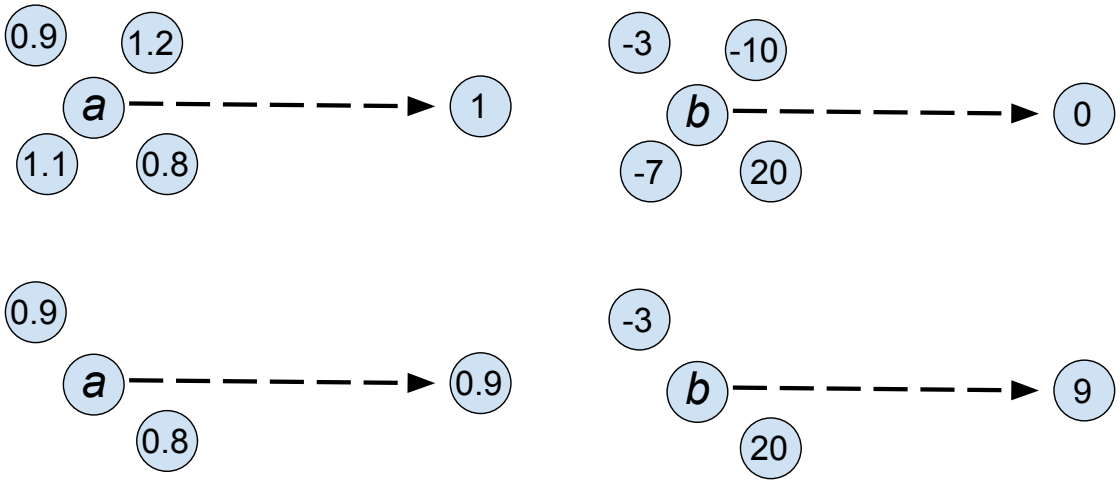


Figure 3.40: Idea of the confidence intervals method: difference between samples with a small confidence interval (a, left) and a large confidence interval (b, right) comes from a dataset. Using a subset of training data affects b much more than a.

It is worth noting that if confidence intervals for all data samples are equal, the corresponding standard deviation is given by a Root Mean Squared Error (RMSE) [Bis06]. Confidence intervals are then obtained by scaling a standard deviation to the desired coverage percentage; for instance multiplying by 2 to get 95.4% or by 3 to get 99.7% intervals.

Per-sample Confidence Intervals

Per-sample confidence intervals method is based on an uneven increase of prediction error for different samples, as the training set size shrinks. This increase is larger for large confidence interval samples than for small confidence interval ones. The error increase is taken proportionally to an average error increase on a test set for the same setup, and that proportion is applied as a scaling factor for RMSE to get a per-sample confidence interval.

The dataset is divided into k non-intersecting subsets, so as to find the error increase. As that increase varies greatly with a particular initialization, experiments are repeated with all k subsets and results are averaged. Moreover, the value k is varied from 2 to a large number (for example 10 or 50), at which point the training set becomes so small that ELM over-fits and produces a high error on the validation set. ELM over-fits because the model structure is selected for the whole dataset, and fixed. The experiments ignore ELM models if they produce three times higher RMSE on a reduced dataset than on a full dataset, and stopped when the number of ignored ELMs reaches k (every second model is ignored).

Define a dataset consisting of three parts: training set $(\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$, validation set $(\mathbf{X}^{\text{val}}, \mathbf{Y}^{\text{val}})$ and test set $(\mathbf{X}^{\text{test}}, \mathbf{Y}^{\text{test}})$ where $\mathbf{Y}^{\text{test}}$ is unknown. Let $\hat{\mathbf{Y}}$ denote outputs predicted by ELM, and a subscript $\cdot_k$ denotes data or model obtained on a $1/k^{\text{th}}$ part of the training set.

First, two ELM models are trained as on Figure 3.41 — one on a full dataset, and another on a $1/k^{\text{th}}$ part of it. Denote the second ELM as ELM_k. An RMSE

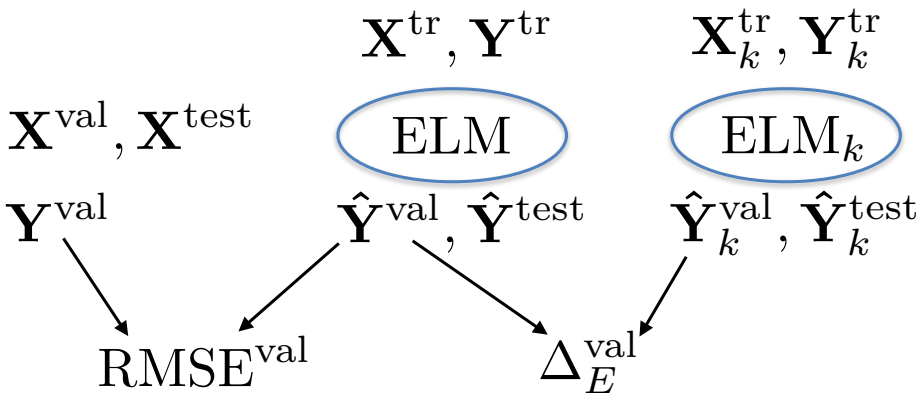


Figure 3.41: First stage of confidence intervals: computing an average statistics. ELM is trained on a whole training set, while ELM_k on a $1/k^{\text{th}}$ part of it.

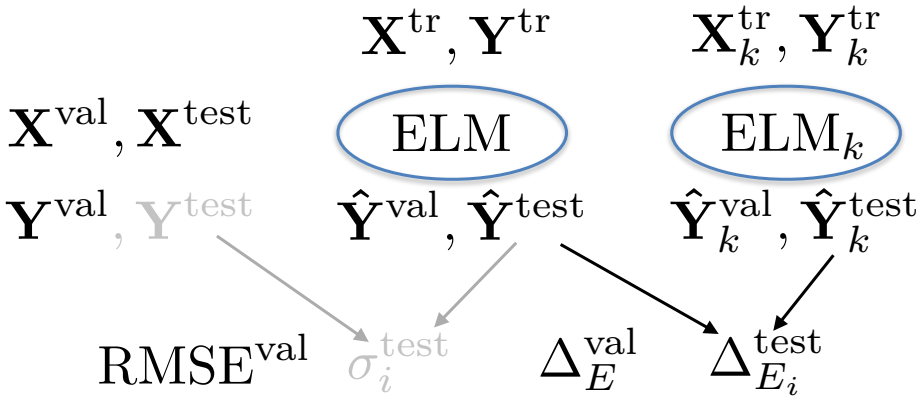


Figure 3.42: Second stage of confidence intervals: finding per-sample standard deviation. As $\mathbf{Y}^{\text{test}}$ does not exist, only $\Delta_{E_i}^{\text{test}}$ can be computed directly, and σ_i^{test} is obtained from a proportion.

is computed on a validation set for ELM. An error increase Δ_E^{val} is computed between predictions of ELM and ELM$_k$, and it describes how a smaller training set increases validation error. These two values are computed per-dataset, averaged over all validation set samples.

Second, per-sample standard deviations σ_i^{test} are found as on Figure 3.42. Unfortunately the $\mathbf{Y}^{\text{test}}$ does not exist, so only per-sample error increase between ELM and ELM$_k$ is possible to compute directly. The σ_i^{test} is then obtained from a following expression.

$$\sigma_i^{\text{test}} = \text{RMSE}^{\text{val}} \frac{\Delta_{E_i}^{\text{test}}}{\Delta_E^{\text{val}}} \tag{3.35}$$

The expression is obvious because for constant confidence intervals, a Mean Squared Error equals to the variance of prediction error as explained in [Bis06], so an RMSE equals to the standard deviation of the same. Confidence intervals themselves are found by multiplying σ_i^{test} by a factor corresponding to the desired confidence level, for examples a factor 2 gives 95.4% confidence intervals.

ELM Confidence Intervals Algorithm

Values $\Delta_{E_i}^{\text{test}}$ are sensitive to a random initialization of ELM and the particular training samples of the $1/k^{\text{th}}$ part of a dataset. Thus they are averaged twice: first by repeating the experiment k times with all $1/k^{\text{th}}$ parts of a dataset, and second by varying k from 2 until ELM$_k$ becomes unstable (about 10-50). A side advantage of repeating the method multiple times are precise predictions $\hat{\mathbf{Y}}^{\text{test}}$ averaged over

Algorithm 3.3 ELM Confidence Intervals Algorithm

given data sets $(\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$, $(\mathbf{X}^{\text{val}}, \mathbf{Y}^{\text{val}})$ and $\mathbf{X}^{\text{test}}$

Select optimal ELM model parameters on $(\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$

for $k = 2 \dots \frac{N}{10}$ **do**

Split $(\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$ into k parts $(\mathbf{X}^{\text{tr}}_j, \mathbf{Y}^{\text{tr}}_j)$, $j \in [\![1, k]\!]$

for $j = 1 \dots k$ **do**

initialize ELM model m and train it with $(\mathbf{X}^{\text{tr}}, \mathbf{Y}^{\text{tr}})$

predict $\hat{\mathbf{Y}}^{\text{test}}$, predict $\hat{\mathbf{Y}}^{\text{val}}$ and find $\text{RMSE}^{\text{val}}_{k,j}$

re-train m with $(\mathbf{X}^{\text{tr}}_j, \mathbf{Y}^{\text{tr}}_j)$

predict $\hat{\mathbf{Y}}^{\text{val}}_j$ and find $\Delta^{\text{val}}_{E,k,j}$

if $\Delta^{\text{val}}_{E,k,j} > 3 * \text{RMSE}^{\text{val}}_{k,j}$ **then**

Stop the main loop, ignore results for current j

end if

predict $\hat{\mathbf{Y}}^{\text{test}}_j$ and find per-sample $\Delta^{\text{test}}_{E_i,k,j}$

end for

end for

find RMSE^{val}, Δ^{val}_E and $\Delta^{\text{test}}_{E_i}$ by averaging over j and k

Find σ^{test}_i as in equation (3.35)

Multiply σ^{test}_i by a factor corresponding to desired confidence interval percentage

Report average $\hat{\mathbf{Y}}^{\text{test}}$ with the confidence intervals

numerous ELM models. An algorithm of confidence intervals method for ELM is presented in Algorithm 3.3.

3.4.2 Experiments

Artificial Dataset

This artificial dataset has one-dimensional inputs and targets, for ease of visualization. The task is to predict a highly nonlinear function f (shown on Figure 3.43), constructed as a sum of two sine waves. The dataset is created by adding normally distributed zero-mean noise, which has input-dependent variance to simulate variable confidence intervals. The training and validation sets have 1000 data samples each.

Effect of Splitting Dataset on Predictions

The ELM used in the method is tuned for the full dataset size. It gives precise predictions when trained on a full training set, or on a large part of it at a low split level k. When k raises, the training set size shrinks and ELM learns worse predictions. It over-fits ofter at high split levels; so the trained model is validated and refused if it exceeds the validation error threshold. The method stops if the current split level starts producing over-fitted models. Effect of training an ELM at high split levels is shown on Figure 3.44.

Effect of splitting dataset on particular samples is shown on Figure 3.45. The increase in error for each particular sample as the dataset size shrinks varies a lot (top left plot), but averaging over multiple k splits provides a reliable estimate. The original $\Delta_{E_i}^{\text{test}}$ is scaled with RMSE^{val} to obtain σ_i^{test}. Then this σ_i^{test} is multiplied by

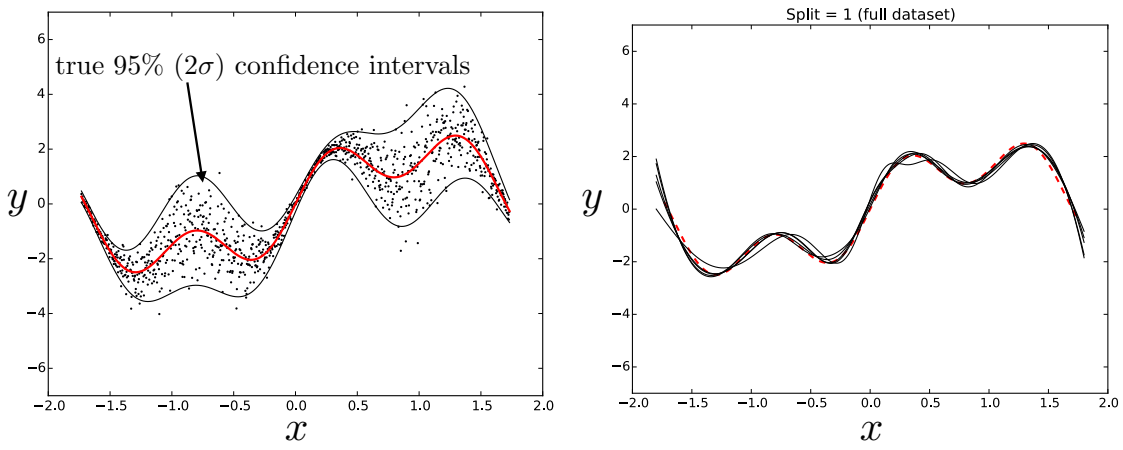


Figure 3.43: Toy dataset and its predictions with five different ELM models, trained on the whole dataset. Points are samples, red line is the original function.

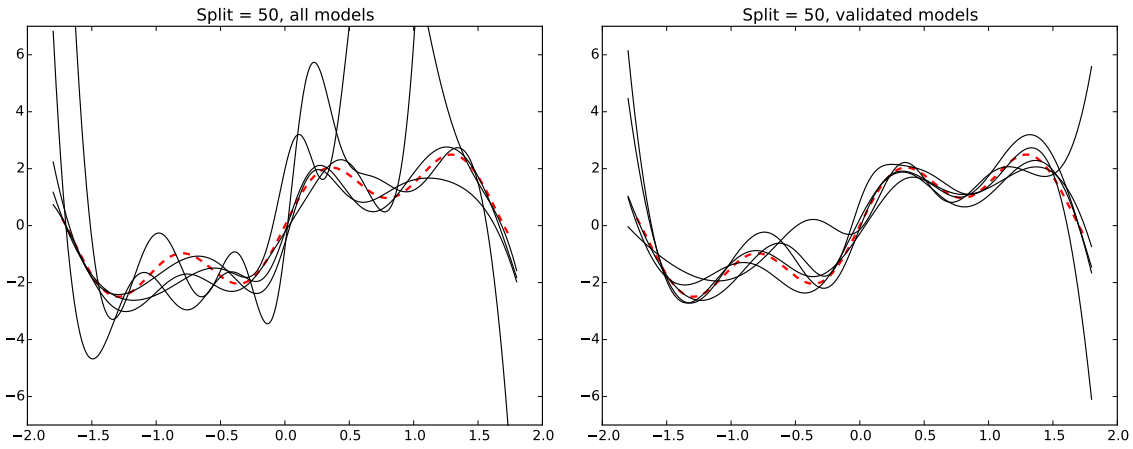


Figure 3.44: Validation of ELM models in confidence intervals. At high split levels, trained ELM model easily overfits (*top*) due to a smaller training set. Overfitted models are sorted out by a validation error threshold (*bottom*).

a scaling factor to show the confidence intervals themselves (top right plot).

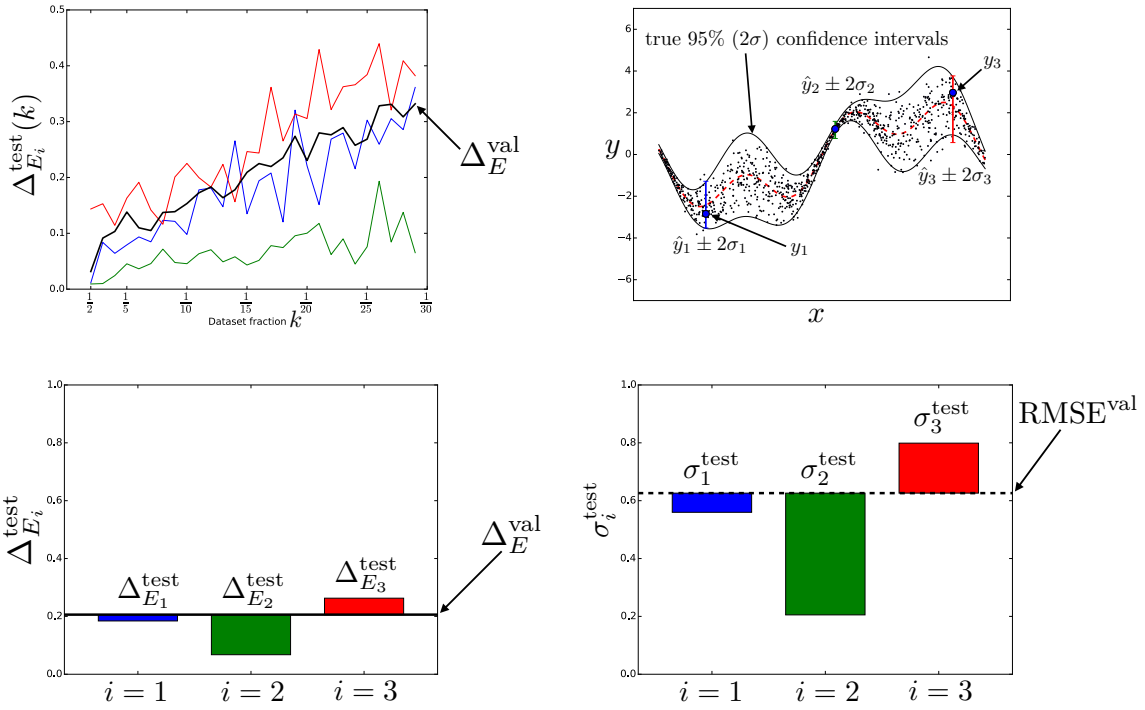


Figure 3.45: Example of finding confidence intervals for three test samples, denoted by red, green and blue colour.

Results on an Artificial Dataset

The 95% confidence intervals on an artificial dataset are shown on Figure 3.46. The obtained confidence intervals follow the true ones well in the middle of prediction range, but grow larger towards the boundaries. The reason is that many ELM models give unstable predictions at boundaries (see Figure 3.44, right) so that follows from a property of an ELM itself. In general the confidence intervals are little exaggerated

compared to the true ones, but this again can be explained by some uncertainty of ELM predictions in general.

Confidence intervals (both true and estimated ones) for datasets with the same projection function but a constant variance noise are shown on Figure 3.47. The method learns and predicts constant variance noise as well (except on boundaries), so it can be used universally for evaluating both variable and constant noise confidence boundaries.

The experiments use ELM model with 11 sigmoid neurons. The runtime is 0,5-1 minute on a 1.4GHz Core i5 laptop.

3.4.3 Skin Color Dataset

Confidence intervals for ELM predictions are tested on a Face/Skin Detection dataset [PBC05], a useful Big Data benchmark [CAK+15] with a non-trivial prediction task. It includes 4000 real-world photos of different people under various lighting conditions, with manually created masks for skin and faces. The paper uses the original dataset separation into 1300 training, 700 validation and 2000 test images.

A skin detection task is used for a real-world illustration of method performance in the paper, because of a good visualization it provides. It is a binary classification task, turned into a regression one by assigning skin pixels a +1 output, non-skin pixels a -1 output, and choosing valid skin/non-skin predictions that are $\alpha\sigma$ distance from zero (with α corresponding to a desired confidence level). The prediction is done per each pixel independently, with a 7×7 pixel mask for an input

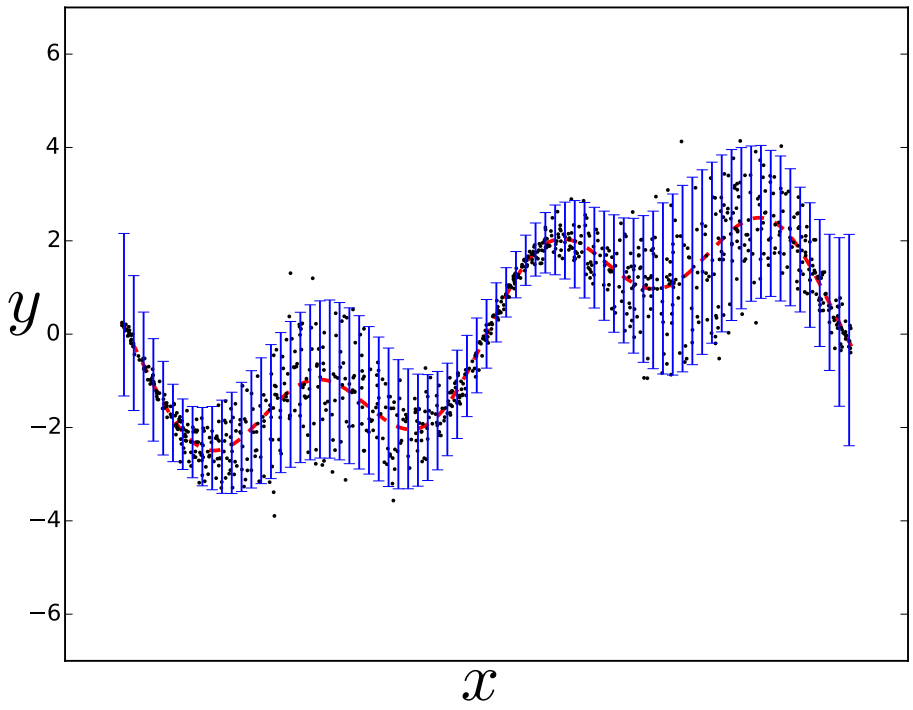


Figure 3.46: 95% confidence intervals on an artificial dataset. These intervals are larger towards the boundaries of prediction range, but the general trend is observed well.

($7 \times 7 \times 3 = 147$ input features in RGB color space). A 3-pixel image border is ignored because a full pixel mask is not obtainable there.

Training and validation sets are balanced in skin/non-skin classes, and made by randomly selecting 800 skin and 800 non-skin pixels from each training and validation image. A test set consists of all pixels from a single image (shown on Figure 3.48). This results in two million training samples, one million validation and 470.000 test. The method uses ELM with 147 linear and 1000 sigmoid neurons, runs for a maximum of 15 splits in single or double precision with GPU acceleration (GTX Titan Black) provided by an ELM toolbox [ABML15]. Experiment in single precision takes only 11 minutes to complete.

Skin predictions for different confidence levels are shown on Figure 3.49. It works well by correctly predicting the skin even at a high confidence. The skin-coloured floor is predicted as skin at lower confidence levels, but is excluded as confidence threshold increases, which is an expected behaviour and confirms that the method is working well. Recall for skin is good even at 99.5% confidence because most of skin in a photo is predicted correctly.

The non-skin prediction is shown on Figure 3.50. It is apparently a much harder task than skin prediction because non-skin has higher variety, and indeed the photo has skin-toned background. Still the white dress and walls are predicted as non-skin, with white color predicted as a highly confident non-skin.

The confidence level is an interesting thing to analyze. It starts from a histogram of σ values, shown on Figure 3.51. The histogram shows three different

regions in the value of σ: $\sigma < 0.4$, $0.4 < \sigma < 0.7$ and $\sigma > 0.7$. Images corresponding to those regions are presented on Figure 3.52. The most confident image (Figure 3.52, *left*) clearly shows the skin, the non-skin walls at the back, and some of the yellow floor which is indeed skin color. The average confidence image (Figure 3.52, *center*) shows most of the floor and some hair. The least confident image (Figure 3.52, *right*) presents edges of the skin, facial features (which are not labeled as skin in the original dataset) and the rest of hair. The skin and face edges are predicted least confidently (with the largest confidence interval), that confirms correct behavior of the method and it's ability to separate uniform high-confidence areas from low-confidence transition regions.

Another interesting discovery emerges at ELM computed and solved in single-precision, shown on Figure 3.53 (note that all the previous results are obtained with double-precision ELM). Single-precision computations are much faster on GPU due to GPU inner structure, and the availability of corresponding BLAS functions and solvers. The predicted skin results are similar to double-precision ELM at 66% and 95% confidence levels, but single precision apparently is not enough to ensure 99.7% confidence of the results. This behaviour was not predicted in the experiments, but it illustrates the connection between the provided "confidence interval" and the actual confidence in the predicted results.

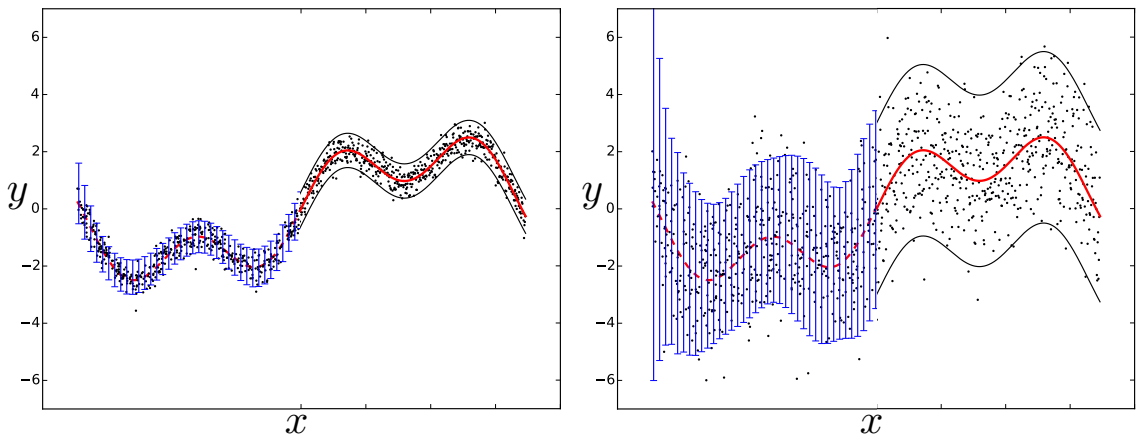


Figure 3.47: 95% confidence intervals on an artificial dataset with constant noise. These intervals are larger towards the boundaries of prediction range, but the general trend is observed well.

Figure 3.48: The original test image for skin pixels classification.

Figure 3.49: Predicted skin with 66%, 95%, 99.5% confidence.

Figure 3.50: Predicted non-skin with 66%, 95%, 99.5% confidence. Non-skin is harder to detect due to its variety.

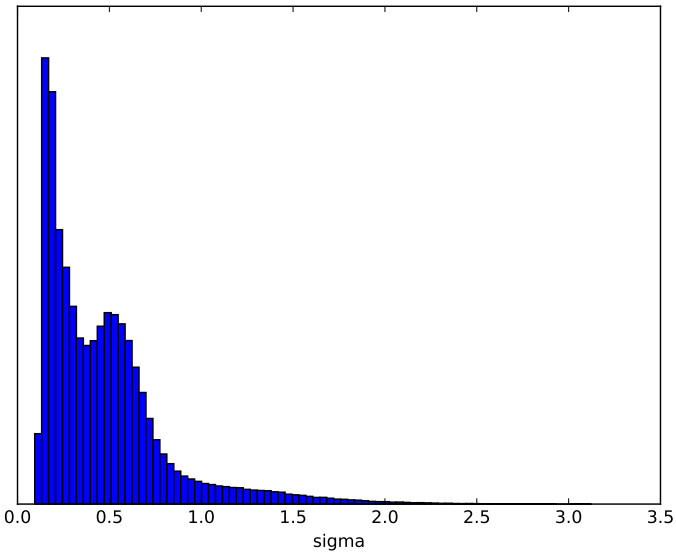


Figure 3.51: Histogram of σ values, showing a separation in three regions: $\sigma < 0.4$, $0.4 < \sigma < 0.7$ and $\sigma > 0.7$.

Figure 3.52: Images with transparency mask corresponding to different regions of σ: $\sigma < 0.4$ (*left*), $0.4 < \sigma < 0.7$ (*center*) and $sigma < 0.7$ (*right*).

Figure 3.53: Predicted skin with 66%, 95%, 99.5% confidence for single-precision ELM. Single precision is not enough to provide highly confident predictions. This is an interesting finding rather than an expected functionality.

CHAPTER 4
EXTREME LEARNING MACHINES FOR BIG DATA

Big Data is different from conventional Machine Learning or Data Mining in the sense that the available data has the potential to become larger than can be processed by simple tools, is more unstructured and heterogeneous in various aspects (e.g., different sources, sampling rates, data types, reliability and precision, etc.) and often has a crowd-sourcing aspect. Rather than defining Big Data by size (in, e.g., terabytes or petabytes), Big Data is defined here as data that require entirely novel and specific evaluation procedures. The Machine Learning process must be revised in order to place significantly more emphasis on investigating novel ways of data cleansing, facilitating the transformation from raw, unstructured data to structured data, aligning (temporal) data, data fusion, inferring missing data, and resolving contradicting data.

Extreme Learning Machine is one of the best suited methods for dealing with Big Data, because it immediately resolves the data size limitation. Only things need to be addressed are an adjusted algorithms to satisfy limited computer memory constraints and a basic parallelization; and a data pre-processing which is very problem-specific and is tuned to adapt for the particular demands of the task in question.

4.1 ELMs for Processing Big Data

Extreme Learning Machine is a good candidate method for solving Machine Learning problems on Big Data, or large datasets in general. But an original ELM faces multiple challenges when dealing with such tasks.

ELMs easily run out of memory for storing the matrix $\mathbf{H}$ with large number of data samples and hidden neurons. Previously this problem was tackled by iteratively updating the output weights. However, these methods are computationally slower because they perform updates of large matrices for each data sample [vT15], or need to calculate a solution repeatedly [HZDZ12].

This chapter introduces Big Data adaptations of an ELM model itself, as well as the corresponding software and hardware tools to help addressing the Big Data challenges.

4.1.1 Iterative Solution of ELMs

The original solution of an output layer equation in Extreme Learning Machines require a pseudo-inverse of matrix $\mathbf{H}_{[N\times L]}$ to solve $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$ matrix equation. The size of $\mathbf{H}$ grows linearly with the number of training samples in a dataset, exceeding an available computer memory for very large or Big Data problems, and restricting the use of accelerators with typically a relatively small amount of memory compared to computer nodes in a cluster. Also, the pseudo-inverse $\mathbf{H}^{\dagger}$ is a computationally heavy operation, which takes a large amount of time to complete on a single machine.

However, there exist an iterative batch solution of the same $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$ equation; *batch* means that matrix $\mathbf{H}$ is processed by a single batch $\tilde{\mathbf{H}}_{[b\times L]}$ of arbitrary size $b < N$ at a time, iterating until the whole matrix $\mathbf{H}$ is used. The obtained solution $\boldsymbol{\beta}$ is numerically equal to the pseudo-inverse solution and has the same complexity in terms of $\mathcal{O}()$ notation. The memory requirements are reduced from NL to $(L+b)L$ and are independent of the number of training data samples. Also, most of the computational complexity ($> 99\%$ in large practical tasks) resides in the batch stage. This allows an embarrassingly parallel [Fos95] implementation with a large theoretical speedup allowed by Amdahl's law [Amd67]. The final stage requires a simple summation of batch results, followed by a solution with complexity $\mathcal{O}(L^3)$ instead of $\mathcal{O}(NL^2)$.

ELM Solution with Best Linear Unbiased Estimator

Iterative batch solution of ELM is derived from the best linear unbiased estimator, that gives the optimal least squares solution to the matrix equation $\mathbf{X}\boldsymbol{\beta} = \mathbf{T}$ for stochastic vectors $\mathbf{x}$ and $\mathbf{t}$ combined into the corresponding matrices. It uses two theoretical correlation matrices

$$\mathbb{E}[\mathbf{x}^T\mathbf{x}] = \mathbf{C}_{xx},\ \mathbb{E}[\mathbf{x}^T\mathbf{t}] = \mathbf{C}_{xt} \tag{4.1}$$

which are assumed to be known. The best linear unbiased estimator of $\mathbf{T}$, denoted by $\mathbf{Y}$, is then

$$\mathbf{Y} = \mathbf{C}_{xx}^{-1}\mathbf{C}_{xt}\mathbf{X} = \boldsymbol{\beta}\mathbf{X}. \tag{4.2}$$

The inverse of $\mathbf{C}_{xx}$ exists because x is a stochastic variable for which $\mathbf{C}_{xx} = \mathbb{E}[x^T x]$ has a full rank.

The output layer problem of an ELM has a finite amount of projected data samples $\mathbf{H}$ and corresponding targets $\mathbf{T}$, so the correlation matrices are replaced by their estimations

$$\mathbf{C}_{xx} \approx \mathbf{H}^T\mathbf{H} = \mathbf{\Omega}_h, \ \mathbf{C}_{xt} \approx \mathbf{H}^T\mathbf{T} = \mathbf{\Omega}_t, \tag{4.3}$$

and the ELM output weights are computed from those estimates

$$\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{H})^{-1}(\mathbf{H}^T\mathbf{T}) = \mathbf{\Omega}_h^{-1}\mathbf{\Omega}_t. \tag{4.4}$$

The inverse of $\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H}$ matrix exists if it has full rank. In ELM model, the nonlinear random projection produces almost orthogonal features which are linearly independent. If the number of hidden neurons (columns of $\mathbf{H}$) is smaller than the number of training samples (rows of $\mathbf{H}$), then the rank of matrix $\mathbf{H}$ equals its number of columns, thus matrix $\mathbf{H}^T\mathbf{H} = \mathbf{\Omega}_h$ is full rank and its inverse exists. Otherwise, an ELM requires some regularization to obtain a non-overfitted solution, which again leads to a full rank of $\mathbf{\Omega}_h$ matrix.

Batch ELM Solution Complexity Analysis

The comparison of computational complexity and memory requirements for the pseudo-inverse versus correlation matrices ELM solutions are presented in table 4.1.

A memory requirement of a correlation ELM solution is constant for any number of training samples, because the correlation matrices $\mathbf{\Omega}_h$ and $\mathbf{\Omega}_t$ can be computed for batches of training data. The batch computation replaces the number of samples N in the memory requirements by a batch size. A good trade-off in terms of memory

Table 4.1: ELM computation and memory requirements; computations along the dimension $\tilde{N}$ can be performed in parallel in L-size batches.

Operation	Comp. complexity	Memory
Projection to hidden layer		
$\mathbf{X}_{N\times d}$	$\mathcal{O}(\tilde{N}d)$	$\mathcal{O}(\tilde{N}d)$
$\mathbf{H}_{N\times L} = f(\mathbf{XW} + \mathbf{b})$	$\mathcal{O}(\tilde{N}Ld + \tilde{N}L)$	$\mathcal{O}(\tilde{N}L)$
Pseudo-inverse solution		
$\mathbf{A}_{L\times N} = \mathbf{H}^{\dagger}$	$\mathcal{O}(NL^2 + L^3)$	$\mathcal{O}(NL)$
$\boldsymbol{\beta}_{L\times c} = \mathbf{AT}$	$\mathcal{O}(\tilde{N}Lc)$	$\mathcal{O}(\tilde{N}L + \tilde{N}c)$
Correlation matrices solution		
$\mathbf{A}_{L\times L} = \mathbf{H}^T\mathbf{H}$	$\mathcal{O}(\tilde{N}L^2)$	$\mathcal{O}(L^2)$
$\mathbf{B}_{L\times c} = \mathbf{H}^T\mathbf{T}$	$\mathcal{O}(\tilde{N}Lc)$	$\mathcal{O}(Lc)$
$\boldsymbol{\beta}_{L\times c} = \mathbf{A}^{-1}\mathbf{B}$	$\mathcal{O}(L^3 + L^2c)$	$\mathcal{O}(L^2 + 2Lc)$
Comparison of solutions		
$\boldsymbol{\beta}_{\dagger} = \mathbf{H}^{\dagger}\mathbf{T}$	$\mathcal{O}(NL^2 + L^3)$	$\mathcal{O}(NL)$
$\boldsymbol{\beta}_{\texttt{corr}} = (\mathbf{H}^T\mathbf{H})^{-1}(\mathbf{H}^T\mathbf{T})$	$\mathcal{O}(\tilde{N}L^2 + L^3)$	$\mathcal{O}(L^2 + 2Lc)$

requirement and computational overhead is achieved with a batch size equal to L. The final $\boldsymbol{\Omega}_h$ and $\boldsymbol{\Omega}_t$ are computed from batches by a simple summation. The summation operation adds no runtime overhead, because in software and hardware implementations, matrix multiplication and summation are performed in a single operation[1]: $\texttt{gemm}(\mathbf{A}, \mathbf{B}, \mathbf{C}) = \mathbf{AB} + \mathbf{C}$.

[1] `http://www.netlib.org/blas/#_level_3`

4.1.2 Accelerated ELM

In ELM implementation with best linear unbiased estimator, only the $\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H}$ and $\mathbf{\Omega}_t = \mathbf{H}^T\mathbf{T}$ matrices need to be kept in memory — with sizes being independent of the number of data samples N. Furthermore, these matrices may be computed in k separate batches, which reduces the memory requirement for storing the $\mathbf{H}$ matrix k times.

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^1 \\ \vdots \\ \mathbf{H}^k \end{bmatrix}, \quad \mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H} = \mathbf{H}^{1T}\mathbf{H}^1 + \ldots + \mathbf{H}^{kT}\mathbf{H}^k. \tag{4.5}$$

The batch computation of $\mathbf{H}^T\mathbf{H}$ is a simple matrix multiplication with relatively low memory requirements and a high computational cost (constituting more than 95% of runtime for an ELM with $L > 10,000$), so this, along with $\mathbf{H}^T\mathbf{T}$, are ideal parts for GPU acceleration, which significantly reduces the total ELM computational time. The output matrices are accumulated in the GPU memory, and the solution of $\boldsymbol{\beta}$ from $(\mathbf{H}^T\mathbf{H})\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{T})$ is also accelerated by GPU, although this operation is fast anyway because of a very fast available solution for the symmetric positive semi-definite matrices involved.

4.1.3 Parameters of Generated Random Weights

Sigmoid function is a common choice of a nonlinear transformation function for hidden nodes of ELM. However, it is sensitive to the range of input weights, which are $\mathbf{XW} + \mathbf{b}$. If inputs to the sigmoid function have small magnitude, it performs similarly to linear function. If these inputs have very large magnitude, it performs as a cutoff value. The effect can be seen by checking the difference between predictions of

SLFNs with the same weights and sigmoid/linear/threshold transformation functions. If the input data has zero mean and unit variance, the range of inputs to SLFN is governed by the range of weights $\mathbf{W}$ generated from $\mathbf{W} = \mathcal{N}(0, s)$ with different values of standard deviation s. The range of weights $\mathbf{W}$ also affects the performance. The effects are shown on Figure 4.1.

Another issue is an increase in standard deviation of inputs to the transformation function, if the dataset has high dimensionality. For a single additive hidden layer neuron, an input to the transformation function $a_k = \sum_{i=1}^{d} x_i w_{i,k}$ is a sum of d components. If a single component has standard deviation s, then that sum has a larger standard deviation $\sqrt{d}s$. This leads to larger magnitudes of inputs to a transformation function and sub-optimal performance for large d, for example in MNIST dataset (see Figure 4.2). The effect of large input dimensionality is fixed by dividing the standard deviation s by $\sqrt{d}$, and generating weights as $\mathbf{W} = \mathcal{N}(0, s/\sqrt{d})$ (see Figure 4.3).

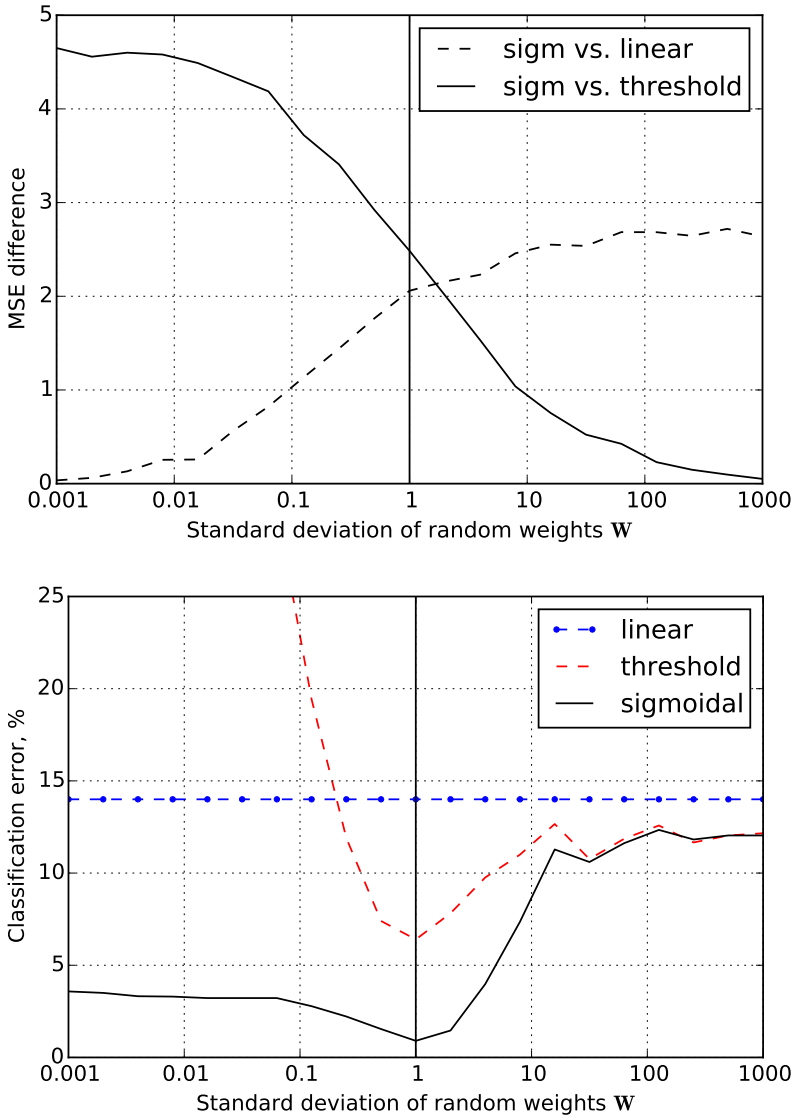


Figure 4.1: Mean squared error difference (*top*) of predictions of SLFNs with 5 hidden neurons, and test error of SLFNs (*bottom*) with 25 hidden neurons, for different values of s in $\mathbf{W} = \mathcal{N}(0, s)$. For small s, outputs of sigmoid SLFN are similar to linear SLFN, and for large s they are similar to threshold SLFN. The data is Iris dataset (average over 100 runs) with 100 training and 50 test samples, balanced over the 3 classes.

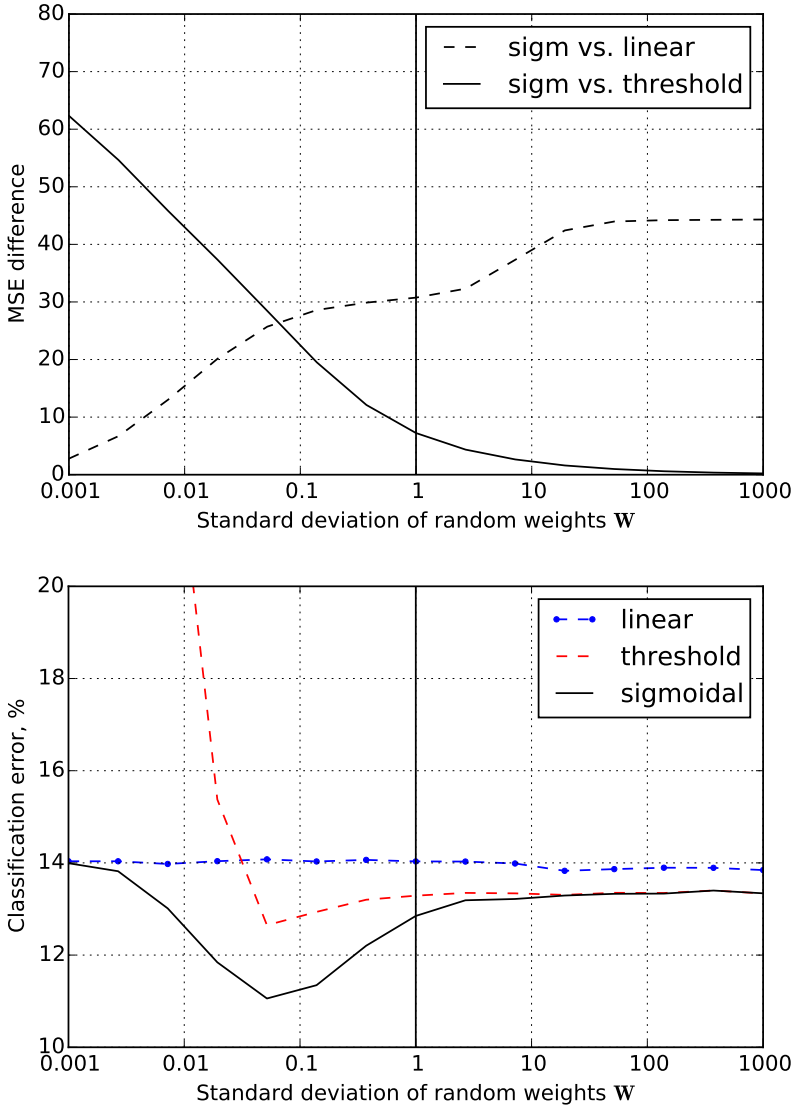


Figure 4.2: MSE difference (*top*) of predictions, and test error (*bottom*) of SLFNs with 500 hidden neurons on MNIST dataset, for different values of s in $\mathbf{W} = \mathcal{N}(0, s)$. The data has 60000 training and 10000 test samples, results are averaged over 10 runs. Due to high dimensionality of inputs, the optimal value of s differs from 1.

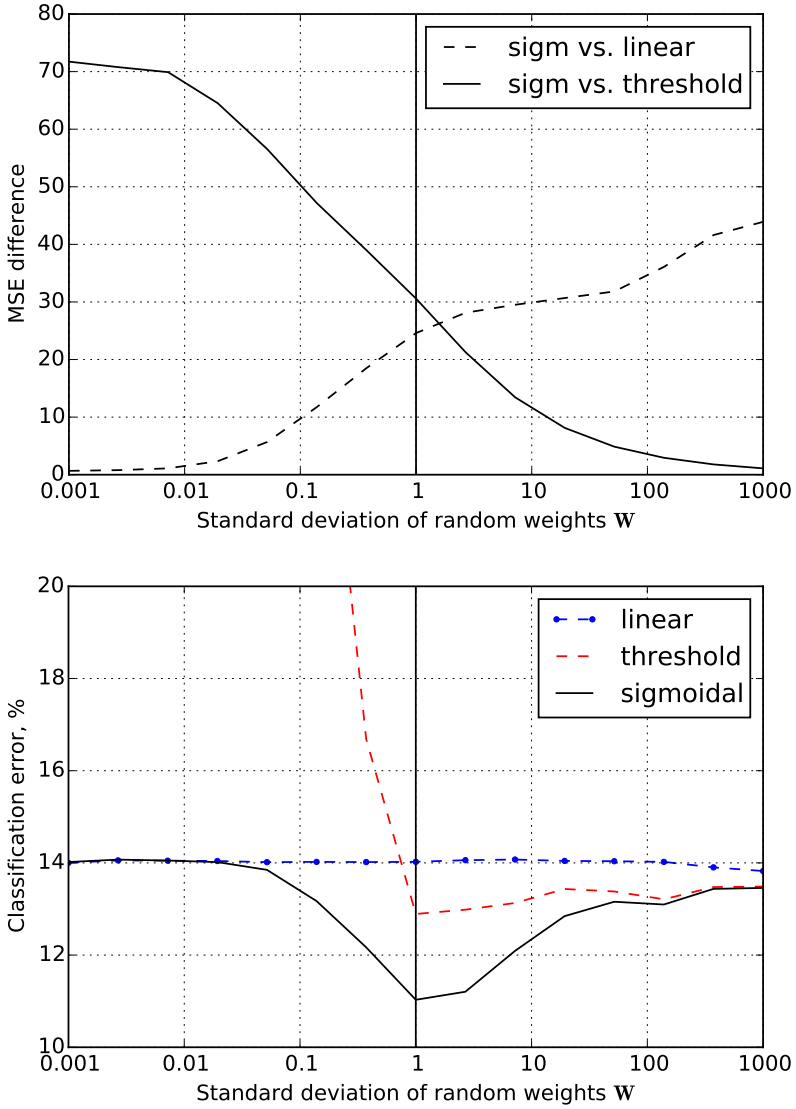


Figure 4.3: MSE difference (*top*) of predictions, and test error (*bottom*) of SLFNs with 500 hidden neurons on MNIST dataset, for different values of s. Results are averaged over 10 runs. With input dimensionality fix $\mathbf{W} = \mathcal{N}(0, s/\sqrt{d})$, the optimal value of s is around 1 even for high dimensional data.

4.2 HP-ELM Toolbox for Big Data Processing

Prior to this work, there exists only two ELM toolboxes [LHSS06, vT15] of all[2] available can process a dataset larger than a given computer memory, and they both implement a particular method rather than focus on overall ELM performance. A GPU acceleration [vML+09, vMOL11] speeds up the computation significantly, but there is no ready to use implementation before the proposed toolbox.

The purpose of developing and publishing a new high-performance ELM implementation is to approach the vast field of Extreme Learning Machines from a practical performance point of view, and to provide an efficient and easy toolbox, which saves time of researchers and data analysts desiring to apply ELM to their existing problems. An analysis of training methods is done in this piece of software, to select the fastest, least bounded by memory, scalable and simplest way of training ELMs. An efficient implementation is created which suits even old machines of low performance, and the software also handles Big Data on modern workstations with accelerators. The proposed toolbox includes all major model structure selection options and regularization methods, tools for Big Data pre-processing and parallel computing. In the next two sections we explain theoretical and practical aspects of the ELMs methodology. Section 4.2.2 explains the actual ELM toolbox, and section 4.2.3 compares and discusses on the toolbox performance on various datasets, including test sets of Big Data.

[2] `http://www.ntu.edu.sg/home/egbhuang/elm_codes.html`

4.2.1 ELMs in Practice

Data Normalization

Input data normalization is a critical preprocessing step for many Machine Learning methods, including ELMs. Raw data often has features of different scales, for example an age of a man is at a scale 1-100, and his annual salary in dollars is 3 orders of magnitude larger. Without normalization, small relative variations in the salary make large relative variations in the age negligible. Normalization sets all features at the same scale. Then all features have the same influence, and the training method learns which ones to use for the prediction.

In the ELM toolbox, weights can be given explicitly or generated automatically. Automatic weights generation assumes that the data has zero mean and unit variance. The generated weights keep the performance of neural network with sigmoid neurons near the optimum, and compensate for large number of inputs. For the explanation and experimental evaluation of the automatic random weights parameters, refer to section 4.1.3.

Numerical Stability of an ELM Solution with Correlation Matrices

If numerical instabilities are faced in the inverse, a regularization term is applied to the correlation matrix $\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H} + \alpha\mathbf{I}$, where α is a small positive constant. This approach is called Ridge Regression [HK70] aka. Tikhonov regularization [Tik63]. A greater than zero parameter α reduces the effective number of variable in the model, increasing the inverse stability but decreasing predictive power. The

default Ridge regression is used in all matrix inverse functions of Python (Numpy) and Matlab® with $\alpha = 50\epsilon$ where ϵ is a machine precision constant.

Extreme Learning Machines also benefit greatly from model structure selection and regularization, which reduces possible negative effects of random initialization and over-fitting. The methods include L^1 (see section 2.2) and L^2 (section 2.3) regularization, as well as other methods [YME+13] like handling imbalance classification [ZHC13]. A reason to include these methods is that they are typically computationally intensive, and challenging to implement efficiently for Big Data.

Weighted Classification with ELMs

In a classification task with highly uneven number of data samples for different classes, ELM predictions are biased towards the class with the most data. This behaviour is improved by using a weighted linear system solution in the output layer of an ELM [ZHC13]. A weighted linear system has a Least Squares solution similar to the BLUE solution:

$$\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{A}\mathbf{H},\ \mathbf{\Omega}_t = \mathbf{H}^T\mathbf{A}\mathbf{T}, \tag{4.6}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is an arbitrary weight matrix. If only sample weights are used, the $\mathbf{A}$ matrix is diagonal; but these weights are complicated to obtain if they are not given explicitly. In a classification task, diagonal elements of $\mathbf{A}$ for all samples of class $j \in [\![1, c]\!]$ are given the same weight a_j

$$a_j = \frac{N}{\sum_{i=1}^{N} \mathbf{T}_{ij}},\ j \in [\![1, c]\!]. \tag{4.7}$$

The solution of ELM obtained this way is unbiased for any class. An addi-

tional multiplication by $\mathbf{A}$ is avoided by applying weights $\sqrt{a_j}$ directly to the rows of matrices $\mathbf{H}, \mathbf{T}$ which correspond to the data samples of a class j.

Alternatively, the correlation matrices can be computed for each class separately $\mathbf{\Omega}_h^1, \mathbf{\Omega}_t^1, \ \ldots, \ \mathbf{\Omega}_h^c, \mathbf{\Omega}_t^c$. Then the weights are applied during the summation of the correlation matrices $\mathbf{\Omega}_h$ and $\mathbf{\Omega}_t$:

$$\mathbf{\Omega}_h = \alpha_1 \mathbf{\Omega}_h^1 + \ldots + \alpha_c \mathbf{\Omega}_h^c, \tag{4.8}$$

$$\mathbf{\Omega}_t = \alpha_1 \mathbf{\Omega}_t^1 + \ldots + \alpha_c \mathbf{\Omega}_t^c. \tag{4.9}$$

4.2.2 Toolbox Overview

The HP-ELM toolbox implements the state-of-the-art knowledge in ELMs and high-performance programming. It is written to save the time of end users on creating yet another implementation of ELM, which is better spent on their own research or application area instead.

An ELM is a simple method which can be written in three lines in Matlab®. But performance of such ELMs is sub-optimal. ELMs achieve best accuracy with parameter selection, regularization and pruning for small datasets, and best scalability with out-of-memory accelerated processing on Big Data. The toolbox is written to provide the best performing ELM implementation to all interested researches and end users.

How to Get the Toolbox

The toolbox is a Python library, also available from Matlab®. It is written in Python programming language using efficient numerical libraries *Numpy*[3] and *Scipy*[4].

The toolbox requires Python and the following libraries: *Numpy*, *Scipy*, *Numexpr* and *pyTables*[5]. The easiest way to get Python with all required libraries is to use the Anaconda[6] Python distribution. It is a one-click install on Windows/Linux/OSX, free and provides free MKL acceleration to all university affiliates. Any other Python installation will work as well.

To install the toolbox for CPU, open the console and type `pip install hpelm`. This will download and install the toolbox with all required libraries. Anaconda provides a python console on Windows; Linux and OSX have built-in ones.

To obtain an accelerated toolbox, first download and install MAGMA[7] math library for your accelerator: Nvidia GPU with CUDA, AMD GPU with OpenCL or Xeon Phi accelerator card (called MIC architecture). All versions of MAGMA are available from the website; it also has a forum for installation support. To build MAGMA, rename one of the `make.inc.xxxx` files as `make.inc` and edit that file according to your system installation. Then install MAGMA by running `make`, `make shared` and `make install` in console from MAGMA directory.

---

[3] `http://www.numpy.org`

[4] http://www.scipy.org

[5] `http://www.pytables.org`

[6] `http://continuum.io/downloads`

[7] `http://icl.cs.utk.edu/magma/`

Second, download the toolbox archive from its repository `https://pypi.python.org/pypi/hpelm` or the latest version from Github[8], extract it and go to a sub-folder `./hpelm/acc`. There is an accelerated code which must be compiled. To get compilation flags, add your MAGMA library to `pkg-config` path, or use the same flags as MAGMA used to compile its tutorial files during an installation. To compile an accelerated ELM library, run `python setup_gpu.py build_ext --inplace` replacing `_gpu` by `_ocl` for OpenCL MAGMA or `_mic` for Xeon Phi MAGMA. You can test an acceleration by running `pyhton try_gpu.py` from the same folder. After that, go to the root directory of the toolbox and install the now-accelerated toolbox with `python setup.py install`.

Big Data Versus Small Data

Based on the number of training samples and underlying model complexity, all Machine Learning tasks can be separated into two categories: *big data* and *small data*. In the *big data*, the number of samples is enough to learn the model accurately without over-fitting, but the training time is a limiting factor. For the *small data*, there is not enough samples for learning an underlying model exactly, thus a model structure selection is necessary to find an optimal model complexity.

Training a *small data* model is computationally intensive, but the whole data is kept in the working memory for quick access. The *big data* training algorithm relies on iterative processing of small chunks of data (which normally does not fit

[8] `https://github.com/akusok/hpelm`

into memory), but with a huge amount of training samples there is no need for a model regularization (there is enough samples for a model to learn ignoring noise). Processing a *small data* which does not fit into memory is not implemented, because a typical server node has up to 256-512GB RAM, and anything large would certainly be limited by the computational speed. A *big data* for an easy problem which fits into memory is solved by either of the two first methods.

Out-of-memory Accelerated Big Data ELM

The HP-ELM toolbox for *big data* is provided by the `hpelm.HPELM` class. All data is stored in HDF5[9] format. The toolbox takes names of HDF5 files as inputs and outputs. Thus a size of processed data is limited only by a disk capacity.

The HDF5 file format provides a fast and convenient access to huge data matrices on a hard drive as if they are in memory: data can be read from or written to any place of a matrix. It also supports transparent data compression, and is native to Matlab®. A convention is used to *store only one matrix in one HDF5 file*, which allows to avoid providing the path to a matrix inside an HDF5 file. Additional utility functions `make_hdf5` and `normalize_hdf5` create HDF5 files from text/csv or matrix data, and normalize these files.

The `HPELM` class supports a GPU or Xeon Phi acceleration. The accelerated functions are provided by MAGMA library, an accelerated linear algebra library similar to LAPACK. It must be compiled by a user to get the acceleration, but it supports

[9] `http://www.hdfgroup.org/HDF5/`

any brands of GPUs and Xeon Phi accelerators. Accelerated parts are correlation matrices computation from BLUE ELM solution, and the calculation of $\boldsymbol{\beta}$. These two operations take more than 95% of runtime for ELMs with very large numbers of hidden neurons.

The ELM solution is computed iteratively by reading chunks of data from HDF5 files. Only the $\mathbf{\Omega}_h$, $\mathbf{\Omega}_t$ and $\boldsymbol{\beta}$ matrices are stored in memory. The large $\mathbf{H}$ matrix is never obtained explicitly. Targets for new inputs are also predicted iteratively and saved into an HDF5 file; and the error is computed iteratively. This makes *Big Data* ELM independent of the number of samples in the dataset, so even the largest problems can be solved on a workstation with GPU.

`HPELM` has one model structure selection function that tests different numbers of hidden neurons on a validation set. It takes pre-computed $\mathbf{\Omega}_h$, $\mathbf{\Omega}_t$ as an input, and creates solutions $\boldsymbol{\beta}_k$ for different $k \in [\![3, L]\!]$ spaced equally on a logarithmic scale. Then the validation data is projected iteratively, and errors for all values of k are computed from the same projected data. This function does the most time consuming process of projecting the data (see section 4.2.3) only once. The optimal number of hidden neurons is chosen by a minimum validation error.

Model Structure Selection for Small Data ELM

The *small data* support in the HP-ELM toolbox is provided by the `hpelm.ELM` class. It has three types of model structure selection alternatives: with a validation set, with cross-validation and with a LOO validation error computed by PRESS

statistics. All model structure selection methods find an optimal number of hidden neurons less of equal than current L. Neurons are ordered randomly, except when the L^1 regularization is used. These methods remove the extra neurons from the model and re-calculate the solution.

Both L^1 and L^2 regularization are available in ELM. The L^1 regularization is done by MRSR (section 2.2.2), a multi-output version of LARS [EHJT04]. It ranks the neurons starting from the most relevant to the problem. All model structure selection methods work better with such ranked neurons, with a trade-off of extra runtime.

The toolbox includes another method of performing L^1-regularization, based on an updated MRSR algorithm [ST06]. The original MRSR includes a part with $\mathcal{O}(2^c)$ complexity w.r.t. number of outputs c. It takes noticeable runtime with 10 outputs, and makes the method impractically slow with more than 15 outputs. The complexity of an updated version scales linearly with the number of outputs. It allows L^1 regularization for a larger set of problems, including an auto-encoder for ELMs in image processing [HBKV15].

L^2 regularization is a class parameter of ELM called `alpha` which can be changed freely. A notable benefit from L^2 regularization is making an ill-conditioned ELM solvable. One can use any single-variable optimization method to find an optimum value of `alpha` parameter.

What kind of data does HP-ELM support?

The `ELM` supports matrices (second order tensors) as inputs, and `HPELM` uses

names of HDF5 files as inputs. The utility function `make_hdf5` creates an HDF5 file from a matrix, or a text/csv file.

What about Classification?

The HP-ELM toolbox supports three kinds of classification: multi-class (one correct class for each sample), multi-label (arbitrary number of correct classes for each sample) and weighted multi-class (each class has a weight, it is independent of the number of samples in a class). ELM targets must have one feature per class (binary classification is a two-class multi-class classification), where the true class(es) for a sample are set to one and irrelevant classes are set to zero. This convention is required for correct work of the classification error and model structure selection. Classification is set with an argument while training, see section 4.2.2 below.

How to create an ELM?

An ELM is an object of `ELM` or `HPELM` class. Two mandatory parameters are numbers of input and output features. The `HPELM` also accepts a batch size for iterative processing, and a type of accelerator.

An ELM is created without any neurons. Neurons are added with `elm.add_neurons` function. It has two mandatory parameters: a number of neurons and their type, and two optional ones: projection matrix **W** and bias vector **b**. Types of neurons are the following: `lin`, `sigm`, `tanh`, `rbf_l1`, `rbf_l2`, `rbf_linf`. For RBF neurons, **W** are coordinates of RBF centers and **b** are corresponding kernel widths. Multiple different types of neurons can be added to a single ELM.

How to train an ELM?

The `train` function provides a universal wrapper for training an ELM. Two mandatory parameters are data samples $\mathbf{X}$ and targets $\mathbf{T}$, and optional arguments and keyword arguments specify the selected way of training:

- `"V"` — perform a model structure selection using validation error; requires keyword arguments `Xv` and `Tv` for validation dataset
- `"CV"` — perform a model structure selection using cross-validation error; optional keyword argument `k` for number of data splits ($k \geq 3$)
- `"LOO"` — perform a model structure selection using PRESS LOO error
- `"OP"` — perform L^1 regularization that ranks neurons starting from the most useful one; works with any model structure selection
- `"c"`, `"mc"`, `"wc"` — use classification multi-class/multi-label/weighted multi-class error instead of MSE, see explanations above; `"wc"` requires keyword argument `w` for class weights vector

How to train a large ELM in parallel?

For an ELM with a large number of neurons trained on a huge dataset, almost all the running time is spent on computing $\mathbf{\Omega}_h$. Hopefully, an operation of computing $\mathbf{\Omega}_h$ is conveniently parallel: a large dataset can be split in n parts, matrices $\mathbf{\Omega}_h^i, \mathbf{\Omega}_t^i,\ i \in [\![1, n]\!]$ computed simultaneously for all the parts of a dataset using the same ELM parameters (loading the same ELM model). The results are combined together by a simple summation $\mathbf{\Omega}_h = \sum_{i=1}^n \mathbf{\Omega}_h^i,\ \mathbf{\Omega}_t = \sum_{i=1}^n \mathbf{\Omega}_t^i$. The output weights $\boldsymbol{\beta}$ will take seconds to compute.

To perform ELM training in parallel, first split the data into multiple parts

and store them in HDF5 format required by `HPELM`. Then compute partial matrices $\boldsymbol{\Omega}_h^i, \boldsymbol{\Omega}_t^i$ using function `HPELM._project` on each data part separately. This operation takes the most runtime, and is easy to run in parallel. Save the outputs on a disk as they are computed. When all partial matrices are ready, obtain the final correlation matrices by a summation $\boldsymbol{\Omega}_h = \sum_i \boldsymbol{\Omega}_h^i$, $\boldsymbol{\Omega}_t = \sum_i \boldsymbol{\Omega}_t^i$. The output weights $\boldsymbol{\beta}$ are computed from $\boldsymbol{\Omega}_h, \boldsymbol{\Omega}_t$ by function `HPELM._solve_corr`.

To validate multiple different numbers of hidden neurons efficiently, use function `HPELM.train_hpv` with pre-computed $\boldsymbol{\Omega}_h, \boldsymbol{\Omega}_t$ and a validation data set. It outputs errors for each of the given numbers of neurons, and solves $\boldsymbol{\beta}$ for an optimal number of neurons.

How to use a trained ELM?

The `predict` function takes inputs $\mathbf{X}$ and returns corresponding calculated outputs $\mathbf{Y}$. Works only on a trained ELM. For `HPELM`, the second input gives an HDF5 file name for $\mathbf{Y}$ where the predicted outputs are written, and the function returns nothing. ELM predictions $\mathbf{Y}$ are always real numbers, predicted classes are found by taking the maximum number (multi-class) or a threshold $\mathbf{Y} > 0.5$ (multi-label).

How to get an error of an ELM?

Error of model predictions is given by `error` function of ELM, which takes true targets $\mathbf{T}$ and predicted targets $\mathbf{Y}$ as inputs. It uses the same classification settings as the ones used for training, if any. For `HPELM`, the `error` function takes file names of HDF5 files containing $\mathbf{T}$ and $\mathbf{Y}$ matrices.

Three examples of HP-ELM toolbox

Below there are three examples of running the `ELM` and `HPELM` toolboxes in Python, with the data obtained from Matlab®. The input data has 9 features and the output has one. Example ELMs using 100 sigmoid and 9 linear neurons are given. If the data is already in Python, one can skip the import from Matlab® section.

Matlab® section for Examples 1 and 2. Here four variables: `x`, `y`, `xtest` and `ytest` are saved as a comma separated values (`.cvs` files).

```
csvwrite('x.csv',x)
csvwrite('t.csv',y)
csvwrite('xtest.csv',xtest)
csvwrite('ttest.csv',ytest)
```

Example 1, Python part. Here the `.csv` files are converted to HDF5 ones in Python, and an `HPELM` is trained with those files. Training and test errors are printed.

```
import hpelm

hpelm.make_hdf5('x.csv', 'x.h5', delimiter=',')
hpelm.make_hdf5('t.csv', 't.h5', delimiter=',')
hpelm.make_hdf5('xtest.csv', 'xtest.h5', delimiter=',')
hpelm.make_hdf5('ttest.csv', 'ttest.h5', delimiter=',')

model=hpelm.HPELM(9,1)
model.add_neurons(100,'sigm')
model.add_neurons(9,'lin')

model.train('x.h5','t.h5')
model.predict('x.h5','y.h5')
print model.error('y.h5','t.h5')
```

```
model.predict('xtest.h5','ytest.h5')
print model.error('ytest.h5','ttest.h5')
```

Example 2, Python part. Here the `ELM` model is trained with different model structure selection. Previously created `.csv` files are loaded into Python and normalized to zero mean and unit variance. Then a basic ELM is trained printing the training and test error. After that a 10-fold cross-validation is used to reduce the number of neurons, showing an updated test error and selected neurons in the model. Finally the model is re-trained using an L^1 regularization (`OP` parameter), showing again re-calculated test error and model neurons.

```
import hpelm
import numpy

x=numpy.loadtxt('x.csv',delimiter=',')
t=numpy.loadtxt('t.csv',delimiter=',')
xtest=numpy.loadtxt('xtest.csv',delimiter=',')
ttest=numpy.loadtxt('ttest.csv',delimiter=',')

xx=(x-x.mean(0))/x.std(0)
tt=(t-t.mean(0))/t.std(0)
xxtest=(xtest-x.mean(0))/x.std(0)
tttest=(ttest-t.mean(0))/t.std(0)

model=hpelm.ELM(9,1)
model.add_neurons(100,'sigm')
model.add_neurons(9,'lin')

model.train(xx,tt)
tth=model.predict(xx)
print model.error(tt,tth)
```

```
yytest=model.predict(xxtest)
print model.error(yytest,tttest)

model.train(xx,tt,'CV',k=10)
yytest=model.predict(xxtest)
print model.error(yytest,tttest)
print str(model)

model.train(xx,tt,'LOO','OP')
yytest=model.predict(xxtest)
print model.error(yytest,tttest)
print str(model)
```

Matlab® section for Example 3. The training data: `x` and `y` is saved as HDF5 files using build-in Matlab® functions. Note the *transpose operation*, as Matlab® uses Fortran matrix ordering by default for HDF5 files.

```
h5create('x.h5','/data',size(x'));
h5create('t.h5','/data',size(y'));
h5write('x.h5','/data',x');
h5write('t.h5','/data',y');
```

Python part for Example 3. An `HPELM` is built and trained using the HDF5 files created by Matlab®.

```
import hpelm
model=hpelm.HPELM(9,1)
model.add_neurons(100,'sigm')
model.add_neurons(9,'lin')
model.train('x.h5','t.h5')
model.predict('x.h5','y.h5')
print model.error('y.h5','t.h5')
```

How to use Gaussian (RBF) Neurons?

The ELM toolbox has Gaussian neurons. Centroids are given instead of a projection matrix $\mathbf{W}$ and kernel widths in a bias vector $\mathbf{b}$. There are three kinds of distance functions: L^2 (Euclidean), L^1 and L^∞. They are chosen by a type of neurons: *rbf_l2*, *rbf_l1* or *rbf_linf* correspondingly. The RBF neurons are about 10 times slower to compute than sigmoid ones, even though the computation is parallelized.

My ELM solution does not exist!

An ELM may not converge if there are a few input features (2-3) with a large number of hidden neurons, if the data features are strongly correlated and not independent, or if the number of data samples is close to the number of hidden neurons. In these cases, matrix $\mathbf{\Omega}_h$ will be almost singular, and it's inverse is numerically unstable.

The numerical stability problem is solved by increasing the value of the L^2 regularization parameter α (an ELM parameter called `alpha`). The default value of $\alpha = 10^{-9}$ can be increased up to 10^{-2} or higher. This reduces the effective number of parameters in the model. The regularization parameter α should be increased if the output matrix $\boldsymbol{\beta}$ has elements with a large magnitude (larger than $10^2 \ldots 10^3$). However, it is not worthwhile increasing the parameter excessively, as this may reduce the accuracy of ELM predictions.

4.2.3 Experiments

Datasets

The HP-ELM toolbox is tested in three scenarios: regular datasets with regularization, large datasets and a Big Data problem.

Small datasets are 11 regression and 4 classification problems from the University of California at Irvine (UCI) Machine Learning Repository [Lic13]. Ten different permutations of the datasets are taken without replacements, and for each of them 2/3 of the data is used for training and 1/3 for testing. Comparison results for Support Vector Machines [CL11] (SVM), Multilayer Perceptron [Bis06] (MLP), Gaussian Processes [Ras04] (GP) are taken from the previous work [MSB+10]. Small datasets are tested on ELMs with model structure selection and without.

Large datasets are 6 relatively large datasets, available from UCI Machine Learning repository with clear prediction targets. They are *Banana* dataset of two banana species, *Adult* dataset of people with annual income below/above $50,000, *MNIST* handwritten digits dataset for classification of 10 digits based on their image representation, Record *Linkage* dataset for detecting duplicate person records with 5.5 millions samples, and *HIGGS* dataset for detecting processes which produce a Higgs boson or not, with 11 million samples (one of the largest UCI datasets available). Each dataset is split into training and test parts (respecting the guidelines where applicable), stored in HDF5 file format and normalized to zero mean and unit variance for all features. Categorical features from *Adult* dataset are encoded as binary inputs (one per each category); these are not normalized. Large datasets are tested without

model structure selection, but multiple ELMs are built with different numbers of hidden neurons.

The **Big Data** is obtained from a Face/Skin Detection dataset [PBC05]. It consists of 4000 photos of people with hand-made masks for skin and faces, under various lightning conditions, surrounding and human skin colors. Skin occupies roughly 20% of the pixels in all images. The dataset is separated into 2000 training and 2000 test images. The problem is to classify each image pixel to be a skin or a non-skin. Dataset inputs are RGB color values of 7×7 pixel mask centered on a classified pixel. The 3-pixel wide boundaries of images are omitted. There are $7 \times 7 \times 3(\text{RGB}) = 147$ features and 10^9 (one billion) data samples in total. It gives a 1.1 TB dataset in HDF5 format when stored in double precision. Two separate datasets for all training and all test samples are created from training/test images. Data features (color values of pixels) are normalized to zero mean and unit variance. A single ELM is trained with 19,000 neurons, limited by the available GPU memory. Performance is tested on differently sized subsets of these 19,000 neurons, as explained in section 4.2.2.

In the following experiments, ELM is used with automatically generated weights from $\mathbf{W} = \mathcal{N}(0, s/\sqrt{d})$. The input data is normalized to zero mean and unit variance. Biases are initialized from $\mathcal{N}(0, 1)$.

Performance on Small Datasets

The performance results and runtime for regular size datasets are presented on tables 4.2 and 4.3. The datasets are chosen similarly to those in section 2.2.4

for comparison purposes. Three ELM setups are tested using the toolbox: a basic ELM (*ELM*), an ELM with pruning of hidden neurons (*P-ELM*) using a Leave-One-Out error, and an OP-ELM (*OP-ELM*) which is an L^1 regularized *P-ELM*. They are initialized with 100 hidden neurons and sigmoid activation function. The actual number of neurons in P-ELM and OP-ELM is smaller after pruning. In three regression problems the pruning algorithm has selected $> 95\%$ of neurons, pointing to an insufficient model complexity. For these tasks (denoted by an asterisk), the number of neurons is increased to 500 where the pruning algorithm selects $< 90\%$ of neurons on average; the accuracy and runtime for 500 L are reported. Experiments are run on a single 2.6GHz core on a cluster for comparable runtimes.

The MSE and classification performance of the proposed HP-ELM toolbox is consistent with the results of other methods. The basic ELM performs worse in some cases (Auto Price), but *P-ELM* and *OP-ELM* results are comparable to the best result between the other three methods.

Considering runtime, ELM is much faster than other methods, and this speedup does not decrease the performance. A basic ELM is 6 orders of magnitude faster than SVM in *Computer* regression problem and 5 orders of magnitude faster in *Wisconsin Breast Cancer* classification problem, and it has better performance in both cases.

Performance on Large Datasets

Large datasets are classified with the toolbox on a workstation with 4-core 4GHz CPU and GTX Titan Black GPU. Additional experiments show runtime com-

Table 4.2: Mean Squared Error (bold) and runtime in seconds for the regression datasets. Results denoted by * are computed with 500 hidden neurons, as suggested by pruning. All ELM results are from the new implementation.

	Abalone	Ailerons	Elevators	Computer	Auto P.	CPU
ELM	**4.6** 0.07	**2.9e-8** 0.08	**2.1e-6** 0.11	**1.4e+1*** 0.18	**8.4e+9** 0.04	**6.8e+4** 0.01
P-ELM	**4.6** 0.12	**2.9e-8** 0.24	**2.1e-6** 0.26	**1.3e+1*** 2.5	**1.5e+7** 0.06	**9.5e+3** 0.06
OP-ELM	**4.6** 1.2	**2.9e-8** 1.2	**2.1e-6** 1.2	**1.4e+1*** 8.9	**1.5e+7** 0.80	**6.3e+3** 0.78
SVM	**4.5** 6.6e+4	**1.3e-7** 4.2e+2	**6.2e-6** 5.8e+2	**1.2e+2** 3.2e+5	**2.8e+7** 2.6e+2	**6.5e+3** 3.2e+2
GP	**4.5** 9.5e+2	**2.7e-8** 2.9e+3	**2.0e-6** 6.5e+3	**7.7** 6.3e+3	**2.0e+7** 2.9	**6.7e+3** 3.2
MLP	**4.6** 2.1e+3	**2.7e-7** 3.5e+3	**2.6e-6** 3.5e+3	**9.8** 8.2e+3	**2.2e+7** 7.3e+2	**1.4e+4** 5.8e+2

	Servo	Breast C.	Bank	Stocks	Boston	
ELM	**5.6** 0.02	**6.3e+3** 0.03	**1.1e-3*** 0.10	**1.1*** 0.02	**2.2e+1** 0.06	
P-ELM	**7.3e-1** 0.08	**1.2e+3** 0.05	**1.1e-3*** 1.1	**8.1e-1*** 0.10	**2.1e+1** 0.11	
OP-ELM	**7.9e-1** 0.78	**1.2e+3** 0.83	**1.1e-3*** 6.4	**7.8e-1*** 1.9	**2.3e+1** 0.79	
SVM	**6.9e-1** 1.3e+2	**1.2e+3** 3.2e+2	**2.7e-2** 1.6e+3	**5.1e-1** 2.3e+3	**3.4e+1** 8.5e+2	
GP	**4.8e-1** 2.2	**1.3e+3** 8.8	**8.7e-4** 1.7e+3	**4.4e-1** 4.1e+1	**1.1e+1** 8.5	
MLP	**2.2e-1** 5.2e+2	**1.5e+3** 8.0e+2	**9.1e-4** 2.7e+3	**8.8e-1** 1.2e+3	**2.2e+1** 8.2e+2	

Table 4.3: Accuracy in % (bold) and runtime in seconds for the classification datasets.

	Iris	Wisconsin B.C.	Pima I.D.	Wine
ELM	**92.6**	**96.7**	**71.8**	**90.0**
	3.4e-3	0.02	0.02	0.01
P-ELM	**95.0**	**96.6**	**73.6**	**95.8**
	6.4e-3	0.12	0.09	0.07
OP-ELM	**95.6**	**95.7**	**75.0**	**95.3**
	0.08	0.83	0.82	0.81
SVM	**95.4**	**91.6**	**72.7**	**95.8**
	2.3e+2	2.9e+3	3.3e+3	3.8e2
GP	**95.6**	**97.3**	**76.3**	**96.1**
	0.76	6.1	5.8	1.9
MLP	**94.8**	**95.6**	**75.2**	**96.0**
	7.6e+2	1.7e+3	4.1e+2	1.2e+3

parison with a cluster node having two 8-core 2.6GHz CPUs, and with a Macbook Air laptop having a 2-core 1.4GHz CPU.

Datasets is split into training and test sets, stored in HDF5 format. They are processed by HPELM toolbox class on both CPU (up to 4096 hidden neurons) and GPU (up to 19,000 hidden neurons, limited by the GPU memory). The classification is done by a basic ELM model with sigmoid hidden neurons. Multiple models are trained for different numbers of hidden neurons. Prediction performance on a test set and training time are shown on Figure 4.4.

The results show fast training times even for the largest datasets with moderate numbers of neurons. Only the largest ELM models surpass the 1 hour training time. With low number of neurons, even HIGGS datasets is processed in a few seconds on any hardware including the laptop.

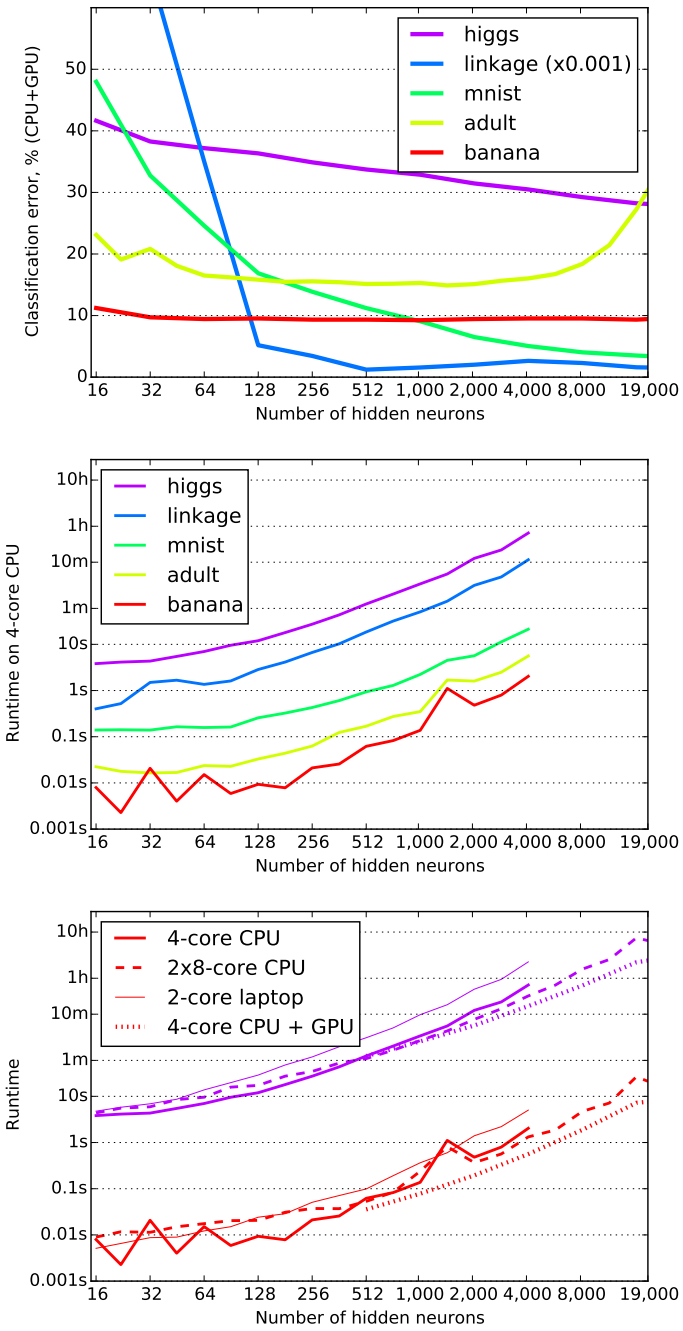


Figure 4.4: Test errors (*top*) and runtimes (*center*) on different hardware (*bottom*) for large datasets, on *logarithmic scale*. Runtime on different hardware is shown for two datasets only for image clarity. The 4-core CPU runs at 4 GHz, 2x8-core CPU run at 2.6 GHz, a 2-core laptop CPU runs an 1.4 GHz and GPU is GTX Titan Black.

High computational power devices like GPU on multi-processor nodes speedup ELM training with more than 1000 hidden neurons. This happens because operations with small matrices cannot fully utilize those devices, thus the sequential performance and disk access become limiting factors. With very high L, a speedup provided by the GPU is roughly 5 times, which is consistent with the relative theoretical CPU:GPU = 1:5 performance in double precision.

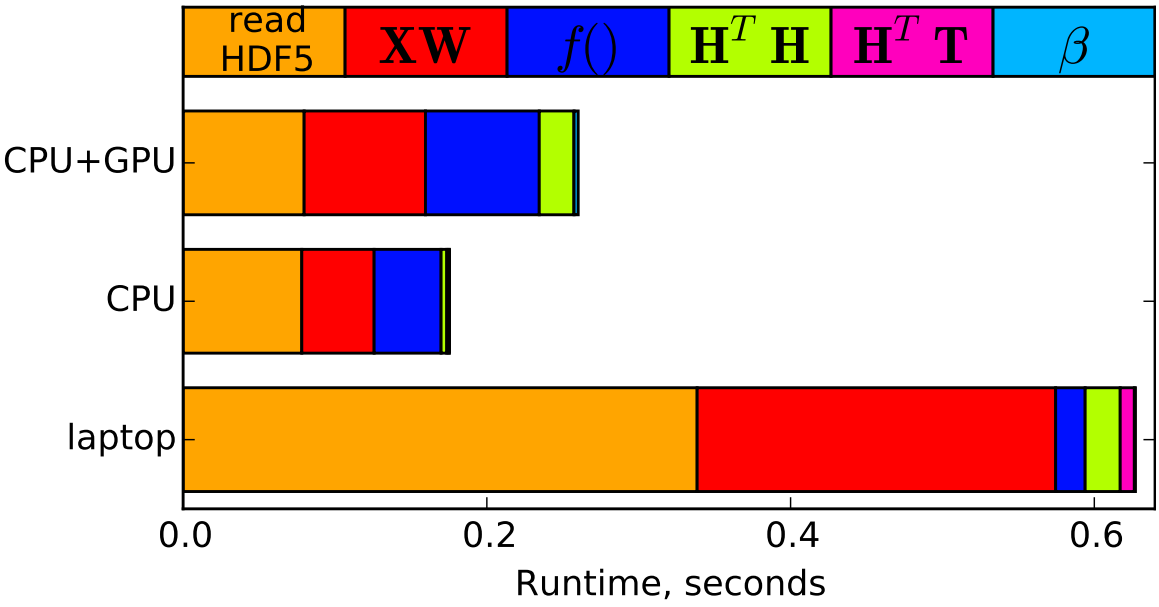
A low-power laptop performs surprisingly well in comparison with other hardware. The maximum difference in runtime (vs. a GPU at $L = 4096$) is only 10 times. For smaller numbers of neurons the runtime difference is even less. Thus a medium size ELM model can be trained fast even on a common laptop with a low-power CPU.

Runtime Analysis of HPELM with MNIST Dataset

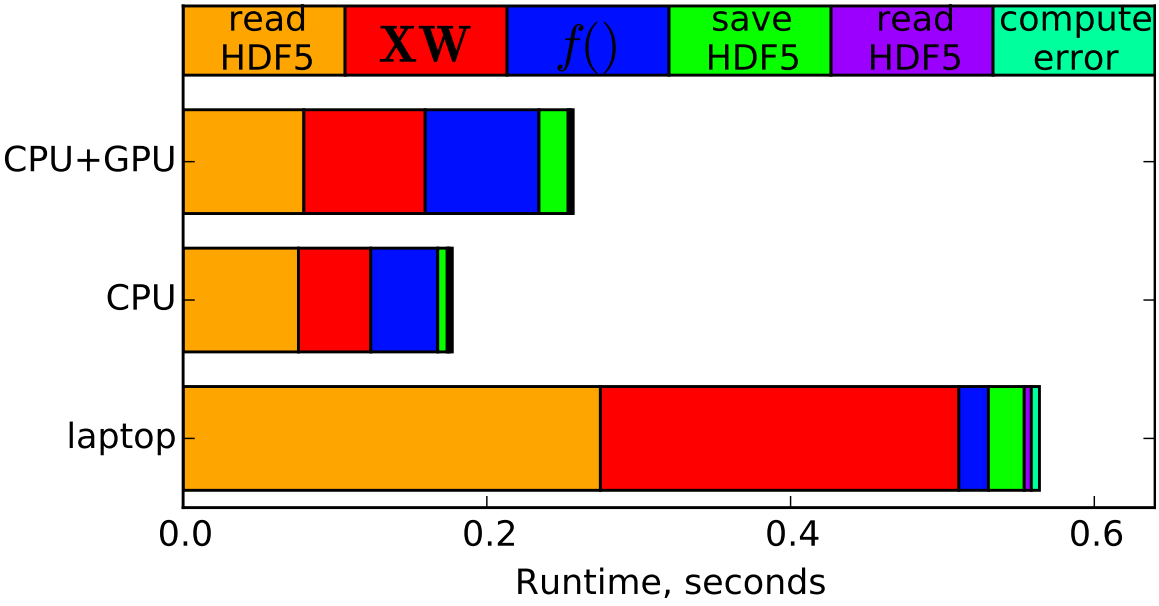
The runtime analysis of the `HPELM` implementation from the toolbox is done on MNIST classification dataset. It has 60,000 training samples with 784 features, and 10 target classes. The training and test data is stored in HDF5 file format. Experiments are performed using a basic ELM with small (64) and large (4096) numbers of sigmoid hidden neurons. First, an ELM model is trained for each number of neurons. Second, classes are predicted for the training data and a mis-classification error is computed. The training data is used for prediction to obtain a comparable runtime.

The runtime for 3 different hardware setups is shown on Figure 4.5 (64 neurons) and Figure 4.6 (4096 neurons). The runtime is obtained with a Python line profiler[10]

[10] `https://pypi.python.org/pypi/line_profiler`



(a) Training with 64 hidden neurons



(b) Prediction with 64 hidden neurons

Figure 4.5: Training (a) and prediction (b) runtimes of a basic ELM for MNIST dataset with 64 neurons. ELM predictions are obtained on the same training dataset for comparable runtimes.

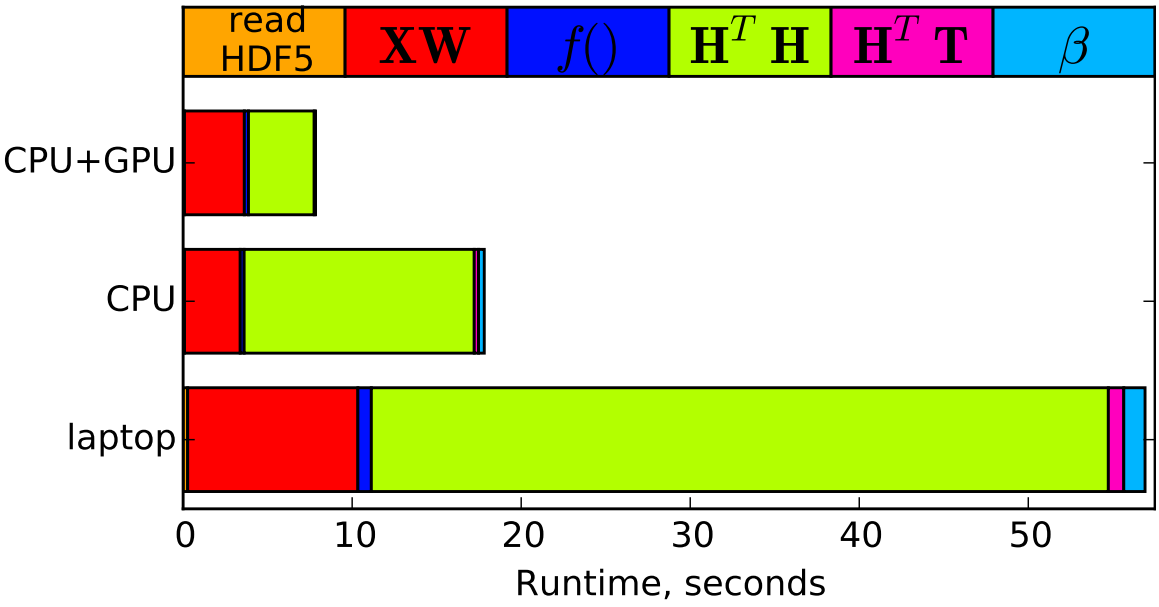
tool. Processing steps with insignificant runtime are omitted; altogether they take less than 1% of runtime.

For a small number of hidden neurons, the training takes only 0.2 seconds on 4-core CPU. The runtime is spent on loading, projecting the data and applying a nonlinear function. The largest overhead is reading data from an HDF5 file, where a laptop with a slow CPU spends half of the runtime. Applying a function has a larger overhead on 4-core CPU because it starts in parallel on all cores. Also, there is a small overhead for using a GPU to compute $\mathbf{H}^T\mathbf{H}$. The useful work ($\mathbf{XW}$, $\mathbf{H}^T\mathbf{H}$ and $f()$) takes about a half of the runtime, which is normal for such short runtimes with a universal toolbox.

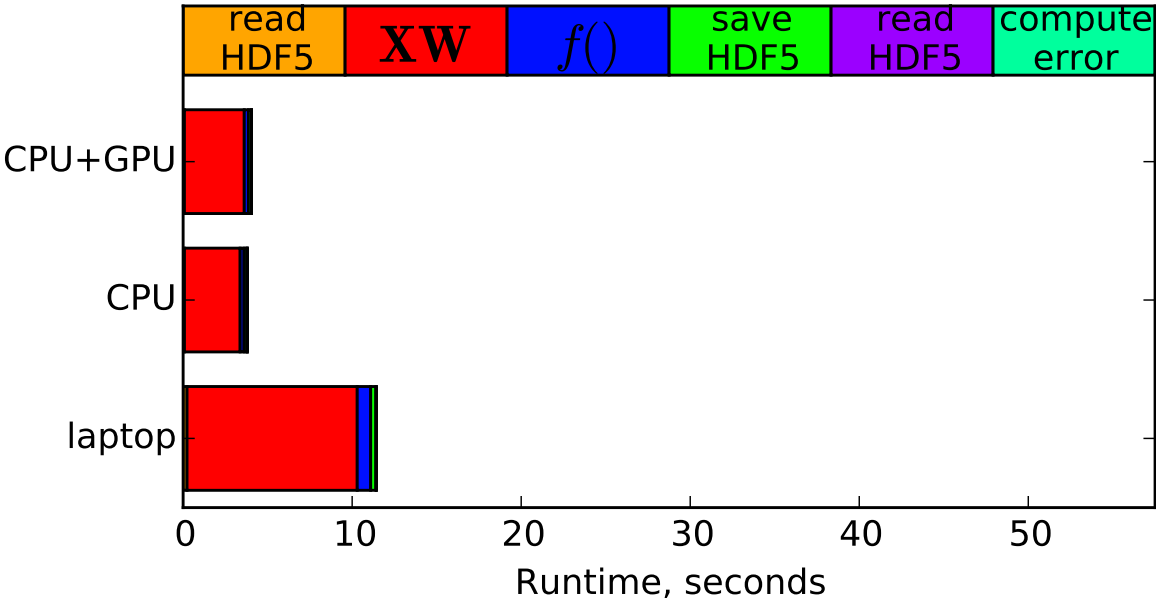
With 4096 hidden neurons, $> 98\%$ of runtime is spent on actual computations. File access time and other overheads are negligible. Computing the covariance matrix $\mathbf{H}^T\mathbf{H} = \mathbf{\Omega}_h$ takes the most of runtime during training. The time to compute $\mathbf{\Omega}_h$ is reduced significantly by GPU acceleration. The prediction runtime on all devices is completely dominated by the cost of projecting the input data into hidden layer, which is not accelerated by the GPU.

Interestingly, computing weights $\boldsymbol{\beta}$ has an insignificant runtime when done from correlation matrices $\mathbf{\Omega}_h$ and $\mathbf{\Omega}_t$. Data read and write with HDF5 files is fast, thus the HDF5 file format is used in HP-ELM. Also, an application of a nonlinear function takes little time, which is noticeable only on slow hardware and small models.

The GPU in the current HP-ELM accelerates the computation of $\mathbf{\Omega}_h$, $\mathbf{\Omega}_t$ and $\boldsymbol{\beta}$. It speeds up an ELM with large number of neurons (see Figure 4.6) because ELM



(a) Training with 4096 hidden neurons



(b) Prediction with 4096 hidden neurons

Figure 4.6: Training (a) and prediction (b) runtimes of a basic ELM for MNIST dataset with 4096 neurons. ELM predictions are obtained on the same training dataset for comparable runtimes.

computational complexity is cubic w.r.t. the number of neurons.

Overall, the runtime analysis shows high efficiency of the proposed toolbox. The effective runtime starts at 50% with a small ELM and goes over 98% for larger models. These computations are done by calling extremely well optimized BLAS matrix subroutimes, which guarantee the smallest possible runtime. BLAS subroutimes are called by Numpy Python library, which can use various implementations of BLAS including open source ones.

Also the analysis clearly shows the part which requires acceleration in large ELM mode: the computation of $\mathbf{\Omega}_h = \mathbf{H}^T\mathbf{H}$. It is combined with $\mathbf{H}^T\mathbf{T}$ and $\boldsymbol{\beta}$ in a simple GPU-accelerated part, which however greatly benefits the training time of larger ELMs (reducing it twice on Figure 4.6).

4.2.4 Big Data Processing and Performance

An example Big Data problem has 0.5 billion training samples in 147-dimensional space. It is solved by training an ELM with 19,000 sigmoid hidden neurons. A weighted classification is used to counter imbalance between the two target classes. The computations are done by splitting the data into small parts with about 1 hour of processing time each. This prevents the loss of data in case of computer failure, and allows for parallelization.

The skin detection Big Data dataset sets two additional challenges to the ELM model compared to large datasets. It requires a balanced classification as the amount of data in the two classes is uneven (17% of skin and 83% of non-skin). It also requires

testing different numbers of neurons to find out whether 19,000 hidden neurons is enough and if there is any over-fitting. The two aforementioned requirements become challenges, because the training time with 0.5 billion training data samples is so long that an ELM can be trained only once. More specifically, the matrix $\mathbf{\Omega}_h$ can be computed once as it takes more than 99% of the runtime. The model structure selection and class balancing must rely on one particular computed $\mathbf{\Omega}_h$. Same holds true for the matrix $\mathbf{\Omega}_t$, but with only two outputs it's much faster to compute.

An ELM is trained using one workstation with 4-core 4GHz CPU with GTX Titian Black GPU (similar to Tesla K40). Due to GPU acceleration, it took 135 hours in total which is less than a week. Runtimes for other hardware are estimated in Table 4.4. Without GPU acceleration, the processing time of Big Data problem becomes prohibitively large — almost 2 months using a laptop.

Table 4.4: Training time of an ELM with 19,000 hidden neurons on 0,5 billion samples with 147 features.

4-core CPU + GPU	**4-core CPU**	**laptop**	**2x8-core CPU**
5d 15h	≈ 16d 4h	≈ 51d 2h	≈ 15d 5h

The final test accuracy with 19,000 hidden neurons is 86,46%, and the confusion matrix is presented on Table 4.5.

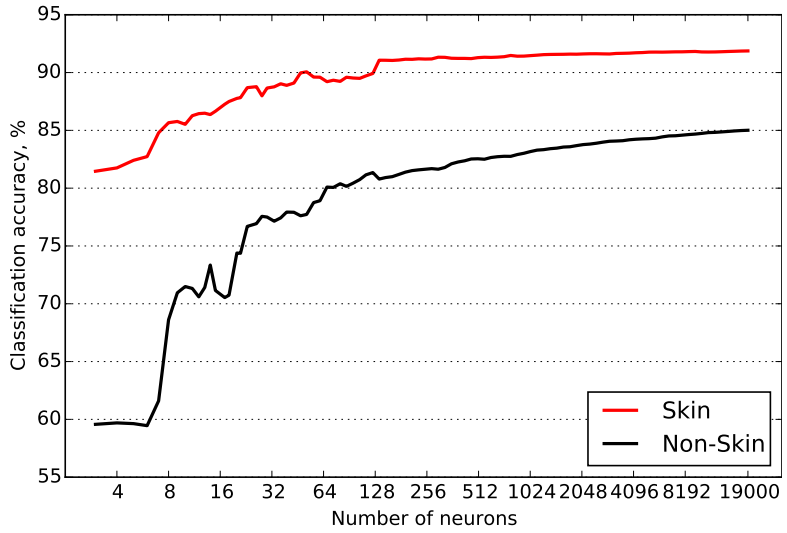
Test accuracy for different numbers of neurons is computed from a single matrix $\mathbf{\Omega}_h$, as explained in section 4.2.2. 100 different numbers of neurons are tested,

Table 4.5: Test Confusion matrix for ELM with 19,000 neurons.

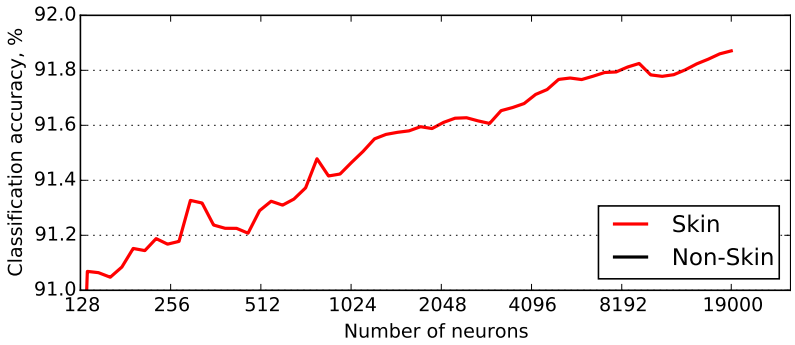
True class	Predicted class	
	Non-skin	**Skin**
Non-skin	374,840,727	66,107,142
Skin	9,626,072	108,786,059

spaced equally on a logarithmic scale from 3 to 19,000. For each number, ELM output weights $\boldsymbol{\beta}$ are solved and a separate confusion matrix is computed. The classification results for skin and non-skin from these confusion matrices are shown on Figure 4.7. Getting this test accuracy plot took 60 hours: 13 hours to obtain hidden layer output $\mathbf{H}$, and 47 hours to compute confusion matrices for all the 100 different numbers on hidden neurons.

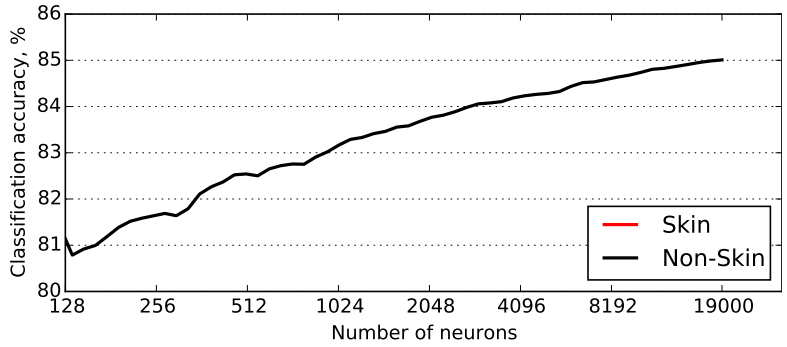
The test accuracy plot shows very good results for skin pixels classification, owing to a balanced classification method used. An ELM without class balancing would be heavily biased towards predicting non-skin pixels, which are 83% of samples in the dataset. The improvement of skin classification accuracy slows down past 128 hidden neurons, so a smaller ELM can be used for detecting skin. However, the non-skin classification accuracy grows steadily up to the maximum number of neurons. This can be explained by a higher variety of non-skin pixel masks than skin ones. ELM does not overfit even with 19,000 neurons, although the performance gain decreases at large numbers of hidden neurons.



(a) Test classification accuracy.



(b) Zoom on skin accuracy.



(c) Zoom on non-skin accuracy.

Figure 4.7: Test classification accuracy for skin and non-skin pixels. Model does not overfit with 19,000 neurons. Note the logarithmic x axis.

4.3 Image Classification with ELM

Analyzing web content is one of the most basic tasks in the Big Data environment, which emerged with the first Internet search engines. Being able to describe or classify a webpage is essential when returning relevant search results [FFPZ10], finding similar pages [JB08], or blocking unwanted or dangerous websites [LHF02] like phishing ones.

Traditional webpage analysis relies on text processing methods [JM09] (webpage text body, address, keywords and links), but with the increase in bandwidth, storage, and processing power, image data becomes omnipresent in webpages as an expressive format easily understood by humans. A modern user will probably be surprised at seeing a text-only webpage without any graphical elements.

Image data, while being an important source of information on the web, is not easily analyzed due to its extreme variability and large amount of data compared to text. Even if image understanding is too challenging today, a simpler task of multi-class image classification can be addressed instead. Existing classification methods include target-specific algorithms [ZZWW04], which cannot be generalized to arbitrary classification. Examples can be found in adult content detection methods based on the amount of skin color in the image [MS10]. Other methods are very elaborate and aim at image understanding with object extraction and recognition [CSH12, Vii12]. They are commonly present in annual competitions like PASCAL VOC [EGW+10] or benchmark datasets like Caltech101 [FFP07]. They are also impractical due to a complicated adaptation to other problems and there is a lack of ready-made toolboxes

as well as long training and running times.

This section challenges a greater goal of solving the web content filtering problem with the power of ELMs. This is an integral part of an automated Internet security framework. Internet security is not new as a domain, but the need for its automation has emerged quite recently. While computer networks are penetrating into all parts of human activity, and the potential danger of their misuse rises, a demand for capable safety measures remains high. Due to a large volume of information produced every day (both content and malicious software), human experts are unable to respond to every potential threat in time. One possibility to address this task is to pre-process data automatically, giving machine decision for easy or well-known cases, and leaving much lower number of cases for human analysis. This work addresses dangers or inconveniences that web content of some kind may give to people, by creating a tunable web content filtering system. The work is done in collaboration with F-Secure Corp.[11].

Web content filtering is done by estimating whether a particular website belongs to one of the offensive classes, and blocking it if needed. The scientific part of the method consists of a procedure to achieve a multi-label classification of websites, based on a large volume of image data. The available dataset has 20 target classes — 19 offensive ones, which reflect concepts like "Hate" or "Beer", and a benign class called "Unknown". A large dataset of URLs (web addresses) for these classes is provided by F-Secure Corp. A list of classes with the number of corresponding URLs is

[11]F-Secure `http://www.f-secure.com/en/web/home_global/home`

given in Figure 4.8.

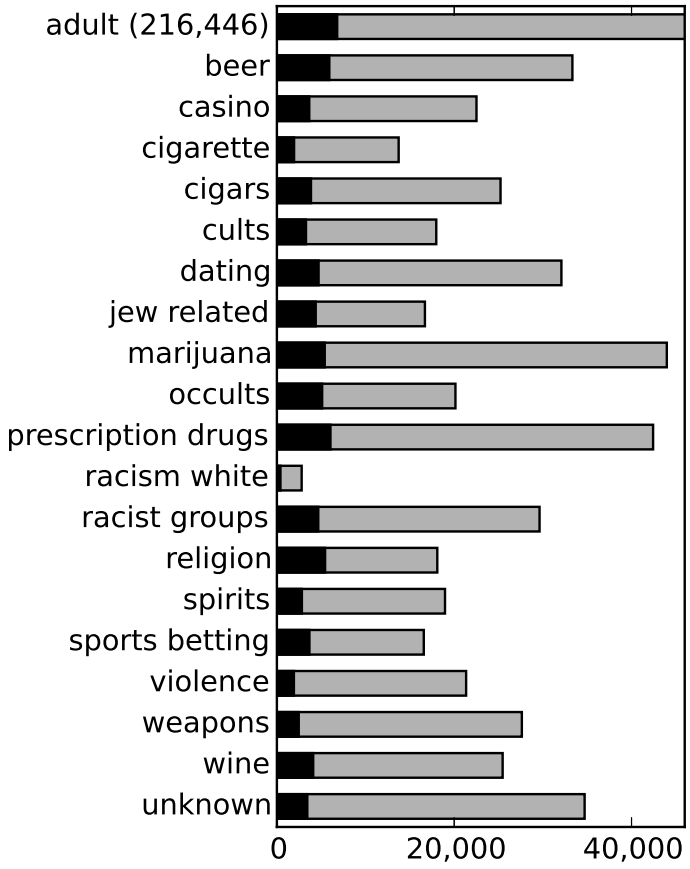


Figure 4.8: Details of the dataset of websites (black) and images (grey), provided by F-Secure Corp. "Adult" class is not shown at scale.

A typical approach for classification of webpages is text-based. This is done indeed on the same set of URLs, but in the context of the given problem domain, a text-based approach has several limitations. First, it performs poorly on websites with little or no text. Such websites are commonly present in important categories like

"Adult". Second, a text-based classifier is learned for one language. It requires adaptation to cover multiple languages, which is problematic as the number of languages is high, and for many of them there is very little training data. An image-based classifier could overcome these difficulties by being invariant to the language of websites and able to use training data from all languages altogether.

The image-based classification task belongs to the field of Big Data, with very specific challenges. The first attribute of Big Data presented in the task is volume — the dataset has 600,000 images which take 50 GB of disk space originally, and the 200,000,000 extracted image features take 200 GB in a database. Processing such volumes cannot be done in a few minutes on a desktop computer like with small sized datasets. A specific high performance database and tailored parallel processing programs can handle the data. But they are slow and risky to develop, because programming bugs and methodology errors are often found only after a few hours of program execution. The available alternatives for classification methods are also limited. In addition, the methods with the quadratic complexity with respect to a number of data samples may not be feasible in this realm.

The second attribute of Big Data in the image-based websites classification is variety, see examples on Figure 4.9. Interestingly, the image dataset has too much and too little variety at the same time. Too much variety comes from the fact that the correct class labels are known only for websites. An assumption is made that all the images in a website have the same class label. This is obviously not always true, as some images may be irrelevant to the class of a website. They are called *semantic*

noise. The proportion of semantic noise significantly varies per class: the "Adult" class has relatively low amount of irrelevant images, about 5%; and webpages of the "Cults" class mostly contain relevant text and lots of irrelevant images, like avatars of people chatting at forums, with a proportion of image semantic noise close to 70%. But even aside of the semantic noise, images from abstract classes like "Hate" are very different from each other. A useful methodology must recognize similar images in previously unseen websites to produce a decision about their class.

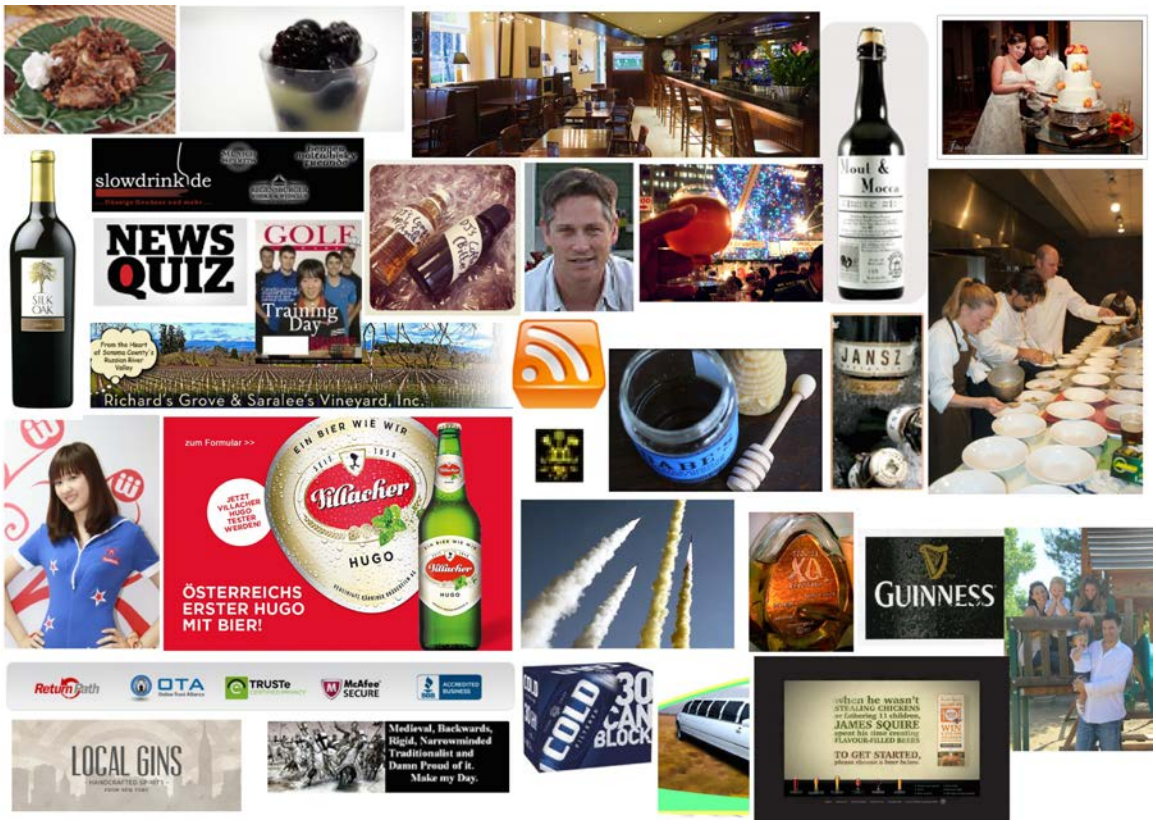


Figure 4.9: Randomly selected images related to alcohol from the dataset (provided by F-Secure Corp.).

But there is also too little variety, because images from different URLs are far from being unique and independent. Different pages of the same forum will have the same images at a header and a footer; advertisements tend to repeat in websites on the same topic. A number of webpages an individual opens per month is huge but the amount of unique websites is usually limited; for example, one can browse tens of news webpages daily from the same news web portal. Removing previously observed images from a webpage will not work, because most of webpages then will be left with one random image, or with no images at all.

This work takes the dataset of URLs as the ground truth about the Internet, and considers different URLs to be different websites. It works with webpages, but might not work with photos or benchmark datasets of unique images. This is the third attribute of Big Data, called veracity. An interesting recent overview about Big Data properties independent of the application area is found in [ZOT14].

Image understanding and machine vision are an active research frontier in Machine Learning. However very few problems are successfully solved so far. The next section discusses about modern image classification, image retrieval and object recognition methods. It describes typical approaches to image-based Machine Learning, and highlights those which form the image processing core of the proposed website classifier. Additional steps of methods for False Positives-optimized classification and merging image predictions into website labels are discussed in section 4.3.1.

Related Work

Extracting semantic information from image data is currently an active research frontier [DRS+13, OBLS14]. It includes topics like image classification, retrieval or segmentation, object detection in images, image labeling, and video processing as well. Problems connected with image processing are hard to solve, partially because most images are 2-dimensional projections of a 3-dimensional world, and successful usage of images by humans is based on an extensive use of prior and context knowledge.

The authors are aware of only two complex image-related tasks, which have a satisfactory solution so far. One is a face detection method, which is found in most modern smartphones and digital cameras. It is based on the Viola-Jones face detector [VJ01], which scans an image with a sliding window at different scales. It uses an ensemble of simple classifiers, selected and trained on a large dataset of labeled faces. That sliding window, being a slow method in general, achieves extreme speedup by using integral images [Cro84] with its simple classifiers, which allows for a real-time performance even on relatively slow devices.

The second problem with an accurate solution available is traffic signs recognition[12]. The approach is similar — an image is scanned with a sliding window of different sizes and shapes, and each window is processed separately by a neural network. Without rolling sum trick, the performance is far from real-time even with accelerated computations on GPU, but the method's accuracy exceeds 99% [MTBG13].

[12] `http://benchmark.ini.rub.de`

Both these successful cases use the prior knowledge of objects they aim to find: faces or road signs. Another interesting research classified 100,000 classes reasonably fast [DRS+13], but again they represent typical objects mostly in a canonical view. This will not hold for a general-purpose image classifier of object detector. One well-known annual competition for general image-based classification and detection methods is PASCAL VOC [EGW+10]. Other popular publicly available complex image datasets include Caltech-101 [FFP07], Caltech-256 [GHP07], MIRFLICKR-1M [HL08] and an 80 million tiny images [TFF08] dataset. The state-of-the-art methods are found among the winners of PASCAL VOC [CSH12] or recent publication on the other image datasets [BBD+12]; links can be found on corresponding websites.

The developed algorithms are large and complex (see [Vii12, BBD+12, DZ11]), but most of them utilize the same basic building blocks — local image features ("local features" or "image features"). The idea is that useful information in image is not distributed uniformly, and some parts (i.e. corners, see Figure 4.10) are more important for image understanding than others (i.e. a uniform background).

The process of obtaining local image features involves two major steps: feature detection and feature extraction. The detection phase finds potential informative regions in an image. A good overview of image feature detectors is given in [TM08]. Feature detection phase can be skipped with dense image sampling, but this method results in more local features, and not all of them are useful. Common feature detection methods are Harris-Laplace [MS04] and Harris-Affine [MTS+05]; the former method is both scale- and rotation-invariant. The latter is also affine-invariant, al-

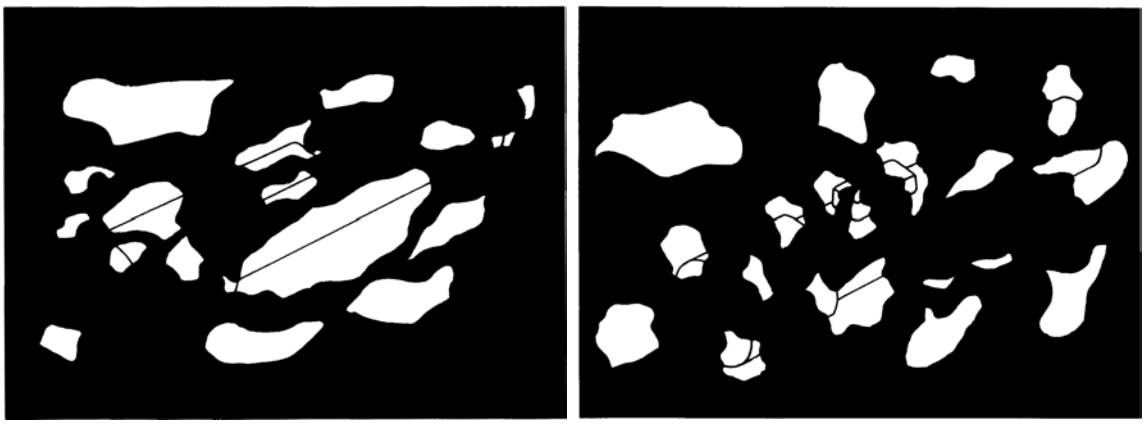


Figure 4.10: Local image features: non-informative (left) and informative (right). Informative features capture corners and junctions, while non-informative represent lines or uniform image regions. An object (flashlight) is recoverable with informative image features. This figure is from a human image understanding paper [Bie87].

though it detects less stable features across different images of the same scene.

In a feature extraction stage, pixel values of a found image patch are transformed into a special fixed-size descriptor vector. Its major property is that two such vectors calculated from two similar patches on different images (two patches of the same object part, which look similar to humans, but may have very different pixel values) lie close in the Euclidean space [Low04]. This property allows matching descriptor vectors for detecting similar objects across images, for instance pedestrians [MLS05, BMTG12]. Different methods of calculating descriptor vectors produce features of various dimensionality, but in general it should be large enough for accurate matching [BG09]. Descriptors are calculated for intensity maps, which are gray-scale images in general; color descriptors are possible by concatenating particular descrip-

tors from intensity maps of colors in different color spaces [VGS10]. Common choices of descriptors are different variants of SIFT [Low04, VGS10] and SURF [BETV08], the latter is less precise but much faster. Modern development train descriptors with Convolutional Neural Networks to obtain the state-of-the-art performance [OBLS14].

General image classification frameworks often use the approach to search over image in order to produce confidence maps [CSH12]. This works well in class-specific methods, like in a search for a naked body in the Adult images detection problem [MS10, WHY09]. Such methods often use a Bag-of-Visual-Words (BoVW) approach [BBD+12, SZ03, HKQ10], similar to Bag-of-Words in text processing [BNJ03]. The occurrence histogram produced by a BoVW method is useful for an SVM classifier with a histogram kernel [CHV99].

Table 4.6: Approaches to semantic image processing

Method	Example application	Limitations
Exhaustive search (sliding window)	Face detection [VJ01] Road sign recogn. [MTBG13]	Well-defined objects to be searched
Deep representations	80 Million tiny images [TFF08] 100,000 classes [DRS+13]	Image shows one object in canonical view
Bag-of-visual-words	Image classification [CSH12]	"Typical" objects to build bag-of-words

A summary of current semantic image processing methods is presented in table 4.6. The website classification task does not satisfy any limitations from the summary. Thus a different approach is chosen. It is inspired by works on per-

feature analysis [CXWZ11], based on k-NN feature matching [AF10]. The idea is that all parts of an image hold some class information. For all the parts of a test image, similar parts of different training images can be found [BSI08], and the test image class derived from classes of those training images. This uses an image-to-class distance [CXWZ11], which is useful for learning with abstract non-uniform classes.

The next section introduces the proposed methodology for learning abstract image classes from a large and noisy dataset. It uses color SIFT image features, and a per-feature k-NN based image classification. Predicted image classes are merged together for websites, as websites are the targets for classification in the imposed problem.

4.3.1 Image-based Classification Methodology

Overview of the Methodology

The methodology (presented on Figure 4.11) is created from an idea of meaningful local regions in images (as illustrated on Figure 4.10), some of which are relevant to the classification problem [BSI08]. It starts by extracting all local feature descriptors from training images and inheriting the class of their corresponding image. These features define a class distribution in the descriptor space. The first step creates a large set of classified local features (200,000,000 features for the proprietary dataset from F-Secure Corp.).

Next, the created dataset is used for classifying feature descriptors of new images. Classification is based on the Nearest Neighbor idea, i.e. that the class of a

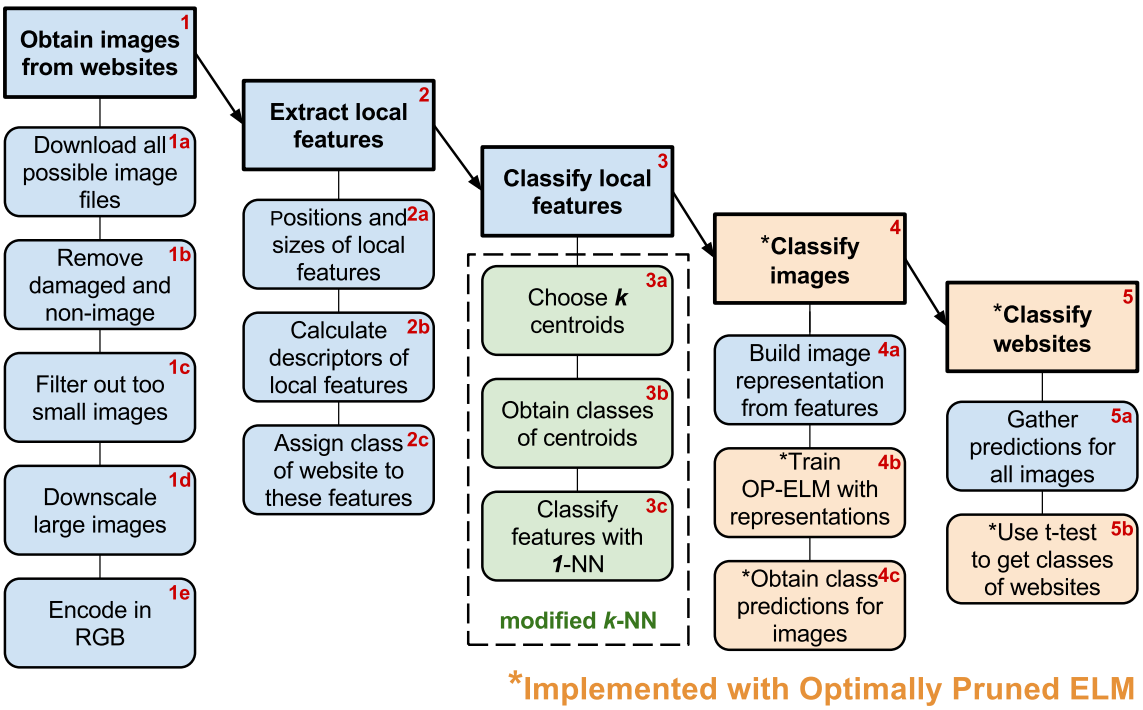


Figure 4.11: Diagram of image classification process. Five major steps are given in bold rectangles, with corresponding sub-steps depicted below. See explanation in the text.

test feature is similar to the classes of its nearest neighbors. Calculation of the exact nearest neighbor with 200,000,000 samples is infeasible. Thus a smaller representative subset of training features is used to define the class distribution in the feature space. Samples in this subset are called "centroids" for convenience, in resemblance to the centroids in k-means clustering. These centroids are classified once on a full training set (which takes thousands of core-hours in the local computer cluster). Then the several closest centroids are found for each test local feature, and the class description of a test feature is derived from the classes of these centroids. Details are explained

in subsection 4.3.1.

With the classes of test local features, test images can be classified by a general method like OP-ELM, a regularized variation of the Extreme Learning Machine [ZQSH05, HMZ+06] model. This method has only one parameter with a low sensitivity to its exact value (the *number of hidden neurons,* which should be large enough; an exact number is chosen by the Optimal Pruning procedure). OP-ELM scales linearly with the number of training samples, and even for big problems the runtime remains sufficiently short (order of hours) to validate the parameter. This makes OP-ELM well suited for large-scale Machine Learning problems [vMOL11]. A comparison with other common classification methods is provided in section 4.3.2.

The last step is the classification of websites. It is based on class predictions of images on a website. Such predictions are not binary, but are continuous numbers (one per each available class) where a higher value means higher class likelihood. An assumption is made that only a few of available classes are correct for one website. Prediction values for irrelevant classes are distributed normally, while predictions for correct classes in images of that website are generated from another normal distribution with a higher mean value. This hypothesis is tested by a paired t-test with an adjustable threshold between the mean values of the two normal distributions, and results in a multi-label website classification. An advantage of such method is an ability to reject websites with not enough confidence. The rejection is important for this particular problem, because low False Positives rate is preferred over a high coverage in the given problem setup: missing an offensive website label sometimes

is acceptable, because the method is not perfect; but blocking a benign website can force a user to remove the website filtering tool, which is an unwanted situation.

Obtaining Images from Websites

An input to the methodology is a single website URL (and a class of that website for the training set). The data collection stage (Figure 4.11, 1a-1e) transforms website image data into a uniform representation.

A website URL is searched for any files that have image extensions. The list of useful extensions is taken from an image processing toolbox for Python language (Python Image Library, PIL[13]), and includes .bmp, .dib, .dcx, .gif, .im, .jpg, .jpeg, .jpe, .pcd, .pcx, .png, .pbm, .pgm, .ppm, .psd, .tif, .tiff, .xbm and .xpm.

Then any damaged or non-image files are deleted. Image files can be detected by a file header, but this does not detect damaged images (created by errors during download, incorrect encoding, or something else). An easy way of finding valid images is to load image content from all files with a toolbox. If an error occurs at any stage, the file is deleted as an invalid image.

Downloaded image content from a website often includes decoration elements like lines or uniform background. They are unlikely to convey any class-relative meaning, and can introduce class noise. Thus they are discarded in the methodology. An empirical threshold on a minimum file size of a meaningful image is estimated by manually browsing a large collection of downloaded images sorted by file size. This

[13] `http://www.pythonware.com/products/pil/`

threshold is fixed at 2400 bytes. Most image files smaller than 2400 bytes are either decoration elements, or tiny previews of other images, and can be safely discarded.

Images of a huge resolution create another problem. One example was found in a rasterized vector image with a size of 6400×6400 pixels. It contained many sharp edges, and produced over 100,000 local features, significantly slowing down the whole method without any benefits in accuracy of predictions. Thus an upper bound in the longest edge is set for all images. The current system sets this bound to 500 pixels; the value can be adjusted for smaller runtime or better performance, but not having such value at all significantly slows down the method at some websites. Down-scaling of images which exceed upper size boundary is done with an anti-aliasing algorithm, keeping the original image proportions.

The final step of obtaining image data is to encode all images in the RGB color space. This step is performed on all images, even the ones which are already in RGB — because they often include an alpha channel, which makes them different for the feature extractor. A low compression level is chosen for JPEG algorithm to prevent an occurrence of visual artifacts.

An important remark is that although website images are pre-processed as explained before, some of them will be irrelevant to the problem. These images form *semantic noise*, which is empirically estimated to range from 5% for "easy" classes like *Adult* up to 70% for "difficult" classes like *Cults*. In "easy" classes, most information is presented by picture and video content, which is a favorable case for an image-based methodology. In "difficult" classes, however, information is transferred

mostly in textual form. For instance, a major part of images from the *Cults* category is formed by avatars of people who chat on forums. Such images don't help much in classification; but an abundance of textual information makes text-based classifiers good complimentary methods for these areas. Highly variable quality of initial image content is an unavoidable difficulty for image-based methods, that is why a rejection option for low-confidence predictions is added to final classifiers of the proposed methodology.

Extraction of Local Image Features

Local image features are "meaningful" or "informative" regions of an image [BETV08, Low99]. Think about about a picture of an airplane in the sky — a patch of a uniform blue sky is not very informative, whereas a patch containing an airplane is. Local features usually contain corners, edges or strong changes in color and contrast [TM08]. These features are specifically made to be invariant to image transformations (scaling, rotation) and noise (for instance from different image encodings). Thus they are useful for finding similar objects in different images [Low04, MS05, VS05]. These features can also be used for image classification, by finding image patches similar to those from some particular classes [BSI08].

Harris-Laplace Image Feature Detector

Edge and corner detection in an image uses derivative (or gradient) of image intensity map — pixel values of a gray-scale image. Edge detection is based on image smoothing. A difference between an original image and a smoothed image will be

small at uniform areas and large at edges.

Corners provide better positions for local image features than just edges. The word "corners" here includes corners, junctions, occlusion boundaries and strong texture — all areas with a high curvature. Corners are found by analyzing two orthogonal pixel gradients. Small values of both gradients point to a uniform area, and one large gradient points to an edge. If an edge changes its direction, the second orthogonal gradient becomes non-zero at that pixel, pointing to a corner. Formally, the two orthogonal gradients are given by eigenvalues of the second derivative matrix around a pixel. Practically, there is a fast way of estimating only the magnitudes of these gradients. The obtained number is called "cornerness", and the detected corners are located at local maximums of the "cornerness" across an image. This is the Harris corner detector [TM08].

For finding an optimal size of each feature, the cornerness is estimated with different size of image smoothing kernel (which is proportional to a size of local features). The results are stacked in a 3-dimensional tensor, and the corners are found as local maximums across the tensor. Two dimensions give the position of a feature and the third one estimates the scale (a so-called "characteristic scale") [Lin98]. A good overview of this and other feature detection methods is given in the following summary paper [TM08].

The main benefit of the Harris-Laplace detector is that a large portion of similar regions are found on images with the same scene but different resolutions and noise levels [MS05] (for instance from different encoding). The detected regions are

still composed from very different pixel values, and an encoding method is required for their comparison, which is invariant to noise and particular pixel brightness. One such commonly used method, called SIFT (Scale-Invariant Feature Transform) is described below.

SIFT Image Feature Descriptor

The Scale Invariant Feature Transform [Low99] is a local image feature descriptor, based on histograms of oriented gradients (HOG) [DT05]. As its inputs, SIFT takes position and size (scale) of a local feature. The image feature orientation is defined as the orientation of an average gradient of that image patch.

The SIFT method starts by placing a 16×16 regular square grid at an image feature, aligned with that feature's orientation. Then local gradients are calculated for each cell of that grid, based from values of pixels inside the grid cells. Magnitudes of those gradients are weighted with a Gaussian kernel to give more importance to gradients near the center of the feature. Then these weighted gradients are accumulated into 4×4 orientation histograms with 8 discrete orientations each. Values of histograms are obtained as vectors by reading them counter-clockwise. Finally, vectors for each histogram are concatenated to get the whole SIFT descriptor vector. Its length is thus 16 histograms $\times$ 8 directions $=$ 128 features. An example of SIFT descriptor calculation is shown on Figure 4.12.

The original SIFT descriptor uses only image intensities, and is suitable for gray-scale images. Color SIFT descriptors are obtained by concatenating particular

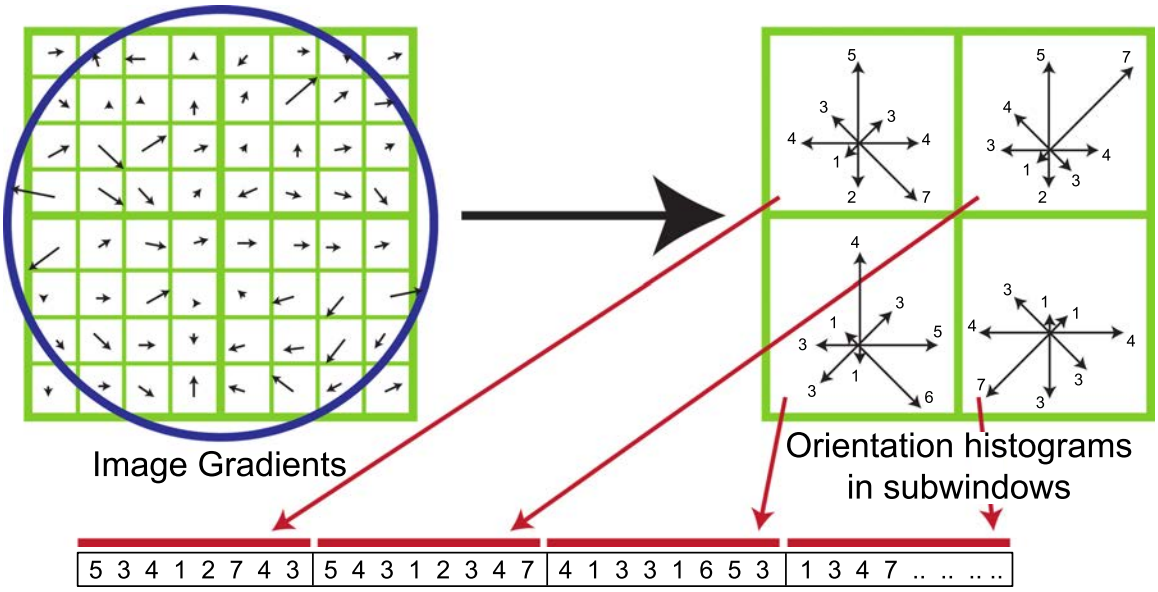


Figure 4.12: Example of SIFT descriptor calculation; reduced dimensionality is used for better visibility. First, local gradients are obtained for each cell of 8×8 grid (16×16 in full SIFT). These gradients are weighted with a Gaussian kernel, denoted by a circle. Then an oriented histogram is calculated on 2×2 grid (4×4 in full SIFT); a length of each arrow in a histogram equal to the sum of gradients in the same direction. Finally, histogram values are read counter-clockwise and concatenated into a SIFT descriptor.

SIFT vectors for each color in some color space. Recent surveys [BG09, VGS10] suggest that a weighted opponent color space (cSIFT descriptor) is a good choice. This type of image feature descriptor is used in the methodology.

The next two subsections introduce the method of utilizing cSIFT image descriptor for image classification. It is based on the k-Nearest Neighbor classification of image descriptors, so that a test image is compared to whole training classes, not

on an image-per-image basis.

Nearest Neighbor Classification of Local Features

A major assumption behind the proposed image-based classifier is that local image features convey class-related meaning. That is, there is a distribution of image classes in the local feature space (384-dimensional space for cSIFT features). This distribution can be estimated, and allows for the classification of local features of a test image.

A suitable method of estimating the class distribution is k-Nearest Neighbors (k-NN). For each local feature of a test image, the k closest features from the training set are found using a space metric (L_p norm is a common choice, with $p = 1$ or $p = 2$). An estimated class of a test feature is obtained by, for instance, a majority vote between classes of k closest training features.

Unfortunately, an exact k-NN method is infeasible with the amount of features in the training set (200,000,000 for F-Secure dataset). The feature dimensionality of 384 makes approximate nearest neighbor computations ineffective as well [WSB98]. Thus a reduced version of k-NN method is used, where all features of the training set are represented by their smaller representative subset. Samples of that subset are called *centroids*, as they are often found as centroids of a k-means clustering algorithm with a large number of clusters. These centroids with corresponding classes store the information about class distribution in the local feature space. In practice, centroids can be selected randomly from a dataset. The effect of k-means versus random

selection, and the optimal number of centroids are discussed in the Experiments section 4.3.2.

Given a smaller set of centroids, it is feasible to find the closest k centroids of each class for each local feature of a test image, with corresponding distances. Then for each class, distances to the closest centroids are pooled together for all the local features of an image. Three parameters are extracted per class: the smallest distance, the average distance and the standard deviation of all distances. It produces 20 classes $\times$ 3 parameters = 60 numbers. These 60 numbers create an image representation vector, which has the same dimensionality for images of any shape and with any number of local features. Feature vectors of images are used in general purpose OP-ELM classifiers, as explained in the next section.

Combining Image Predictions to Website

The previous step provides labels for particular local features, and image labels can be inferred from them by a simple majority vote. But such approach disregards possible interactions between classes, and the effect of an uneven number of training samples per class. Also, an increased tolerance against the False Positive predictions is desired for the specific problem, even at the cost of decreased coverage. All these constraints are satisfied by a general classifier algorithm. The algorithm of choice is an OP-ELM.

The OP-ELM classifier provides predictions for single images. For website classification, these predictions are combined together. Another goal is a multi-label

website classification, because the target labels are not mutually exclusive — for instance, the same website can belong to "Spirits" and "Cigarette" classes.

The combination uses the fact that most of the 20 classes will be negative for a given website. The image classifier can predict all negative classes (if website images are non-informative), or one or more classes can be predicted as positive. Per-image prediction of a positive class may vary, and will always have the highest confidence (ELM output) value. However, among all the website images, predictions for a positive class have a significantly higher average value than predictions for a negative class. This significance level is evaluated formally with a t-test.

An example is shown on Figure 4.13. The top left frame shows classifier outputs for a website with 5 images, and 3 possible classes. The exact output values incorporate some randomness. Thus a normal distribution can be fit to them, as on a top right frame. If one class is positive, then a normal distribution fitted to that class will have significantly higher mean than a normal distribution fitted to all other classes, as on a bottom right frame. The bottom left frame shows a fitted distribution of a negative class, which does not have a significantly higher mean value.

The t-test makes a parametric website classifier, because minimum thresholds between the global mean and particular class means must be found for the multi-label classification task. It can be implemented as another generic classifier, which uses mean and standard deviation of samples of each class as inputs. One additional useful input is the number of images in a website, which gives 41 input features in total. Using mean and standard deviation values imply the normality of data distri-

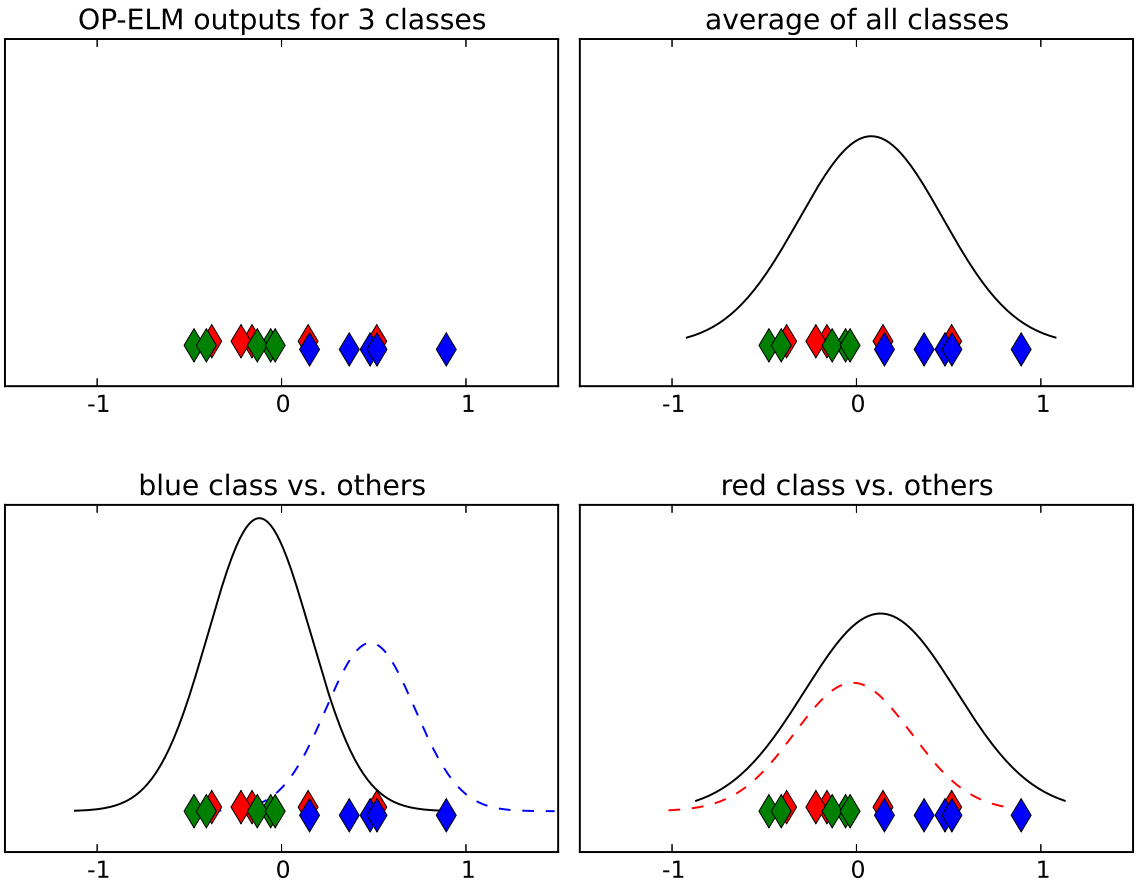


Figure 4.13: An example of a multi-label website classification using t-tests. A website has 5 images and 3 classes (red, green and blue). *Top left*: the original classifier outputs for all images and all classes (a small vertical variation is added for visibility). *Top right*: all outputs together can be approximated by a single Gaussian distribution. *Bottom left*: an example of a positive class (blue) for a website - a normal distribution fitted to the blue samples has significantly higher mean than a normal distribution fitted to all except blue samples. *Bottom right*: an example of a negative class (red) for a website.

bution, and gives a similar method to t-test; but all inner parameters are evaluated automatically from the training set. For a multi-label classification, only threshold on the final classifier outputs is to be evaluated; and for a simpler multi-class classification there are no parameters at all, because the largest output is taken as the predicted class. Such simplified multi-class classifier is used in the experiments.

4.3.2 Experiments

Selecting a Number of Centroids

An image classifier is based on a class distribution in the local feature space. Due to infeasibility of an exact k-NN, only a subset of all training local features is used; samples in this subset are called *centroids*. Centroids can be selected randomly, or taken as centroids of k-means clustering with a high number of clusters. Centroids need to have a class; this can be a class of an image they are taken from (for random centroids), or it is selected with a majority vote among k-Nearest Neighbors from training local features. A comparison between random and k-means centroids, with different centroids classification methods, is given on Figure 4.14. For the experiment, images from the Caltech-101 dataset are used because classes of Caltech are well-defined and certain.

As Figure 4.14 shows, local feature classification performance grows for both training and validation set as the number of centroids increases. It is probably limited by an exact Nearest Neighbor approach, which is computationally infeasible except for toy data. Among centroids selection method, k-means outperforms random selection

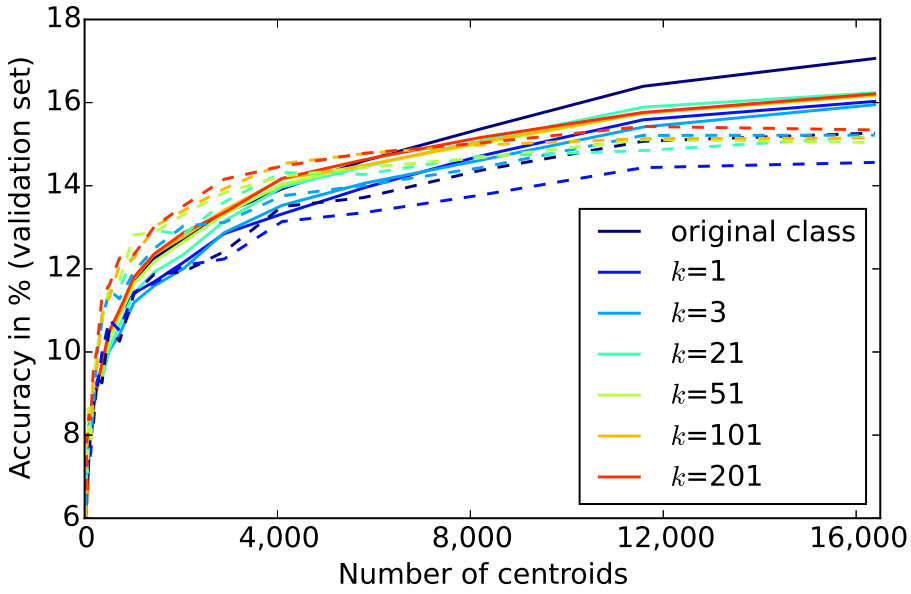


Figure 4.14: Classification accuracy of each local image feature for a validation set, using 1-NN on a set of centroids. Solid lines correspond to randomly chosen centroids, dashed lines to centroids obtained by a k-means algorithm (initialized by the same randomly chosen centroids). Dark blue label correspond to original class of local features, which are chosen to be centroids. Other colors correspond to centroids, classes of whose are obtained by the majority vote between k-NN on all local features from the training set, values of k shown on the legend. The dataset used is 20 classes of Caltech-101 with 5 images for training and validation, around 53,000 training local features in total.

on the training set, but on a validation set the accuracy with k-means grows slower for large numbers of centroids. Taking into account a significant computational cost of obtaining k-means centroids even on the given toy example with 53,000 training local features (not to mention the 200,000,000 local features of the realistic web images dataset), the random sampling centroids selection method is chosen.

The best method for centroids classification also depends on the experiment scale. At less than 4,000 centroids, re-calculating the class of centroids with a majority vote between k-NN of a training set improves accuracy. Larger values of k ($k = 201$ on the figure) provide better accuracy than smaller values ($k = 3$ or $k = 1$). This approach is often used in conjunction with Bag-of-Visual-Words and SVM classification in the literature [BBD+12]. However, the best classification accuracy is reached with a high number of randomly selected centroids, using original classes of corresponding local features. The original centroids' classes and a high number of randomly chosen centroids are therefore used in the experiments.

Classification Results

The experiments are performed on a large dataset of images and websites, provided by F-Secure Corp., with 20 target classes (19 offensive and one benign, called "Unknown"). For training and validation datasets, websites with all of their images are chosen randomly (without repetitions) and added to a training dataset until it has 10,000 images per class, and a validation dataset until it has 5,000 images per class. For classes with less than 15,000 images, 2/3 of websites are taken for

training and 1/3 for validation. The total size of a training dataset is 190,600 images and a validation is 96,000 images.

For the final website classification, only websites with at least 3 images are retained. This gives 14,000 training websites and 7,200 test websites. Ten training/-validation datasets are created with this method (which is similar to a Monte-Carlo cross-validation), and averaged results of ten experiments are reported.

Centroids are selected randomly from the available image descriptors. There are 2^{20} centroids in total, with equal amounts from each class (roughly 53,000 per class). Prediction accuracy is tested with different amounts of centroids, and a smaller set of centroids is always a subset of a larger one. The maximum number of centroids 2^{20} is limited by the computational complexity of processing the dataset (more than 10,000 core $\times$ hours for the given number of centroids), and the practical limitation on runtime for website class predictions at F-Secure Corp. (target runtime is less than one minute per website).

The final results for image and website classification are presented on Figure 4.15. Classification performance grows steadily as the number of centroids increases, possibly reaching the maximum with an exact k-NN (all training local features are centroids). The benign class is especially hard to predict, because it includes all possible images (except for the other 19 offensive classes) and has a high intro-class variability. Mis-predicting a benign website as an offensive is effectively a False Positive in the website classification methodology, because it will deny access to a good website and can make a user unhappy about the website filtering tool.

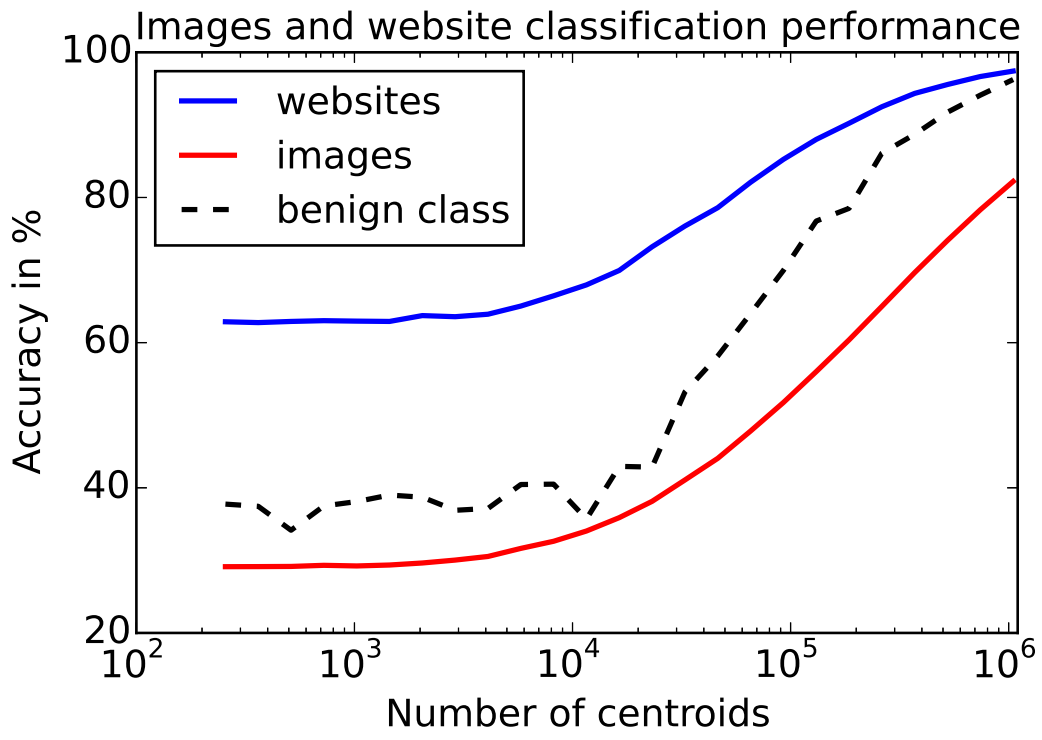


Figure 4.15: Website, image and benign website classification accuracy for the F-Secure Corp. dataset. The number of centroids is given on a logarithmic scale, from 256 to 2^{20}. Images and websites accuracy is averaged over all 20 classes; results are for validation. Random guessing accuracy is 5%.

Comparison with Other Classification Methods

An OP-ELM two-stage classifier is compared with other commonly used classification methods: SVM, linear model, ELM, RBF-ELM (a Radial Basis Function network) and a Multilayer Perceptron (MLP). Results are shown for the largest number of centroids, which corresponds to the rightmost part of Figure 4.15. All the methods are run on a whole training dataset except SVM, which is run on 1/10 of the

dataset due to computational complexity constraints (this is the only method of the aforementioned with the quadratic complexity with respect to the number of training samples). The results for image and website classification are shown on Figure 4.16.

A bad performance of SVM can be explained by a smaller amount of training data. The training time of a linear method is 5 seconds, ELM and RBF-ELM methods — 30 seconds, OP-ELM — 3 minutes, SVM and MLP — 30 minutes. Difference between all the classification methods is within 1% because most of the classification information is obtained during the k-NN stage, and the final classification stage is an easy problem where a sophisticated method cannot get a significant advantage. OP-ELM and MLP classifiers perform similarly well, but OP-ELM is ten times faster in training, thus it is used through the rest of the chapter.

Runtime

The methodology includes three main parts: feature extraction, reduced k-NN classification of descriptors, and two OP-ELMs for image and website classification.

- Feature extraction (image pre-processing, detection of local image features and calculation of SIFT descriptors) takes 0.5-5 seconds per image. This part is done in parallel on a cluster with 2,000 cores in several hours.
- Reduced k-NN classification runtime is dominated by distance calculation. An average image has 300 descriptors with 384 features, and there are 1,050,000 centroids. Computing 300×1,050,000 distances takes 155 seconds with single core, 17 seconds with optimized C code running in parallel on an i7 CPU, or 6

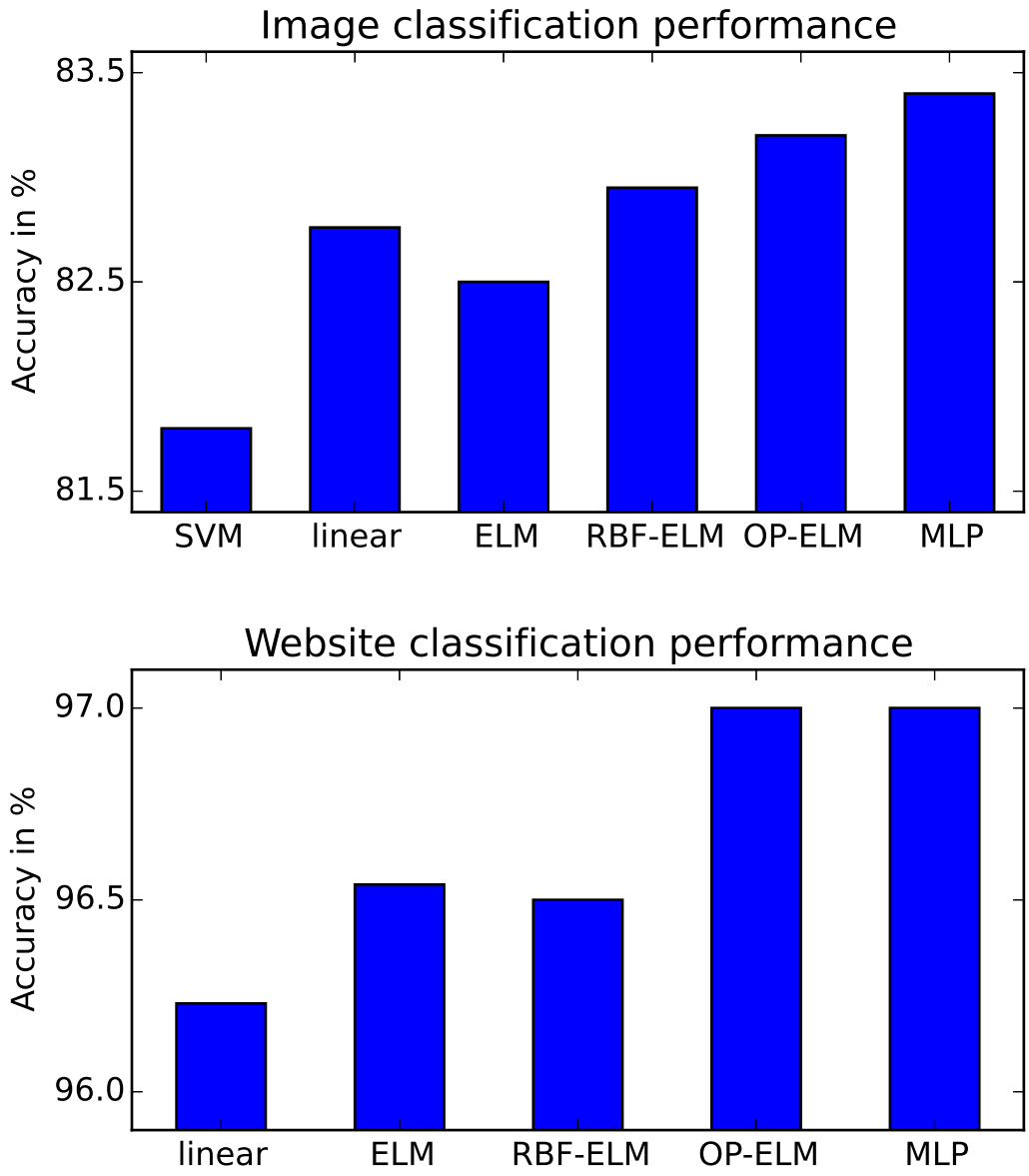


Figure 4.16: Image and website classification results using different methods combined with the k-NN local features classifier. SVM uses only 1/10 of the dataset due to training time limitations, and is not applied to website classification because of a bad performance.

seconds with an OpenCL code running on Iris Pro 5200 integrated GPU. The k closest centroids are found by a partial sorting algorithm with a 1 second runtime on an i7 CPU (see `partition()` function in `numpy` Python module). All these computations are also done in parallel with a fixed set of centroids, which took 5 days on a cluster with 2,000 cores.

- The two OP-ELM have insignificant runtime, an order of milliseconds. Training an OP-ELM for image classification with the whole image dataset takes 3 minutes on an i7 CPU, if there is enough memory.

The computations can be accelerated with GPUs. But this requires using non-free libraries for SIFT descriptors (or re-implementing a complicated algorithm), also an efficient partial sorting algorithm is non-trivial to implement on a GPU. GPU acceleration will work well for testing new websites if end-to-end runtime is important, but for the training set a large computer cluster provides higher throughput (an average number of processed images per minute) with a lower programming complexity.

4.4 Malware Detection with ELM

Classification problems typically involve putting in relation certain knowledge, in the form of a data matrix, with a discretized output, with the goal of maximizing accuracy, e.g. Despite the existence of many possible classification techniques, very few put an emphasis on the matter of decision making under constraints in non-Euclidean space (in which the data matrix lies).

Cognitive computation is a challenging research topic which focuses on solving problems related to recognition, perception, as well as learning and decision making. The interest in this paper is in using a cognitive-based technique, the Extreme Learning Machine, in the context of learning and decision making under a very specific set of constraints, in a nominal data space problem. The reason for this choice lies in the need for a human cognitive ability, that of not providing an actual decision: The very nature of the classification task studied in this paper calls for a model that is able to simply say "I do not know" in situations where a False Positive or Negative is likely to happen. This paper thus focuses on a classification problem in a non-Euclidean data space, with the constraints of zero False Positives, high coverage as well as the smallest computational time possible.

Classification problems relying solely on the distances between the different samples are common in genetics [Lel93], or syntactic and document resemblance problems [BGMZ97, Bro97]. The reason for the direct use of the distance matrix in these setups is that the original data does not lie in a Euclidean space, but is usually nominal data, i.e. without any sense of ordering between two different values. As such,

distance matrices need to be calculated using non-Euclidean metrics, usually.

While the high coverage constraint is rather typical (achieving the highest True Positive and True Negative rates possible), the zero False Positive constraint is not. In addition, the False Negatives are not regarded as very important in this problem setup: even if lowering False Negatives means increasing the coverage, the most highly regarded requirement is on the False Positives. In this regard, the ability of the cognitive model making the final decision to state that the uncertainty in said decision is too great is paramount.

As mentioned, the fact that the data is nominal makes it mandatory to use methods which directly deal with the distance matrix. A means of computing this distance matrix is first described, by the use of an approximation technique based on Min-Wise independent hash function families.

The following section 4.4.1 describes a very specific application of this proposed methodology to Malware detection for computer security. This application is exactly framed by the previously mentioned constraints. In addition, this application provides experimental data on which the proposed methodology is tested in section 4.4.3. Section 4.4.2 describes first the matter of calculating distances between samples and then how the use of the Jaccard distance remains possible with the low-computational time imperative, by estimating it using Locality Sensitive Hashing. A 1-Nearest Neighbor classifier is then proposed as a first step and its shortcomings listed, while section 4.4.3 details the complete two-step methodology which addresses these issues using a combination of the distance based K-Nearest Neighbours and cognitive-based

Extreme Learning Machine, along with the experimental results.

4.4.1 A Specific Application

The goal of Anomaly Detection in the context of computer intrusion detection [RRZ+09] is to identify abnormal behavior — defined as deviating from what is considered "normal" behavior — and signal the anomaly in order to take appropriate measures: identification of the anomaly source, shutdown/closing of sensitive information or software...

Most current anomaly detection systems rely on sets of heuristics or rules to identify this abnormality. Such rules and heuristics enable some flexibility on the detection of new anomalies, but still require action from the expert to tune the rules according to the new situation and the potential new anomalies identified. One ideal goal is then to have a global system capable of "learning" what constitutes normal and abnormal behavior and therefore be able to identify reliably new anomalies [SG10, BOA+07]. In such a context, the only human interaction required is the monitoring of the system, to ensure that the learning phase happened properly.

A small part of the whole anomaly detection problem is studied in this paper, in the form of a binary classification problem for malware and clean samples. While the output of this problem is quite typical, the input is not. In order to compare files together and compute a similarity between them, a set of features is needed. F-Secure Corporation devised such a set of features [F-S06], based partly on sandbox execution (virtual environment for a sample execution [WHF07, YHOM10]). This sandbox is

capable of providing a wide variety of behavioral information (events), which as a whole can be divided into two main categories: hardware-specific or OS-specific. The hardware-specific information is related to the low-level, mostly CPU-specific, events occurring during execution of the application being analyzed in the virtual environment (up to the CPU instruction flow tracing). The other category mostly relates to the events caused by interaction of the application with the virtual OS (the sandbox). This category includes information such as General Thread/Process events (e.g. Start/Stop/Switching), API call events, specific events like Structure Exception Handling, system module loading etc. Besides, the sandbox can provide (upon user request) some other information about application execution, like reaching pre-set breakpoints, detecting behavioral patterns, which are not typical for traditional well-written benign applications (e.g., so-called anti-emulation and anti-debugging tricks), etc.

The sandbox features used in the following research are thus the dynamic component of the collected features. Dynamic features in this context refer to those gathered from the Sandbox while an inspected application was executed in it. Some examples of those are what API calls were called and with what parameters, various types of memory and code fingerprints. Static features refer to some of the features gathered from the executable binary itself without actually executing it. Some examples of those are what packer it was compressed with and various code and data fingerprints. There are 15 features from the static domain and as many from the dynamic domain, containing up to tens of thousands of values each. Each of these

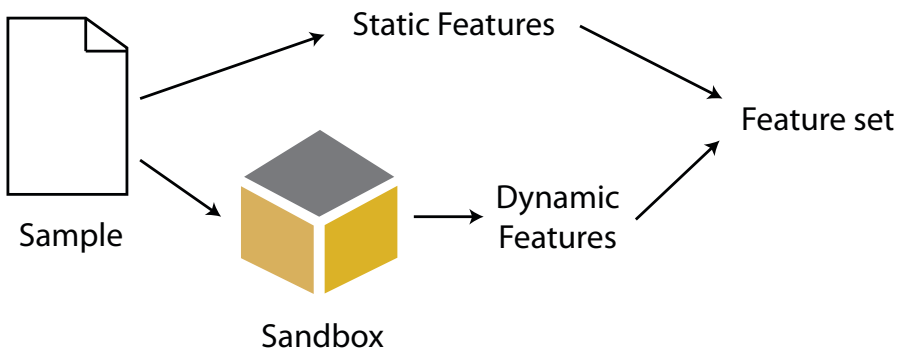


Figure 4.17: Feature extraction from a file (sample): The sandbox runs the sample in a virtual environment and extracts dynamic (run-time specific) information; meanwhile a set of static features are extracted and both sets are combined in the whole feature set.

features can be present or absent for one sample (e.g. if the sample studied does not perform some classical operations in the sandbox, some features do not get activated). As such, the input data obtained per sample usually consists of tens of thousands of values for each feature number. The feature values are represented by CRC64 hashes.

One of the major challenges is related to this data size: Each sample having some tens of thousands (on average) of feature-value pairs (at most 30 features per sample, with thousands of values per feature for one sample), sample to sample comparisons are non-trivial computationally speaking. Also, due to the nature of the data, measuring similarities between files requires specific metrics that can be applied to nominal data (i.e. with no sense of order between values, as opposed to ordinal data). Indeed, since the actual feature values are encoded as hashes (and represent

function strings and series of arguments, parameters...), classical measures used in Euclidean spaces do not apply. The Jaccard similarity enables such comparisons and is detailed in section 4.4.2, with the computational challenges it poses.

In addition to this specificity of the data, the requirements on the performance of the classifier are particular as well. As a security company, F-Secure Corporation needs to have very low false positives on any anomaly detection system deployed: If a clean file is labeled as a malware (i.e. is a false positive), it is likely that several clients will see this same error deployed on their machines as well. This single mistake will potentially hinder seriously the work on all the affected machines, making the clients unhappy about the product and thus deactivating it or switching to a concurrent one. Therefore, while typical binary classification problems addressed by Machine Learning focus on optimizing the accuracy, one of the goals of the methodology presented in this paper is to lower the false positives to achieve 0, at the obvious cost of lowered coverage. The ability of the cognitive model ELM making the final decision to give the "I do not know" decision is what enables the whole methodology to have control over said false positives and coverage. To clarify notations, table 4.7 summarizes the confusion matrix used in this paper.

Table 4.7: Confusion Matrix for this binary classification problem.

		Actual	
		Malware	Clean
Prediction	Malware	True Positive (TP)	False Positive (FP)
	Clean	False Negative (FN)	True Negative (TN)

Some additional practical constraint also makes this problem particular. Since the goal is the identification and classification of new malware samples, there is an imperative on the time it takes to have a decision per sample: The fastest an answer is provided, the quicker will be the deployment of the information concerning a new sample, possibly preventing infection at many other sites. As such, computational times need to be reduced as much as possible.

4.4.2 Problem Description

This section first describes the problem in terms of the nature of the data at hand, and a way to calculate distances between files, using this very data. The matter of the computational requirements for such calculations are addressed by an approximation based on Min-Wise independent families of hash functions. The parameters of this approximation are then determined and its effects investigated.

Data Specifics

Distances in a traditional Euclidean sense are usually calculated for points which have a set of coordinates to locate them in the space. Having a data set consisting of multiple hashes with different hashes representing incomparable properties or attributes, makes that data effectively categorical, and does not allow to calculate distances in a classical manner. The specifics and origin of the data set used in this paper are confidential as the data is provided by F-Secure Corporation. Original values present in the data have been hashed using the CRC64 hash function, so as to obfuscate the original details.

The data set is composed of a large amount of files (samples), each having the following structure:

- 30 possible feature numbers (each representing a different class of information recorded about the sample)
- For each of these feature numbers, a variable amount of hashes (from 0 to tens of thousands).

The reason for this structure is that some feature numbers are standing for a wide range of possible informations: if one such feature number stands for "the names of all the functions called in this sample", e.g., the number of values associated to it is bound to be large for some samples. It is important to note that the number of feature values per feature number can be very different from file to file.

With this data structure, it is impossible to use traditional Machine Learning techniques, as most of them rely on the data points position in the sample space (usually expected to be Euclidean). In this paper, distances between samples are calculated by using the Jaccard index [Jac01, TSK05], as presented in the next subsection.

Distance Calculation for Nominal Data

One of the most classical similarity statistics for nominal data is the Jaccard index [Jac01]. It enables the computation of the similarity between two sets of nominal attributes as the ratio between the cardinalities of their intersection and of their union. Denoting A and B as two sets of nominal attributes, the Jaccard index is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \tag{4.10}$$

This index intuitively gives a good sense of overlap (similarity) between the two sets; the more common attributes (hashes in this case) they have, the more statical and dynamical properties the corresponding files — each associated with one set — share, thus the higher the chance that they are of the same class. In addition, considering the Jaccard distance $J_\delta(A,B) = 1 - J(A,B)$ yields an actual metric, which enables to use Machine Learning techniques directly.

In the case of this paper, the files not only have one set of attributes, but multiple, identified by their feature number. As such let us redefine $A = \{A_i\}_{i \in \mathcal{A}}$, where A_i is the set of hashes associated to feature number i, and $\mathcal{A}$ is the set of all feature numbers available for file A. Therefore, the Jaccard index needs to take into account all such feature numbers. A straightforward modification of the Jaccard index for this case is to define it as

$$J(A,B) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \frac{|A_i \cap B_i|}{|A_i| + |B_i| - |A_i \cap B_i|} \tag{4.11}$$

where A_i and B_i are the sets of feature values for feature number i for file A and B respectively, and $\mathcal{C} = \mathcal{A} \bigcap \mathcal{B}$, with $\mathcal{A}$ (resp. $\mathcal{B}$) the set of the feature numbers for file A (resp. B).

This way, only feature numbers present in both files are accounted for. In addition, expressing the index like this enables to avoid computing the cardinality of the union, which saves some computational time, as the cardinality of the sets A_i and

B_i are known.

The computational time required for the multiple calculations of the Jaccard distance remains a problem, due to the intersection cardinality calculation. This problem is addressed in the following subsection by approximating the Jaccard distance.

Speeding Up the Distance Calculations

The main drawback of the original Jaccard distance lies in the computational time required for its calculation. While the intersection of two sets (the upper part of the fraction in Eq. (4.11)) is relatively fast — for example, the Python language implementation of it has an average complexity of $O(\min\{|A_i|, |B_i|\})$ and a worst case of $O(|A_i| \times |B_i|)$[14]—, the intersection of such large sets repeated multiple times makes the total computational time intractable. As mentioned before, the sets A_i for one single feature number i can total some tens of thousands of elements.

As such, the direct Jaccard distance calculations using Eq. (4.11) cannot be used. The specific requirement for this problem of "near real-time" computations raises the need for an fast approximation of the Jaccard distance.

Resemblance as an Alternative to Jaccard Index

Consider a file named A, and denote by $|A|$ the number of hashes in this file (to avoid heavy notations, it is considered that only one feature number is present in the files; the following extends directly to the practical case of multiple feature

[14] http://wiki.python.org/moin/TimeComplexity#set

numbers per file). Let us define by $S(A, l)$ the set of all contiguous subsequences of length l of hashes of A. Using these notations, one can define [Bro97] the *resemblance* $r_l(A, B)$ of two files A and B based on their hashes as

$$r_l(A, B) = \frac{|S(A, l) \cap S(B, l)|}{|S(A, l) \cup S(B, l)|}, \tag{4.12}$$

which is similar to the original definition of the Jaccard index. Defining the *resemblance distance* as

$$d_l(A, B) = 1 - r_l(A, B) \tag{4.13}$$

yields an actual metric [BGMZ97, Bro97].

Let us fix the size of the contiguous subsequences of hashes l and denote by Ω_l the set of all such subsequences of length l. Let us assume that Ω_l is totally ordered and set a number of elements n. For any subset $\omega_l \subseteq \Omega_l$ denote by $\mathrm{MIN}_n(\omega_l)$ the set of the smallest n elements (using the order on Ω_l) of ω_l defined as

$$\mathrm{MIN}_n(\omega_l) = \begin{cases} \text{the set of the smallest } n \text{ elements from } \omega_l, & \text{if } |\omega_l| \geq \mathrm{n} \\ \omega_l, & \text{otherwise.} \end{cases} \tag{4.14}$$

From [Bro97], the following theorem gives an unbiased estimate of the resemblance $r_l(A, B)$.

Theorem 4.1. *Let $\pi : \Omega_l \to \Omega_l$ a permutation on Ω_l chosen uniformly at random and let $M(A) = MIN_n(\pi(S(A, l)))$. Defining $M(B)$ similarly, the following is an unbiased estimate of $r_l(A, B)$:*

$$\hat{r}_l(A, B) = \frac{|MIN_n(M(A) \cup M(B)) \cap M(A) \cap M(B)|}{|MIN_n(M(A) \cup M(B))|}.$$

The proof can be found in [Bro97].

As such, once a random permutation is chosen, it is possible to only use the set $M(A)$ (instead of the whole of A) for resemblance-based calculations.

Weak Universal Hashing and Min-Wise Independent Families

Note that while CRC64 cannot be considered as a random hash function, the notion of *weak universality* for a family of hash functions proposed in [CW79] makes it possible to further extend the former approximation to families of hash functions satisfying

$$\Pr\left(h\left(s_1\right) = h\left(s_2\right)\right) \leq \frac{1}{M}, \tag{4.15}$$

with h a hash function chosen uniformly at random from the family $\mathcal{H}$ of functions $\mathcal{U} \rightarrow \mathcal{M}$, s_1 and s_2 elements from the origin space $\mathcal{U}$ of the hash function in $\mathcal{H}$ and $M = |\mathcal{M}|$. More precisely, in [BCFM00], the definition of *min-wise independent* family of functions is proposed in the spirit of the weak universality concept, and the authors show that for such families of functions, the resemblance can be computed directly.

Define as *min-wise independent* a family $\mathcal{H}$ of functions such that for any set $X \subseteq [\![1, N]\!]$ and any $x \in X$, when the function h is chosen at random in $\mathcal{H}$, we have

$$\Pr\left(\min\left\{h(X)\right\} = h(x)\right) = \frac{1}{|X|}. \tag{4.16}$$

That is, all elements of the set X must have the same probability to become the minimum element of the image of X under the function h. Assuming such a

min-wise independent family $\mathcal{H}$, then

$$\Pr\left(\min\left\{h(S(A,l))\right\} = \min\left\{h(S(B,l))\right\}\right) = r_l(A,B), \tag{4.17}$$

for files A and B and a function h chosen uniformly at random from $\mathcal{H}$; it is therefore possible to compute the resemblance $r_l(A,B)$ of files A and B by computing the cardinality of the intersection

$$\begin{matrix}\left\{\min\left(h_1\left(S\left(A,l\right)\right)\right),\ldots,\min\left(h_k\left(S\left(A,l\right)\right)\right)\right\} \\ \left\{\min\left(h_1\left(S\left(B,l\right)\right)\right),\ldots,\min\left(h_k\left(S\left(B,l\right)\right)\right)\right\}\end{matrix} \bigcap, \tag{4.18}$$

where $h_1,\ldots,h_k$ are a set of k independent random functions from $\mathcal{H}$. This way of calculating the resemblance of two files is sometimes called *min-hash*, and this name is used in the rest of this paper to denote this approach.

For computational and practical reasons, in this paper only one hash function is used (CRC64) and the cardinality of the intersection of equation (4.18) is approximated as the cardinality of

$$\left\{\min_k\left(h\left(S\left(A,l\right)\right)\right)\right\} \bigcap \left\{\min_k\left(h\left(S\left(B,l\right)\right)\right)\right\}, \tag{4.19}$$

where the notation $\min_k(X)$ denotes the set of the k smallest elements in X (assuming X is fully ordered). While this is a crude approximation, experiments show that the convergence with respect to k towards the true value of the resemblance is assured, as shown in the following subsection.

Influence of the Number of Hashes on the Proposed Min-hash Approximation

Figure 4.18 illustrates experimentally the validity of the proposed approximation of the Jaccard distance by the min-hash based resemblance. These plots use a small subset of 3000 samples from the whole dataset, used only for this purpose of validating the amount of hashes k required for a proper approximation.

As can be seen, with low amounts of hashes, such as $k = 10$ or 100 (subfigures in the top row), quantization effects appear on the estimation of the resemblance, and the estimation errors are large. These quantization problems are especially important in regard to the method using these distances — K-Nearest Neighbors —, as presented in the next section: Since distances are so much quantized, samples being at different distances appear to be at the same, and can thus be taken as nearest neighbors wrongly.

The quantization effects are lessened when k reaches the hundreds of hashes, as in subfigures on a middle row, while the errors on the estimation remain large. $k = 2000$ hashes reduces such errors to only the largest distances, which are of less importance for the following methodology. While $k = 10000$ hashes reduces these errors further (and even more so for larger values of k), the main reason for using the min-hash approximation described is to reduce drastically the computational time.

Figure 4.19 is a plot of the average time required per sample for the determination of the distances to the whole reference set, with respect to the number of hashes k used for the min-hash. Thanks to the use of the Apache Cassandra backend

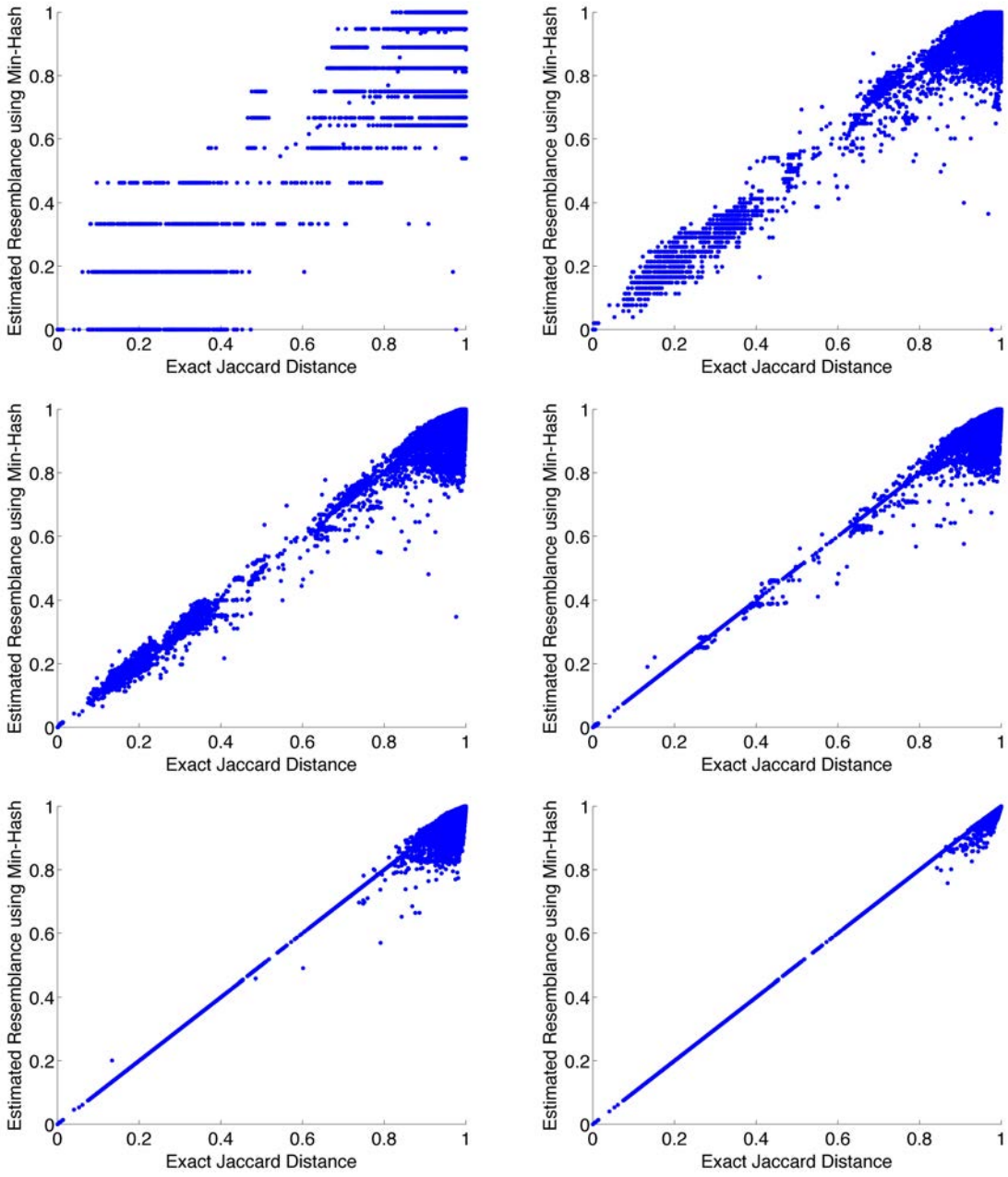


Figure 4.18: Influence of the number of hashes $k \in \{10, 100, 500, 1000, 2000, 10000\}$ (*top left* to *bottom right*) over the min-hash approximation of the resemblance r. The exact Jaccard distance is calculated using the whole amount of the available hashes for each sample.

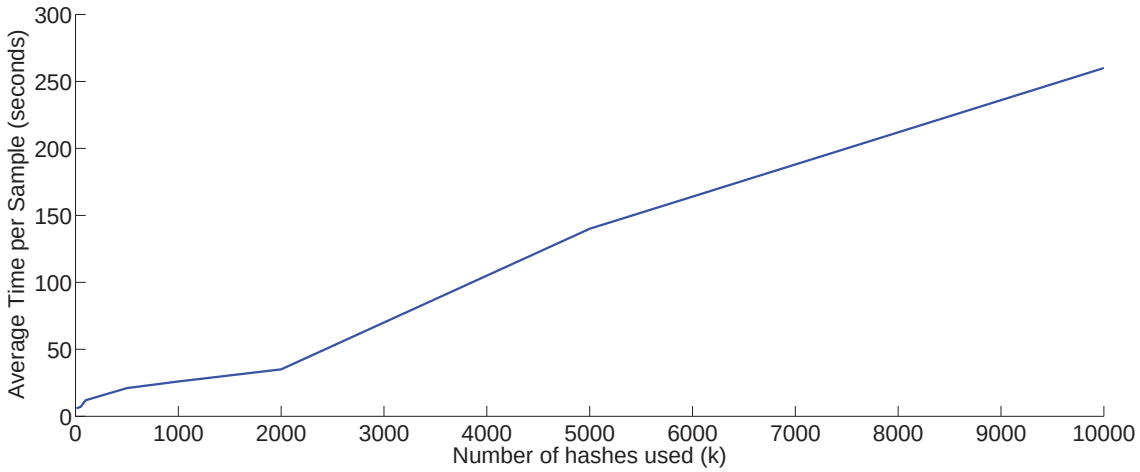


Figure 4.19: Average time per sample (over 3000 samples) versus the number k of hashes used for the min-hash approximation.

(with three nodes) for these calculations, the computational time only grows linearly with the number of hashes (and also linearly with the number of samples in the reference set, although this is not depicted here). Unfortunately, large values of k do not decrease the computational time sufficiently for the practical application of this methodology. Therefore, in the following, $k = 2000$ hashes is used for the min-hash approximation of the Jaccard distance, as a good compromise between computational time and approximation error. Note that the choice of this value for k is based on the empirical analysis above and is therefore directly tied to the nature of the data used in this paper.

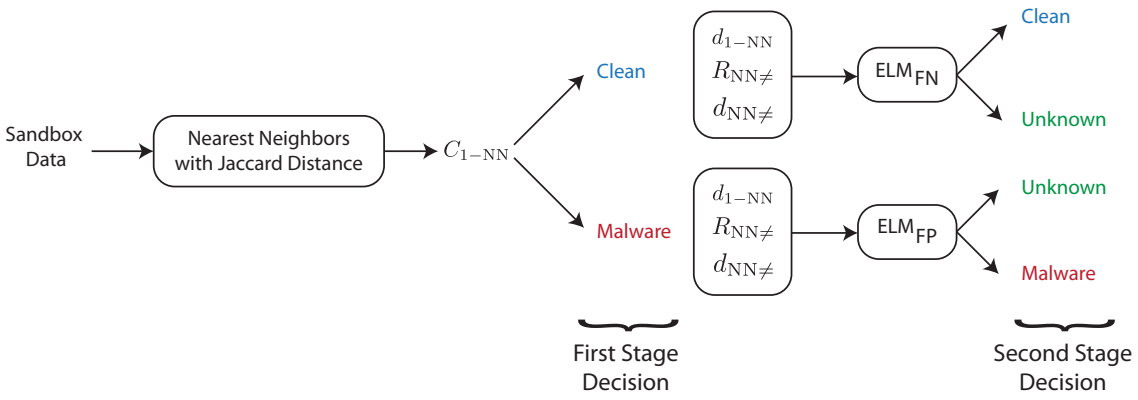


Figure 4.20: 1-NN-ELM: Two stage methodology using first a 1-NN and then specialized ELM models to lower false positives and false negatives. The first stage uses only the class information $C_{1-\mathrm{NN}}$ of the nearest neighbor, while the second stage uses additional neighbors information: the distance $d_{1-\mathrm{NN}}$ to the nearest neighbor, the distance $d_{\mathrm{NN}\neq}$ to the "nearest neighbor of the opposite class" and the rank $R_{\mathrm{NN}\neq}$ (i.e. which neighbor is it) of this opposite class neighbor.

4.4.3 Methodology Using Two Stage Classifiers

This section details the use of a two-stage decision strategy so as to avoid False Positives while retaining high coverage. The first stage decision uses a 1-NN, which still yields too high False Positive rates; this rate is lowered by using an optimized Extreme Learning Machine model, specialized either for False Positives or False Negatives minimization; this model has the very specific cognitive ability to also decide that it would rather not take a decision in some cases than create such False Positive or Negative.

First Stage Decision using 1-NN

The K-Nearest Neighbor [CH67] method for classification is one of the most natural techniques to use in this setup, since it relies directly and only on distances. As mentioned in the previous subsection, for this classifier to perform well, it requires the proper identification of the "real" nearest neighbors: the approximation made using the min-hash cannot be too crude.

Using $k = 2000$ hashes, a reference set is devised — by F-Secure Corporation — which contains samples that are considered to be representative of most current malware and clean samples. This set contains about 10000 samples (for each of which the $k = 2000$ minimum hashes have been extracted per feature number), balanced equally between clean and malware samples. The determination of this reference set is especially important as it should not contain samples for which there are some uncertainties about the class: Only samples with the highest probability of being either malware or clean are present in the reference set.

Once this reference set is fixed, samples can be compared against it using the min-hash based distances and a K-NN classifier.

Determining K for this problem is done using a validation set for which the certainty of the class of each sample is very high as well. The validation set contains 3000 samples, checked against the reference set of 10000 samples. Figure 4.21 depicts the classification accuracy (average of True Positive and True Negative rates) versus the value of K used for the K-NN. Surprisingly, the decision based on the very first nearest neighbor is always the best in terms of classification accuracy. Therefore, in

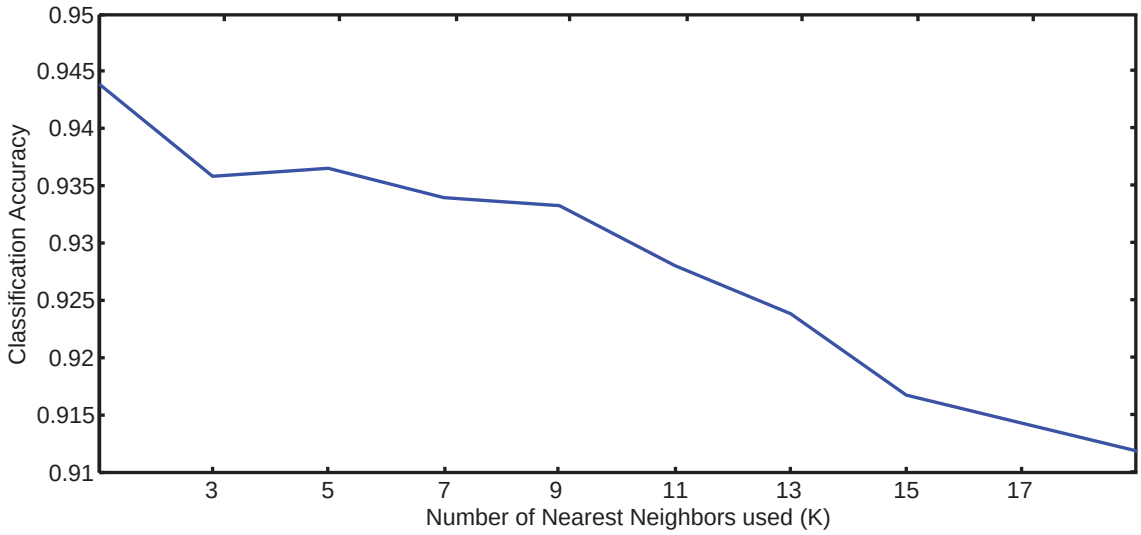


Figure 4.21: $K = 1$ is the best for this specific data regarding classification accuracy.

the following methodology presented in Section 4.4.3, a 1-NN is used as the first step classifier.

1-NN is Not Sufficient

As mentioned earlier, one of the main imperatives in this paper is to achieve 0 False Positives (in absolute numbers). As Table 4.8 depicts, by using a test set (totally separate from the validation sets used above) composed of 28510 samples for which the class is known with the highest confidence, with the 1-NN approach still yields large amounts of False Positives. Note that this test set is unbalanced, although not significantly.

The results of the 1-NN are not satisfactory regarding the constraint on the False Positives. An obvious way of addressing directly the amount of False Positives is to set a maximum threshold on the distance to the first nearest neighbor: Above

Table 4.8: Confusion Matrix for the sole 1-NN on the test set. If only the first stage of the methodology is used, results are unacceptable in terms of False Positive rates.

		Actual	
		Malware	Clean
Prediction	Malware	18160	183
	Clean	277	9890

this threshold, the sample is deemed too far from its nearest neighbor, and no decision is taken.

While this strategy would effectively reduce the number of False Positives, it lowers significantly the number of True Positives as well, i.e. the coverage. For this reason, and to keep a high coverage, the following methodology using a second stage classifier as the ELM, is proposed.

As can be seen from Figure 4.19, the computational time required to calculate the distance from a test sample to the whole 10000 reference set samples is about 35 seconds on average, using $k = 2000$ hashes. This is still acceptable, from the practical point of view, but adding a second stage classifier has the obvious drawback of increasing this time.

In order to make this increase the smallest possible, an Extreme Learning Machine model specialized for False Positives (and another for False Negatives) is used. Figure 4.20 illustrates the global idea of this two-stage methodology.

The motivation for an additional classifier comes from the fact that the single information from the 1-NN is not sufficient: the distance to that first neighbor is

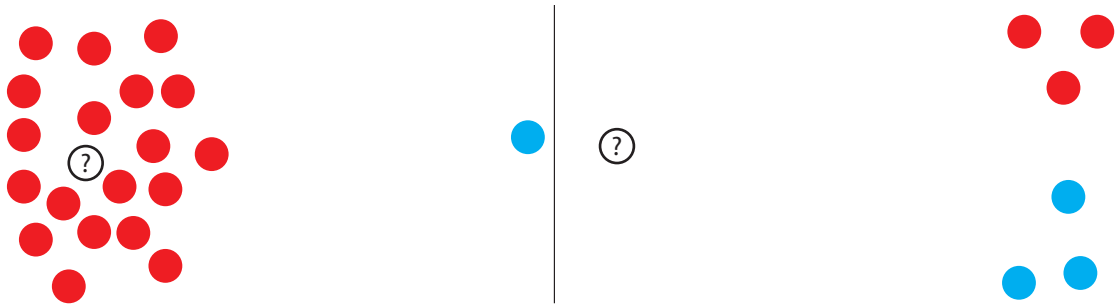


Figure 4.22: Illustration of different situations with identical 1-NN: in (a) the density of reference samples of the same class around the test sample gives the decision high confidence; in (b) while the 1-NN is of the same class as for (a), the confidence should be very different on the decision.

important as well, and so is the distance and the "rank" of the nearest neighbor of the opposite class. Figure 4.22 attempts to illustrate two different situations for which a test sample has its first nearest neighbor in the same class — note that the position of the samples has no meaning here, due to the nominal nature of the data; the distances are the interesting fact. In the first case (a), the confidence on the decision must be high, as many of the neighbors of the test sample are near and of the same class. The case (b) is very different and needs to have a much lower confidence on the decision taken, if any.

A means of describing such situations is to account for:

1. The distance to the nearest neighbor $d_{1-\mathrm{NN}}$: If the nearest neighbor is far, it is likely that the test sample is in a part of the original space where the reference samples density is insufficient;

2. The distance to the "nearest neighbor of the opposite class" $d_{\mathrm{NN}\neq}$: If $d_{1-\mathrm{NN}}$ is very similar to $d_{\mathrm{NN}\neq}$, the test sample lies in a part of the space where reference samples of both classes are present and at similar distances;
3. The "rank" of this neighbor of opposite class $R_{\mathrm{NN}\neq}$ (is it the 3rd or 100th neighbor?): This information gives a rough sense of the "density" of the reference samples of the same class as that of the nearest neighbor around the test sample.

The combination of these additional three pieces of information describes roughly the situation in which the test sample lies. This is the information fed to the second stage classifier for the final decision.

Second Stage Decision using modified ELM

As depicted on Figure 4.22 , the single information of the class of the nearest neighbor is not sufficient to obtain zero False Positives. The proposed second stage classifier uses modified ELM models for lowering the amounts of False Positives — one of the two modified ELM models reduces False Negatives as well; only the False Positive minimizing one is mentioned in the following.

The modified ELM model used in the second stage of the methodology is specially optimized so as to minimize the False Positives (a similar model to minimize the False Negatives is used as well, in the same fashion). It uses additional information gathered while searching for the nearest neighbor (so no additional computational time is required to obtain the training data): the distance to the nearest neighbor $d_{1-\mathrm{NN}}$, the distance to the nearest neighbor of the opposite class $d_{\mathrm{NN}\neq}$, and the rank

of this neighbor of opposite class $R_{\mathrm{NN}\neq}$. With this input data, the False Positive Optimized ELM is trained using a weighted classification accuracy criterion.

While for binary classification problems, the classification rate Acc defined as the average of the True Positive Rate TPR and True Negative Rate TNR,

$$\mathrm{Acc} = \frac{\mathrm{TNR} + \mathrm{TPR}}{2}, \tag{4.20}$$

is typically used as a performance measure, the proposed modified ELM uses the following weighted accuracy $\mathrm{Acc}(\alpha)$

$$\mathrm{Acc}(\alpha) = \frac{\alpha\mathrm{TNR} + \mathrm{TPR}}{1 + \alpha}. \tag{4.21}$$

By changing the α weight, it becomes possible to give precedence to the True Negative Rate and thus to avoid false positives. The output of the proposed False Positive Optimized ELM is calculated using Leave-One-Out PRESS statistics as explained in section 2.2.3.

In order to obtain a parsimonious model in the shortest possible time, the proposed modified ELM uses the idea of the TROP-ELM 2.3.3 and OP-ELM 2.2.1 to prune out neurons from an initially large ELM model. In addition, for computational time considerations, the maximum number M of selected neurons desired for the final model is taken as a parameter. Overall, the False Positive Optimized ELM used in this paper follows the steps of Algorithm 4.1.

Algorithm 4.1 False Positive Optimized ELM.

Given a training set $(\mathbf{x}_i, y_i), \mathbf{x}_i \in \mathbb{R}^3, y_i \in \{-1, 1\}$, an activation function $\phi : \mathbb{R} \mapsto \mathbb{R}$, a large number of hidden nodes N and the maximum number $M < N$ of neurons to retain for the final model:

1: Randomly assign input weights $\mathbf{w}_i$ and biases b_i, $i \in [\![1, N]\!]$;

2: Calculate the hidden layer output matrix $\mathbf{H}$;

3: **for** $i = 1$ to M **do**

4: Perform Forward Selection of the i best neurons (among N) using PRESS LOO output with Acc(α) criterion, and ELM determination of the output weights β_i;

5: **end for**

6: Retain the best combination out of the M different selections as the final model structure.

The selection of the optimal α is done experimentally, following the two constraints of 0 False Positives and highest possible coverage (i.e. as many True Positives as possible). Figure 4.23 is the Receiver Operating Characteristic curve for various values of α, plotted for a balanced 3000 samples validation set. As can be seen, the requirement on absolutely 0 False Positives has a strong influence on the coverage (represented by the True Positives rate here). If one allows as low as 0.06% False Positives, the coverage reaches 92% already.

Figure 4.24 depicts the plot of the False Positive rate against the α value.

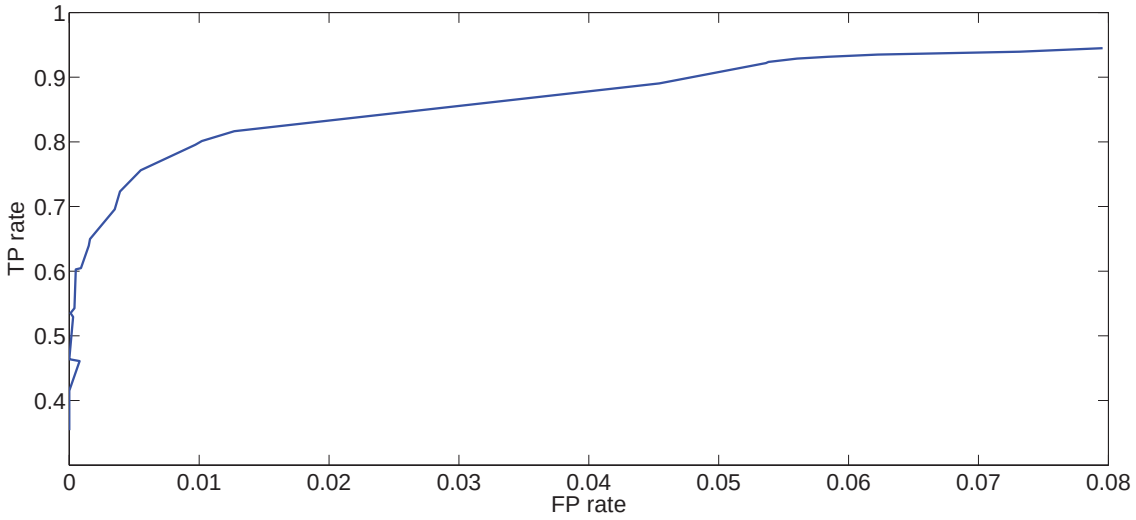


Figure 4.23: ROC curve (True Positive Rate versus False Positive Rate) for varying values of α.

This plot is using the same validation data as Figure 4.23. The value of α for which the 0 False Positives requirement is met while keeping highest possible coverage is $\alpha = 30$, from Figure 4.24.

The case of evaluating α for the False Negative minimizing ELM is not depicted here as the false negatives are of less interest to the practical application at hand in this paper. Nonetheless, the α value of the False Negative minimizing ELM has been found to be 27 using the same experimental setup as for the False Positive minimizing one. This value can be seen as a tuning parameter of the model which enables the user to have control over the specific decision making ability of the model: by adjusting it, it is possible to get a smaller amount of false positives, at the cost of coverage, as the final cognitive model then prefers to take no decision than to make a mistake.

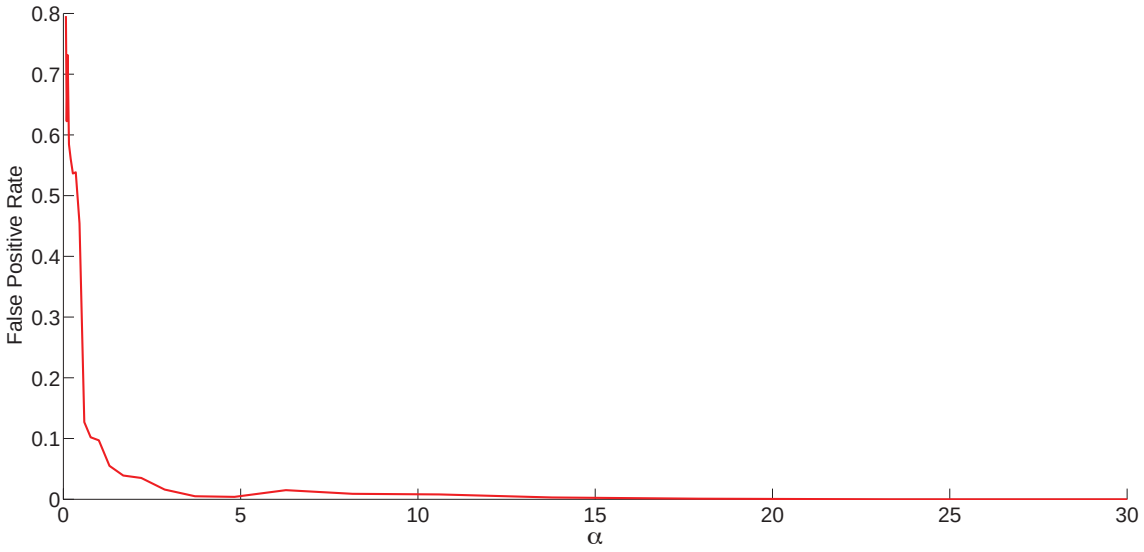


Figure 4.24: Evolution of the False Positive Rate as a function of the α weight. The first attained zero False Positive Rate is for $\alpha = 30$.

Final Results on Test Data

With the parameters of the two-stage methodology determined as above, i.e.:

- $k = 2000$ hashes used for the min-hash approximation of the Jaccard distance;
- $K = 1$ for the K-NN first stage classifier;
- α=30 for the False Positive Optimized ELM second stage classifier,
- $\alpha = 27$ for the False Negative Optimized ELM second stage classifier,

the presented methodology is applied to a test set of 28510 samples spanning from early 2008 until late 2011. The reference set of 10000 samples mentioned before is within the same time frame and balanced between malware and clean so as to reflect the "real" proportions, i.e. that of the samples received by F-Secure Corporation. The proportions are — roughly — 2/3 malware and 1/3 clean.

Tables 4.9 give the previous results of the sole 1-NN to be compared against the ones of the 1-NN and False Positive Optimized ELM methodology.

Table 4.9: Confusion matrices for (a) the training data (Leave-One-Out results) when training the False Positive/Negative Optimized ELMs; on the whole test set, (b) using only the 1-NN approach and (c) using the proposed 1-NN and ELM two-stage methodology. The reduction in coverage from the second stage ELM is noticeable, as False Positives and Negatives are decreased significantly.

		Actual	
		Malware	Clean
Prediction	Malware	1930	1
	Clean	1	908
	Unknown	2473	1623

(a) Confusion Matrix for the two-stage classifier methodology on the training data.

		Actual	
		Malware	Clean
Prediction	Malware	18160	183
	Clean	277	9890

(b) Confusion Matrix for the sole 1-NN on the test set.

		Actual	
		Malware	Clean
Prediction	Malware	8393	2
	Clean	7	4115
	Unknown	10037	5956

(c) Confusion Matrix for the two-stage classifier methodology on the test set.

It can be seen that the False Positive rate achieved in test is in line with the results from the Leave-One-Out in (a).

The results depicted in Table 4.9 (c) use not only a False Positive Optimized ELM but also a False Negative Optimized ELM, to reduce the False Negatives, as mentioned on Figure 4.20. The improvements in the reduction of the False Positives and the coverage achieved are satisfying for this test set.

A value of 2 False Positives in this test set is probably acceptable in practice. If the strict goal of 0 False Positives in test is to be enforced, then one possibility is to increase the α parameter to a higher value, more conservative. This has the effect of lowering further the coverage, though.

Two additional experiments are presented in Tables 4.10 and 4.11 for classification datasets from UCI Machine Learning repository [Lic13]. In order to apply the exact same methodology as presented in this paper, pairwise distances have been computed between samples so as to be in the same situation as for the specific data treated in this paper. Data has been shuffled and two thirds are taken for training and the remaining third is kept for test. As can be seen from Tables 4.10 and 4.11, the proposed methodology also enables to reduce the amount of false positives at the cost of reduced coverage. It is worth noting that the quality of the results for these two other datasets is less than for the main dataset considered in this paper. This is possibly due to the difference in the metric used. For the UCI datasets, the Euclidean metric has been used, while the proposed approximation to the Jaccard distance behaves probably differently in terms of the distance matrices it produces.

Note on Hardware and Computational Time Considerations

While the details of the implementation are not mentioned in this paper, the proposed methodology uses a set of three computers, each equipped with 8GB of RAM, and Intel Core2 Quad CPUs. Apache Cassandra is the distributed database framework used for performing efficient min-hash computations in batches, and a memory-held queueing system (based on memcached) is holding jobs for execution against Cassandra database. All additional computations are performed using Python code on one of the three computers mentioned.

With this setup, as seen on Figure 4.19, the average per sample evaluation time — i.e. calculating pairwise distances to the 10000 reference samples and finding the closest elements — is about 35 seconds. The choice of Cassandra as a database backend is meant so that the computational time grows only linearly if the precision of the min-hash or the number of reference samples is increased linearly: growing the number of reference samples linearly or the number k of hashes used for the min-hash approximation only requires a linear growth in the number of Cassandra nodes for the computational time to remain identical.

Table 4.10: Confusion matrices for Pima Indians Diabetes dataset from UCI [Lic13]. (a) the training data (Leave-One-Out results) when training the False Positive/Negative Optimized ELMs; on the whole test set, (b) using only the 1-NN approach and (c) using the proposed 1-NN and ELM two-stage methodology.

	Actual	
	Mal.	Clean
Mal.	105	0
Clean	0	227
Unkn.	87	93

(a) 2-stage in training

	Actual	
	Mal.	Clean
Mal.	34	44
Clean	42	136

(b) 1-NN in test

	Actual	
	Mal.	Clean
Mal.	5	10
Clean	41	135
Unkn.	30	35

(c) 2-stage in test

Table 4.11: Confusion matrices for Wisconsin Breast Cancer dataset from UCI [Lic13]. (a) the training data (Leave-One-Out results) when training the False Positive/Negative Optimized ELMs; on the whole test set, (b) using only the 1-NN approach and (c) using the proposed 1-NN and ELM two-stage methodology.

	Actual	
	Mal.	Clean
Mal.	124	0
Clean	0	229
Unkn.	15	11

(a) 2-stage in training

	Actual	
	Mal.	Clean
Mal.	69	3
Clean	4	114

(b) 1-NN in test

	Actual	
	Mal.	Clean
Mal.	63	1
Clean	4	114
Unkn.	6	2

(c) 2-stage in test

4.5 Improvements to HP-ELM Toolbox

The HP-ELM Toolbox remains in active development and support after publication of the paper, to keep it a useful practical tool for all Machine Learning practitioners around the world. A summary of most important recent changes is given below.

First, the GPU support is significantly improved. The code is based on a very recent "scikit-cuda" library [Giv15] that was released after the publication of [ABML15]. This library provides linear algebra computations on GPU in native Python code, and is easily installed with standard package manager. Scikit-CUDA enables GPU support in HP-ELM without additional compilation step, providing the power of accelerated ELM to people not experienced with C or C++ languages.

This convenient library also facilitated moving all computation-heavy steps on GPU (including prediction with ELM), and saving runtime by computing one triangle of a symmetric helper matrix in some cases. It also allowed writing an asynchronous code that runs GPU computations while on CPU it prepares the next batch of data and sends it to the graphics card. Asynchronous code removes the time of waiting for disc access from ELM training time, thus ELM runs at the maximum speed allowed by an accelerator.

Another improvement provided a distributed parallel training of ELM. It requires only a shared storage between different computers and suits well to clusters or any machines in a shared network. Additional ELM training methods run asynchronously with CPU and GPU alike, saving time for slow network access, for example

to a shared drive. They are tested and proved useful on several machines in a classroom.

The documentation[15] is significantly improved, which is a very important part of any toolbox or library aimed to be useful for a vast range of programmers. All methods are comprehensively documented, and the modern documentation website is automatically updates when a new version of code is uploaded online. The documentation includes tutorial for distributed computation with ELM.

[15] `hpelm.readthedocs.org`

CHAPTER 5
CONCLUSIONS

This thesis is dedicated to the Extreme Learning Machines (ELMs), a non-linear neural network-like structures with linear system solution. ELMs exist at the intersection between linear and nonlinear methods of Machine Learning, and take the best from both of these worlds. nonlinear methods provide a universal approximation property, which is an ability to learn (approximate with any precision) any function or dependency in the world, if only there is enough training data and computational power. Linear methods give robust and fast solution (based on Single Value Decomposition), or simply an extremely fast solution (based on LU decomposition), which both are unique solutions with exact computation algorithm, implemented in an extremely efficient standard subprograms on CPUs or accelerators. Also linear methods provide all sorts of existing computational tricks like computing LOO error with a single non-iterative equation, rank neurons by usefulness and retain only a necessary minimum, or easily control range of solution coefficient with L^2 regularization.

The biggest impact of Extreme Learning Machines in a whole field of Machine Learning lies in their practical applicability. The core of ELM is a PCA method between a nonlinearly transformed inputs and output; thus ELMs are basically a nonlinear version of PCA. While basic linear systems are outdated in scientific community, fascinated by Deep Learning and Quantum Computing nowadays, they are a backbone of most practical applications used at a large scale even by an advanced hi-tech company like Google. Extreme Learning Machines can add nonlinear capabilities

to all these practical systems without virtually any change in infrastructure.

That idea, and reviewing multiple beautiful application papers with ELMs, motivated the author for creation of an ELM toolbox that would provide the fastest, the most convenient and the most easy to use implementation of ELM for all those practical programmers and researchers. The toolbox also aims at addressing one significant drawback, that some regularization and model selection practices of ELM are not straightforward to understand and implement, while they benefit greatly to the performance of the method. The toolbox includes all these improvements activated by a simple option without extensive parameter selection process. ELM toolbox is an ambitious project for a single person, but it is feasible with the simplicity of the method, and is currently a work under active development.

The fast solution and universal approximation property make Extreme Learning Machines a good candidate for Big Data analysis. The aforementioned toolbox presents the raw performance on Big Data applications, which is limited only by hard disk capacity and common sense. ELMs solution can be conveniently distributed across multiple machines, because it requires very few synchronizations (precisely – one). Big Data is about complexity as well, and often requires complex solutions. ELM serves well in such setups as a building block for a large and complex Machine Learning approach; and it simply works without much tuning, leaving time and effort for addressing harder parts of the whole methodology. Two successful examples of such methodologies are presented in the thesis in sections 4.3 and 4.4. The first problem required analysis of image data collected online, in various sizes, resolutions and

encodings. The input data in the second problem consisted of hashes, with equality check being the only operation possible with them. Both methodologies put large effort at just handling the data and extracting features from it, then ELMs helped to test and compare multiple feature extraction methods without worrying about the ELM classifier, and tune the final solution according to requirements of the specific problem.

The linear algebra tricks allow to change ELM method for non-typical applications, with promising results not available if ELM is replaced by a more common nonlinear method like SVM. A good example of that is ELMVIS+, a visualization method based on ELM. Visualization methods work well if the data manifold is two-dimensional, and fits natively on a two-dimensional visualization space. With higher manifold dimensions, it is hard to calculate meaningful coordinates of visualization point. An ELMVIS+ method approaches the problem from another perspective, simply taking any set of points as visualization coordinates, and finding the best possible visualization with them. The problem becomes an assignment problem, where ELM provides the global reconstruction error and ELMVIS+ randomly searches for pairs of data samples which would decrease the global error if swapped their places. A random search may be slow, but numerical tricks allow testing over a million candidates per second in a set of 10,000 MNIST handwritten digits, with dimensionality around 600. The visualization is obtained much faster than for any other nonlinear method, while the results are comparable to the best of these nonlinear methods. With a similar approach one can search for samples in the dataset with wrong labels

(mislabeled samples) and fix them, filtering the errors of dataset creators and getting better generalization performance for any method trained on a filtered dataset. More details are given in chapter 3

An important and under-researched are in Extreme Learning Machines is the reliability of predictions. Sure there are metrics like Mean Squared Error, but it is hard to tell how reliable is a particular predicted value for a particular test input. The predictions may be stable for some inputs while rough and approximate for others, and averaging their quality is unacceptable in many applications. Such applications include for example financial institutions, and nowadays they often use linear models, predictions of which are easily explainable despite a generally worse performance. A proposed method computes confidence intervals (estimates standard deviation of error, assuming it is normally distributed) individually for particular test samples. It can tell which samples are predicted reliably and for which an ELM could not compute exact answer. Such confidence intervals can be used to consider only reliable predictions and leave unreliable ones for human experts. They facilitate the automated processing and promote higher efficiency in reliability-oriented practical applications like financial institutions, where current analysis is limited to linear models or simply humans.

What is the future of ELM? What has still to be done? In our opinion, one of the challenges for ELM is the problem of multi-labels classification for Big Data. And researchers will have to focus on this issue. One possibility is to transform ELM into a model that can provide probabilities as outputs. A second challenge is to

combine classification and visualization in a single method. Researchers and users would benefit a lot of such merge. ELMVIS+ is probably a good starting point to go in that direction. I will continue such research and will develop novel methods which base on ELM in the future since I consider that ELM is the keystone of Machine Learning for Big Data.

REFERENCES

[AAAY13] Anton Akusok, Alexander Grigorievskiy, Amaury Lendasse, and Yoan Miche, *Image-based Classification of Websites*, Machine Learning Reports 02/2013 (Saarbrücken, Germany) (Thomas Villmann and Frank-Michael Schleif, eds.), Machine Learning Reports, vol. ISSN: 18, Workshop of the GI-Fachgruppe Neuronale Netze and the German Neural Networks Society in connection to GCPR 2013, September 2013, pp. 25–34.

[ABML15] Anton Akusok, Kaj-Mikael Björk, Yoan Miche, and Amaury Lendasse, *High-Performance Extreme Learning Machines: A Complete Toolbox for Big Data Applications*, IEEE Access **3** (2015), 1011–1025.

[ADY+14] Anton Akusok, David Veganzones, Yoan Miche, Eric Séverin, and Amaury Lendasse, *Finding Originally Mislabels with MD-ELM*, Proceedings of ESANN2014: 22nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, i6doc, Bruges, Belgium, 23-25 April 2014, pp. 689–694.

[AF10] Giuseppe Amato and Fabrizio Falchi, *kNN Based Image Classification Relying on Local Feature Similarity*, Proceedings of the Third International Conference on SImilarity Search and APplications (New York, NY, USA), SISAP '10, ACM, 2010, pp. 101–108.

[AGM+16] Anton Akusok, Andrey Gritsenko, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Paula Lauren, and Amaury Lendasse, *Adding Reliability to ELM Predictions by Confidence Intervals*, Neurocomputing (submitted 2016).

[All74] David M. Allen, *The Relationship between Variable Selection and Data Agumentation and a Method for Prediction*, Technometrics **16** (1974), no. 1, 125–127.

[AMB+16a] Anton Akusok, Yoan Miche, Kaj-Mikael Björk, Rui Nian, Paula Lauren, and Amaury Lendasse, *ELMVIS+: Improved Nonlinear Visualization Technique Using Cosine Distance and Extreme Learning Machines*, Proceedings of ELM-2015 Volume 2: Theory, Algorithms and Applications (II) (Jiuwen Cao, Kezhi Mao, Jonathan Wu, and Amaury Lendasse, eds.), Springer International Publishing, Cham, 2016, pp. 357–369.

[AMB+16b] ———, *Evaluating Confidence Intervals for ELM Predictions*, Proceedings of ELM-2015 Volume 2: Theory, Algorithms and Applications (II) (Jiuwen Cao, Kezhi Mao, Jonathan Wu, and Amaury Lendasse, eds.), Springer International Publishing, Cham, 2016, pp. 413–422.

[AMB+16c] ———, *ELMVIS+: Fast Nonlinear Visualization Technique based on Cosine Distance and Extreme Learning Machines*, Neurocomputing (forthcoming 2016).

[Amd67] Gene M. Amdahl, *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*, Proceedings of the April 18-20, 1967, Spring Joint Computer Conference (New York, NY, USA), AFIPS '67 (Spring), ACM, 1967, pp. 483–485.

[AMH+14] Anton Akusok, Yoan Miche, Jozsef Hegedus, Rui Nian, and Amaury Lendasse, *A Two-Stage Methodology Using K-NN and False-Positive Minimizing ELM for Nominal Data Classification*, Cognitive Computation **6** (2014), no. 3, 432–445.

[AMK+15] Anton Akusok, Yoan Miche, Juha Karhunen, Kaj-Mikael Björk, Rui Nian, and Amaury Lendasse, *Arbitrary Category Classification of Websites Based on Image Content*, IEEE Computational Intelligence Magazine **10** (2015), no. 2, 30–41.

[AVM+15] Anton Akusok, David Veganzones, Yoan Miche, Kaj-Mikael Björk, Philippe du Jardin, Eric Séverin, and Amaury Lendasse, *MD-ELM: Originally Mislabeled Samples Detection using OP-ELM Model*, Neurocomputing **159** (2015), 242–250.

[BBB98] Gianluca Bontempi, Mauro Birattari, and Hugues Bersini, *Recursive lazy learning for modeling and control*, Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings (Claire Nédellec and Céline Rouveirol, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 292–303.

[BBD+12] Lamberto Ballan, Marco Bertini, Alberto Del Bimbo, Andrea M Serain, Giuseppe Serra, and Benito F Zaccone, *Combining generative and discriminative models for classifying social images from 101 object categories*, Pattern Recognition (ICPR), 2012 21st International Conference on, 11-15 Nov. 2012, pp. 1731–1734.

[BCFM00] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher, *Min-Wise Independent Permutations*, Journal of Computer and System Sciences **60** (2000), no. 3, 630–659.

[BDM12] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*, Society for Industrial and Applied Mathematics, Philadelphia, 2012.

[Ben13] Benoit Frénay, *Uncertainty and label noise in machine learning*, Ph.D. thesis, Université catholique de Louvain, September 2013.

[BETV08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, *Speeded-Up Robust Features (SURF)*, Similarity Matching in Computer Vision and Multimedia **110** (2008), no. 3, 346–359.

[BF99] Carla E Brodley and Mark A Friedl, *Identifying mislabeled training data*, Journal Of Artificial Intelligence Research **11** (1999), 131–167.

[BG09] Gertjan J. Burghouts and Jan-Mark Geusebroek, *Performance evaluation of local colour invariants*, Computer Vision and Image Understanding **113** (2009), no. 1, 48–62.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig, *Syntactic clustering of the Web*, Papers from the Sixth International World Wide Web Conference **29** (1997), no. 8–13, 1157–1166.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft, *When Is "Nearest Neighbor" Meaningful?*, Database Theory — ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings (Catriel Beeri and Peter Buneman, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 1999, pp. 217–235.

[BHV99] H.-U. Bauer, Michael Herrmann, and Thomas Villmann, *Neural maps and topographic vector quantization*, Neural Networks **12** (1999), no. 4–5, 659–676.

[Bie87] I Biederman, *Recognition-by-components: a theory of human image understanding*, Psychological Review **94** (1987), no. 2, 115–147.

[Bis96] Christopher M. Bishop, *Neural Networks for Pattern Recognition*, Advanced Texts in Econometrics, Clarendon Press, January 1996.

[Bis06] Christopher M Bishop, *Pattern Recognition and Machine Learning*, Information science and statistics, vol. 4, Springer Science+Business Media, Singapore, 2006.

[BMTG12] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool, *Pedestrian detection at 100 frames per second*, Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, 16-21 June 2012, pp. 2903–2910.

[BN01] Mikhail Belkin and Partha Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, Advances in Neural Information Processing Systems, vol. 14, MIT Press, 2001, pp. 585–591.

[BN03] ———, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural computation **15** (2003), no. 6, 1373–1396.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan, *Latent Dirichlet Allocation*, Journal of Machine Learning Research **3** (2003), no. 4-5, 993–1022.

[BOA+07] Michael Bailey, Jon Oberheide, Jon Andersen, Z. Morley Mao, Farnam Jahanian, and Jose Nazario, *Automated Classification and Analysis of Internet Malware*, Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings (Christopher Kruegel, Richard Lippmann, and Andrew Clark, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 178–197.

[BP92] H. U. Bauer and Klaus R. Pawelzik, *Quantifying the neighborhood preservation of self-organizing feature maps*, IEEE Transactions on Neural Networks **3** (1992), no. 4, 570–579.

[Bro97] Andrei Z Broder, *On the resemblance and Containment of Documents*, Proceedings of Compression and Complexity of Sequences 1997, SEQUENCES '97, IEEE Computer Society, 1997, pp. 21–29.

[BSI08] Oren Boiman, Eli Shechtman, and Michal Irani, *In defense of Nearest-Neighbor based image classification*, Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, June 2008, pp. 1–8.

[BSW98] Christopher M Bishop, Markus Svensén, and Christopher K I Williams, *GTM: The generative topographic mapping*, Neural computation **10** (1998), no. 1, 215–234.

[CAK+15] Colin Swaney, Anton Akusok, Kaj-Mikael Björk, Yoan Miche, and Amaury Lendasse, *Efficient Skin Segmentation via Neural Networks: HP-ELM and BD-SOM*, INNS Conference on Big Data 2015 Program San Francisco, CA, USA 8-10 August 2015 **53** (2015), 400–409.

[CCG91] S. Chen, C. F. N. Cowan, and P. M. Grant, *Orthogonal least squares learning algorithm for radial basis function networks*, IEEE Transactions on Neural Networks **2** (1991), no. 2, 302–309.

[CH67] Thomas M. Cover and Peter E. Hart, *Nearest neighbor pattern classification*, IEEE Transactions on Information Theory **13** (1967), no. 1, 21–27.

[CHV99] Olivier Chapelle, Patrick Haffner, and Vladimir N Vapnik, *Support vector machines for histogram-based image classification*, IEEE Transactions on Neural Networks / a publication of the IEEE Neural Networks Council **10** (1999), no. 5, 1055–1064.

[CKW12] Jingjing Cao, Sam Kwong, and Ran Wang, *A noise-detection based AdaBoost algorithm for mislabeled data*, Pattern Recognition **45** (2012), no. 12, 4451–4465.

[CL11] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology **2** (2011), no. 3, 27:1–27:27.

[Co13] Eric Cambria and others, *Extreme Learning Machines [Trends & Controversies]*, IEEE Intelligent Systems **28** (Nov.-Dec. 2013), no. 6, 30–59.

[Cro84] F C Crow, *Summed-area tables for texture mapping*, ACM SIGGRAPH computer graphics **18** (1984), no. 3, 207–212.

[CSFS02] Barbara Caputo, K. Sim, F. Furesjo, and Alexander Smola, *Appearance-based Object Recognition using SVMs: Which Kernel Should I Use?*, Proc of NIPS workshop on Statistical methods for computational experiments in visual processing and computer vision (Whistler), 2002.

[CSH12] Qiang Chen, Zheng Song, and Yang Hua, *Hierarchical matching with side information for image classification*, Proc. of 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, June 2012, pp. 3426–3433.

[CV95] Corinna Cortes and Vladimir Vapnik, *Support-vector networks*, Machine learning **20** (1995), no. 3, 273–297.

[CW79] J. Lawrence Carter and Mark N. Wegman, *Universal classes of hash functions*, Journal of Computer and System Sciences **18** (1979), no. 2, 143–154.

[CXWZ11] Xinyuan Cai, Baihua Xiao, Chunheng Wang, and Rongguo Zhang, *A local learning based Image-To-Class distance for image classification*, Pattern Recognition (ACPR), 2011 First Asian Conference on, 28-28 Nov. 2011, pp. 667–671.

[DH97] Pierre Demartines and Jeanny Herault, *Curvilinear component analysis: a self-organizing neural network for nonlinear mapping of data sets*, IEEE Transactions on Neural Networks **8** (1997), no. 1, 148–154.

[DRS+13] Thomas Dean, Mark A. Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik, *Fast, Accurate Detection of 100,000 Object Classes on a Single Machine*, Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on, 23-28 June 2013, pp. 1814–1821.

[DT05] N. Dalal and B. Triggs, *Histograms of oriented gradients for human detection*, Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, 25-25 June 2005, pp. 886–893 vol. 1.

[du 07] P. du Jardin, *Prévision de la défaillance et réseaux de neurones: l'apport des méthodes numériques de sélection de variables*, Université de Nice-Sophia-Antipolis, 2007.

[DZ11] Bala S Divakaruni and Jie Zhou, *Image Categorization using Codebooks Built from Scored and Selected Local Features*, Proc. of The 2011 International Conference on Image Processing, Computer Vision and Pattern Recognition (IPCV), vol. 1, 2011, pp. 3–9.

[DZC09] Wanyu Deng, Qinghua Zheng, and Lin Chen, *Regularized Extreme Learning Machine*, Computational Intelligence and Data Mining, 2009. CIDM '09. IEEE Symposium on, March 30 2009-April 2 2009, pp. 389 –395.

[EGW+10] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman, *The Pascal Visual Object Classes (VOC) Challenge*, International Journal of Computer Vision **88** (2010), no. 2, 303–338.

[EHJT04] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani, *Least Angle Regression*, The Annals of Statistics **32** (2004), no. 2, 407–499.

[F-S06] F-Secure Corporation, *F-Secure DeepGuard – A proactive response to the evolving threat scenario*, November 2006.

[FFP07] Li Fei-Fei, Rob Fergus, and Pietro Perona, *Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories*, Special issue on Generative Model Based Vision **106** (2007), no. 1, 59–70.

[FFPZ10] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman, *Learning Object Categories From Internet Image Searches*, Proceedings of the IEEE **98** (2010), no. 8, 1453–1466.

[Fos95] Ian Foster, *Designing and building parallel programs*, Addison-Wesley, 1995.

[FV14] Benoit Frénay and Michel Verleysen, *Classification in the Presence of Label Noise: A Survey*, IEEE Transactions on Neural Networks and Learning Systems **25** (2014), no. 5, 845–869.

[GHP07] Gregory Griffin, Alex Holub, and Pietro Perona, *Caltech-256 object category dataset*, Tech. report, Technical Report. California Institute of Technology, Pasadena, CA, 2007, http://resolver.caltech.edu/CaltechAUTHORS:CNS-TR-2007-001.

[GHW79] Gene H. Golub, Michael Heath, and Grace Wahba, *Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter*, Technometrics **21** (1979), no. 2, 215–223.

[Giv15] Lev Givon, *Scikit-CUDA*, `https://scikit-cuda.readthedocs.org`, 2015.

[GLG99] Dragan Gamberger, Nada Lavra**v**c, and Ciril Gro**v**selj, *Experiments with noise filtering in a medical domain*, Proc. 16th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1999, pp. 143–151.

[GP02] Gregory Gutin and Abraham P. Punnen (eds.), *The traveling salesman problem and its variations*, Combinatorial optimization, Kluwer Academic, Dordrecht, London, 2002.

[GS96] Geoffrey J Goodhill and Terrence J Sejnowski, *Quantifying neighbourhood preservation in topographic mappings*, Proceedings of the 3rd Joint Symposium on Neural Computation, vol. 6, Citeseer, 1996, pp. 61–82.

[Hay98] Simon Haykin, *Neural Networks: A Comprehensive Foundation (2nd Edition)*, 2nd ed., Prentice Hall, July 1998.

[HBKV15] Guang-Bin Huang, Zuo Bai, L.L.C. Kasun, and Chi Man Vong, *Local Receptive Fields Based Extreme Learning Machine*, IEEE Computational Intelligence Magazine **10** (2015), no. 2, 18–29.

[HCS06] Guang-Bin Huang, Lei Chen, and Chee-Kheong Siew, *Universal approximation using incremental constructive feedforward networks with random hidden nodes*, IEEE Transactions on Neural Networks **17** (2006), no. 4, 879–892.

[HK70] Arthur E Hoerl and Robert W Kennard, *Ridge Regression: Biased Estimation for Nonorthogonal Problems*, Technometrics **12** (1970), no. 1, 55–67.

[HKQ10] Jian Hou, Jianxin Kang, and Naiming Qi, *On Vocabulary Size in Bag-of-Visual-Words Representation*, Advances in Multimedia Information Processing - PCM 2010: 11th Pacific Rim Conference on Multimedia, Shanghai, China, September 21-24, 2010, Proceedings, Part I (Guoping Qiu, Kin Man Lam, Hitoshi Kiya, Xiang-Yang Xue, C.-C. Jay Kuo, and Michael S. Lew, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 414–424.

[HL08] Mark J Huiskes and Michael S Lew, *The MIR Flickr retrieval evaluation*, MIR '08: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval (Vancouver, Canada), ACM, 2008, pp. 39–43.

[HMZ+06] Guang-bin Huang, Senior Member, Qin-yu Zhu, KZ Z Mao, Chee-kheong Siew, P Saratchandran, and Narashiman Sundararajan, *Can threshold networks be trained directly?*, IEEE Transactions on Circuits and Systems II: Express Briefs **53** (2006), no. 3, 187–191.

[HRT04] Nicholas P. Hughes, Stephen J. Roberts, and Lionel Tarassenko, *Semi-supervised learning of probabilistic models for ECG segmentation*, Engineering in Medicine and Biology Society, 2004. IEMBS '04. 26th Annual International Conference of the IEEE, vol. 1, 1-5 Sept. 2004, pp. 434–437.

[Hua14] Guang-Bin Huang, *An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels*, Cognitive Computation **6** (2014), no. 3, 376–390.

[Hua15] _____, *What are Extreme Learning Machines? Filling the Gap Between Frank Rosenblatt's Dream and John von Neumann's Puzzle*, Cognitive Computation **7** (2015), no. 3, 263–278.

[HZDZ12] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang, *Extreme learning machine for regression and multiclass classification.*, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **42** (2012), no. 2, 513–529.

[HZS04] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, *Extreme learning machine: a new learning scheme of feedforward neural networks*, Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, vol. 2, 25-29 July 2004, pp. 985–990 vol.2.

[HZS06] _____, *Extreme learning machine: Theory and applications*, Neural NetworksSelected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04)7th Brazilian Symposium on Neural Networks **70** (2006), no. 1–3, 489–501.

[IP95] B Igelnik and Y.-H. Pao, *Stochastic choice of basis functions in adaptive function approximation and the functional-link net*, Neural Networks, IEEE Transactions on **6** (1995), no. 6, 1320–1329.

[Jac01] Paul Jaccard, *Étude comparative de la distribution florale dans une portion des Alpes et du Jura*, Bulletin de la Société Vaudoise des Sciences Naturelles **37** (1901), 547–579.

[JB08] Yushi Jing and Shumeet Baluja, *VisualRank: Applying PageRank to Large-Scale Image Search*, IEEE Transactions on Pattern Analysis and Machine Intelligence **30** (2008), no. 11, 1877–1890.

[JM09] Dan Jurafsky and James H Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*, 2nd ed., Prentice Hall series in artificial intelligence, Pearson Prentice Hall, Upper Saddle River, N.J., 2009.

[JWF10] Piyasak Jeatrakul, Kok Wai Wong, and Chun Che Fung, *Data cleaning for classification using misclassification analysis*, Journal of Advanced Computational Intelligence and Intelligent Informatics **14** (2010), no. 3, 297–302.

[KL51] Solomon Kullback and Richard A Leibler, *On information and sufficiency*, The Annals of Mathematical Statistics **22** (1951), no. 1, 79–86.

[KME+11] L. Kainulainen, Yoan Miche, Emil Eirola, Qi Yu, Benoit Frénay, Eric Séverin, and Amaury Lendasse, *Ensembles of Local Linear Models for Bankruptcy Analysis and Prediction*, Case Studies in Business, Industry and Government Statistics (CSBIGS) **4** (2011), no. 2, 116–133.

[Koh82] Teuvo Kohonen, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics **43** (1982), no. 1, 59–69.

[KP03] Samuel Kaski and Jaakko Peltonen, *Informative discriminant analysis*, Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), vol. 20, AAAI Press, Menlo Park, CA, 2003, pp. 329–336.

[KP11] ———, *Dimensionality Reduction for Data Visualization [Applications Corner]*, IEEE Signal Processing Magazine **28** (2011), no. 2, 100–104.

[Kru64] Joseph B Kruskal, *Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis*, Psychometrika **29** (1964), no. 1, 1–27.

[Lac74] Peter A. Lachenbruch, *Discriminant Analysis When the Initial Samples Are Misclassified II: Non-Random Misclassification Models*, Technometrics **16** (1974), no. 3, 419–424.

[LAV03] John Aldo Lee, Cédric Archambeau, and Michel Verleysen, *Locally linear embedding versus isotop*, ESANN'2003 proceedings - European Symposium on Artificial Neural Networks, d-side publi., Bruges (Belgium), 23-25 April 2003, pp. 527–534.

[LCLo09] Giampaolo Luiz Libralon, André Carlos Ponce de Leon Carvalho, Ana Carolina Lorena, and others, *Pre-processing for noise detection in gene expression classification data*, Journal of the Brazilian Computer Society **15** (2009), no. 1, 3–11.

[Lel93] Subhash Lele, *Euclidean Distance Matrix Analysis (EDMA): Estimation of mean form and mean form difference*, Mathematical Geology **25** (1993), no. 5, 573–602.

[LHF02] Pui Y. Lee, Siu C. Hui, and Alvis Cheuk M. Fong, *Neural networks for web content filtering*, IEEE Intelligent Systems **17** (Sep/Oct 2002), no. 5, 48–57.

[LHSS06] Nan-Ying Liang, Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan, *A fast and accurate online sequential learning algorithm for feedforward networks*, Neural Networks, IEEE Transactions on **17** (2006), no. 6, 1411–1423.

[Lic13] M. Lichman, *UCI machine learning repository*, 2013.

[Lin98] Tony Lindeberg, *Feature Detection with Automatic Scale Selection*, International Journal of Computer Vision **30** (1998), no. 2, 79–116.

[LJRV05] Amaury Lendasse, Yongnan Ji, Nima Reyhani, and Michel Verleysen, *LS-SVM Hyperparameter Selection with a Nonparametric Noise Estimator*, Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005, Lecture Notes in Computer Science, vol. 3697, Springer Berlin Heidelberg, 2005, pp. 625–630.

[LLDV00] John Aldo Lee, Amaury Lendasse, Nicolas Donckers, and Michel Verleysen, *A robust nonlinear projection method*, ESANN'2000 proceedings - European Symposium on Artificial Neural Networks (Bruges (Belgium)), D-Facto public., 26-28 April 2000, pp. 13–20.

[LLV04] John Aldo Lee, Amaury Lendasse, and Michel Verleysen, *Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis*, New Aspects in Neurocomputing: 10th European Symposium on Artificial Neural Networks 2002 **57** (2004), 49–76.

[LND09] Hanna Lukashevich, Stefanie Nowak, and Peter Dunker, *Using one-class SVM outliers detection for verification of collaboratively tagged image training sets*, Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on, June 28 2009-July 3 2009, pp. 682–685.

[Low99] David Lowe, *Object recognition from local scale-invariant features*, Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on (J Tsotsos, ed.), ICCV '99, vol. 2, IEEE, 1999, pp. 1150–1157.

[Low04] David G. Lowe, *Distinctive Image Features from Scale-Invariant Keypoints*, International Journal of Computer Vision **60** (2004), no. 2, 91–110.

[LV07] John A Lee and Michel Verleysen, *Nonlinear dimensionality reduction*, Springer, 2007.

[MAV+15] Yoan Miche, Anton Akusok, David Veganzones, Kaj-Mikael Björk, Eric Séverin, Philippe du Jardin, Maite Termenon, and Amaury Lendasse, *SOM-ELM—Self-Organized Clustering using ELM*, Neurocomputing **165** (2015), 238 – 254.

[MBJ+08] Yoan Miche, Patrick Bas, Christian Jutten, Olli Simula, and Amaury Lendasse, *A Methodology for Building Regression Models using Extreme Learning Machine: OP-ELM*, Proceedings of the European Symposium on Artificial Neural Networks (ESANN), 2008, pp. 247–252.

[MLS05] Krystian Mikolajczyk, Bastian Leibe, and Bernt Schiele, *Local features for object class recognition*, Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, vol. 2, 17-21 Oct. 2005, pp. 1792–1799.

[MS04] Krystian Mikolajczyk and Cordelia Schmid, *Scale & Affine Invariant Interest Point Detectors*, International Journal of Computer Vision **60** (2004), no. 1, 63–86.

[MS05] ———, *A performance evaluation of local descriptors*, IEEE Transactions on Pattern Analysis and Machine Intelligence **27** (2005), no. 10, 1615–1630.

[MS10] Mahmoud A Mofaddel and Samy Sadek, *Adult image content filtering: A statistical method based on Multi-Color Skin Modeling*, Signal Processing and Information Technology (ISSPIT), 2010 IEEE International Symposium on, 15-18 Dec. 2010, pp. 366–370.

[MSB+10] Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse, *OP-ELM: Optimally-Pruned Extreme Learning Machine*, IEEE Transactions on Neural Networks **21** (2010), no. 1, 158–162.

[MSL08] Yoan Miche, Antti Sorjamaa, and Amaury Lendasse, *OP-ELM: Theory, Experiments and a Toolbox*, LNCS - Artificial Neural Networks - ICANN 2008 - Part I, Lecture Notes in Computer Science, vol. 5163/2008, Springer Berlin / Heidelberg, September 2008, pp. 145–154.

[MTBG13] Markus Mathias, Radu Timofte, Rodrigo Benenson, and Luc Van Gool, *Traffic sign recognition — How far are we from the solution?*, Neural Networks (IJCNN), The 2013 International Joint Conference on, 4-9 Aug. 2013, pp. 1–8.

[MTS+05] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, *A Comparison of Affine Region Detectors*, International Journal of Computer Vision **65** (2005), no. 1, 43–72.

[MvB+11] Yoan Miche, Mark van Heeswijk, Patrick Bas, Olli Simula, and Amaury Lendasse, *TROP-ELM: A double-regularized ELM using LARS and Tikhonov regularization*, Advances in Extreme Learning Machine: Theory and ApplicationsBiological Inspired Systems. Computational and Ambient IntelligenceSelected papers of the 10th International Work-Conference on Artificial Neural Networks (IWANN2009) **74** (2011), no. 16, 2413–2421.

[Mye90] R. H. Myers, *Classical and Modern Regression with Applications*, 2 ed., Duxbury, Pacific Grove, CA, USA, 1990.

[NM65] John A. Nelder and Roger Mead, *A Simplex Method for Function Minimization*, The Computer Journal **7** (1965), no. 4, 308–313.

[OBLS14] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic, *Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks*, Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, 23-28 June 2014, pp. 1717–1724.

[ON97] Seishi Okamoto and Yugami Nobuhiro, *An Average-case Analysis of the K-nearest Neighbor Classifier for Noisy Domains*, Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1 (San Francisco, CA, USA), IJCAI'97, Morgan Kaufmann Publishers Inc., 1997, pp. 238–243.

[Owe06] Art B. Owen, *A robust hybrid of lasso and ridge regression*, Contemporary Mathematics (Joseph Stephen Verducci, Xiaotong Shen, and John Lafferty, eds.), vol. 443, Stanford University, 2006.

[PBC05] S L Phung, A Bouzerdoum, and Sr. Chai D., *Skin segmentation using color pixel classification: analysis and comparison*, Pattern Analysis and Machine Intelligence, IEEE Transactions on **27** (2005), no. 1, 148–154.

[Pea01] Karl Pearson, *LIII. On lines and planes of closest fit to systems of points in space*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2** (1901), no. 11, 559–572.

[PG90] Tomaso Poggio and Federico Girosi, *Networks for approximation and learning*, Proceedings of the IEEE **78** (1990), no. 9, 1481–1497.

[PLB10] Federico Montesino Pouzols, Amaury Lendasse, and Angel Barriga Barros, *Autoregressive time series prediction by means of fuzzy inference systems using nonparametric residual variance estimation*, Fuzzy Sets and Systems **161** (2010), no. 4, 471–497, Theme: Forecasting, Classification, and Learning.

[PPS94] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J Sobajic, *Learning and generalization characteristics of the random vector functional-link net*, Neurocomputing **6** (1994), no. 2, 163–180.

[Ras04] CarlEdward Rasmussen, *Gaussian Processes in Machine Learning*, Advanced Lectures on Machine Learning (Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, eds.), Lecture Notes in Computer Science, vol. 3176, Springer Berlin Heidelberg, 2004, pp. 63–71.

[Red98] Thomas C. Redman, *The Impact of Poor Data Quality on the Typical Enterprise*, Communications of the ACM **41** (1998), no. 2, 79–82.

[RM72] C. Radhakrishna Rao and Sujit Kumar Mitra, *Generalized inverse of a matrix and its applications*, Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Theory of Statistics (Berkeley, Calif.), University of California Press, 1972, pp. 601–620.

[Ros58] Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), no. 6, 386–408.

[ROTZ08] Hai-Jun Rong, Yew-Soon Ong, Ah-Hwee Tan, and Zexuan Zhu, *A fast pruned-extreme learning machine for classification problem*, Machine Learning for Signal Processing (MLSP 2006) / Life System Modelling, Simulation, and Bio-inspired Computing (LSMS 2007) **72** (2008), no. 1–3, 359–366.

[RRZ+09] Y Robiah, S Siti Rahayu, M Mohd Zaki, S Shahrin, M A Faizal, and R Marliza, *A New Generic Taxonomy on Hybrid Malware Detection Technique*, International Journal of Computer Science and Information Security **5** (2009), no. 1, 56–61.

[RS00] Sam T. Roweis and Lawrence K. Saul, *Nonlinear Dimensionality Reduction by Locally Linear Embedding*, Science **290** (2000), no. 5500, 2323–2326.

[RW06] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, The MIT Press, 2006.

[Sam69] John W. Sammon, *A Nonlinear Mapping for Data Structure Analysis*, IEEE Transactions on Computers **C-18** (1969), no. 5, 401–409.

[SG10] Abhinav Srivastava and Jonathon Giffin, *Automatic Discovery of Parasitic Malware*, Recent Advances in Intrusion Detection: 13th International Symposium, RAID 2010, Ottawa, Ontario, Canada, September 15-17, 2010. Proceedings (Somesh Jha, Robin Sommer, and Christian Kreibich, eds.), Lecture Notes in Computer Science, vol. 6307, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 97–117.

[SH15] Zhigen Shang and Jianqiang He, *Confidence-weighted extreme learning machine for regression problems*, Neurocomputing **148** (2015), 544–550.

[SLWV03] G. Simon, Amaury Lendasse, V. Wertz, and Michel Verleysen, *Fast Approximation of the Bootstrap for Model Selection*, ESANN'2003 proceedings - European Symposium on Artificial Neural Networks (Bruges (Belgium)), d-side Publications, 23-25 April 2003, pp. 475–480.

[SOJN08] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng, *Cheap and Fast—but is It Good?: Evaluating Non-expert Annotations for Natural Language Tasks*, Proceedings of the conference on empirical methods in natural language processing, EMNLP '08, Association for Computational Linguistics, Stroudsburg, PA, USA, 2008, pp. 254–263.

[SSSM98] Bernhard Schölkopf, Alexander Smola, Er Smola, and Klaus-Robert Müller, *Nonlinear component analysis as a kernel eigenvalue problem*, Neural Computation **10** (1998), no. 5, 1299–1319.

[ST83] David W Scott and James R Thompson, *Probability density estimation in higher dimensions*, Computer Science and Statistics: Proceedings of the Fifteenth Symposium on the Interface, vol. 528, North-Holland, Amsterdam, 1983, pp. 173–179.

[ST05] Timo Similä and Jarkko Tikka, *Multiresponse Sparse Regression with Application to Multidimensional Scaling*, Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and Their Applications - Volume Part II (Berlin, Heidelberg), ICANN'05, Springer-Verlag, 2005, pp. 97–102.

[ST06] T. Simila and J. Tikka, *Common Subset Selection of Inputs in Multiresponse Regression*, Neural Networks, 2006. IJCNN '06. International Joint Conference on, 2006, pp. 1908–1915.

[SZ03] Josef Sivic and Andrew Zisserman, *Video Google: a text retrieval approach to object matching in videos*, Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on, 13-16 Oct. 2003, pp. 1470–1477 vol.2.

[TdL00] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford, *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science **290** (2000), no. 5500, 2319–2323.

[Ten98] Joshua B Tenenbaum, *Mapping a manifold of perceptual observations*, Advances in Neural Information Processing Systems 10 (M. I. Jordan, M. J. Kearns, and S. A. Solla, eds.), MIT Press, 1998, pp. 682–688.

[TFF08] Antonio Torralba, Rob Fergus, and William T Freeman, *80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence **30** (2008), no. 11, 1958–1970.

[Thi76] Ronald A. Thisted, *Ridge regression, minimax estimation, and empirical bayes methods*, Tech. Report 28, Division of Biostatistics, Stanford University, 1976.

[Tib96] Robert Tibshirani, *Regression Shrinkage and Selection Via the Lasso*, Journal of the Royal Statistical Society, Series B (Methodological) **58** (1996), 267–288.

[Tik63] A N Tikhonov, *Solution of incorrectly formulated problems and the regularization method*, Soviet Math. Dokl. **5** (1963), 1035–1038.

[TK07] Grigorios Tsoumakas and Ioannis Katakis, *Multi-Label Classification: An Overview*, International Journal of Data Warehousing and Mining (IJDWM) **3** (2007), no. 3, 1–13.

[TM08] Tinne Tuytelaars and Krystian Mikolajczyk, *Local Invariant Feature Detectors: A Survey*, Foundations and Trends in Computer Graphics and Vision **3** (2008), no. 3, 177–280.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, *Introduction to Data Mining*, 1 ed., Pearson, May 2005.

[VGS10] Koen EA Van De Sande, Theo Gevers, and Cees GM Snoek, *Evaluating color descriptors for object and scene recognition*, IEEE Trans. on Pattern Analysis and Machine Intelligence **32** (2010), no. 9, 1582–1596.

[Vii12] Ville Viitaniemi, *Visual Category Detection: An Experimental Perspective*, Ph.D. thesis, Aalto University, Helsinki, Finland, 2012.

[VJ01] Paul Viola and Michael Jones, *Rapid object detection using a boosted cascade of simple features*, Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on, vol. 1, 2001, pp. I–511–I–518.

[VK01] Jarkko Venna and Samuel Kaski, *Neighborhood Preservation in Nonlinear Projection Methods: An Experimental Study*, Artificial Neural Networks — ICANN 2001: International Conference Vienna, Austria, August 21–25, 2001 Proceedings (Georg Dorffner, Horst Bischof, and Kurt Hornik, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 485–491.

[vML+09] Mark van Heeswijk, Yoan Miche, Tiina Lindh-Knuutila, Peter A. Hilbers, Timo Honkela, Erkki Oja, and Amaury Lendasse, *Adaptive Ensemble Models of Extreme Learning Machines for Time Series Prediction*, Proceedings of the 19th International Conference on Artificial Neural Networks: Part II (Berlin, Heidelberg) (Cesare Alippi, Marios Polycarpou, Christos Panayiotou, and Georgios Ellinas, eds.), ICANN '09, Springer-Verlag, 2009, pp. 305–314.

[vMOL11] Mark van Heeswijk, Yoan Miche, Erkki Oja, and Amaury Lendasse, *GPU-Accelerated and Parallelized ELM Ensembles for Large-scale Regression*, Neurocomputing **74** (2011), no. 16, 2430–2437.

[VPN+10] Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski, *Information retrieval perspective to nonlinear dimensionality reduction for data visualization*, The Journal of Machine Learning Research **11** (2010), 451–490.

[VS05] Andrea Vedaldi and Stefano Soatto, *Features for recognition: viewpoint invariance for non-planar scenes*, Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on, vol. 2, 17-21 Oct. 2005, pp. 1474–1481 Vol. 2.

[vT15] André van Schaik and Jonathan Tapson, *Online and adaptive pseudoinverse solutions for ELM weights*, Advances in neural networksAdvances in Extreme Learning MachinesSelected papers from the Tenth International Symposium on Neural Networks (ISNN 2013)Selected articles from the International Symposium on Extreme Learning Machines (ELM 2013) **149, Part A** (2015), 233–238.

[Wel47] Bernard L. Welch, *The Generalization of 'Student's' Problem when Several Different Population Variances are Involved*, Biometrika **34** (1947), no. 1/2, 28–35.

[WHF07] C. Willems, T. Holz, and F. Freiling, *Toward Automated Dynamic Malware Analysis Using CWSandbox*, IEEE Security & Privacy **5** (March-April 2007), no. 2, 32–39.

[Whi89] Stephen H. White, *An additional hidden unit test for neglected nonlinearity in multilayer feedforward networks*, Neural Networks, 1989. IJCNN., International Joint Conference on, 1989, pp. 451–455 vol.2.

[Whi06] Halbert White, *Chapter 9 Approximate Nonlinear Forecasting Methods*, Handbook of Economic Forecasting (C.W.J. Granger G. Elliott and A. Timmermann, eds.), vol. Volume 1, Elsevier, 2006, pp. 459–512.

[WHY09] Xiaoyin Wang, Changzhen Hu, and Shuping Yao, *An adult image recognizing algorithm based on naked body detection*, Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on, vol. 4, 8-9 Aug. 2009, pp. 197–200.

[WMR92] W.F. Schmidt, M.A. Kraaijveld, and R.P.W. Duin, *Feedforward neural networks with random weights*, Pattern Recognition, 1992. Vol.II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on, 30 Aug-3 Sep 1992, pp. 1–4.

[WSB98] Roger Weber, HJ Schek, and Stephen Blott, *A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces*, Proceedings of the 24rd International Conference on Very Large Data Bases (San Francisco, CA, USA), VLDB '98, Morgan Kaufmann Publishers Inc., 1998, pp. 194–205.

[YHOM10] Katsunari Yoshioka, Yoshihiko Hosobuchi, Tatsunori Orii, and Tsutomu Matsumoto, *Vulnerability in Public Malware Sandbox Analysis Systems*, Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on (Washington, DC, USA), IEEE Computer Society, 19-23 July 2010, pp. 265–268.

[YKL11] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung, *A Survey of Crowdsourcing Systems*, Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on, 9-11 Oct. 2011, pp. 766–773.

[YME+13] Qi Yu, Yoan Miche, Emil Eirola, Mark van Heeswijk, Eric Séverin, and Amaury Lendasse, *Regularized extreme learning machine for regression with missing data*, Advances in Extreme Learning Machines (ELM 2011) **102** (2013), 45–51.

[ZH05] Hui Zou and Trevor Hastie, *Regularization and variable selection via the elastic net*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **67** (2005), no. 2, 301–320.

[ZHC13] Weiwei Zong, Guang-Bin Huang, and Yiqiang Chen, *Weighted extreme learning machine for imbalance learning*, Neurocomputing **101** (2013), 229–242.

[ZOT14] Yiteng Zhai, Yew-Soon Ong, and I. W. Tsang, *The Emerging "Big Dimensionality"*, IEEE Computational Intelligence Magazine **9** (2014), no. 3, 14–26.

[ZQSH05] Qin-Yu Zhu, A. Kai Qin, Ponnuthurai N. Suganthan, and Guang-Bin Huang, *Evolutionary extreme learning machine*, Pattern Recognition **38** (2005), no. 10, 1759–1763.

[ZRY09] Peng Zhao, Guilherme Rocha, and Bin Yu, *The composite absolute penalties family for grouped and hierarchical variable selection*, Annals of Statistics **37** (2009), no. 6A, 3468–3497 (en).

[ZW04] Xingquan Zhu and Xindong Wu, *Class Noise vs. Attribute Noise: A Quantitative Study*, Artificial Intelligence Review **22** (2004), no. 3, 177–210.

[ZWC03] Xingquan Zhu, Xindong Wu, and Qijun Chen, *Eliminating class noise in large datasets*, Proceedings of the 20th International Conference on Machine Learning (ICML-03), 2003, pp. 920–927.

[ZZWW04] Qing-Fang Zheng, Wei Zeng, Gao Wen, and Wei-Qiang Wang, *Shape-based adult images detection*, Image and Graphics (ICIG'04), Third International Conference on, 18-20 Dec. 2004, pp. 150–153.