Review and performance comparison of SVM- and ELM- based classifiers[☆]

Jan Chorowski[a], Jian Wang[b,a,c], Jacek M. Zurada[a]

[a] *Computational Intelligence Laboratory, Department of Electrical and Computer Engineering, University of Louisville, Louisville, KY 40292, USA*
[b] *School of Mathematical Sciences, Dalian University of Technology, Dalian 116024, China*
[c] *College of Science, China University of Petroleum, Qingdao, 266580, China*

Abstract

This paper presents how commonly used machine learning classifiers can be analyzed using a common framework of convex optimization. Four classifier models, the Support Vector Machine (SVM), the Least-Squares SVM (LSSVM), the Extreme Learning Machine (ELM), and the Margin Loss ELM (MLELM) are discussed to demonstrate how specific parametrizations of a general problem statement affect the classifier design and performance, and how ideas from the four different classifiers can be mixed and used together. Furthermore, twenty one public domain benchmark datasets are used to experimentally evaluate five performance metrics of each model and corroborate the theoretical analysis. Comparison of classification accuracies under a nested cross-validation evaluation shows that with on exception all four models perform similarly on the evaluated datasets. However, the four classifiers command different amounts of computational resources for both testing and training. These requirements are directly linked to their formulations as different convex optimization problems.

Keywords: Classifiers, convex quadratic programming, SVM, LSSVM, ELM, MLELM, randomization.

[☆] J.W. and J.C. contributed equally to this paper.

This work was supported by the China Scholarship Council (CSC) and the China Postdoctoral Science Foundation (No. 2012M520624).

Email addresses: `jan.chorowski@louisville.edu` (Jan Chorowski), `wangjiannl@mail.dlut.edu.cn` (Jian Wang), `jacek.zurada@louisville.edu` (Jacek M. Zurada)

1. Introduction

The discovery of Support Vector Machines in 1995 [1, 2] marks the beginning of development of a family of very efficient and unique classifiers. They operate by first mapping input samples into a highly dimensional feature space using a fixed nonlinear transformation. The samples are then classified in this feature space with a linear decision function chosen in a way that maximizes the margin separating samples from different classes. This contribution discusses how similarly operating classifiers can be analyzed in a common framework of convex optimization problems. This is done by portraying classifiers' similarities and by highlighting how the differences in their definition affect the properties of the resulting models.

The SVM embodies many important principles. It solves the problem of classification directly without trying to solve the much harder problem of estimating the distribution of data samples [3]. It provides efficient means of trading the training error for generalization error. Furthermore, even in the nonlinear case, the very central minimization task is stated as a convex optimization problem for which efficient numerical methods of finding the globally optimum solution exist.

The SVM uses two main ideas. First, kernel functions are used to transform the problem from the original input space into a highly dimensional one, called the feature space, where linear separation of training samples belonging to different classes is possible. Second, to find the best separating hyperplane, the concept of maximum margin is introduced. Finally, the optimization problem which defines the SVM is convex and quadratic, and therefore it can be solved efficiently.

We begin by formulating a convex optimization problem general enough to encompass all classifiers of interest [4]. Let there be a training set $\{(\mathbf{x}_i, y_i)\} \subseteq \mathbb{R}^n \times \mathbb{R}$ consisting of N training samples $\mathbf{x}_i$ and corresponding labels y_i. The following analysis is restricted to the case of two classes, which are encoded as $\{-1, +1\}$. In section 2.6 we discuss extensions to multi-class problems. Let $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ denote a nonlinear transformation which projects data samples $\mathbf{x}$ into a $m-$dimensional feature space (m can be infinite). Consider the following optimization problem:

$$\underset{\mathbf{w},b}{\text{minimize}}\ A||\mathbf{w}||_p^p + C\sum_{i=0}^{N} L(\mathbf{w}^T \cdot \Phi(\mathbf{x}_i) + b, y_i) \tag{1}$$

where $\mathbf{w}$ is a vector of tunable weights, b is the bias term, $|| \cdot ||_p$ denotes a convex norm ($p \geq 1$), $L(o, y)$ is a loss function penalizing the classifier output o when the desired class is y, and A and C are nonnegative constants. Assume that $L(\cdot, y)$ is convex in the first argument for all possible y. Then $L(\mathbf{w}^t \cdot \Phi(x_i) + b, y_i)$ is a convex function because convexity is invariant under affine mappings. The problem (1) is then convex because it is a weighted nonnegative sum of convex terms [4]. Furthermore, when the transformation Φ has an explicit formula small instances can be readily solved using convex programming tools, such as CVX [5].

In the special case of the identity feature transformation $\Phi(\mathbf{x}) = \mathbf{x}$, and when the L_2 ($p = 2$) norm is used, the problem can be reduced to one of:

1. regularized least-squares regression (ridge regression) when $L(o, y) = (o - y)^2$,
2. regularized logistic regression when $L(o, y) = \log(1 + \exp(-y \cdot o))$,
3. linear SVM when $L(o, y) = \max(0, 1 - y \cdot o)$ (margin loss or hinge loss).

Those three classifiers differ only by the loss function whose minimum is sought. The three different loss functions computed when $y = 1$ have been compared in Fig. 1 along with the (not convex) misclassification count which is 1 for a misclassified sample and 0 otherwise. The least-squares loss function can be derived by finding a maximum likelihood estimation of weights under the assumption of Gaussian noise, while the logistic regression loss function stems from the assumption of Bernoulli binomial noise [6]. The loss function used in the SVM was introduced as a convex approximation to the number of misclassifications [1]. It can be seen that it is closely linked to the loss function used in the logistic regression. Probabilistic interpretations of outputs of the SVM have been studied in [7, 8]. We have included in Fig. 1 the plot of the loss function $L(o, y) = (\tanh(o) - y)^2$ which commonly arises when an artificial neural network uses the hyperbolic tangent activation function in the output layer. We see that it is not convex and can be interpreted as a smooth scaled approximation of the misclassification count. The addition of the hyperbolic tangent function may seem insignificant, but it has important consequences – the optimization problem no longer has a single global optimum and training algorithms converge only to local optima.

To solve nonlinear problems, a transformation needs to be performed to project the data samples from the input space into a highly dimensional

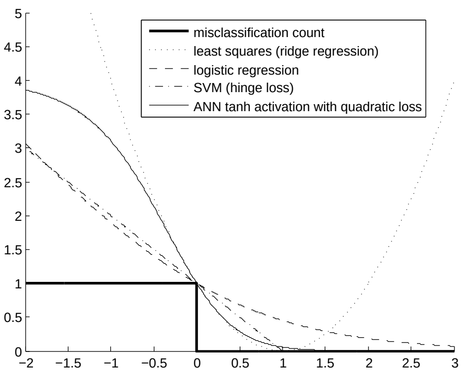


Figure 1: Comparison between loss functions: misclassification count, least squares (ridge regression), logistic regression, SVM (margin loss), and artificial neural network with tanh output activation function and quadratic loss. The curve of logistic regression has been scaled by $1/\ln 2$ such that all curves contain the point $(0, 1)$ (modified after [6]).

feature space in which linear separation of the data is possible. In the case of an explicitly known transformation $\Phi(\mathbf{x})$, the training of the classifier can be performed by directly solving the problem (1). For example, the Extreme Learning [9] approach advocates the use of randomly chosen transformations. In its original formulation, the Extreme Learning Machine (ELM) does not only use random transformations, but also chooses to minimize the least squares loss, for which a closed-form solution given by the Moore-Penrose pseudo-inverse exists to facilitate very fast training.

In many cases it is beneficial to define the transformation Φ indirectly with the use of a kernel function $K(\mathbf{x}_1, \mathbf{x}_2)$ [1, 10–12]. If K satisfies the conditions of Mercer's theorem, then there exist a space in which K defines an inner product operation $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2)$. The kernel function can be used when the optimization problem (1) can be transformed into a form which depends only on the inner products between projections of samples. This is possible when the weights are regularized using the L_2 norm. Without the loss of generality, we can substitute $A = 1/2$ into (1). Furthermore, dummy variables o_i are introduced to represent classifier output:

$$\begin{aligned} &\underset{\mathbf{w},b,\boldsymbol{o}}{\text{minimize}}\, \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N} L(o_i, y_i) \\ &\text{subject to: } o_i = \mathbf{w}^T\Phi(\mathbf{x}_i) + b \end{aligned} \tag{2}$$

When L is differentiable with respect to o_i, (2) can be solved using the method of Lagrange multipliers [12, 13]. Construct the Lagrangian

$$\Lambda = \frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N} L(o_i, y_i) - \sum_{i=1}^{N} \alpha_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b - o_i), \tag{3}$$

where α_i are Lagrange multipliers – there is one for each constraint or, equivalently, one multiplier for each training sample. A stationary point is computed by solving the following system of equations [12, 13]:

$$\nabla_{\mathbf{w}}\Lambda = 0 \Rightarrow \mathbf{w}^T = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)^T \tag{4a}$$

$$\nabla_b\Lambda = 0 \Rightarrow \sum_{i=1}^{N} \alpha_i = 0 \tag{4b}$$

$$\nabla_{\boldsymbol{o}}\Lambda = 0 \Rightarrow C\frac{\partial L(o_i, y_i)}{\partial o_i} = -\alpha_i \ \forall_i \tag{4c}$$

$$\nabla_{\boldsymbol{\alpha}}\Lambda = 0 \Rightarrow \mathbf{w}^T\Phi(\mathbf{x}_i) + b = o_i \ \forall_i. \tag{4d}$$

Furthermore, since the problem is convex the stationary point is the global minimizer of (2).

From (4a) we can see that the weight vector $\mathbf{w}$ can be expressed as a weighted sum of projections of training samples [11, 12]. Furthermore, it can be plugged into the last equation (4d). Substituting the kernel function in place of $\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_j)$ leads to a solution for $\boldsymbol{\alpha}$. The relation $\mathbf{w}^T\Phi(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{x})$ can be used to classify new samples. Observe that unless special care is taken, most of the coefficients α_i can be nonzero and classification of new samples will be computationally expensive. This technique has been used to derive kernelized versions of many commonly used classifiers, that include kernelized logistic regression [14] and Least Squares SVM [15, 16] (LSSVM, or kernelized least squares) classifier.

It is informative to compare four exemplary architectures of classifiers (presented in more detail in Sections 2 and 3) with the intuitive Fig. 2. The figure illustrates classifiers' optimization goals for the objective function as columns, and the feature transformation methods from input space to a highly dimensional feature space as rows. Since the choice of the loss function is independent from the choice of the feature transformation we can form four different classifier architectures.

		Optimization goal	
		Margin Loss	Least Squares
Feature Transformation	Kernel	$\frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N}\max(0, 1 - o_i y_i)$ SVM (1995)	$\frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N}(o_i - y_i)^2$ LSSVM (1999)
	Randomization	$\frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N}\max(0, 1 - o_i y_i)$ MLELM (2010)	$\frac{\mathbf{w}^T\mathbf{w}}{2} + C\sum_{i=1}^{N}(o_i - y_i)^2$ ELM (2006)

where $o_i = \mathbf{w}^T\Phi(\mathbf{x}_i) + b$ and b may be 0 for ELM

Figure 2: Relationship between SVM and ELM algorithms.

2. Algorithms and related basic concepts

This section discusses in more detail how the four categories of classifiers highlighted in Fig. 2 operate.

2.1. Support Vector Machine (SVM)

SVM was first introduced by Vapnik [1, 2, 10, 17] for solving pattern classification and nonlinear function approximation problems [18–20]. For pattern classification, the main paradigm is to find the optimal separating hyperplane as the decision boundary located in such a way that the margin of separation between classes is maximized. The derivation of the margin loss $L(o, y) = \max(0, 1 - o \cdot y)$ used in the SVM and leading to a large margin separating classifier is presented in [1, 6, 12, 21]. For non-linear classification, training patterns are first mapped into a highly dimensional space, for which kernel functions are used.

The optimization problem solved by the SVM [1, 12, 20, 21] is:

$$\underset{\mathbf{w},b}{\text{minimize}}\ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\max(0, 1 - y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)) \tag{5}$$

Because the loss function is not smooth, it is not possible to solve the Lagrangian (3). We can, however, transform (5) into the following equivalent minimization task [1, 12, 20, 21]:

$$\begin{aligned} \underset{\mathbf{w},b,\boldsymbol{\xi}}{\text{minimize}}\ & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ \text{subject to: } & \xi_i \geq 1 - y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)\ \forall_i \\ & \xi_i \geq 0\ \forall_i, \end{aligned} \tag{6}$$

where ξ_i are artificial slack variables representing classifier errors and C is a constant.

We can solve (6) using KKT optimality conditions [13]. Unlike the case of a differentiable loss function which can be reduced to the system of equations (4), the solution now requires maximizing an auxiliary problem, which is also called the dual [1, 12, 20, 21]:

$$\begin{aligned}
&\underset{\boldsymbol{\alpha}}{\text{maximize}} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_j) + \sum_{i=1}^{N}\alpha_i \\
&\text{subject to: } \sum_{i=1}^{N} y_i \alpha_i = 0 \ \forall_i \\
&\qquad\qquad 0 \le \alpha_i \le C \ \forall_i
\end{aligned} \tag{7}$$

The optimum weights are again recovered as a linear combination of Lagrange multipliers α_i: $\mathbf{w}^T = \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)^T$.

An important property of the SVM, linked directly to its non-smooth loss function and the resulting problem formulation with inequality constraints, is that the vector $\boldsymbol{\alpha}$ is sparse [1]. This means that classifying new data using the SVM requires the iteration over a small fraction of the training set. This provides important performance benefits.

2.2. Least Squares Support Vector Machine (LSSVM)

The original solution of the kernelized least squares classifier, called LSSVM has been presented in [15, 16]. The essential difference between SVM and LSSVM is the use of the least squares loss function $L(o, y) = \frac{1}{2}(o - y)^2$. Substituting this loss function into the equality constrained problem formulation (2) defines LSSVM problem as[1]

$$\begin{aligned}
&\underset{\mathbf{w},b,\mathbf{o}}{\text{minimize}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{N}(o_i - y_i)^2 \\
&\text{subject to: } o_i = \mathbf{w}^T\Phi(\mathbf{x}_i) + b
\end{aligned} \tag{8}$$

[1]In [15, 16] the authors defined LSSVM using a slightly different, yet equivalent formulation.

The system of equations obtained by constructing the Lagrangian becomes (compare with (4) and [15, 16]):

$$\begin{aligned} \mathbf{w}^T &= \sum_{i=1}^{N} \alpha_i \Phi(\mathbf{x}_i)^T \\ 0 &= \sum_{i=1}^{N} \alpha_i \\ \alpha_i &= C(y_i - o_i)\ \forall_i \\ o_i &= \mathbf{w}^T \Phi(\mathbf{x}_i) + b\ \forall_i \end{aligned} \tag{9}$$

A closed form solution for the Lagrange multipliers α_i can readily be obtained.

The training of the LSSVM is much simpler than the training of the SVM, which requires solving (5). In case of large-scale problems many efficient iterative methods exist to solve systems of linear equations [22]. Conjugate gradient method, one of the common iterative optimization methods, has been applied in [23] to solve (9). Benchmarking classification tests with UCI Machine Learning Repository data have been reported in 2004 for SVM and LSSVM classifiers, and the results demonstrate consistently good performance of these two methods [24].

However, the main shortcoming of LSSVM is that of the loss of sparseness of the multipliers α_i due to the choice of a smooth loss function. With large majority of multipliers α_i usually nonzero, the LSSVM classifier commands a lot of resources to classify new samples. Several heuristic remediations [25–29] have therefore been proposed to improve their sparseness and robustness. In [25], a weight pruning procedure is used based on removing training samples by the sorted support value spectrum. An improvement method for achieving sparsity in LSSVM is presented in [26] by considering the residuals for all training samples instead of only those incorporated in the sparse kernel expansion.

2.3. Extreme Learning Machine (ELM)

ELM architecture was originally proposed in 2006 as a single hidden layer feedforward network in [9, 30]. It didn't use the regularization term used in (11).

The most important contribution of the ELM is the proposition of using random independent nonlinear feature transformations Φ. An input vector $\mathbf{x}$

is transformed into an M dimensional vector whose jth component is defined by one of:

$$(\Phi_A(\mathbf{x}))_j = \sigma(\boldsymbol{\omega}_j^T \mathbf{x} + \beta_j) \tag{10a}$$

$$(\Phi_M(\mathbf{x}))_j = \sigma(||\boldsymbol{\omega}_j - \mathbf{x}||_2/\beta_j) \tag{10b}$$

where $\sigma(\cdot)$ is a nonlinear function, such as the logistic sigmoid, exponent of the negated square, or the sign function, $\boldsymbol{\omega}_j$ is a vector of i.i.d. random weights sampled from the normal or uniform distribution, and β_j is a randomly sampled bias term. The transformations defined by (10a) are sometimes called random additive nodes, while those defined by (10b) are called multiplicative nodes.

The ELM is similar to an artificial neural network, whose weights and biases in the first layer are randomly initialized and kept constant, while the weights (and optionally biases) of the second layer are selected by minimizing the least squares error. There are two important characteristics of feedforward neural networks [31–39]: interpolation capability and universal approximation capability. The interpolation ability of ELM is rigorously proved in [9] and any N arbitrary distinct samples can be learned precisely with at most N hidden nodes and under the condition that the activation functions of hidden layer are infinitely differentiable in any interval. As an important theoretical contribution, the universal approximation ability of ELM for any continuous target function is presented in detail for the activation functions that are nonconstant, piecewise continuous and the feature space of hidden layer is dense in L^2 [30, 40, 41].

In terms of the training performance and the above theoretical guarantee, ELM and its variants such as fully complex ELM [42–44], online sequential ELM [44–46] and ensemble ELM [46–48] have been extensively studied. Detailed survey for interested readers is in [49] and a comparison of ELM, SVM, LSSVM is in [50]. Regularized extreme learning machine (RELM) is based on structural risk minimization principle and weighted least squares. The RELM has led to a better generalization performance than the original ELM algorithm [51, 52].

The ELM classifier is usually designed without the bias term b, since theoretical analysis shows that it is not needed. Also, the regularization term was first used in [51–53]. The ELM solves the following problem:

$$\underset{\mathbf{w}}{\text{minimize}}\ \mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{N}(\mathbf{w}^T\Phi(\mathbf{x}_i) - y_i)^2 \tag{11}$$

where the transformation Φ is one of (10) with $\boldsymbol{\omega}$ and β sampled randomly. This problem can be readily solved via the Moore-Penrose matrix pseudo-inverse [49, 50]:

$$\mathbf{w} = \left(\frac{\mathbf{I}}{C} + \Phi(\mathbf{X})\Phi(\mathbf{X})^T\right)^{-1} \Phi(\mathbf{X})\mathbf{Y} \tag{12a}$$

$$= \Phi(\mathbf{X}) \left(\frac{\mathbf{I}}{C} + \Phi(\mathbf{X})^T\Phi(\mathbf{X})\right)^{-1} \mathbf{Y}, \tag{12b}$$

where $\mathbf{X}$ is a matrix whose ith column is the ith training sample $\mathbf{x}_i$, $\Phi(\mathbf{X})$ is a matrix whose ith column is $\Phi(\mathbf{x}_i)$, $\mathbf{Y}$ is a vector of class labels, and $\mathbf{I}$ is the identity matrix. The solutions (12a) and (12b) exist when the matrix $\Phi(\mathbf{X})$ is full column rank or full row rank, respectively. They can be readily derived by solving (9) with setting $b = 0$. When one directly solves for the weight vector, the solution (12a) is obtained, while solving first for $\boldsymbol{\alpha}$ and then plugging into the formula for $\mathbf{w}$ yields (12b).

2.4. Margin Loss Extreme Learning Machine (MLELM)

Perhaps the most striking property of the ELM method is that the randomly chosen input transformations are sufficient to approximate any function [30] and moreover yield good classification accuracy. One can consider to use a kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2)$ in which the transformation $\Phi(\mathbf{x})$ is given by (10) and is generated prior to classifier training. Using this ELM-inspired kernel for SVM training was analyzed in [54, 55]. We observe that since the formula for the feature transformation is known, training can be accomplished either by supplying transformed inputs to a linear SVM solver, or by setting the kernel $K(\mathbf{x}_1, \mathbf{x}_2) = \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2)$ in a nonlinear one.

One can consider the case of using infinitely many randomly chosen hidden neurons. Under a Gaussian prior on weights, it is possible to obtain an analytical formula of the ELM kernel. We direct the interested reader to [56–58].

2.5. L_1 weight regularization

When the transformation $\Phi(\mathbf{x})$ is explicitly known, as it is the case with e.g. the ELM method, one may wish to minimize the L_1 norm of the weight vector $\mathbf{w}$. This enforces sparsity of the weights and can be used for feature selection as in the LASSO method [59]. In fact, the LASSO method has been used to minimize the number of hidden neurons used by the ELM [60].

In [61] an extension of LASSO is used to rank hidden neurons of an ELM for multivariate regression.

Finally, when $\Phi(\mathbf{x})$ is known and one wishes to minimize the L_1 norm on weights and use the margin loss penalty, then the 1-norm Support Vector Machine is obtained [62–64]. The optimization problem (1) can be solved efficiently using linear programming. Substituting the L_1 norm and margin loss function yields the problem:

$$\underset{\mathbf{w},b}{\text{minimize}} \; ||\mathbf{w}||_1 + C\sum_{i=1}^{N} \max(0, 1 - y_i(\mathbf{w}^T\Phi(\mathbf{x}_i) + b)). \tag{13}$$

Introducing slack variables ξ_i and decomposing $\mathbf{w}$ into its positive and negative parts $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$ results in:

$$\begin{aligned} \underset{\mathbf{w}^+,\mathbf{w}^-,b,\boldsymbol{\xi}}{\text{minimize}} \; & \sum_{i=1}^{N}(\mathbf{w}_i^+ + \mathbf{w}_i^-) + C\sum_{i=1}^{N}\xi_i \\ \text{subject to: } & \xi_i \geq y_i((\mathbf{w}^+ - \mathbf{w}^-)^T\Phi(\mathbf{x}_i) + b) - 1 \\ & \xi_i \geq 0 \\ & \mathbf{w}^+, \mathbf{w}^- \geq 0 \end{aligned} \tag{14}$$

which is indeed a linear programming problem. An algorithm recovering the full solution path obtained while varying the parameter C is presented in [62].

2.6. Multi-class operation

The presented classifiers are essentially binary ones. Multi-class operation is possible by training many binary classifiers. Suppose there are K classes. One possibility is to train K one-versus-rest classifiers, each distinguishing between class k and all remaining classes. This scheme was proposed for the ELM as "multioutput multiclass" [50]. Usually, the class with the highest classifier output is selected. Another option is to train $K(K-1)/2$ one-versus-one classifiers, which is the default in [65]. A voting scheme is the used to select the class of new samples. A full discussion of this topic is outside of the scope of this paper and we refer to reviews [66, 67].

3. Relationship between the discussed classifiers

All four above described classifier models share many common properties. Considering the optimization goal, they all belong to the same mathematical category of convex quadratic programming, which guarantees that the training will converge to a single global optimum. Furthermore, they all consist of a chosen a priori and fixed nonlinear transformation followed by a trained linear classifier. As demonstrated by the experiments presented in the next section, the accuracies of different classifier models in this family tested on popular datasets from the UCI repository are comparable.

Despite these similarities, the classifier models differ in many important ways with respect to their distinct inspirations. We summarize those differences and their consequences in the next sections.

3.1. Differences Due to Loss Function

The quadratic loss function adopted in LSSVM and ELM may seem to be unsuitable for classification because it not only penalizes wrong answers, but also penalizes correct answers which are far from the decision boundary, which can be seen in Fig. 1. In contrast, the margin loss used in SVM and MLELM penalizes only answers that are incorrect or that are correct but lie close to the decision boundary. However, the results of the experiments do not indicate that classification accuracy is much influenced by the adopted loss function.

The choice of the loss function determines how the classifier can be trained and how expensive is the classification of new data. The quadratic loss function is smooth and the resulting KKT system has a closed form solution [15, 55]. Thus the LSSVM and ELM can be easily trained. However, all Lagrange multipliers are nonzero and the decision boundary depends on all training samples. On the other hand, the margin loss function adopted in SVM and MLELM is not smooth and the resulting KKT dual system needs to be solved in an iterative way by maximizing an auxiliary problem. Due to the singularity of the loss function only a fraction of the Lagrange multipliers is nonzero [1]. The decision boundary depends on the training samples (support vectors) corresponding to the nonzero multipliers.

3.2. Differences Due to the Feature Transformation

During nonlinear operation, SVM and LSSVM use a feature transformation given implicitly by the kernel function [1, 15]. Therefore, they are always trained in the dual space and the decision boundary is given by the Lagrange multipliers. In contrast, ELM classifiers randomly choose a feature transformation according to (10) [32]. Since the transformation is known, ELM-based classifiers can be trained using both the primal and dual formulation. Furthermore, the weights determining the decision boundary can be computed and no training samples must be stored to classify new data.

3.3. Meta-parameters Used by the Classifiers

While all classifiers discussed in this paper are defined by a convex optimization problem, their operation requires the selection of a number of meta-parameters.For linear operation, the SVM and LSSVM require the selection of the cost parameter C which balances the penalty for errors and the regularization terms in (1) [1]. For nonlinear operation, a kernel function needs to be chosen, possibly requiring the selection of a parameter [12]. Popular choices are the Gaussian kernel $K(x,y) = \exp\left(-||x-y||^2/(2\gamma^2)\right)$ which requires the parameter γ or the polynomial kernel $K(x,y) = (x \cdot y + 1)^p$ which requires the parameter p.

ELM-based classifiers also require the selection of the regularization parameter C. Furthermore, parameters of the feature transformations (10) must be specified. These are the number of hidden units which determines the dimensionality of the transformation, the nonlinear activation function σ, and the probability distribution of the weights and biases of the random hidden neurons. However, the original introduction of the ELM method suggested that the only important parameter is the number of hidden units [30]. The regularization constant C was originally not used. The probability distributions of the weights and biases of the hidden units were only required to match mild requirements of the universal approximation theorems [30] and were not specified. The official ELM toolkit always uses $\omega_{ij} \sim Uniform(-1,-1)$, $\beta_j \sim Uniform(0,1)$ [68]. The regularization constant was added subsequently in [49] and there is evidence that variance of the chosen weight distribution affects the generalization performance [57].

3.4. Discussion of Training and Testing Times

The ELM and LSSVM have a closed-form solution that can be used to compute the weights and it is easy to determine their training times [15, 55]. In contrast, training of the SVM and MLELM is iterative and the determination of the training times is more difficult. We will use N to denote the number of training samples, n to denote the dimensionality of the input data, and m to denote the dimensionality of the feature space, which corresponds to the number of hidden neurons in ELM training.

To ELM requires the computations of the matrix $\Phi(\boldsymbol{X})$ which takes $O(Nnm)$ operations. The weights $\mathbf{w}$ are then computed using the formula (12a) which requires $O(Nm^2 + m^3)$ operations or (12b) which requires $O(N^2m + N^3)$ operations [50]. When the number of training samples is much greater than the number of hidden neurons and the dimensionality of the input, training an ELM requires $O(Nm^2)$ operations and $O(N(m + n))$ memory if formula (12a) is used. A sample can be classified using $O(mn)$ operations.

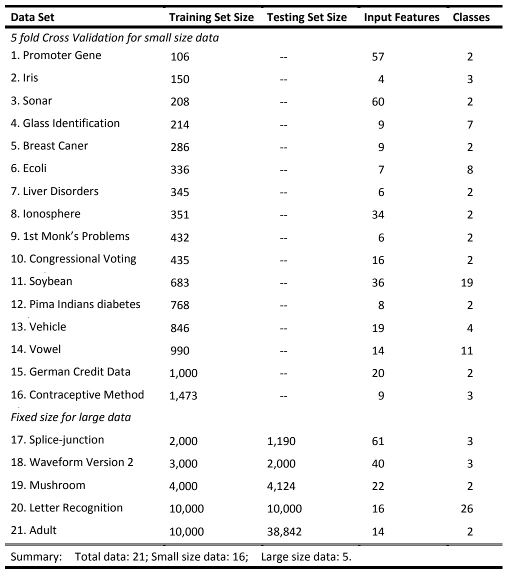
An implementation of the nonlinear LSSVM that solves the system of equations (9) by matrix inversion requires to first compute the kernel matrix in $O(N^2n)$ operations when the Gaussian or polynomial kernels are used. The inversion of the kernel matrix requires $O(N^3)$ operations using $O(N^2)$ memory. More efficient algorithms have been experimentally found to scale quadratically with the training set size [69]. Since the majority of Lagrange multipliers $\boldsymbol{\alpha}$ is non-zero [15], classification of a sample requires the iteration over the whole training set which takes $O(Nn)$ operations.

Due to the iterative nature of SVM training, determining the required number of operations is difficult. In general, training time of the SVM depends on the the number of support vectors N_S [10]. It has been found experimentally that the SMO algorithm scales between linearly and quadratically with the number of training examples [70]. Classification of new samples requires $O(N_Sn)$ operations [10] since we need to only account for the nonzero Lagrange multipliers which correspond to support vectors.

MLELM can be trained in either the primal or dual form. To be compatible with most solvers [13] the non-differentiable loss function needs to be transformed into a differentiable equivalent form in the primal problem formulation. This requires the introduction of artificial variables, similar to the transformation of the L_1-norm SVM as shown in (13) and (14) [62]. In the dual form any solver used to train the SVM can be used. We have experimentally observed, however, that the MLELM commands much more computations than an SVM to train, even for the same solver used.

4. Experimental analysis

The four discussed classifier models have been compared using 21 classification datasets, which include 11 binary classification and 10 multi-class classification problems. Due to the differences of scale, the datasets have been split into two categories based on data volumes as in Fig. 3: small size



Data Set	Training Set Size	Testing Set Size	Input Features	Classes
5 fold Cross Validation for small size data				
1. Promoter Gene	106	--	57	2
2. Iris	150	--	4	3
3. Sonar	208	--	60	2
4. Glass Identification	214	--	9	7
5. Breast Caner	286	--	9	2
6. Ecoli	336	--	7	8
7. Liver Disorders	345	--	6	2
8. Ionosphere	351	--	34	2
9. 1st Monk's Problems	432	--	6	2
10. Congressional Voting	435	--	16	2
11. Soybean	683	--	36	19
12. Pima Indians diabetes	768	--	8	2
13. Vehicle	846	--	19	4
14. Vowel	990	--	14	11
15. German Credit Data	1,000	--	20	2
16. Contraceptive Method	1,473	--	9	3
Fixed size for large data				
17. Splice-junction	2,000	1,190	61	3
18. Waveform Version 2	3,000	2,000	40	3
19. Mushroom	4,000	4,124	22	2
20. Letter Recognition	10,000	10,000	16	26
21. Adult	10,000	38,842	14	2
Summary: Total data: 21; Small size data: 16; Large size data: 5.				

The training set size was reduced to 5000 samples for MLELM on the "Adult" and "Letter" due to its long running times.

Figure 3: Data set summary for classification performance comparison.

(No. $1-16$) and large size (No. $17-21$). For small size datasets, 5-fold cross-validation has been performed, i.e., each dataset is randomly split into 5 subsets, then each subset is used as test sets for a classifier built on the remaining four. The large size datasets have been split into fixed size training set and test set separately as indicated in Fig. 3. For all the classifiers we have normalized all input variables to the $(-1, 1)$ range. Furthermore, we have used the 1-of-N encoding of nominal inputs, i.e. each nominal input was expanded into N dummy inputs, one for each possible value, with only one input equal to 1. In some datasets a few nominal attributes contained missing values. We encoded them as vectors of all zeros. For datasets with many target labels, we have used the one-vs-one strategy for the SVM, and one-vs-all for the other classifiers. In all the cases a 5-fold cross-validation was performed to select meta-parameters on each training set. Furthermore, with the exception of MLELM on the "Letter" dataset, each experiment was repeated three times. All benchmark datasets are from the UCI Machine Learning Repository [71].

To compare the computational performance of different classifier models, SVM results have been obtained by directly using the LIBSVM library [65];

the current version of ELM [68] has been employed for the performance analysis. For LSSVM classifier, the classic algorithm [15, 16] has been rewritten to verify its performance in terms of different parameters which are consistent with other classifiers. We have implemented the $O(N^3)$ matrix inversion algorithm because it is the only algorithm implemented in the LS-SVMlab 1.8 official toolbox [72]. To implement the MLELM we have experimented with different quadratic programming solvers working both in primal and dual space. Considered solvers included the Matlab's Optimization Toolbox, LibSVM configured to use a precomputed kernel matrix, CVX [5, 73], and SVM and Kernel Methods Matlab Toolbox [74], which experimentally proved to be the fastest. The randomization of the weights of hidden layer is the same as in the ELM implementation [68]. The detailed simulation parameters for the experiments are specified in the Fig. 4.

All simulations have been carried out in MATLAB 2010 environment running in an Intel Core i4, 2.67GHZ CPU. The samples of each training dataset are first randomized and then fixed for training of the four classifiers. To analyze numerical results, four performance metrics have been graphed: training accuracy, testing accuracy, training time and testing time. Three learning session trials were executed for each experiment. The average performance across three trials was gathered in Table 1 and plotted in Fig. 5 and Fig. 6. The confidence interval based on testing accuracy has also been computed using the methodology outlined in [12]. As indicated in Eqs. (15)-(18), accuracies for small datasets represent the average accuracy rate on the training sets with 5-fold cross validation averaged over three runs, while accuracies for large datasets are for the fixed training/testing set split. Training times incorporate 5-fold cross-validation parameter tuning. Testing time is the running time on the test sets after the completed training procedure. The definitions of the above metrics are specified as following:

$$\text{trAcc\_CV} = \frac{1}{5}\sum_{i=1}^{5}\left(1 - \frac{e_i}{N_{tr\_CV}}\right), \tag{15}$$

$$\text{teAcc\_CV} = \frac{1}{5}\sum_{1}^{5}\left(1 - \frac{\varepsilon_i}{N_{te\_CV}}\right), \tag{16}$$

$$\text{trAcc\_F} = 1 - \frac{e}{N_{tr\_F}}, \tag{17}$$

$$\text{teAcc\_F} = 1 - \frac{\varepsilon}{N_{te\_F}}, \tag{18}$$

Objective Function	$\frac{1}{2}\|\|\mathbf{w}\|\|_2^2 + C\sum_{i=0}^{N}\max(0, 1 - o_i y_i)$
Kernel Function	Gaussian
Reg. constant C	$2^{-5}, 2^{-2}, \ldots 2^{16}$
Kernel Param. γ	$2^{-15}, 2^{-12}, \ldots 2^{3}$

(a) SVM

Objective Function	$\frac{1}{2}\|\|\mathbf{w}\|\|_2^2 + C\sum_{i=0}^{N}(o_i - y_i)^2$
Kernel Function	Gaussian
Reg. constant C	$2^{-5}, 2^{-2}, \ldots 2^{16}$
Kernel Param. γ	$2^{-15}, 2^{-12}, \ldots 2^{3}$

(b) LSSVM

Objective Function	$\frac{1}{2}\|\|\mathbf{w}\|\|_2^2 + C\sum_{i=0}^{N}(o_i - y_i)^2$
Activation Function	Sigmoid
Hidden weights and biases distribution	weights: $\mathcal{U}(-1, 1)$ biases: $\mathcal{U}(0, 1)$
Reg. Constant C	$2^{-5}, 2^{-2}, \ldots 2^{16}$
Number of Hidden Neurons m	10, 30, 100, 300, 1000, 2000, 5000

(c) ELM

Objective Function	$\frac{1}{2}\|\|\mathbf{w}\|\|_2^2 + C\sum_{i=0}^{N}\max(0, 1 - o_i y_i)$
Activation Function	Sigmoid
Hidden weights and biases distribution	weights: $\mathcal{U}(-1, 1)$ biases: $\mathcal{U}(0, 1)$
Reg. constant C	$2^{-5}, 2^{-2}, \ldots 2^{16}$ or $2^{-5}, 2^{-1}, \ldots 2^{15}$ on "Letter" and "Adult"
Number of Hidden Neurons	10, 30, 100, 300, 1000, 2000, 5000 or 10, 30, 100, 300, 1000, 2000 on "Letter" and "Adult"

(d) MLELM

where $o_i = \mathbf{w}^T\Phi(\mathbf{x}_i) + b$ and b may be 0 for ELM

Figure 4: Experiment designs for classifiers with detailed parameters.

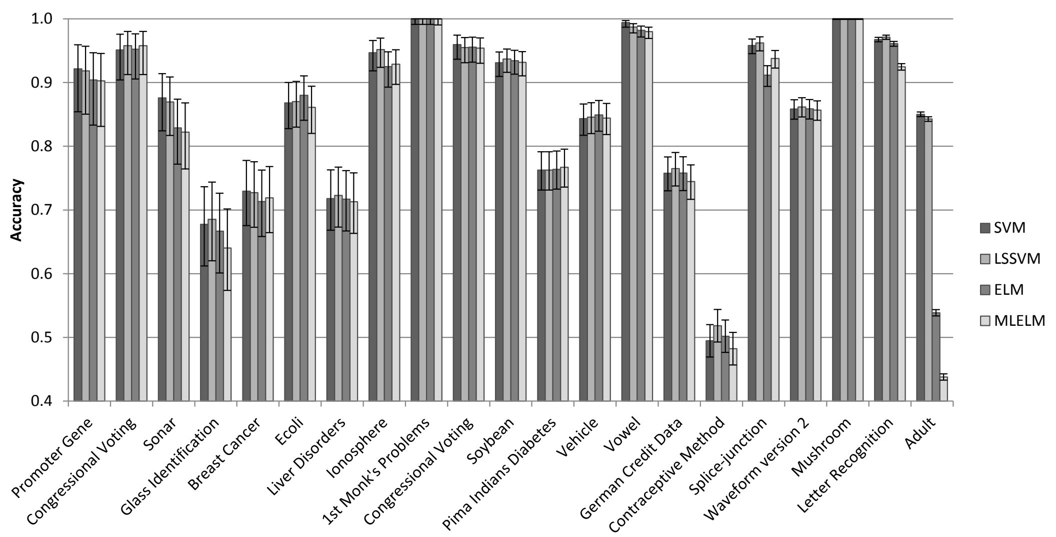


Figure 5: Testing Accuracy for different classifiers with indicated 95% confidence intervals.

where trAcc_CV and teAcc_CV represent the training and testing accuracy with 5 fold cross-validation tests for small size datasets separately, while trAcc_F and teAcc_F stand for the training and testing accuracy with the fixed training and test samples for large size datasets as in Fig. 3, N_{tr_CV} and N_{te_CV} are the number of training and test samples of each of the 5-fold cross validation runs, e_i and ε_i $(i = 1, \cdots, 5)$ represent the number of misclassifications in the ith cross validation run on N_{tr_CV} and N_{te_CV}, respectively. Similarly, N_{tr_F} and N_{te_F} mean the number of the fixed training and test samples for large size datasets, then e and ε stand for the number of misclassifications on N_{tr_F} and N_{te_F}, respectively.

From Table 1 and Fig. 5, we can conclude that each of the four different classifiers has a very similar accuracy performance for most classification problems. The biggest difference is seen on the "Adult" dataset on which the ELM-based classifiers fail to choose proper parameters under the employed cross-validation framework. We have established that the maximum cross-validation accuracy on the training set was usually obtained for 2000 hidden neurons and $C = 0.03$, while the best testing accuracy requires fewer number of hidden neuron (limiting the search to 1000 gave better result). Other differences are seen on datasets "Glass" and "Sonar". The similar performance of SVM and LSSVM shows that on the benchmark problems that we used

Table 1: Average performance of the tested classifiers.

	SVM				LSSVM				ELM				MLELM			
	Train Acc.	Test Acc.	Train Time [s]	Test Time [s]	Train Acc.	Test Acc.	Train Time [s]	Test Time [s]	Train Acc.	Test Acc.	Train Time [s]	Test Time [s]	Train Acc.	Test Acc.	Train Time [s]	Test Time [s]
Promoter Gene	1.00	0.92	6.05	0.01	1.00	0.92	4.91	0.02	1.00	0.90	41.40	0.03	1.00	0.90	77.54	0.03
Iris	0.98	0.95	1.84	0.01	0.98	0.96	3.26	0.01	0.97	0.95	16.45	0.00	0.98	0.96	107.07	0.00
Sonar	1.00	0.88	7.45	0.01	1.00	0.87	6.63	0.02	1.00	0.83	32.49	0.02	0.99	0.82	105.80	0.02
Glass Identification	0.89	0.68	7.51	0.00	0.90	0.69	5.02	0.02	0.86	0.67	24.57	0.01	0.87	0.64	359.21	0.01
Breast Cancer	0.86	0.73	15.57	0.01	0.82	0.73	12.73	0.04	0.79	0.71	39.21	0.01	0.80	0.72	318.78	0.01
Ecoli	0.90	0.87	11.11	0.01	0.90	0.87	9.09	0.02	0.90	0.88	37.63	0.01	0.90	0.86	464.37	0.01
Liver Disorders	0.77	0.72	61.54	0.01	0.77	0.72	9.48	0.02	0.77	0.72	36.48	0.01	0.76	0.71	248.41	0.01
Ionosphere	0.98	0.95	12.51	0.01	0.99	0.95	13.85	0.04	0.99	0.92	39.90	0.03	0.98	0.93	117.60	0.02
1st Monk problems	1.00	1.00	212.49	0.01	1.00	1.00	17.62	0.04	1.00	1.00	56.02	0.02	1.00	1.00	280.45	0.02
Congressional Voting	0.98	0.96	11.52	0.01	0.99	0.95	23.09	0.06	0.97	0.96	56.41	0.05	0.98	0.95	119.44	0.01
Soybean	0.98	0.93	179.97	0.11	0.98	0.94	119.03	0.23	1.00	0.93	113.30	0.07	0.99	0.93	2274	0.04
Pima Indians Diabetes	0.79	0.76	244.70	0.02	0.80	0.76	52.01	0.07	0.79	0.76	102.98	0.03	0.78	0.77	904.25	0.02
Vehicle	0.95	0.84	95.15	0.03	0.95	0.85	67.32	0.13	0.96	0.85	116.74	0.06	0.97	0.84	1807	0.07
Vovel	1.00	0.99	131.60	0.08	1.00	0.99	100.04	0.19	1.00	0.98	146.32	0.16	1.00	0.98	7406	0.10
German Credit Data	0.85	0.76	199.44	0.09	0.89	0.77	293.49	0.39	0.91	0.76	141.73	0.04	0.89	0.74	2337	0.06
Contraceptive Method	0.62	0.49	1089	0.15	0.61	0.52	251.94	0.38	0.60	0.50	266.37	0.05	0.61	0.48	37108	0.04
Splice-junction	0.98	0.96	17063	16.23	1.00	0.96	29996	153.88	1.00	0.91	1007	3.05	1.00	0.94	13254	6.25
Waveform version 2	0.88	0.86	294.90	0.46	0.89	0.86	476.91	2.93	0.92	0.86	227.88	0.36	0.89	0.86	13151	0.06
Mushroom	1.00	1.00	1112	2.57	1.00	1.00	3588	32.83	1.00	1.00	407.66	0.25	1.00	1.00	3792	0.10
Letter Recognition	1.00	0.97	2543	7.58	1.00	0.97	8109	57.46	0.99	0.96	1826	5.20	1.00	0.92	132636	3.12
Adult	0.86	0.85	12823	45.29	0.87	0.84	23470	766.19	0.86	0.54	1894	2.02	0.85	0.44	152793	1.17

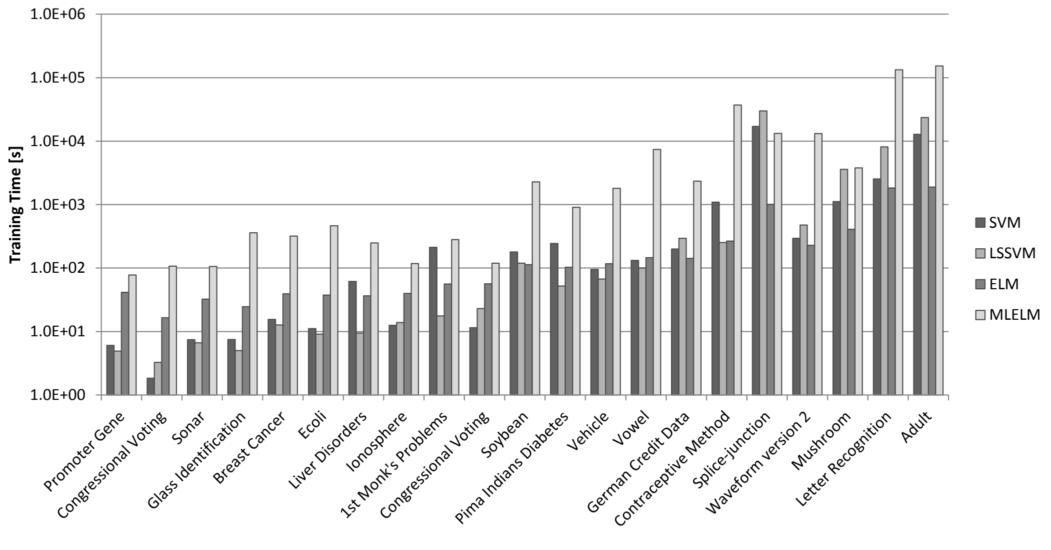


Figure 6: Training Time for different classifiers.

the choice of the loss function doesn't have much impact on classification accuracy. Also, with the exception of the "Adult" data, ELM-based methods demonstrate that the random choice of the hidden layer results in similar performance to SVM-based methods.

We have included the time necessary for tuning parameters with the grid search into the reported training times. While all the classifiers solve optimization problems with unique global optima, they are non-convex with regard to their parameters that we discuss in section 3.3. Tuning those parameters is often necessary for good performance and we believe that the time consumed by this search should be included into the total training time.

Comparing the training times on Fig. 6, it can be seen that MLELM is a significantly more time-consuming algorithm than the remaining ones. All of the QP solvers tested yielded similar unsatisfactory running times. This may indicate that the optimization problem is inherently hard due to correlations between hidden neuron activations obtained from the ELM random projection of training samples. Running times on small datasets show that LibSVM is the fastest method. Our analysis in section 3.4 has shown that the ELM requires a time proportional to the number of hidden neurons, which was larger that the size of small datasets. Also, the highly tuned solver used in LibSVM may have a lower overhead. Running times on larger datasets for which the number of training samples is larger than the maximum number of ELM hidden nodes shows that the ELM is the most computationally efficient of the tested classifiers. The SVM is the second fastest, which corresponds with experimental results of the performance of the SMO solver.

Inspection of the testing time shown in Table 1 indicates that LSSVM is the most time-consuming method to test for all datasets. This is consistent with the theoretical analysis of its computational complexity and demonstrates that the sparseness of the support vectors is lost due to the choice of L_2 norm in the objective function of LSSVM. Similar as with training time analysis, SVM's testing time varies from data to data However, it is generally more time-consuming than ELM and MLELM, whose evaluations require only a feedforward pass through the network.

Fig. 5 shows the confidence intervals for testing accuracy at a 95% confidence level. We noticed that the confidence intervals on the same dataset are generally identical due to the similar accuracies for the four different classifiers. Another characteristic is that the confidence intervals for large size datasets have been typically much tighter than those for small size datasets. The main reason is that the confidence interval is dependent on the number of test samples besides the testing accuracy. Finally, it can be seen that there are several low testing accuracies such as for cases "Breast Cancer", "Ecoli" and "Ionosphere" for small datasets. This indicates that the performance of the classifiers really depends on the nonlinear degree of datasets themselves rather than the number of samples.

Conclusions

This paper discusses and compares four closely related classification models based on margin loss and least squares objective functions. A uniform framework of convex optimization is introduced to highlight the similarities and differences between the models. Implications of different parametrizations of the convex problem (1) are stated and justified in the experimental section.

But for the "Adult" dataset all four investigated models offer comparable classification accuracies. The choice of the one to use is, therefore, problem dependent. The classical ELM algorithm yields usually good testing accuracy while being fast to train and fast to apply to new data. However, the results on the "Adult" dataset demonstrate that the parameters of the ELM must be carefully chosen and validated. The MLELM fusion of SVM's loss function and ELM's feature transformation performs poorly, because it is often unacceptably slow to train and yields similar accuracy to the classical ELM algorithm. The SVM offers state-of-the art accuracy, it is however more expensive than the ELM to both train and apply to new data. The LSSVM often matches or surpasses SVM's accuracy, an has a simpler than the SVM training algorithm. However, its application to new data requires processing of the whole training set, which can be unacceptable for certain applications.

Acknowledgments

The authors wish to thank the anonymous reviewers for many insightful comments and suggestions which greatly improved this work.

The authors thank Dr. Tolga Ensari for his advice in preparation of this manuscript.

[1] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297.

[2] V. Vapnik, The nature of statistical learning theory, Springer-Verlag New York Inc, 1995.

[3] V. Cherkassky, F. Mulier, Learning from data: concepts, theory, and methods, Wiley-IEEE Press, 2007.

[4] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge Univ Pr, 2004.

[5] M. Grant, S. Boyd, Cvx: Matlab software for disciplined convex programming, version 1.21 (2011).
URL `http://cvxr.com/cvx`

[6] C. Bishop, Pattern recognition and machine learning, springer New York, 2006.

[7] Y. Grandvalet, J. Mariéthoz, S. Bengio, A probabilistic interpretation of svms with an application to unbalanced classification, in: Advances in Neural Information Processing Systems, Vol. 18, 2006, pp. 467–474.

[8] P. Sollich, Probabilistic methods for support vector machines, Advances in neural information processing systems 12 (2000) 349–355.

[9] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.

[10] B. Boser, I. Guyon, V. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144–152.

[11] G. Wahba, et al., Support vector machines, reproducing kernel hilbert spaces and the randomized gacv, Advances in Kernel Methods-Support Vector Learning 6 (1999) 69–87.

[12] P. Tan, M. Steinbach, V. Kumar, et al., Introduction to data mining, Pearson Addison Wesley Boston, 2006.

[13] J. Nocedal, S. Wright, Numerical optimization, Springer verlag, 1999.

[14] J. Zhu, T. Hastie, Kernel logistic regression and the import vector machine, Journal of Computational and Graphical Statistics 14 (1) (2005) 185–205.

[15] J. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural processing letters 9 (3) (1999) 293–300.

[16] J. Suykens, J. Vandewalle, Multiclass least squares support vector machines, in: Neural Networks, 1999. IJCNN'99. International Joint Conference on, Vol. 2, IEEE, 1999, pp. 900–903.

[17] V. Vapnik, Statistical learning theory, Wiley, New York, 1998.

[18] A. Smola, B. Schölkopf, K. Müller, The connection between regularization operators and support vector kernels, Neural Networks 11 (4) (1998) 637–649.

[19] B. Schölkopf, C. Burges, A. E. Smola, Advances in kernel methods: support vector learning, The MIT press, 1999.

[20] N. Cristianini, J. Shawe-Taylor, An introduction to support Vector Machines: and other kernel-based learning methods, Cambridge Univ Pr, 2000.

[21] C. Burges, A tutorial on support vector machines for pattern recognition, Data mining and knowledge discovery 2 (2) (1998) 121–167.

[22] G. Golub, C. Van Loan, Matrix computations, Vol. 3, Johns Hopkins Univ Pr, 1996.

[23] J. Suykens, L. Lukas, P. V. Dooren, B. D. Moor, J. Vandewalle, Least squares support vector machine classifiers: a large scale algorithm, in: European Conference on Circuit Theory and Design, 1999, 1999, pp. 839–842.

[24] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, J. Vandewalle, Benchmarking least squares support vector machine classifiers, Machine Learning 54 (1) (2004) 5–32.

[25] J. Suykens, J. De Brabanter, L. Lukas, J. Vandewalle, Weighted least squares support vector machines: robustness and sparse approximation, Neurocomputing 48 (1-4) (2002) 85–105.

[26] G. Cawley, N. Talbot, Improved sparse least-squares support vector machines, Neurocomputing 48 (1-4) (2002) 1025–1031.

[27] L. Wei, Z. Chen, J. Li, W. Xu, Sparse and robust least squares support vector machine: a linear programming formulation, in: Grey Systems and Intelligent Services, 2007. GSIS 2007. IEEE International Conference on, IEEE, 2007, pp. 1134–1138.

[28] J. Liu, J. Li, W. Xu, Y. Shi, A weighted lq adaptive least squares support vector machine classifiers-robust and sparse approximation, Expert Systems with Applications 38 (3) (2011) 2253–2259.

[29] L. Wei, Z. Chen, J. Li, Evolution strategies based adaptive lp ls-svm, Information Sciences 181 (2011) 3000–3016.

[30] G. Huang, L. Chen, C. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, Neural Networks, IEEE Transactions on 17 (4) (2006) 879–892.

[31] S. O. Haykin, Neural Networks: A Comprehensive Foundation, 2E, Prentice Hall, 1999.

[32] G. Huang, H. Babri, Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions, Neural Networks, IEEE Transactions on 9 (1) (1998) 224–229.

[33] M. Sartori, P. Antsaklis, A simple method to derive bounds on the size and to train multilayer neural networks, Neural Networks, IEEE Transactions on 2 (4) (1991) 467–471.

[34] S. Huang, Y. Huang, Bounds on the number of hidden neurons in multilayer perceptrons, Neural Networks, IEEE Transactions on 2 (1) (1991) 47–55.

[35] M. Leshno, V. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural networks 6 (6) (1993) 861–867.

[36] B. Gao, Y. Xu, Univariant approximation by superpositions of a sigmoidal function, Journal of mathematical analysis and applications 178 (1) (1993) 221–226.

[37] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks 4 (2) (1991) 251–257.

[38] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.

[39] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural networks 2 (3) (1989) 183–192.

[40] G. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16-18) (2007) 3056–3062.

[41] G. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16) (2008) 3460–3468.

[42] M. Li, G. Huang, P. Saratchandran, N. Sundararajan, Fully complex extreme learning machine, Neurocomputing 68 (2005) 306–314.

[43] G. Huang, M. Li, L. Chen, C. Siew, Incremental extreme learning machine with fully complex hidden nodes, Neurocomputing 71 (4) (2008) 576–583.

[44] H. Rong, G. Huang, N. Sundararajan, P. Saratchandran, Online sequential fuzzy extreme learning machine for function approximation and classification problems, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 39 (4) (2009) 1067–1072.

[45] N. Liang, G. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, Neural Networks, IEEE Transactions on 17 (6) (2006) 1411–1423.

[46] Y. Lan, Y. Soh, G. Huang, Ensemble of online sequential extreme learning machine, Neurocomputing 72 (13-15) (2009) 3391–3395.

[47] Z. Sun, T. Choi, K. Au, Y. Yu, Sales forecasting using extreme learning machine with applications in fashion retailing, Decision Support Systems 46 (1) (2008) 411–419.

[48] M. Van Heeswijk, Y. Miche, T. Lindh-Knuutila, P. Hilbers, T. Honkela, E. Oja, A. Lendasse, Adaptive ensemble models of extreme learning machines for time series prediction, Lecture Notes in Computer Science 5769 (2009) 305–314.

[49] G. Huang, D. Wang, Y. Lan, Extreme learning machines: a survey, International Journal of Machine Learning and Cybernetics 2 (2011) 107–122.

[50] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 42 (2) (2012) 513 –529.

[51] W. Deng, Q. Zheng, L. Chen, Regularized extreme learning machine, in: Computational Intelligence and Data Mining, 2009. CIDM'09. IEEE Symposium on, IEEE, 2009, pp. 389–395.

[52] W. Deng, L. Chen, Color image watermarking using regularized extreme learning machine, Neural Network World 20 (3, Sp. Iss.{SI}) (2010) 317–330.

[53] Q. Liu, Q. He, Z. Shi, Extreme support vector machine classifier, in: Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining, Springer-Verlag, 2008, pp. 222–233.

[54] B. Frénay, M. Verleysen, Using svms with randomised feature spaces: an extreme learning approach, in: Proceedings of the 18th European symposium on artificial neural networks (ESANN), Bruges, Belgium, Vol. 28, 2010, p. 30.

[55] G. Huang, X. Ding, H. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (1) (2010) 155–163.

[56] C. Williams, Computation with infinite neural networks, Neural Computation 10 (5) (1998) 1203–1216.

[57] E. Parviainen, J. Riihimäki, Y. Miche, A. Lendasse, Interpreting extreme learning machine as an approximation to an infinite neural network, in: KDIR 2010: Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Valencia, Spain, 2010.

[58] B. Frénay, M. Verleysen, Parameter-insensitive kernel in extreme learning for non-linear support vector regression, Neurocomputing 74 (2011) 2526–2531.

[59] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) 58 (1996) 267–288.

[60] J. Martínez-Martínez, P. Escandell-Montero, E. Soria-Olivas, J. Martín-Guerrero, R. Magdalena-Benedito, J. Gómez-Sanchis, Regularized extreme learning machine for regression problems, Neurocomputing 74 (17) (2011) 3716 – 3721.

[61] Y. Miche, A. Sorjamaa, A. Lendasse, Op-elm: Theory, experiments and a toolbox, in: Proceedings of the 18th international conference on Artificial Neural Networks, Part I, ICANN '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 145–154.

[62] J. Zhu, S. Rosset, T. Hastie, R. Tibshirani, 1-norm support vector machines, Advances in neural information processing systems 16 (1) (2004) 49–56.

[63] O. Mangasarian, Exact 1-norm support vector machines via unconstrained convex differentiable minimization, The Journal of Machine Learning Research 7 (2006) 1517–1530.

[64] L. Wang, X. Shen, On l 1-norm multiclass support vector machines, Journal of the American Statistical Association 102 (478) (2007) 583–594.

[65] C. Chang, C. Lin, Libsvm: a library for support vector machines, ACM Transactions on Intelligent Systems and Technology (TIST) 2 (3) (2011) 27.

[66] C. Hsu, C. Lin, A comparison of methods for multiclass support vector machines, Neural Networks, IEEE Transactions on 13 (2) (2002) 415–425.

[67] K. Duan, S. Keerthi, Which is the best multiclass svm method? an empirical study, Multiple Classifier Systems (2005) 732–760.

[68] Official elm toolbox, accessed 6/9/2012.
URL `http://www.ntu.edu.sg/home/egbhuang/ELM_Codes.htm`

[69] S. Keerthi, S. Shevade, Smo algorithm for least-squares svm formulations, Neural computation 15 (2003) 487–507.

[70] J. C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schölkopf, C. J. C. Burges, A. J. Smola (Eds.), Advances in kernel methods, MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.

[71] D. N. A. Asuncion, UCI machine learning repository (2007).
URL `http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html`

[72] K. D. Brabanter, P. Karsmakers, C. A. F. Ojeda, J. D. Brabanter, K. Pelckmans, B. D. Moor, J. Vandewalle, J. Suykens, Ls-svmlab toolbox, accessed 6/9/2012.
URL `http://www.esat.kuleuven.be/sista/lssvmlab/`

[73] M. Grant, S. Boyd, Graph implementations for nonsmooth convex programs, in: Recent Advances in Learning and Control (tribute to M. Vidyasagar), V. Blondel, S. Boyd, and H. Kimura, editors, Lecture Notes in Control and Information Sciences, Springer, 2008, pp. 95–110.

[74] S. Canu, Y. Grandvalet, V. Guigue, A. Rakotomamonjy, Svm and kernel methods matlab toolbox, Perception Systemes et Information, INSA de Rouen, Rouen, France 2 (2005) 2.