Hindawi Publishing Corporation
Advances in Artificial Neural Systems
Volume 2010, Article ID 597373, 6 pages
doi:10.1155/2010/597373

Research Article

OP-KNN: Method and Applications

Qi Yu,[1] Yoan Miche,[1] Antti Sorjamaa,[1] Alberto Guillen,[2] Amaury Lendasse,[1] and Eric Séverin[3]

[1] *Department of Information and Computer Science, Aalto University School of Science and Technology, FI-00076 Aalto, Finland*
[2] *Department of Computer Technology and Architecture, University of Granada, 18017 Granada, Spain*
[3] *Department GEA, University of Lille 1, 59653 Villeneuve d'ascq cedex, France*

Correspondence should be addressed to Qi Yu, qiqiyu1981@gmail.com

Received 6 October 2009; Revised 25 January 2010; Accepted 2 February 2010

Academic Editor: Songcan Chen

This paper presents a methodology named Optimally Pruned K-Nearest Neighbors (OP-KNNs) which has the advantage of competing with state-of-the-art methods while remaining fast. It builds a one hidden-layer feedforward neural network using K-Nearest Neighbors as kernels to perform regression. Multiresponse Sparse Regression (MRSR) is used in order to rank each kth nearest neighbor and finally Leave-One-Out estimation is used to select the optimal number of neighbors and to estimate the generalization performances. Since computational time of this method is small, this paper presents a strategy using OP-KNN to perform Variable Selection which is tested successfully on eight real-life data sets from different application fields. In summary, the most significant characteristic of this method is that it provides good performance and a comparatively simple model at extremely high-learning speed.

1. Introduction

In many application fields, the regression problem is widely been paid attention to, in order to predict a dependent variable (target) from a number of independent variables (observations), or to model numerical data consisting of values of variables (input) and of one or more variable (output). However, there are two main difficulties facing regression problems: accuracy and computational time.

In the recent years, many different techniques have been investigated to solve the regression problems. Support Vector Machines (SVMs) are one of the most popular ones among these techniques, initially developed for classification tasks and lately has been extended to the domain of regression [1]. Briefly, SVM is a universal constructive learning procedure based on statistical learning theory. Its basic idea is to transform the signal into a higher-dimensional feature space and find the optimal hyperplane for classification and regression problems. Different from Multiple Layer Perception (MLP), the nonlinear classification and model regression are solved using convex optimization leading to a unique solution, which avoids the problem of local minima of MLP [2]. Least Squares Support Vector Machines (LS-SVMs) are reformulations of standard SVM [3]. In LS-SVM, the complexity of solving quadratic programs in SVM is deduced to solving linear Karush-Kuhu-Tucker (KKT) conditions. Only linear equations need to be solved which makes the approach much simpler. As a consequence, LS-SVM loses the property of sparseness in SVM.

However, there are some limitations of SVM that weaken its performance: the hyperparameters of the kernel have to be chosen; the training speed is slow, especially when the number of variables is large. Even from a practical point of view perhaps the most serious problem with SVM is the high algorithmic complexity and extensive memory requirements of the required quadratic programming in large-scale tasks. Thus, another group of methods like K-nearest neighbors (KNNs) or Lazy Learning (LL) [4] is taken into account. The key idea behind KNN is that similar training samples have similar output values and it keeps avoiding the local minima problem as SVM, but performs more simple and fast.

On the other hand, Variable Selection has several important advantages when the number of input variables increases. It helps to decrease the redundancy of the original data. It can also reduce the complexity of the modeling process. Moreover, it contributes to the interpretability of the input variables.

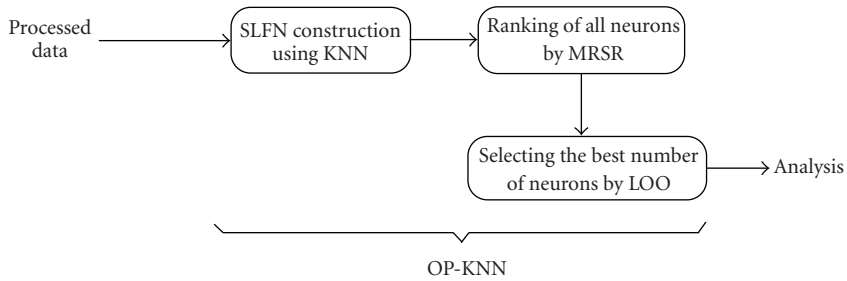

Figure 1: The three steps of the OP-KNN algorithm.

Thus, in this paper, we present a methodology: Optimally Pruned K-Nearest Neighbors (OP-KNNs) which builds a single-hidden layer feedforward neural networks (SLFNs) using KNN as the kernel. The most significant characteristic of this method is that it tends to provide good generalization performance at a fast computing speed and select the most important variables at the same time.

In the next section, the three steps of OP-KNN are introduced. And the strategy we used to solve regression problem using OP-KNN is showed in Section 3. Section 4 gives the results for a toy example and nine real-life datas using OP-KNN and four other methods, and the last section summarizes the whole methodology.

2. Optimal Pruned K-Nearest Neighbors

In this section, a methodology called Optimally Pruned K-Nearest Neighbors (OP-KNNs) is presented. The three main steps of the OP-KNN are summarized in Figure 1.

2.1. Single-Hidden Layer Feedforward Neural Networks (SLFNs). Recently, Huang et al. in [5] proposed an original algorithm called Extreme Learning Machine (ELM). This method makes the selection of the weights of the hidden neurons very fast in the case of single-layer feedforward neural network (SLFN). A more thorough presentation of the ELM algorithm can be found in the original paper [6, 7]. Furthermore, a methodology named Optimally Pruned Extreme Learning Machine (OP-ELM) [8], based on the original ELM algorithm, is proved to be more efficient when encountering irrelevant or correlated data.

The first step of the OP-KNN algorithm is building a single-layer feedforward neural network. This is similar to the core of Extreme Learning Machine (ELM). The difference is that OP-KNN is deterministic, rather than randomly choosing hidden nodes like in ELM and OP-ELM.

In the context of a single-hidden layer perceptron network, let us denote the inputs by $\mathbf{x}$, outputs by $\mathbf{y}$, and the weight vectors between the hidden layer and the output by $\mathbf{b}$. Activation functions used with the OP-KNN differ from the original SLFN choice since the original sigmoid activation functions of the neurons are replaced by the K-Nearest Neighbors, hence it named OP-KNN. For the output layer, the activation function remains as a linear function, meaning that the relationship between hidden layer and output layer is linear.

A theorem proposed in [5] states that the output weights $\mathbf{b}$ can be computed from the real output and the hidden layer output matrix $\mathbf{H}$, where the columns $\mathbf{h_i}$ of $\mathbf{H}$ are the corresponding output of the K-nearest neighbors. Finally, the output weights $\mathbf{b}$ are computed by $\mathbf{b} = \mathbf{H}^{\dagger}\mathbf{y}$, where $\mathbf{H}^{\dagger}$ stands for the Moore-Penrose inverse [9] and $\mathbf{y} = (y_1, \ldots, y_M)^T$ is the output.

The only remaining parameter in this process is the initial number of neurons N of the hidden layer.

2.2. K-Nearest Neighbors. The K-Nearest Neighbors (KNNs) model is a very simple, but powerful tool. It has been used in many different applications and particularly in classification tasks. The key idea behind the KNN is that similar training samples have similar output values for regression problems [10]. In OP-KNN, the approximation of the output is the weighted sum of the outputs of the k-nearest neighbors. The model introduced in the previous section becomes

$$\widehat{y}_i = \sum_{j=1}^{k} b_j y_{P(i,j)}, \tag{1}$$

where $\widehat{y}_i$ represents the output estimation, $P(i, j)$ is the index number of the jth nearest neighbor of sample $\mathbf{x}_i$, and b represents the results of the Moore-Penrose inverse introduced in the previous section.

In this sense, for each different neuron, different nearest neighbors are used, in other words, the only remaining hyperparameter that has to be chosen is the neighborhood size K. Besides choosing K, there is no other hyperparameter in method KNN, as well as in OP-KNN.

2.3. Multiresponse Sparse Regression (MRSR). For the removal of the useless neurons of the hidden layer, the Multiresponse Sparse Regression proposed by Similä and Tikka in [11] is used. It is an extension of the Least Angle Regression (LARS) algorithm [12] and hence it is actually a variable ranking technique, rather than a selection one. The main idea of this algorithm is the following: denote by $\mathbf{T} = [\mathbf{t}_1 \cdots \mathbf{t}_p]$ the $n \times p$ matrix of targets, and by $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_m]$ the $n \times m$ regressors matrix. MRSR adds each regressor one by one to the model $\mathbf{Y}^k = \mathbf{X}\mathbf{W}^k$, where $\mathbf{Y}^k = [\mathbf{y}_1^k \ldots \mathbf{y}_p^k]$ is the target approximation by the model. The $\mathbf{W}^k$ weight matrix

has k nonzero rows at kth step of the MRSR. With each new step a new nonzero row, and a new regressor to the total model, is added.

An important detail shared by the MRSR and the LARS is that the ranking obtained is exact in the case where the problem is linear. In fact, this is the case, since the neural network built in the previous step is linear between the hidden layer and the output layer. Therefore, the MRSR provides the exact ranking of the neurons for the problem [12].

Details on the definition of a cumulative correlation between the considered regressor and the current model's residuals and on the determination of the next regressor to be added to the model can be found in the original paper about the MRSR [11].

MRSR is hence used to rank the kernels of the model: the target is the actual output y_i while the "variables" considered by MRSR are the outputs of the k-nearest neighbors.

2.4. Leave-One-Out (LOO) Method. Since the MRSR only provides a ranking of the kernels, the decision over the actual best number of neurons for the model is taken using a Leave-One-Out method. One problem with the LOO error is that it can get very time consuming if the dataset tends to have a high number of samples. Fortunately, the (or PREdiction Sum of Squares) PRESSs statistics provides a direct and exact formula for the calculation of the LOO error for linear models. See [4, 13] for details on this formula and implementations:

$$\epsilon^{\text{PRESS}} = \frac{y_i - \mathbf{h}_i\mathbf{b}}{1 - \mathbf{h}_i\mathbf{P}\mathbf{h}_i^T}, \tag{2}$$

where $\mathbf{P}$ is defined as $\mathbf{P} = (\mathbf{H}^T\mathbf{H})^{-1}$ and $\mathbf{H}$ the hidden layer output matrix is defined in Section 2.1.

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 (properly ranked by MRSR already).

3. Strategy for Regression Using OP-KNN

3.1. Variable Selection (VS). Variable Selection is one of the most important issues in machine learning, especially when the number of observations (samples) is relatively small compared to the numbers of input variables. It has been the subject in application domains like pattern recognition, time series modeling, and econometrics. The necessary size of the data set increases exponentially with the number of dimensions. To circumvent this, one solution is to select a subset of the features or variables which best describes the output variables (targets) [14]. Then, it is possible to capture and reconstruct the underlying regularity or relationship (that is approximated by the regression model) between input variables and output variables.

Variable Selection has several important advantages. It helps to decrease the redundancy of the original data. It can also reduce the complexity of the modeling process. Moreover, it contributes to the interpretability of the input variables.

3.2. Variable Selection Using OP-KNN. Whether using KNN, OP-KNN, SVM, LS-SVM, or some other regression method, an optimization criterion is needed to do Variable Selection. In fact, there are many ways to deal with the Variable Selection problem, a common one is using the generalization error estimation. In this methodology, the set of features that minimizes the generalization error is selected using Leave one out. Other techniques such as Bootstrap or resampling techniques [15, 16] exist but they are very time consuming and may lead to an unacceptable computational time. In this paper, Variable Selection is performed using the Leave-One-Out error of OP-KNN as criterion, since OP-KNN is very fast.

3.2.1. Wrapper Method. As is well known, Variable Selection can be roughly divided into two broad classes: filter method and wrapper method. As the name implies, our strategy belongs to the wrapper methods which means that the variables are selected according to the criterion directly from the training algorithm.

In other words, our strategy is to select the input subset that can give the best OP-KNN result. Once the input subset is fixed, OP-KNN is repeated to build the model. Furthermore, for the training set and test set, selection procedure is performed on the training set, and then OP-KNN is used on the selected variables of the test set. In this paper, the input subset is selected by means of Forward Selection algorithm.

3.2.2. Forward Selection. This algorithm starts from the empty set S which represents the selected set of the input variables. Then the best available variable is added to the set S one by one until running through all the variables.

To clarify Forward selection, suppose a set of inputs X_i, $i = 1, 2, \ldots, M$, and the output Y, then the algorithm is as follows.

(1) Set F to be the initial set of the original M input variables, and S to be the empty set like mentioned before.

(2) Find

$$X_S = \arg\min_{x_i}\{\text{Opknn}(S \cup X_i)\}, \quad x_i \in F, \tag{3}$$

where X_S represents the selected variable, save the OP-KNN results, and move X_S from F to S.

(3) Continue the same procedure, till the size of S is M.

(4) Compare the OP-KNN values for all the sizes of the sets S, the final selection result is the set S which the corresponding OP-KNN gives the smallest value.

Forward-Backward Selection [17] can be also used instead of Forward Selection in the algorithm but will increase the computational time.

4. Experiments

This section shows the speed and accuracy of the OP-KNN method, as well as the strategy we introduced before, using

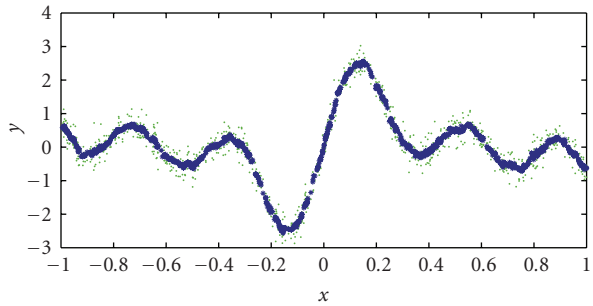

Figure 2: Sine Toy example. Original data is depicted by green points and the model in blue crosses.

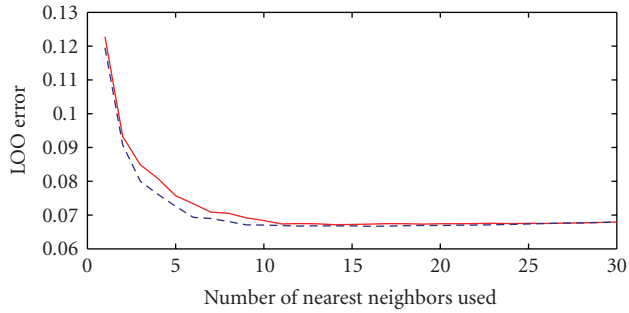

Figure 3: Comparison of LOO error with (dashed blue line) and without (continuous red line) the MRSR.

several different regression data sets. For the comparison, Section 4.2 provides also the performances using Support Vector Machine (SVM) [18].

The following subsection shows a toy example to illustrate the performance of OP-KNN on a simple case that can be plotted.

4.1. Sine Example. In this toy example, a set of 1000 training points (x) are generated (and represented as green points in Figure 2), the output y is a sum of two sines. This single dimension example is used to test the method without the need for variable selection beforehand.

The model built by OP-KNN is showed as blue crosses in Figure 2. As seen from the figure, it approximates the data very well.

The dashed blue line in Figure 3 shows the LOO error for different numbers of nearest neighbors. From the analysis of the figure, by using 16 nearest neighbors, the algorithm reaches the smallest LOO error (0.0666) which is close to the real noise introduced in the dataset which is 0.0625. The computational time for the whole OP-KNN is one second (using Matlab implementation).

Thus, in order to have a very fast and still accurate algorithm, each of the three presented steps has a special importance in the whole OP-KNN methodology. The K-nearest neighbor ranking by the MRSR is one of the fastest ranking methods providing the exact best ranking, since the model is linear (for the output layer), when creating the neural network using KNN. Without MRSR, which can be seen in the solid red line in Figure 3, the number of nearest neighbor that minimizes the Leave-One-Out error is not optimal and the Leave One Out error curve has several local minima instead of a single global minimum. The linearity also enables the model structure selection step using the Leave-One-Out, which is usually very time-consuming. Thanks to the PRESS statistics formula for the LOO error calculation, the structure selection can be done in a small computational time.

Table 1: Key information about the data sets and number of selected variables on average.

Regression	Data			Variables selected by
	Train	Test	Variable	OP-KNN on average
Abalone	2784	1393	8	4.4
Ailerons	4752	2377	5	3.4
Elevators	6344	3173	6	4
Auto Price	106	53	15	2.4
Servo	111	56	4	1.5
Breast cancer	129	65	32	8.9
Bank	2999	1500	8	4
Stocks	633	317	9	7.7
Delve	2000	20732	104	3

Table 2: Test error comparison for the 9 data sets with 5 methods.

Regression	Test error				
	SVM	OP-ELM	ELM	MLP	OP-KNN
Abalone	4.5	4.9	8.3	4.6	4.8
Ailerons	$1.3e-7$	$2.8e-7$	$3.3e-8$	$2.7e-7$	$2.7e-8$
Elevators	$6.2e-6$	$2.0e-6$	$2.2e-6$	$2.0e-6$	$7.5e-5$
Auto price	$2.8e+7$	$9.5e+7$	$7.9e+9$	$2.2e+7$	$3.1e+9$
Servo	$6.9e-1$	$8.0e-1$	7.1	$2.2e-1$	9.7
Breast cancer	$1.2e+3$	$1.4e+3$	$7.7e+3$	$1.5e+3$	$1.5e+3$
Bank	$2.7e-2$	$1.1e-3$	$6.7e-3$	$9.1e-4$	$1.2e-3$
Stocks	$5.1e-1$	$9.8e-1$	$3.4e-1$	$8.8e-1$	$9.0e-2$
Delve	$1.1e+9$	$9.2e+8$	$2.5e+9$	$2.4e+8$	$6.8e+7$

4.2. Real-Data Sets. For the comparison of OP-KNN and four other methods, nine data sets are selected from different application for regression problems [19]. Each data set is randomly permuted (without repetitions) and then divided into training set (two-thirds of the data set) and testing set (one-third of the data set). 10 such rounds are performed (different permutations) such that the results have statistical significance. In this sense, the test error we calculate finally is the average of 10 trials.

The only exception here is the data "Delve," which has 2000 samples in training and 20732 samples in testing. 10-fold test in Monte Carlo way is not necessary in this case since the number of samples in testing is very large.

Table 1 shows some key information about the data sets and the variables selected on average, while Tables 2 and 3 illustrate the test error and Computational time for all methods, respectively.

Table 3: Computational time comparison for the 9 data sets with 5 methods.

Regression	Computational time (in seconds)				
	SVM	OP-ELM	ELM	MLP	OP-KNN
Abalone	$6.6e+4$	5.7	$4.0e-1$	$2.1e+3$	$5.1e+1$
Ailerons	$4.2e+2$	16.8	$9.0e-1$	$3.5e+3$	$5.3e+1$
Elevators	$5.7e+2$	$2.9e+1$	1.6	$3.5e+3$	$8.2e+1$
Auto price	$2.6e+2$	$2.7e-1$	$3.8e-2$	$7.3e+2$	$7.8e-1$
Servo	$1.3e+2$	$2.1e-1$	$3.9e-2$	$5.2e+2$	$2.1e-1$
Breast cancer	$3.2e+2$	$4.2e-1$	$4.8e-2$	$8.0e+2$	2.6
Bank	$1.6e+3$	8.03	$4.7e-1$	$2.7e+3$	$4.39e+1$
Stocks	$2.3e+3$	1.54	$1.1e-1$	$1.2e+3$	6.69
Delve	$4.5e+2$	8.6	$5.7e-1$	$2.9e+4$	$8.75e+3$

As seen from Table 2, the OP-KNN holds the best performance level in most of the cases except two datasets. According to these results, SVM and OP-ELM are reliable in general. However, considering the computational time shown in Table 3, the OP-KNN method clearly has its own advantage. It is faster than SVM, with several orders of magnitude. For example, in the Abalone data set using the OP-KNN is more than 200 times faster than the SVM.

On the other hand, the speed is not the only advantage of OP-KNN; OP-KNN also selects the most significant input variables. This operation highly simplifies the final model, and moreover, makes the data and model more interpretable. The cost is the computational time. According to the forward strategy we used in variable selection part, the higher the dimensionality of the data, the more rounds of OP-KNN. Therefore, OP-KNN is not as fast as OP-ELM in some cases while selecting variables. However, for example, we select 3 most important variables from the original 104 in Delve data, which highly reduces the complexity. This selection of variables was tested with the other methods and yielded much better results—decreasing to $5.6e+7$ the test error for the MLP for example.

5. Conclusions

It is usual to have very long-computational time for training a feedforward network using existing classic learning algorithms even for simple problems, especially when the number of observations (samples) is relatively small compared to the numbers of input variables. Thus, this paper presents OP-KNN method as well as a strategy using OP-KNN to do Variable Selection. This algorithm has several notable achievements:

(i) keeping good performance while being simpler than most learning algorithms for feedforward neural networks,

(ii) using KNN as the deterministic initialization,

(iii) the computational time of OP-KNN being extremely low,

(iv) variable selection highly simplifies the final model, and moreover, makes the data and model more interpretable.

In the experiment section, we have demonstrated the speed and accuracy of the OP-KNN methodology in nine real applications. The aim of OP-KNN is not to be the best method in terms of error, but to prove that OP-KNN is a good tradeoff between performance, computational time, and variable selection possibility. In a word, this makes OP-KNN a valuable tool for real applications.

References

[1] B. Schlkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond Adaptive Computation and Machine Learning*, 2001.

[2] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Mahines and Other Kernal-Based Learning Methods*, Cambridge University Press, New York, NY, USA, 2000.

[3] J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Van dewalle, *Least Squares Support Vector Machines*, World Scientific Publishing, Singapore, 2007.

[4] G. Bontempi,, M. Birattari, and H. Bersini, "Recursive lazy learning for modeling and control," in *Proceedings of the European Conference on Machine learning*, pp. 292–303.

[5] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[6] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16-18, pp. 3056–3062, 2007.

[7] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "Fully complex extreme learning machine," *Neurocomputing*, vol. 68, no. 1–4, pp. 306–314, 2005.

[8] Y. Miche, P. Bas, C. Jutten, O. Simula, and A. Lendasse, "A methodology for building regression models using Extreme Learning Machine: OP-ELM," in *Proceedings of the European Symposium on Artificial Neural Networks*, pp. 23–25, 2008.

[9] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and Its Applications*, John Wiley & Sons, New York, NY, USA, 1972.

[10] A. Sorjamaa, J. Hao, and A. Lendasse, "Mutual information and k-Nearest Neighbors approximator for time series prediction," in *Proceedings of the International conference on Artificial Neural Networks (ICANN '05)*, vol. 3697 of *Lecture Notes in Computer Science*, pp. 553–558, Warsaw, Poland, 2005.

[11] T. Similä and J. Tikka, "Multiresponse sparse regression with application to multidimensional scaling," in *Proceedings of the 15th International Conference on Artificial Neural Networks*, vol. 3697 of *Lecture Notes in Computer Science*, pp. 97–102, 2005.

[12] B. Efron, T. Hastie, I. Johnstone, et al., "Least angle regression," *Annals of Statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[13] R. H. Myers, *Classical and Modern Regression with Applications*, Duxbury, Pacific Grove, Calif, USA, 1990.

[14] A. Lendasse, V. Wertz, and M. Verleysen, "Model selection with cross-validations and bootstraps—application to time series prediction with RBFN models," in *Proceedings of the Joint International Conference on Artificial Neural Networks*, vol. 2714 of *Lecture Notes in Computer Science*, pp. 573–580, Istanbul, Turkey, 2003.

[15] B. Efron and R. Tibshirani, *New Tools in Non-linear Modeling and Prediction*, Chapman and Hall, London, UK, 1993.

[16] M. Verleysen, "Learning high-dimensional data," in *Proceedings of the NATO Advanced Research Workshop on Limitations and Future Trends in Neural Computing*, pp. 22–24, Italy, 2001.
[17] Q. Yu, E. Séverin, and A. Lendasse, "Variable selection for financial modeling," in *Proceedings of the 13th International Conference on Computing in Economics and Finance*, Montréal, Canada, 2007.
[18] C. C. Chang and C. J. Lin, "LIBSVM: a library for support vector machines," 2001, http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[19] http://archive.ics.uci.edu/ml/datasets.html.