한국 퍼지 및 지능시스템 학회 2002, Vol. 13, No. 4, pp. 486-490

Software Reliability Assessment with Fuzzy Least Squares Support Vector Machine Regression[1)]

Changha Hwang · Dug Hun Hong · Jang Han Kim

Department of Statistical Information, Catholic University of Daegu
School of Mechanical and Automotive Engineering, Catholic University of Daegu
Department of Statistics, Keimyung University

ABSTRACT

Software quality models can predict the risk of faults in the software early enough for cost-effective prevention of problems. This paper introduces a least squares support vector machine (LS-SVM) as a fuzzy regression method for predicting fault ranges in the software under development. This LS-SVM deals with the fuzzy data with crisp inputs and fuzzy output. Predicting the exact number of bugs in software is often not necessary. This LS-SVM can predict the interval that the number of faults of the program at each session falls into with a certain possibility. A case study on software reliability problem is used to illustrate the usefulness of this LS-SVM.

Key Words : Least squares support vector machine (LS-SVM), Triangular membership function, Software reliability

1. Introduction

Society is becoming quite dependent on computer-based systems. Today, computers are embedded in wristwatches, vending machines, factory equipment, automobiles and aircraft. A failure in a computer-based system that controls critical applications may lead to significant economic losses or even the loss of human lives. The causes of failures in computer-based systems are manifold: physical faults, maintenance errors, design and implementations mistakes resulting in hardware or software defects, and user or operator mistakes. Here, in particular we focus on software faults. Recent advances in communication technology have led to a rapid proliferation of distributed systems. For example, a cluster of servers provided Web coverage of the 2002 Korea-Japan World Cup Football Games. As distributed systems evolve from the special case to commonplace, ensuring their reliable operation has emerged as an important and challenging problem. In spite of extensive testing and debugging, software faults persist even in commercial grade software. Many distributed systems, especially those employed in safety-critical environments, should be able to operate properly even in the presence of software faults. Hence, predicting software faults is very important in software engineering.

접수일자 : 2003년 1월 2일
완료일자 : 2003년 4월 14일
본 연구는 2003학년도 대구가톨릭대학교 일반연구비에 의해 지원받았습니다.

1) This research is supported by Catholic University of Daegu grant 2003.

Predicting the exact number of bugs in each software is often not necessary. Previous research has focused on classification models to identify fault-prone and not fault-prone softwares. See for details Khoshgoftaar *et al.* [3]. However, available information is often uncertain, imprecise and incomplete. Before software reliability problems become evident, it is difficult to choose an appropriate definition of fault-prone at the time of modeling. In such cases, predicting the range of the numbers of faults or bugs is more appropriate. A predicted interval consists of the possible minimum and maximum numbers of faults in the softwares. A software manager might use this information to allocate testing efforts more effectively.

Recently, Xu *et al.* [8] and D'Urso and Gastaldi [1] have dealt with predicting software faults or bugs using fuzzy nonlinear regression based on neural networks and parametric polynomial model, respectively. Hong and Hwang [2] has proposed the fuzzy nonlinear regression model based on standard support vector machine (SVM) [7]. We can apply this model to such software reliability problems. In this paper we propose much simpler model than previous models.

The rest of this paper is organized as follows. Section 2 illustrates the LS-SVM [5, 6] regression procedures for fuzzy multivariate linear and nonlinear models. Section 3 briefly describes some parameter selection methods. Section 4 gives illustrative example. Section 5 gives some conclusions.

2. LS-SVM for Fuzzy Regression

In this section we will modify the underlying idea of

LS-SVM for the purpose of deriving the convex optimization problems for multivariate fuzzy linear and nonlinear regression models for crisp inputs and fuzzy output. The basic idea of LS-SVM gives computational efficiency in finding solutions of fuzzy regression models particularly for multivariate case. We will focus on fuzzy regression models based on triangular fuzzy number since this type of fuzzy number is mostly used in practice. Fuzzy regression models based on trapezoidal and Gaussian fuzzy numbers can be constructed in a similar manner.

To do this we need some preliminaries. Let $X=(m, \alpha, \beta)$ be a triangular fuzzy numbers when m is the model value of X and α and β are the left and right spreads, respectively. If $\alpha=\beta$, we can write $X=(m, \alpha)$. On the space $T(R)$ of all triangular fuzzy numbers we use the metric d defined by

$$d^2(X, Y)=(m_X-m_Y)^2+((m_X-\alpha_X)-(m_Y-\alpha_Y))^2 +((m_X+\beta_X)-(m_Y+\beta_Y))^2$$

where $X=(m_X, \alpha_X, \beta_X)$ and $Y=(m_Y, \alpha_Y, \beta_Y)$ are any two vectors of triangular fuzzy numbers in $T(R)$. A linear structure is defined on $T(R)$ by

$$(m_X, \alpha_X, \beta_X)+(m_Y, \alpha_Y, \beta_Y) = (m_X+m_Y, \alpha_X+\alpha_Y, \beta_X+\beta_Y)$$

$$t(m, \alpha, \beta)=(tm, t\alpha, t\beta) \text{ if } t\geq 0 \text{ and}$$

$$t(m, \alpha, \beta)=(tm, t\beta, t\alpha) \text{ if } t<0.$$

Now, we will study LS-SVM to be used in estimating fuzzy linear regression model. Suppose we are given training data $\{(\boldsymbol{x}_i, Y_i),\ i=1, \cdots, n\}$, with each input $\boldsymbol{x}_i \in R^d$ and the output $Y_i \in T(R)$. Let x_{ij} be element of $\boldsymbol{x}_i$. Then, we assume $x_{ij} \geq 0$ by simple translation of all vectors. Let $\boldsymbol{W}=(W_1, W_2, \cdots, W_d)$, where $W_i=(m_{W_i}, \alpha_{W_i}, \beta_{W_i})$, $\alpha_{W_i}, \beta_{W_i} \geq 0$, $i=1, \cdots, d$ and let $B=(m_B, \alpha_B, \beta_B)$, $\alpha_B, \beta_B \geq 0$. We now consider the following model:

$$M:\ f(\boldsymbol{x}) = B+\langle \boldsymbol{W}, \boldsymbol{x} \rangle \quad (B \in T(R),\ \boldsymbol{W} \in T(R)^d,\ \boldsymbol{x} \in R^d)$$
$$= B+W_1x_1+W_2x_2+\cdots+W_dx_d$$

where $T(R)^d$ is the set of d-vectors of triangular fuzzy numbers.

We reexpress $\boldsymbol{m}_W=(m_B, m_{W_1}, \cdots, m_{W_d})^t$, $\boldsymbol{\alpha}_W=(\alpha_B, \alpha_{W_1}, \cdots, \alpha_{W_d})^t$ and $\boldsymbol{\beta}_W=(\beta_B, \beta_{W_1}, \cdots, \beta_{W_d})^t$, where the t denotes the transpose of matrix. Then, defining the norm of $\boldsymbol{W}$

$$\| \boldsymbol{W} \|^2 = \| \boldsymbol{m}_W \|^2+\| \boldsymbol{m}_W-\boldsymbol{\alpha}_W \|^2 + \| \boldsymbol{m}_W+\boldsymbol{\beta}_W \|^2,$$

we arrive at the following fuzzy LS-SVM learning procedure for model M as follows:

$$\text{minimize } \frac{1}{2}\| \boldsymbol{W} \|^2+\frac{C}{2}\sum_{k=1}^{3}\sum_{i=1}^{l}\xi_{ki}^2$$

$$\text{subject to } \begin{cases} m_{Y_i}=\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B+\xi_{1i}, \\ m_{Y_i}-\alpha_{Y_i}=\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B -\langle \boldsymbol{\alpha}_W, \boldsymbol{x}_i \rangle-\alpha_B+\xi_{2i}, \\ m_{Y_i}+\beta_{Y_i}=\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B +\langle \boldsymbol{\beta}_W, \boldsymbol{x}_i \rangle+\beta_B+\xi_{3i} \end{cases}$$

Here, the parameter C is a positive real constant and should be considered as a tuning parameter in the algorithm. This controls the smoothness and degree of fit. The cost function with squared error and regularization corresponds to a form of ridge regression.

Introducing Lagrange multipliers α_{1i}, α_{2i} and α_{3i}, $i=1, \cdots, n$, we construct a Lagrange function as follows:

$$\begin{aligned} L = &\frac{1}{2}\| \boldsymbol{W} \|^2+\frac{C}{2}\sum_{k=1}^{3}\sum_{i=1}^{n}\xi_{ki}^2 \\ &-\sum_{i=1}^{n}\alpha_{1i}(\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B+\xi_{1i}-m_{Y_i}) \\ &-\sum_{i=1}^{n}\alpha_{2i}(\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B-\langle \boldsymbol{\alpha}_W, \boldsymbol{x}_i \rangle \\ &-\alpha_B+\xi_{2i}-(m_{Y_i}-\alpha_{Y_i})) \\ &-\sum_{i=1}^{n}\alpha_{3i}(\langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B+\langle \boldsymbol{\beta}_W, \boldsymbol{x}_i \rangle \\ &+\beta_B+\xi_{2i}-(m_{Y_i}+\beta_{Y_i})) \end{aligned}$$

Then, the conditions for optimality are given by

$$\frac{\partial L}{\partial \boldsymbol{m}_W}=\boldsymbol{0} \rightarrow 3\boldsymbol{m}_W-\boldsymbol{\alpha}_W+\boldsymbol{\beta}_W=\sum_{k=1}^{3}\sum_{i=1}^{n}\alpha_{ki}\boldsymbol{x}_i$$

$$\frac{\partial L}{\partial m_B}=0 \rightarrow \sum_{k=1}^{3}\sum_{i=1}^{n}\alpha_{ki}=0$$

$$\frac{\partial L}{\partial \boldsymbol{\alpha}_W}=\boldsymbol{0} \rightarrow \boldsymbol{m}_W-\boldsymbol{\alpha}_W=\sum_{i=1}^{n}\alpha_{2i}\boldsymbol{x}_i$$

$$\frac{\partial L}{\partial \alpha_B}=0 \rightarrow \sum_{i=1}^{n}\alpha_{2i}=0$$

$$\frac{\partial L}{\partial \boldsymbol{\beta}_W}=\boldsymbol{0} \rightarrow \boldsymbol{m}_W+\boldsymbol{\beta}_W=\sum_{i=1}^{n}\alpha_{3i}\boldsymbol{x}_i$$

$$\frac{\partial L}{\partial \beta_B}=0 \rightarrow \sum_{i=1}^{n}\alpha_{3i}=0$$

$$\frac{\partial L}{\partial \xi_{ki}}=0 \rightarrow \xi_{ki}=\frac{\alpha_{ki}}{C},\ k=1,2,3,\ i=1, \cdots, n$$

$$\frac{\partial L}{\partial \alpha_{1i}}=0 \rightarrow \langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B+\xi_{1i}=m_{Y_i}$$

$$\frac{\partial L}{\partial \alpha_{2i}}=0 \rightarrow \langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B-\langle \boldsymbol{\alpha}_W, \boldsymbol{x}_i \rangle-\alpha_B+\xi_{2i} = m_{Y_i}-\alpha_{Y_i},\ i=1, \cdots, n$$

$$\frac{\partial L}{\partial \alpha_{3i}}=0 \rightarrow \langle \boldsymbol{m}_W, \boldsymbol{x}_i \rangle+m_B+\langle \boldsymbol{\beta}_W, \boldsymbol{x}_i \rangle+\beta_B+\xi_{2i} = m_{Y_i}+\beta_{Y_i},\ i=1, \cdots, n$$

with solutions

$$\boldsymbol{m}_W = \sum_{i=1}^{n} \alpha_{1i} \boldsymbol{x}_i, \quad \boldsymbol{\alpha}_W = \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i}) \boldsymbol{x}_i,$$

$$\boldsymbol{\beta}_W = \sum_{i=1}^{n} (\alpha_{3i} - \alpha_{1i}) \boldsymbol{x}_i$$

and

$$\begin{bmatrix} 0 & 0 & 0 & \mathbf{1}^t & \mathbf{1}^t & \mathbf{1}^t \\ 0 & 0 & 0 & \mathbf{0}^t & \mathbf{1}^t & \mathbf{0}^t \\ 0 & 0 & 0 & \mathbf{0}^t & \mathbf{0}^t & \mathbf{1}^t \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \Omega + \frac{1}{C} I & O & O \\ \mathbf{1} & -\mathbf{1} & \mathbf{0} & O & \Omega + \frac{1}{C} I & O \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & O & O & \Omega + \frac{1}{C} I \end{bmatrix} \begin{bmatrix} m_B \\ \alpha_B \\ \beta_B \\ \boldsymbol{\alpha}_1 \\ \boldsymbol{\alpha}_2 \\ \boldsymbol{\alpha}_3 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \\ \boldsymbol{m}_Y \\ \boldsymbol{m}_Y - \boldsymbol{\alpha}_Y \\ \boldsymbol{m}_Y + \boldsymbol{\beta}_Y \end{bmatrix}$$

with

$\boldsymbol{m}_Y = (m_{Y_1}, \cdots, m_{Y_n})^t$, $\boldsymbol{\alpha}_Y = (\alpha_{Y_1}, \cdots, \alpha_{Y_n})^t$,

$\boldsymbol{\beta}_Y = (\beta_{Y_1}, \cdots, \beta_{Y_n})^t$, $\boldsymbol{\alpha}_1 = (\alpha_{11}, \cdots, \alpha_{1n})^t$,

$\boldsymbol{\alpha}_2 = (\alpha_{21}, \cdots, \alpha_{2n})^t$, $\boldsymbol{\alpha}_3 = (\alpha_{31}, \cdots, \alpha_{3n})^t$,

$\mathbf{0} = (0, \cdots, 0)^t$, $\mathbf{1} = (1, \cdots, 1)^t$, $n \times n$ zero matrix O, $n \times n$ identity matrix I and $n \times n$ matrix Ω of $\Omega_{ij} = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Here, the t denotes the traspose of matrix.

Hence, the prediction $\hat{Y} = (m_Y, \alpha_Y, \beta_Y)$ given by the LS-SVM procedure on the new unlabeled example $\boldsymbol{x}$ is

$$\Big(\sum_{i=1}^{n} \alpha_{1i} \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + m_B, \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i}) \langle \boldsymbol{x}_i, , \boldsymbol{x} \rangle + \alpha_B, \sum_{i=1}^{n} (\alpha_{3i} - \alpha_{1i}) \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + \beta_B \Big).$$

Next, we will study LS-SVM to be used in estimating fuzzy nonlinear regression model. In contrast to fuzzy linear regression, there have been only a few articles on fuzzy nonlinear regression. In this paper we treat fuzzy nonlinear regression for data of the form with numerical inputs and fuzzy output, without assuming the underlying model function. In the case where a linear regression function is inappropriate LS-SVM makes algorithm nonlinear. How can the above methods be generalized to the case where the regression function is not a linear function of the data? This could be achieved by simply preprocessing input patterns $\boldsymbol{x}_i$ by a map $\Phi: R^d \rightarrow E$ into some feature space E and then applying LS-SVM regression algorithm. This is an astonishingly straightforward way.

First notice that the only way in which the data appears in the training problem is in the form of dot products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. The algorithm would only depend on the data through dot products in E, i.e. on functions of the form $\langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$. Hence it suffices to know and use $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$ instead of $\Phi(\cdot)$ explicitly. The only difference between LS-SVMs for linear and nonlinear function estimations is the use of mapping function Φ. The well used kernels for regression problem are given below.

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^t \boldsymbol{y} + 1)^p, \quad K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}}.$$

Here, p and σ^2 are kernel parameters. The kernel approach is again employed to address the curse of dimensionality. In final, the fuzzy nonlinear LS-SVM regression solution is given by

$$\Big(\sum_{i=1}^{n} \alpha_{1i} K(\boldsymbol{x}_i, \boldsymbol{x}) + m_B, \sum_{i=1}^{n} (\alpha_{1i} - \alpha_{2i}) K(\boldsymbol{x}_i, \boldsymbol{x}) + \alpha_B, \sum_{i=1}^{n} (\alpha_{3i} - \alpha_{1i}) K(\boldsymbol{x}_i, \boldsymbol{x}) + \beta_B \Big).$$

3. Model Selection Method

When we use LS-SVM for fuzzy linear regression, we still have to determine an optimal choice of the regularization parameter C. In particular, when we use this algorithm for fuzzy nonlinear regression, we have to determine one more parameter, which is the polynomial degree p and the kernel width σ^2 for polynomial and Gaussian kernels, respectively. There could be several parameter selection methods such as cross-validation type methods, bootstraping and Bayesian learning methods. In this paper we use cross-validation methods. If data is not scarce then the set of available input-output measurements can be divided into two parts - one part for training and one part for testing. In this way several different models, all trained on the training set, can be compared on the test set. This is the basic form of cross-validation. A better method is to partition the original set in several different ways and to compute an average score over the different partitions. In this paper the average score is computed by using the squared error based on the following distance

$$d^2(X, Y) = (m_X - m_Y)^2 + ((m_X - \alpha_X) - (m_Y - \alpha_Y))^2 + ((m_X + \beta_X) - (m_Y + \beta_Y))^2.$$

An extreme variant of this is to split the n measurements into a training set of size $n-1$ and a test set of size 1 and average the squared error on the left-out measurements over the n possible ways of obtaining such a partition. This is called leave-one-out (LOO) or 1-fold cross-validation. In the LOO method, we train using all but one training measurement, then test using the left out measurement. We repeat this, leaving out another single measurement. We do this until we have left out each example. Then we average the results on the left out measurements to assess the

generalization capability of our fuzzy regression procedure.

LOO is computationally more demanding. The advantage is that all the data can be used for training - none has to be held back in a separate test set. For large data sets we typically prefer 10-fold cross-validation in order to select regularization and kernel parameters. The LOO for linear models can be calculated analytically, but is slightly awkward to handle. See for details Orr [4]. Its cousin, generalized cross-validation (GCV), is more convenient. The similarity of GCV to LOO cross-validation is apparent. Performance for small samples is different for two criteria. However, performance for large samples is approximately the same for two criteria. For fuzzy linear and nonlinear models in ths paper we can use the following GCV criterion.

$$GCV = \frac{1}{\left(1-\frac{k}{n}\right)^2}\left[\sum_{i=1}^{n}(m_{Y_i}-\hat{m}_{Y_i})^2 + \sum_{i=1}^{n}((m_{Y_i}-\alpha_{Y_i})-(\hat{m}_{Y_i}-\hat{\alpha}_{Y_i}))^2 + \sum_{i=1}^{n}((m_{Y_i}+\beta_{Y_i})-(\hat{m}_{Y_i}+\hat{\beta}_{Y_i}))^2\right]$$

where $\hat{m}_{Y_i}$, $\hat{\alpha}_{Y_i}$ and $\hat{\beta}_{Y_i}$ are the fitted values of m_{Y_i}, α_{Y_i} and β_{Y_i}, respectively and k is the effective number of parameters. See Orr [4] for details on how to compute k for linear model, and Vapnik [7] for nonlinear model.

4. Empirical Study

We study a dataset which comes from D'Urso and Gastaldi [1] for a real case study, carried out during the development of a sophisticated software system for running a virtual mall, i.e., a mall on the Internet on an http server (http is the hypertext protocol used for exchanging data on the Internet). The work of programmers was paralleled by a team formed by seven beta-testers, i.e., individuals whose job was that of testing the program functionality, and of finding possible bugs in the software under development. For all the course of the software development, which took 30 months, the team of beta-testers produced comprehensive monthly reports. In each monthly report, along with the analytical scores attributed to specific aspects of the program under development (such as interface usability, accessibility, credit card transactions security, help system, program reliability, etc.), the team also provided an average global score expressed as a fuzzy number, based on the aggregation (through a weighted average) of the single scores. Such average global scores are reported in Table 1, and are used in the following for the purpose of providing an application of our procedure (the software company made these data available provided their origin remained undisclosed).

It is interesting to note that these data are not monotonic, due to occasional falls of reliability evaluation caused by the discovery of (more or less important) malfunctions or bugs. We wished to find a model which could describe the fuzzy evaluations of the reliability and functionality of the software under development.

D'Urso and Gastaldi [1] use the fourth-order polynomial model. For this particular dataset the fourth-order model succeeded in capturing the "bug-discovery phase", where the beta-testers substantially lowered their grades. Observe that their procedure indicated to choose the polynomial model of the fourth order, which correctly identifies, in the software development process, a period (after around one year) when there was a reengineering of part of the program which had serious problems with the Internet communication protocols. However, their polynomial model is somewhat impractical since it is basically univariate parametric model which can be seldom used in fitting models with several inputs in practice.

Table 1. Fuzzy Data for Software Reliability Assessment

Test Session	Center	Left Spread	Right Spread	Test Session	Center	Left Spread	Right Spread
1	6	2	8	16	10	4	3
2	5	3	6	17	12	6	8
3	8	5	8	18	13	3	9
4	10	9	9	19	20	6	7
5	13	9	8	20	28	7	9
6	19	5	7	21	29	8	9
7	20	8	5	22	30	4	7
8	23	9	5	23	35	2	3
9	25	5	7	24	44	3	5
10	27	6	8	25	45	6	6
11	26	7	6	26	43	5	4
12	25	4	6	27	47	3	9
13	5	3	3	28	48	4	5
14	8	5	3	29	50	6	4
15	9	7	8	30	49	5	4

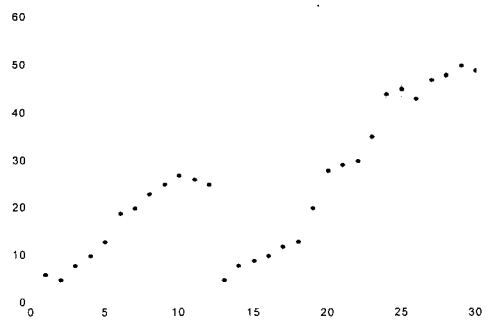

Fig. 1. Fuzzy LS-SVM Regression

For this dataset the fuzzy nonlinear regression model by Xu *et al.* [8] can be utilized. Their procedure can be

also implemented in the case where several inputs are used and there is no information on the underlying model at all. However, their model is rather complicated to implement in practice since this method uses two separate neural networks for left and right spreads and has to make neural networks to keep the relationship between left and right spreads.

We have used LOO cross-validation to determine an optimal combination of C and σ, which are $C=130$ and $\sigma=4.0$. As seen from Figure 1, we observe that our LS-SVM succeeded in capturing the "bug-discovery phase" and correctly identifies, in the software development process, a period (after around one year) when there was a reengineering of part of the program which had serious problems with the Internet communication protocols.

For future research we need to show the outperformance of our LS-SVM over procedures by D'Urso and Gastaldi [1] and Xu et al. [8] for complex data sets with several inputs as in Xu et al. [8].

5. Conclusions

High reliability is an important attribute for high assurance software systems. Consequently, software developers are seeking ways to forecast and improve quality before release. Because many quality factors cannot be measured until after software becomes operational, software quality models are developed to predict quality factors based on measurements that can be collected earlier in the life cycle.

Due to the incomplete information in the early life cycle of software development, a software quality model with fuzzy characteristics can perform better, because fuzzy concepts deal with phenomena that are vague in nature. LS-SVM derives the satisfying solutions and is attractive approach to predicting bugs ranges in the software under development.

The algorithms combine generalization control with a technique to address the curse of dimmensionality. The main formulation results in solving a simple matrix inversion problem instead of solving a global quadratic optimization problem with box constraints unlike SVM in Hong and Hwang [2]. Hence, this is not a computationally expensive way. The hyperparameters of the proposed algorithm were tuned using LOO cross-validation or GCV procedure and a grid search mechanism.

References

[1] P. D'Urso and T. Gastaldi, "An orderwise polynomial regression procedure for fuzzy data," Fuzzy Sets and Systems, Vol. 130, pp. 1-19, 2002.

[2] D. H. Hong and C. Hwang, "Support vector fuzzy regression machines," to be appeared in Fuzzy Sets and Systems, 2003.

[3] T. M. Khoshgoftaar, E. B. Allen, R. Halstead and G. P. Trio, "Detection of fault-prone software modules during a spiral life cycle," In Proceedings of the International Conference on Software Maintenance, pp. 69-76, Monterey, CA, November 1996. IEEE Computer Society.

[4] M. J. L. Orr, "Introduction to radial basis function networks," Centre for Cognitive Science Technical Report, U. of Edinburgh, 1996.

[5] J. A. K. Suykens and J. Vandewalle, "Recurrent least squares support vector machines," IEEE Transactions on Circuits and Systems-I, Vol. 47, No. 7, pp. 1109-1114, 2000.

[6] J. A. K. Suykens, "Nonlinear modelling and support vector machines," Proc. of the IEEE International Conference on Instrumentation and Measurement Technology, pp. 287-294, 2001.

[7] V. N. Vapnik, "Statistical Learning Theory," John Wiley & Sons, New York, 1998.

[8] Z. Xu, T. M. Khoshgoftaar and E. B. Allen, "Prediction of software faults using fuzzy nonlinear regression modeling," In Proceedings: Fifth IEEE International Symposium on High-Assurance Systems Engineering, Albuquerque, New Mexico USA, November 2000. IEEE Computer Society.

저 자 소 개

홍덕헌

제 12 권 6 호 참조

황창하

1978~1982 경북대학교 사범대학 수학교육과 (학사)
1982~1984 서울대학교 대학원 계산통계학과 통계학전공(석사)
1985~1987 한국통신 전임연구원
1987~1991 미국 Michigan 대학교 통계학과 (박사)
1992~1995 경성대학교 전산통계학과 조교수
1995~현재 대구가톨릭대학교 정보통계학과 부교수
관심 분야 : 계산지능, 뉴로-퍼지 시스템, 기계학습, 생명정보학

김장한

1979.02 (졸) 경북대학교 수학교육학전공 학사
1981.02 (졸) 고려대학교 통계학전공 (이론통계학) 석사

관심분야 : 이론통계학 수리통계학 통계자료분석