Lagrangian Support Vector Machines for Fuzzy Regression Problems

A dissertation submitted to the

School of Computer & Systems Sciences

Jawaharlal Nehru University, New Delhi

In the partial fulfillment of requirement for the award of the degree of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND TECHNOLOGY

by

HASAN Z. H. ALHAJHAMAD

School of Computer and Systems Sciences

Jawaharlal Nehru University – JNU

New Delhi – 110067

JULY 2010

DEDICATED TO

GOD, who gave me the divine love and blessings

JAWAHARLAL NEHRU UNIVERSITY

SCHOOL OF COMPUTER & SYSTEMS SCIENCES

NEW DELHI-110067(INDIA)

CERTIFICATE

This is to certify that this dissertation entitled **"Lagrangian Support Vector Machines for Fuzzy Regression Problems"** submitted by **Mr. Hasan Z. H. Alhajhamad**, to the **School of Computer and Systems Sciences**, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of degree of **Master of Technology in Computer Science & Technology**, is a bona fide work carried out under my supervision.

The study concerning to this dissertation is original and has not been submitted, in part or in full, to any other University or Institution for the award of any other degree.

29/7/10

Prof. Sonajharia Minz

Dean

SC&SS, JNU, New Delhi

Prof. S. Balasundaram

Supervisor

SC&SS, JNU, New Delhi

Prof. Sonajharia Minz
Dean
School of Computer & Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

JAWAHARLAL NEHRU UNIVERSITY

SCHOOL OF COMPUTER & SYSTEMS SCIENCES

NEW DELHI-110067(INDIA)

DECLARATION

This is to certify that the dissertation titled "**Lagrangian Support Vector Machines for Fuzzy Regression Problems**", being submitted to the **School of Computer & Systems Sciences**, Jawaharlal Nehru University, New Delhi, in partial fulfillment of the requirements for the award of **Master of Technology in Computer Science & Technology** is a bona fide work carried out by me. This research work has been carried out the under the supervision of **Prof. S. Balasundaram**.

The study concerning to this dissertation is original and has not been submitted, in part or in full, to any other University or Institution for the award of any other degree.

29-7-2010

Hasan Z. H. Alhajhamad

M. Tech, SC & SS, JNU,

New Delhi - 110067

Acknowledgments

I would like to praise the Almighty, who gave me everything I needed and sustained me with faith and blessings to complete this work.

I would like to express my truehearted gratitude and thankfulness to my thesis supervisor, Prof. S. Balasundaram of the School of Computer and Systems Sciences for his admirable help and support during the period of my study. The completion of this research would not have been possible without his guidance and help.

I can never express in words my gratitude to my parents without whose encouragement and support, I would never have been able to complete this work. I thank my sisters for their support to me sharing me love and support.

I would like to show my gratitude to Mr. Ibrahim, Mr. Salaah and Madam Shadi, and all the friends who made me countless favors.

I would like to express my sincere gratitude to Ms. Ainura Aitibaeva for her support and generous kindness.

Abstract

In Machine Learning and Pattern Recognition problems, the Support Vector Machines (SVM) has been a very successful model for producing computational efficient and accurate solutions. In problems where the information is uncertain we combine the concept of fuzzy uncertainty into the Support Vector Machine. This improvement for the Support Vector Machine model will solve the inexact nature of the information. Support Vector Machines solve both Classification and Regression problems. In this work we demonstrated the Lagrangian Support Vector Machine for Fuzzy Regression Problems with fuzzy input and fuzzy output data.

Table of Contents

Chapter 1:

Introduction to Support Vector Machine

1.1 Foreword to Problem Solving

Computers are designed to solve the human being problems by taking the inputs and giving the outputs. The simplest example for that is Printers. To result the required output in problem solving techniques we explicitly use programming and algorithms on the inputs. For a complex inputs or problems it's well-known that we would need algorithms to solve them but even so there might be no precise algorithm will produce the precise and less expensive output. For example, how to model a complex classifier to recognize the mood of person when the input is his portrait photo, where the precise faces of different people are not known, or classification of protein types based on the DNA sequence from which they are generated, or weather forecasting or prediction [1].

It's not possible to solve these tasks by only programming. That we cannot precisely specify the correct output that can be computed from the input data. Problems like those can be solved by learning the input and output functionality from examples or what is called *learning methodology* to learn the inputs and outputs from the multiple examples happening so that the system will recognize the solution by its training. When the examples are input and output pairs it is called *supervised learning.* The examples of input and output functionality are referred to as the *training data.*

The input/output pairings typically reflect a functional relationship mapping inputs to outputs. Function which forms a translation from inputs to outputs it is called the *target function.* The target function is learnt by the learning algorithm as well as it's called the *solution* of the learning problem.

Classification is the case when the target function is making a decision to choose among multiple output options. In this case we can call it also the *decision function.* So, the solution in classification problem is chosen from a set of target functions find the proper output domain or *Class*. For example, so-called *decision trees* are a good classification example. Decision Trees use simple decision functions (if statements) at the internal nodes and output values at the leaves. Therefore, the leaves of the decision functions are the choice after passing the set of the functions and this is one of the key ingredients of the learning strategy. I mean; the algorithm which its inputs are the training data and selects the output through the functions is called here the *learning algorithm.*

As a sensible example, the Classification learning methodology is in the same way that we ask a child to take out the foot-balls from a basket filled with different types of balls (i.e foot-ball, basket-balls, volley-balls...etc). The easy way to teach this child how to distinguish the foot-balls is by telling him a number of the balls in the basket which are foot-ball ones and by giving him a precise specification of the foot-ball. Learning to distinguish foot-ball is simply making the output is a YES if the ball was a foot-ball or NO if the ball was not a foot-ball which is a binary output value. For that, the learning problem with binary outputs is called also a *binary classification* problem.

When we ask the same child to take out not only the foot-balls from the basket but also the basket-balls, the volley-balls, and the cricket-balls by the same way of teaching him with the *binary classification* we call these output values as *regression*. Thereby, the learning problem becomes with a finite number of categories as *multi-class classification,* while for real-valued outputs the problem becomes known as *regression.* With no doubt, it's considered that the binary classification is the simplest case and with having more output values that is to say having more complicated cases [1].

1.2 Presenting the Support Vector Machine

The Support Vector Machines (SVM) technique is a learning methodology proved its efficiency to perform input/output mapping which is applying the supervised learning methodology performing batch training data. This technique was introduced by Vapnik [4].

The strategy of the Support Vector Machine is to presume a decision plane that separates between a set of samples having different types of class memberships in the space. For example, as it's shown in Figure 1, we suppose that we have samples belong either to class Green or Red. The separating decision plane forms a boundary on the right side of which all samples are Red and to the left of which all samples are Green. New objects falling to the right will be classified as Red or if they were falling to the left will be classified as Green [5].

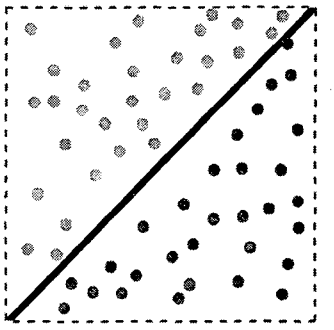

Figure 1: Exposing the strategy of Support Vector Machines

Thereby, the Support Vector Machines aim to create an efficient algorithm to learn the optimal separating planes using a number of samples. The first and simplest model of SVM is the Maximal Margin Classifier which is linearly separating the space into classes.

The Maximal Margin Classifier optimizes a bound separating the data with a maximal margin hyper-plane, whereas given that the bound is independent from the dimensionality of the space [1].

Chapter 2:

Linear and Dual Support Vector Machines Problem

Throughout this work, all vectors are considered as column vectors. For any two vectors x, y in R^n, the inner product of the two vectors will be denoted by $x^t y$ where x^t is the transpose of x. For any vector $x = (x_1, \dots, x_n)^t \epsilon\, R^n$, we define the plus function x_+ as: $(x_+)_i = \max\{0, x_i\}$ for$i = 1, 2, 3, \dots, n$. The column vector of ones of size equals to the number of input samples will be denoted by e.

Consider the problem of classifying m points $\{(x_i, y_i)\}_{i=1..m}$ where $x_i = (x_{i1}, x_{i2}, \dots, x_{in})^t$ in R^n and $y_i = \pm 1$.

Suppose $A = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ \dots & \dots & \dots & \dots \\ x_{m_1} & x_{m_2} & \dots & x_{mn} \end{pmatrix}$, $A_i = (x_{i1}, x_{i2}, \dots, x_{in})$,

and $D = \text{diag}(y_1, y_2, \dots, y_n)$ is the diagonal matrix.

For this problem the standard support vector machine is given by the following quadratic program with parameter $C > 0$:

$\min_{(w,b,\xi)} C\, e^t\xi + \frac{1}{2} w^t w.$

Subject to $\quad D(Aw - eb) + \xi \geq e,$

$\xi \geq 0$

Here w is the normal to the bounding planes:

$x^t w = b \pm 1$ or $x^t w + b = \pm 1$

where b is determining the location of the bounding planes. The plane $x^t w = b + 1$ bounds the positive class A_+ points, with some error, and the plane $x^t w = b - 1$ bounds the negative class A_- points, also possibly with some error. The linear separating surface is the plane:

$$x^t w = b$$

midway between the two planes $x^t w + b = \pm 1$. The two planes bound the two classes with a "soft margin". That is, we bound each set approximately with some error determined by the nonnegative error variable ξ_i:

$A_i w + \xi_i \geq b + 1$ for $D_{ii} = +1$,

$A_i w - \xi_i \leq b - 1$ for $D_{ii} = -1$.

Now, using the Lagrangian multipliers α, β, the Lagrangian function will be:

$$L(w, b, , \xi_i, \alpha, \beta) = {}^1/_2\, w^t w + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i (w^t x_i - b) + \xi_i - 1] - \sum_{i=1}^{N} \beta_i \xi_i$$

$$L(w, b, , \xi_i, \alpha, \beta) = {}^1/_2\, w^t w + \sum_{i=1}^{N} (C - \alpha - \beta)\xi_i$$

$$- \left(\sum_{i=1}^{N} \alpha_i y_i x_i^t \right) w - \left(\sum_{i=1}^{N} \alpha_i \xi_i \right) b + \sum_{i=1}^{N} \alpha_i$$

$$\max_{\alpha} L_{dual} \equiv \sum_{i=1}^{N} \alpha_i - {}^1/_2 \sum_{i'j}^{N} \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

Subject to: $0 \leq \alpha_i \leq C$,

$$\sum_i \alpha_i y_i = 0$$

The solution is given by

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

Recently, Mangasarian (Mangasarian & Musicant) [3] has proposed an "equivalent" SVM formation of the form:

$$\min_{w,b,\xi \in R^{n+1+m}} c\frac{\xi\xi^t}{2} + {}^1/_2 (w^t w + b^2)$$

Subject to

$$D(Aw - eb) + \xi \geq e$$

The dual of this problem is:

$$\min_{0 \leq \alpha \in R^m} \frac{1}{2}\alpha^t({}^I/_C + D(AA^t + ee^t)\mathrm{D})\alpha - e^t\alpha.$$

The variables (w, b) of the primal problem which determine the separating surface $x^t w = b$ are recovered directly from the solution of the dual above satisfying the relations:

$w = A^t D\alpha$

$\xi = \frac{\alpha}{C}$

$b = -e^t D\alpha$

Where α is the lagrangian multiplier.

Now, note that the matrix appearing in the dual objective function is positive definite and that there is no equality constraint and no upper bound on the dual variable α. Unlike the standard SVM formation, the only constraint present is a nonnegativity one.

We define two matrices to simplify our notations:

$H = D[A - e]$, $\quad Q = {}^1/_C + H H^t$

with these definistions the dual problem becomes:

$$\min_{0 \leq \alpha \in R^m} f(\alpha) := {}^1/_2 \alpha^t Q\alpha - e^t\alpha$$

By the result of optimization theory it can be verified that the above minimization problem is equivalent in solving the problem:

$$0 \leq \alpha \perp Q\alpha - e \geq 0$$

Since for any two real numbers a, b,

$$0 \leq a \perp b \geq 0$$

is equivalent to

$$A = (a - ub)_+ \quad , \text{for any } u > 0$$

and

$$x_+ = \max\{0, x\}$$

In our problem, for

$\alpha^t = (\alpha_1, \ldots, \alpha_m)$.

$$Q\alpha = \begin{pmatrix} q_{11} & \cdots & q_{1m} \\ \cdots & \cdots & \cdots \\ q_{m1} & \cdots & q_{mm} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \cdots \\ \alpha_m \end{pmatrix} = \begin{pmatrix} q_{11}\alpha_1 + \ldots + q_{1m}\alpha_m \\ \cdots \\ q_{m1}\alpha_1 + \ldots + q_{mm}\alpha_m \end{pmatrix}$$

$$0 \leq \begin{pmatrix} \alpha_1 \\ \alpha_m \end{pmatrix} \perp \begin{pmatrix} q_{11} & \cdots & q_{1m} \\ \cdots & \cdots & \cdots \\ q_{m1} & \cdots & q_{mm} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \cdots \\ \alpha_m \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \end{pmatrix} \geq 0.$$

meaning,

$$0 \leq \alpha_1 \perp (q_{11}\alpha_1 + \ldots + q_{1m}\alpha_m) - 1 \geq 0$$

$$\ldots$$

$$0 \leq \alpha_m \perp (q_{m1}\alpha_1 + \ldots + q_{mm}\alpha_m) - 1 \geq 0$$

Therefore we have to solve the equivalent problem:

$(Q\alpha - e) = (Q\alpha - e - u\alpha)$.

i.e. we need to find $\alpha \in R^m$ such that,

$Q\alpha = e + (Q\alpha - e - u\alpha)_+$. Or $\alpha = Q^{-1}[e + (Q\alpha - e - u\alpha)_+]$

To solve this problem a simple iteration scheme of the following form is proposed for any input $\alpha_1 \in R^m$:

$$\alpha_2 = Q^{-1}[e + (Q\alpha_1 - e - u\alpha_1)_+]$$

$$\alpha_3 = Q^{-1}[e + (Q\alpha_2 - e - u\alpha_2)_+]$$

$$\alpha_{i+1} = Q^{-1}[e + (Q\alpha_i - e - u\alpha_i)_+]$$

where $i = 1, 2, \ldots$

The Langragian SVM algorithm requires the inversion of a positive definite $n \times n$ matrix, at the beginning of the algorithm followed by a straightforward linearly convergent iterative scheme that requires no optimization package.

In the next chapter, we extend the above study of Lagrangian SVM to fuzzy regression problems for fuzzy input and fuzzy output samples.

Chapter 3:

Lagrangian Support Vector Machines for fuzzy Inputs and Fuzzy Outputs

3.1 Brief introducing to our new problem

In many real world applications, the observed input data can not be measured precisely. For such cases, the concept fuzzy is introduced to characterize imprecision.

In this chapter we study the extension of lagrangian support vector machines for classification problems proposed by Mangasarian and Musicant (2001) [3] to fuzzy regression problems. We assume that both the input and output of the given regression problem are triangular fuzzy numbers.

Let $x = (m_x, s_x, r_x)$ and y= (m_y, s_y, r_y) be two triangular fuzzy numbers where m is the centre, $l > 0$ is the left spread and $r > 0$ is the right spread. For the arithmetic operations and for the definition of a metric or triangular fuzzy numbers we follow the approach of Yan and Xu (2007) [10]. Let $T(R)$ be the space of all triangular fuzzy numbers. For $X, Y \epsilon T(R)$, we define:

$$X + Y = (m_x + m_y\ , \max\{s_x, s_y\}, \max\{r_x, r_y\}\ ),$$

$$k\,X = (km_x, s_x, r_x) \quad , \text{if } \ k > 0$$

$$X - Y = (m_x - m_y\ , \max\{s_x, s_y\}, \max\{r_x, r_y\}\ ),$$

It is shown in Yan and Xu (2007) [11] that the distance $d\,(.,.)$ between two fuzzy numbers X, Y can be obtained as:

$$d\,(X,Y\,) = \max\{\,|(m_x - s_x) - (m_y - s_y)|, |(m_x - m_y)|, |(m_x - r_x) - (m_y + r_y)|\}$$

In particular, when $X$and Y are symmetric triangular fuzzy numbers.

$$d(X,Y) = \max\{|(m_x - s_x) - (m_y - s_y)|, |(m_x + s_x) - (m_y + s_y)|\}$$

where,

$X = (m_x, s_x)$ and Y= (m_y, s_y)

Throughout this work, we assume that triangular fuzzy numbers always symmetric, i.e. $s_x = r_x$ and we denote a triangular fuzzy number X as: $X = (m_x, s_x)$.

For the study of fuzzy regression problem, let us assume that a set of symmetric triangular fuzzy input data$\{(\text{Xi}, \text{Yi})\}_{i=1}^{l}$ be given, where

Xi = $(X_{i1}, \dots X_{in})^t \in T(R)^n, Yi \in T(R)$

be such that,

$$X_{ij} = \left(m_{x_{ij}}, S_{x_{ij}}\right) \in T(R), \qquad Y_i = \left(m_{y_i}, S_{y_i}\right) \in T(R)$$

Assume the regression estimation function $f(.)$ will be defined in the following form:

$$f(X) = (w.m_x + b, \rho(S_{x_i}))$$

for, $x = (x_1, \dots, x_n)^t \in T(R)^n$

Such that,

$X_i = \left(m_{x_i}, S_{x_i}\right) \in T(R)$ (which are the inputs of the function)

$m_x = (m_{x1}, \dots, m_{xn})^t \in R^n$ (m_{xi} is the center of the fuzzy number)

$S_{xi} \in R$ (the spread of the fuzzy number)

$\rho(S_x) = \max\{S_{x1}, \dots, S_{xn}\}$

Here,

$w = (w_1, \ldots, w_n) \in R^n$

$b \in R^n$

w and **b** are the unknowns.

Following the idea of Mangasarian and Musicant (2001) [3] definition of objective function, the ε-insentive (Vapnik, 2000) [8] fuzzy support vector regression for the linear case is formulated as:

$$\min_{w,b,\xi_{i1}+\xi^*_{i1}+\xi_{i2}+\xi^*_{i2}} {}^1/_2 (w^t w + b^2) + {}^c/_2 \sum_{i=1}^{l} \xi_{i1}^2 + \xi_{i1}^{*\,2} + \xi_{i2}^2 + \xi_{i2}^{*\,2}$$

Subject to:

$(m_{yi} + S_{yi}) - (w.m_{xi} + b + \rho(s_{xi})) \leq \varepsilon + \xi_{i1}$,

$(w.m_{xi} + b + \rho(s_{xi})) - (m_{yi} + S_{yi}) \leq \varepsilon + \xi^*_{i1}$,

$(m_{yi} - S_{yi}) - (w.m_{xi} + b - \rho(s_{xi}) \leq \varepsilon + \xi_{i2}$,

$((w.m_{xi} + b) - \rho(s_{xi})) - (m_{yi} + S_{yi}) \leq \varepsilon + \xi^*_{i2}$,

where $c > 0$ is the regularization parameter, and,

$$\xi_1 = (\xi_{11}, \ldots, \xi_{1l})^t, \xi_1^* = (\xi_{11}^*, \ldots, \xi_{1l}^*)^t, \xi_2 = (\xi_{21}, \ldots, \xi_{2l})^t, \xi_2^* = (\xi_{21}^*, \ldots, \xi_{2l}^*)^t \epsilon\, R^l$$

Are the vectors of slack variables.

Note that, for

$X_i = (X_{i1}, \ldots, X_{in})^t \in T(R)^n$,

$Y_i = (m_{y_i}, S_{y_i}) \in T(R)$,

and, $\rho(S_x) = \max\{S_{x1}, \ldots, S_{xn}\}$ such that,

$X_{ij} = \left(m_{x_{ij}}, S_{x_{ij}}\right) \in T(R)$,

we have,

$m_{xi} = (m_{xi1}, \ldots, m_{xin})^t \in R^n$

By introducing the Lagrangian Multipliers, let us consider the Lagrangian function, given by:

$$L = {}^1\!/_2\,(w^t w + b^2) + {}^c\!/_2\,\left(\xi_1^t\xi_1 + \xi_1^{*t}\xi_1^* + \xi_2^t\xi_2 + \xi_2^{*t}\xi_2^*\right)$$

$$+ \sum_{i=1}^{l} u_{1i}[(m_{yi} + S_{yi}) - (w.m_{xi} + b + \rho(s_{xi}) - \varepsilon + \xi_{1i}]$$

$$+ \sum_{i=1}^{l} u_{1i}^*[\left(w.m_{xi} + b + \rho(s_{xi})\right) - \left(m_{yi} + S_{yi}\right) - \varepsilon + \xi_{1i}^*]$$

$$+ \sum_{i=1}^{l} u_{2i}[(m_{yi} - S_{yi}) - (w.m_{xi} + b - \rho(s_{xi}) - \varepsilon + \xi_{2i}]$$

$$+ \sum_{i=1}^{l} u_{2i}^*[(w.m_{xi} + b) - \rho(s_{xi})) - \left(m_{yi} + S_{yi}\right) - \varepsilon + \xi_{2i}^*]$$

and $u_{1i}, u_{1i}^*, u_{2i}, u_{2i}^* > 0$

Now,

$\frac{\partial L}{\partial w} = 0$,

$$\frac{1}{2}[2w + 0] + \sum_{i=1}^{l} u_{1i}(-m_{xi}) + \sum_{i=1}^{l} u_{1i}^*(m_{xi}) + \sum_{i=1}^{l} u_{2i}(-m_{xi}) + \sum_{i=1}^{l} u_{2i}^*(m_{xi}) = 0$$

$$w = \sum_{i=1}^{l} (m_{xi})(u_{1i} + u_{2i}) - \sum_{i=1}^{l} (u_{1i}^* + u_{2i}^*)(m_{xi})$$

Define the matrix:

$$M = \begin{bmatrix} m_{x1}^t \\ \vdots \\ m_{xl}^t \end{bmatrix} = \begin{bmatrix} m_{x1} & \cdots & m_{xn} \\ \vdots & \ldots & \vdots \\ m_{xl1} & \cdots & m_{xln} \end{bmatrix}_{l \times n}$$

$$\sum_{i=1}^{l} (m_{xi})(u_{1i} + u_{2i}) = \sum_{i=1}^{l} \begin{bmatrix} m_{xi1} \\ \vdots \\ m_{xin} \end{bmatrix} (u_{1i} + u_{2i}) = \sum_{i=1}^{l} \begin{bmatrix} m_{xi1}(u_{1i} + u_{2i}) \\ \vdots \\ m_{xin}(u_{1i} + u_{2i}) \end{bmatrix}$$

$$= \begin{bmatrix} m_{x1} & \cdots & m_{xn} \\ \vdots & \ldots & \vdots \\ m_{xl1} & \cdots & m_{xln} \end{bmatrix}_{n*l} \begin{bmatrix} (u_{11} + u_{21}) \\ \vdots \\ (u_{1l} + u_{2l}) \end{bmatrix}_{l \times 1} = \begin{bmatrix} \sum_{i=1}^{l} m_{xi1}(u_{1i} + u_{2i}) \\ \vdots \\ \sum_{i=1}^{l} m_{xin}(u_{1i} + u_{2i}) \end{bmatrix}$$

and the lagrangian vectors,

$$u_1 = (u_{11}, \ldots, u_{1l})^t, u_1^* = (u_{11}^*, \ldots, u_{1l}^*)^t, u_2 = (u_{21}, \ldots, u_{2l})^t, u_2^* = (u_{21}^*, \ldots, u_{2l}^*)^t \epsilon\, R^l$$

Then, we will have:

$$w = M^t(u_1 + u_2) - M^t(u_1^* + u_2^*) \quad ,$$

$$w = M^t(u_1 + u_2 - u_1^* - u_2^*)$$

Now,

$\frac{\partial L}{\partial b} = 0,$

$$\frac{1}{2}[2b] + \sum_{i=1}^{l} u_{1i}(-1) + \sum_{i=1}^{l} u_{1i}^{*}(1) + \sum_{i=1}^{l} u_{2i}(-1) + \sum_{i=1}^{l} u_{2i}^{*}(1) = 0$$

$$b = \sum_{i=1}^{l} u_{1i} + u_{2i} - u_{1i}^{*} - u_{2i}^{*}$$

Since $e = (1 \dots 1)^{t} \epsilon R^{l}$

$$\frac{\partial L}{\partial \xi_1} = 0 \Rightarrow \xi_1 = \frac{u_1}{c}$$

$$\frac{\partial L}{\partial \xi_2} = 0 \Rightarrow \xi_2 = \frac{u_2}{c}$$

$$\frac{\partial L}{\partial \xi_1^{*}} = 0 \Rightarrow \xi_1^{*} = \frac{u_1^{*}}{c}$$

$$\frac{\partial L}{\partial \xi_2^{*}} = 0 \Rightarrow \xi_2^{*} = \frac{u_2^{*}}{c}$$

Substituting the above relations in the lagrangian function and simplifying we can write the dual problem as a minimization problem in the following form:

$$\min_{(u_1,u_2,u_1^*,u_2^* \geq 0)} L = {}^1\!/_2 \left[(u_1^t + u_2^t - u_1^{*t} - u_2^{*t})(MM^t + ee^t)(u_1 + u_2 - u_1^* - u_2^*) \right]$$

$$+\frac{1}{2c}[u_1^t u_1 + u_2^t u_2 + u_1^{*t} u_1^* + u_2^{*t} u_2^*]$$

$$+\varepsilon(u_1^t + u_2^t + u_1^{*t} + u_2^{*t})e$$

$$-(u_1^t + u_2^t - u_1^{*t} - u_2^{*t})m_y - (u_1^t - u_2^t - u_1^{*t} + u_2^{*t})(s_y - \rho(s_x))$$

where,

$$m_y = (m_{y1}, \dots, m_{yl})^t \in R^l$$

$$s_y = (s_{y1}, \dots, s_{yl})^t \in R^l$$

$$\rho(s_x) = (\rho(s_{x1}), \dots, \rho(s_{xl}))^t \in R^l$$

Let,

$$G = [M \quad e] = \begin{bmatrix} {m_{x1}}^t & 1 \\ \dots & \\ {m_{xl}}^t & 1 \end{bmatrix}_{l \times (n+1)}$$

be an augmented matrix. Then the above dual problem as a minimization problem can be written as:

$$\min_{(u_1,u_2,u_3,u_4) \geq 0} \frac{1}{2}(u_1 \quad u_2 \quad u_3 \quad u_4)\, Q \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} - [r_1^t \quad r_2^t \quad r_3^t \quad r_4^t] \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

, where,

$$Q = \begin{bmatrix} \left(\frac{I}{c} + (GG^t)\right) & (GG^t) & -(GG^t) & -(GG^t) \\ (GG^t) & \left(\frac{I}{c} + (GG^t)\right) & -(GG^t) & -(GG^t) \\ -(GG^t) & -(GG^t) & \left(\frac{I}{c} + (GG^t)\right) & (GG^t) \\ -(GG^t) & -(GG^t) & (GG^t) & \left(\frac{I}{c} + (GG^t)\right) \end{bmatrix}_{(4l \times 4l)}$$

$$r_1 = \left(m_y + s_y - \rho(s_x) - \varepsilon e\right)_{l \times 1}$$

$$r_2 = (m_y - s_y + \rho(s_x) - \varepsilon e)_{l \times 1}$$

$$r_3 = (-m_y - s_y + \rho(s_x) - \varepsilon e)_{l \times 1}$$

$$r_4 = (-m_y + s_y - \rho(s_x) - \varepsilon e)_{l \times 1}$$

, i.e. we can write the problem as:

$$\min_{0 \le u \epsilon R^{4l}} \frac{1}{2} u^t Q u - r^t u$$

where,

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}_{(4l \times 1)} \quad \text{and} \quad r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}_{(4l \times 1)}$$

The Karush-Khun-Tucker (KKT) necessary and sufficient conditions for the dual problem will be:

$$0 \leq u \perp (Qu - r) \geq 0$$

However, since for any two real numbers or vectors **a** and **b**, we have:

$$0 \leq a \perp b \geq 0 \quad \Leftrightarrow \quad a = (a - \alpha b)_+ \,, \ \alpha > 0$$

is true, the KKT conditions will be equivalent to:

$(Qu - r) = ((Qu - r) - \alpha u)_+$ for $\alpha > 0$

The above condition on the solution **u** of the problem will lead to the following iterative scheme:

$$u^{i+1} = Q^{-1}\left[r + ((Qu^i - r) - \alpha\, u^i)_+ \right] \,, i = 1,2, \ldots$$

To apply the above iterative scheme we need to compute Q^{-1} at the beginning of the iteration of the algorithm.

Since,

$$w = M^t(u_1 + u_2 - u_1^* - u_2^*) \text{ and } b = e^t(u_1 + u_2 - u_1^* - u_2^*)$$

we have,

$$\begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} m^t(u_1 + u_2 - u_1^* - u_2^*) \\ e^t(u_1 + u_2 - u_1^* - u_2^*) \end{bmatrix} = \begin{bmatrix} M^t \\ e^t \end{bmatrix} (u_1 + u_2 - u_1^* - u_2^*)$$

where,

$$M = \begin{bmatrix} m_{x1}{}^t \\ \vdots \\ m_{x_l}{}^t \end{bmatrix}_{l*n} = \begin{bmatrix} m_{x_{11}} & \cdots & m_{x_{1n}} \\ \vdots & \ldots & \vdots \\ m_{x_{l1}} & \cdots & m_{x_{ln}} \end{bmatrix}$$

and the regression estimation function will become: for any $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n)\, \epsilon\, T(R)^n$ and $\tilde{x}_i = (m_{\tilde{x}_1},\, s_{\tilde{x}_1})$,

$$f(\tilde{x}) = \left(\begin{bmatrix} m_{\tilde{x}}^t & 1 \end{bmatrix} \begin{bmatrix} w \\ b \end{bmatrix}, \rho(\tilde{s_x}) \right)$$

$$= \left(\begin{bmatrix} m_{\tilde{x}}{}^t & 1 \end{bmatrix} \begin{bmatrix} m_{x_1} & \cdots & m_{x_l} \\ 1 & \ldots & 1 \end{bmatrix} (\mathrm{u}_1 + \mathrm{u}_2 - \mathrm{u}_1^* - \mathrm{u}_2^*), \rho(s_{\tilde{x}}) \right)$$

in which,

$$m_{\tilde{x}} = (m_{\tilde{x}1}, \ldots, m_{\tilde{x}n})^t\, \epsilon\, R^n$$

$$\rho(s_{\tilde{x}}) = \max\{s_{\tilde{x}1}, \ldots, s_{\tilde{x}n}\}\, \epsilon\, R$$

Following the approach of Mangasarian and Musicant (2001) [3] the linear SVR formulation can be extended to nonlinear SVR formulation in dual variables of the form:

$$\min_{0 \le u \epsilon R^{4l}} \frac{1}{2} u^t Q u - r^t u,$$

where,

$$Q = \begin{bmatrix} \frac{\mathrm{I}}{\mathrm{c}} + \mathrm{H} & -H \\ -H & \frac{\mathrm{I}}{\mathrm{c}} + \mathrm{H} \end{bmatrix}_{(4l \times 4l)}$$

such that,

$$H = \begin{bmatrix} K(GG^t) & K(GG^t) \\ K(GG^t) & K(GG^t) \end{bmatrix}_{(2l*2l)}$$

$$K(G, G^t) = \begin{bmatrix} k([m_{x1}{}^t \quad 1], [m_{x1}{}^t \quad 1]) & \cdots & k([m_{x1}{}^t \quad 1], [m_{xl}{}^t \quad 1]) \\ \vdots & \cdots & \vdots \\ k([m_{xl}{}^t \quad 1], [m_{x1}{}^t \quad 1]) & \cdots & k([m_{xl}{}^t \quad 1], [m_{xl}{}^t \quad 1]) \end{bmatrix}_{(l \times l)}$$

, and, $k(.,.)$ is the given kernel function.

Now, for any input sample $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n) \epsilon T(R)^n$ with $\tilde{x}_i = (m_{\tilde{x}_1}, s_{\tilde{x}_1})$, $\rho(s_{\tilde{x}}) = \max\{s_{\tilde{x}1}, \ldots, s_{\tilde{x}n}\}$, and $m_{\tilde{x}} = (m_{\tilde{x}1}, \ldots, m_{\tilde{x}n})^t \epsilon R^n$, by defining:

$$K\left([m_{\tilde{x}}{}^t \quad 1], \begin{bmatrix} m^t \\ e^t \end{bmatrix}\right) = (k([m_{\tilde{x}}{}^t \quad 1], [m_{x1}{}^t \quad 1]), \ldots, [m_{\tilde{x}}{}^t \quad 1], [m_{xl}{}^t \quad 1])_{1*l}$$

, where, $K(.,.)$ is the kernel function, the nonlinear regression estimation function is defined to be:

$$f(\tilde{x}) = (K\left([m_{\tilde{x}}{}^t \quad 1], \begin{bmatrix} m^t \\ e^t \end{bmatrix}\right)(u_1 + u_2 - u_1^* - u_2^*), \rho(s_{\tilde{x}}))$$

3.4 Experiments and Results

In our experiment we used "Matlab" [7] code to show the outputs of the algorithm.

For running the code, we fixed the following values for the parameters $c = 100$, $alpha\ \alpha = 1.9/c$, $epsilon\ \varepsilon = 0.01$, $mu = \mu = 1.0$ (mu is a positive parameter for the kernel function), $maxIter = 100$, and $tol = 1e-5$.

The following three datasets has been used to generate the fuzzy outputs:

Table 1. Dataset 1 with prediction

m_x	s_x	m_y	s_y
Training data			
2	0.5	4	0.5
3.5	0.5	5.5	0.5
5.5	1	7.5	1
7	0.5	6.5	0.5
8.5	0.5	8.5	0.5
10.5	1	8	1
Testing data			
11	0.5	10.5	0.5
12.5	0.5	9.5	0.5
Predicted results for the linear case			
-	-	8.96377	0.5
-	-	9.67762	0.5
Predicted results for the nonlinear case			
-	-	4.9482	0.5
-	-	1.1041	0.5

Table 2. Dataset 2 with prediction

m_x	s_x	m_y	s_y
Training data			
1	0.5	-1.6	0.5
3	0.5	-1.8	0.5
4	0.5	-1	0.5
5.6	0.8	1.2	0.5
7.8	0.8	2.2	1
10.2	0.8	6.8	1
11	1	10	1
Testing data			
11.5	1	10	1
12.7	1	10	1
Predicted results for the linear case			
-	-	8.44548	1
-	-	9.81566	1
Predicted results for the nonlinear case			
-	-	7.68446	1
-	-	2.31451	1

Table 3. Dataset 3 with prediction

m_x	s_x	m_y	s_y
Training data			
21	2.1	4	0.8
15	2.25	3	0.3
15	1.5	3.5	0.35
9	1.35	2	0.4
12	1.2	3	0.45
18	3.6	3.5	0.7
Testing data			
6	0.6	2.5	0.38
12	2.4	2.5	0.5
Predicted results for the linear case			
-	-	1.8092	0.6
-	-	2.71367	2.4
Predicted results for the nonlinear case			
-	-	0.100228	0.6
-	-	3.02517	2.4

3.5 Conclusion

In the beginning of this work, we proposed the concept of Support Vector Machines under the subject of Machine Learning and Pattern Recognition problems, the Support Vector Machines (SVM) which are successful models for producing computationally efficient and accurate solutions. We briefly described the mathematical solution introduced by Mangasarian and Musicant (2001) [3]. We studied in this work the Lagrangian Support Vector Regression to a regression problem with fuzzy input and fuzzy output data. Experiments were conducted on three available datasets.

Appendix I

1. The Linear Support Vector Regression Matlab code:

This code consists of three files:

a. **fsvr_main.m**: the main of the code.
b. **fsvr.m**: contains the training algorithm function.
c. **testfsvr.m**: contains the testing algorithm function and error extraction.

File 1: fsvr_main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Support Vector Regression using LSVM                                    %
% (coded by: Hasan Z. H. Alhajhamad)                                            %
%                                                                               %
% Jawaharlal Nehru Universiry, New Delhi 110067                                 %
% Last modified: 09/6/2010                                                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   FSVR Fuzzy Support Vector Regression algorithm
%   FSVR solves an advanced LSVM regression problem
%   using fuzzy inputs and fuzzy outputs


clc;
clear all;
close all;


file1 = fopen('results_fsvr.txt','w+');

trainfile= strcat('dataset/fsvrdataset3_train.txt');
testfile = strcat('dataset/fsvrdataset3_test.txt');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

train_data= load(trainfile);
[no_trainrows, no_traincolumns ] = size(train_data);

train_inputs= train_data(:, 1:no_traincolumns-2);
train_outputs= train_data(:, no_traincolumns-1:no_traincolumns);
```

```
%

test_data= load(testfile);
[no_testrows, no_testcolumns ] = size(test_data);

test_inputs= test_data(:, 1:no_testcolumns-2);
test_outputs= test_data(:, no_testcolumns-1:no_testcolumns);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


c= 100;            %%% c = 1, 10, 100, 1000, 10000, 100000
alpha = 1.9/c;   %%%
epsilon = 0.01;
maxIter = 100;
tol = 1e-5;

A = train_data;


%%%%%%%%%%%%%%% Training %%%%%%%%%%%%%%%%%%%%

start_training_time = cputime;

[iter, w, gamma] = fsvr(A, c, tol, maxIter...
    , alpha, epsilon);

finish_training_time = cputime;

training_time = finish_training_time - start_training_time;



%%%%%%%%%%%%%%% Testing %%%%%%%%%%%%%%%%%%%%

start_testing_time = cputime;

[error_rate_my, error_rate_sy, new_distance] = testfsvr(test_inputs,
test_outputs, w, gamma, file1);

finish_testing_time = cputime;


testing_time = finish_testing_time - start_testing_time;


%%%%%%%%%%%%%%%% Display & Write %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%   the results   %%%%%%%%%%%%%%%%%%
```

```
fprintf(file1, 'CPU processing time for FSVR training: %g \n',
training_time);

disp(sprintf('CPU processing time for FSVR training:'));

training_time



fprintf(file1, 'CPU processing time for FSVR training: %g \n',
testing_time);

disp(sprintf('CPU processing time for FSVR training:'));

testing_time



fprintf(file1, 'Error-rate for my: %g \n', error_rate_my);
fprintf(file1, 'Error-rate for sy: %g \n', error_rate_sy);
fprintf(file1, 'Max distance (error): %g \n', new_distance);

disp(sprintf('Error-rate is: '));

error_rate_my
error_rate_sy
new_distance


fclose(file1);  % File close
```

File 2: fsvr.m

```
function [iter, w, gamma] = fsvr(A, c, tol, maxIter...
    , alpha, epsilon)

%%%%%%%%%%%%%%%%%%%%%%%%

[a_rows,a_columns] = size(A);

train_inputs  = A(:, 1:a_columns-1);
train_outputs = A(:, 3:a_columns);

m_inputs = A(:, 1); % matrix contains all the Centers in the input
s_inputs = A(:, 2); % matrix contains all the Spreads in the input

[m_tuples, m_columns]= size(m_inputs);

m_outputs = A(:, 3); % matrix contains all the Centers in the output
s_outputs = A(:, 4); % matrix contains all the Spreads in the output


% Max(sx) is the same like sx because x has one attribute in our
dataset
s_maxinputs = s_inputs;


%%%%%%%%%%%%%%%%%%%%%%%%%

e = ones(a_rows, 1);

G = [m_inputs e];

GGT = G*G';

H = [ GGT GGT;...
      GGT GGT ];


% To find inverse( Q ) we use the following steps:

[Hrows, Hcolumns] = size(H);

I = speye(Hrows);

IcH = (I/c) + H;

%%%% In order to find inverse( I/c + 2H ) we suppose: %%%%%%%%%

% to find inverse( I/c + 4GGT ) = temp_var
I = speye(size(GGT));
```

```
temp_var = inv(I/c+ 4*GGT);

pstar = (I+2*c*GGT)*temp_var;
qstar = -2*c*GGT*temp_var;
rstar = -2*c*GGT*temp_var;
sstar = (I+2*c*GGT)*temp_var;

% so, inverse( I/c + 2H ) is temp_var2

temp_var2 = [pstar qstar;rstar sstar];


I = speye(2*m_tuples);

Q = [IcH -H;-H IcH];

ptilde = (I+c*H)*temp_var2;
qtilde = c*H*temp_var2;
rtilde = qtilde;
stilde = ptilde;

%%%%%% then, inverser( Q ) is:

Q_inv = [ptilde qtilde;rtilde stilde];


%%%%%%% To find ( r ) %%%%%%%%%%%%%%%%%%%%%%

r1= speye(a_rows, 1);
r2= speye(a_rows, 1);
r3= speye(a_rows, 1);
r4= speye(a_rows, 1);


r1=  m_outputs + s_outputs - s_maxinputs - epsilon*e;
r2=  m_outputs - s_outputs + s_maxinputs - epsilon*e;
r3= -m_outputs - s_outputs + s_maxinputs - epsilon*e;
r4= -m_outputs + s_outputs - s_maxinputs - epsilon*e;

r = [r1;r2;r3;r4];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

iter = 0;

% oldu is a state to start comparison with newu
oldu = 10*ones(4*m_tuples,1);
newu = ones(4*m_tuples,1); % newu= [1 1 1 1 ..]'

while(iter<maxIter&&norm(oldu-newu)>tol)

    oldu = newu;
    newu = Q_inv*(r+max((Q*oldu-r)-alpha*oldu,0));
```

```
    iter = iter+1;
end



%%%%%% We divide newu into 4 parts to find alpha1, alpha1star, alpha2,
%%%%%% and alpha2star

[u_tuples, u_columns] = size(newu);

u1_length = u_tuples/4;
u1 = newu( 1:u1_length,1);

u2_length = u1_length+u_tuples/4;
u2 = newu( u1_length+1:u2_length,1);

u3_length = u2_length+u_tuples/4;
u3 = newu( u2_length+1:u3_length,1);

u4_length = u3_length+u_tuples/4;
u4 = newu( u3_length+1:u4_length,1);


w = m_inputs'*(u1+u2-u3-u4);
gamma = e'*(u1+u2-u3-u4);
```

File 3: testfsvr.m

```
function [error_rate_my, error_rate_sy, new_distance] =
testfsvr(test_inputs, test_outputs, w, gamma, file1)

error_counter_my =0;
error_counter_sy =0;

distance = 0;

[nof_tuples, no_columns] = size(test_inputs);

for i=1 : nof_tuples

    predicted_outputs(i, 1) = dot(w,  test_inputs(i, 1)) + gamma;
    predicted_outputs(i, 2) = test_inputs(i, 2);

    error_counter_my = error_counter_my + (test_outputs(i, 1)-
predicted_outputs(i, 1))*(test_outputs(i, 1)-predicted_outputs(i, 1));
    error_counter_sy = error_counter_sy + (test_outputs(i, 2)-
predicted_outputs(i, 2))*(test_outputs(i, 2)-predicted_outputs(i, 2));

    new_distance = (test_inputs(i,1)-test_inputs(i,2))-
(predicted_outputs(i,1)-predicted_outputs(i,2));



    fprintf(file1, 'Predicted(%g , %g) ', predicted_outputs(i,1),
predicted_outputs(i,2));
    fprintf(file1, '- Exact(%g , %g)\n', test_outputs(i,1),
test_outputs(i,2));


    if(distance < new_distance)
        distance = new_distance;


end  %End for


error_rate_my = sqrt( error_counter_my/nof_tuples );
error_rate_sy = sqrt( error_counter_sy/nof_tuples );

end
```

2. The Nonlinear Support Vector Regression Matlab code:

This code consists of three files:

a. **fsvr_main.m**: the main of the code.
b. **fsvr.m**: contains the training algorithm function.
c. **K.m**: contains the kernel function.
d. **testfsvr.m**: contains the testing algorithm function and error extraction.

File 1: fsvr_main.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fuzzy Support Vector Regression using LSVM                                    %
% (coded by: Hasan Z. H. Alhajhamad)                                            %
%                                                                               %
% Jawaharlal Nehru Universiry, New Delhi 110067                                 %
% Last modified: 27/07/2010
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   FSVR Fuzzy Support Vector Regression algorithm
%   FSVR solves an advanced LSVM regression problem
%   using fuzzy inputs and fuzzy outputs

clc;
clear all;
close all;

file1 = fopen('results_fsvr.txt','w+');

trainfile= strcat('dataset/fsvrdataset3_train.txt');
testfile = strcat('dataset/fsvrdataset3_test.txt');

%

train_data= load(trainfile);
[no_trainrows, no_traincolumns ] = size(train_data);

train_inputs= train_data(:, 1:no_traincolumns-2);
train_outputs= train_data(:, no_traincolumns-1:no_traincolumns);

%

test_data= load(testfile);
[no_testrows, no_testcolumns ] = size(test_data);
```

```
test_inputs= test_data(:, 1:no_testcolumns-2);
test_outputs= test_data(:, no_testcolumns-1:no_testcolumns);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

c= 100;           %%% c = 1, 10, 100, 1000, 10000, 100000
alpha = 1.9/c;    %%%
epsilon = 0.01;
mu= 1.0;          %%% mu is a positive parameter for the kernel function
maxIter = 100;
tol = 1e-5;

[KM] = train_data;

e = ones(no_trainrows, 1);

G = [KM e];


%%%%%%%%%%%%%%% Training %%%%%%%%%%%%%%%%%%%%

start_training_time = cputime;

[iter, u] = fsvr(KM, G, c, tol, maxIter, alpha, epsilon, mu);

finish_training_time = cputime;

training_time = finish_training_time - start_training_time;


%%%%%%%%%%%%%%% Testing %%%%%%%%%%%%%%%%%%%%

start_testing_time = cputime;

[error_rate_my, error_rate_sy, new_distance] = testfsvr(u, test_inputs,
train_inputs, test_outputs, mu, file1);

finish_testing_time = cputime;


testing_time = finish_testing_time - start_testing_time;


%%%%%%%%%%%%%%%%% Display & Write %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%    the results    %%%%%%%%%%%%%%%%%%%


fprintf(file1, 'CPU processing time for FSVR training: %g \n',
training_time);

disp(sprintf('CPU processing time for FSVR training:'));
```

```
training_time


fprintf(file1, 'CPU processing time for FSVR training: %g \n',
testing_time);

disp(sprintf('CPU processing time for FSVR training:'));

testing_time

fprintf(file1, 'Error-rate for my: %g \n', error_rate_my);
fprintf(file1, 'Error-rate for sy: %g \n', error_rate_sy);
fprintf(file1, 'Max distance (error): %g \n', new_distance);

disp(sprintf('Error-rate is: '));

error_rate_my
error_rate_sy
new_distance


fclose(file1);  % File close
```

File 2: fsvr.m

```
function [iter, u] = fsvr(KM, G, c, tol, maxIter...
    , alpha, epsilon, mu)

%%%%%%%%%%%%%%%%%%%%%%%%

[KM_rows,KM_columns] = size(KM);

train_inputs  = KM(:, 1:KM_columns-1);
train_outputs = KM(:, 3:KM_columns);

m_inputs = KM(:, 1); % matrix contains all the Centers in the input
s_inputs = KM(:, 2); % matrix contains all the Spreads in the input

[m_tuples, m_columns]= size(m_inputs);

m_outputs = KM(:, 3); % matrix contains all the Centers in the output
s_outputs = KM(:, 4); % matrix contains all the Spreads in the output


% Max(sx) is the same like sx because x has one attribute in our
dataset
s_maxinputs = s_inputs;


%%%%%%%%%%%%%%%%%%%%%%%%%

time = cputime;

e = ones(m_tuples, 1);

GGT = K(G, G, mu);

%%%%%%%%%%%%%%% Finding H using K( G,G' ) %%%%%%%%%%%%%%%%%%%%

H = [ GGT GGT;...
      GGT GGT ];


% To find inverse( Q ) we use the following steps:

[Hrows, Hcolumns] = size(H);

I = speye(Hrows);

IcH = (I/c) + H;

%%%% In order to find inverse( I/c + 2H ) we suppose: %%%%%%%%%
```

```
% to find inverse( I/c + 4GGT ) = temp_var
I = speye(size(GGT));

temp_var = inv(I/c+ 4*GGT);

pstar = (I+2*c*GGT)*temp_var;
qstar = -2*c*GGT*temp_var;
rstar = -2*c*GGT*temp_var;
sstar = (I+2*c*GGT)*temp_var;

% so, inverse( I/c + 2H ) is temp_var2

temp_var2 = [pstar qstar;rstar sstar];


% I = speye(2*m_tuples);
I = speye(2*m_tuples);

Q = [IcH -H;-H IcH];


ptilde = (I+c*H)*temp_var2;
qtilde = c*H*temp_var2;
rtilde = qtilde;
stilde = ptilde;

%%%%%% then, inverser( Q ) is:

Q_inv = [ptilde qtilde;rtilde stilde];


%%%%%%% To find ( r ) %%%%%%%%%%%%%%%%%%%%%%

r1= speye(KM_rows, 1);
r2= speye(KM_rows, 1);
r3= speye(KM_rows, 1);
r4= speye(KM_rows, 1);


r1=  m_outputs + s_outputs - s_maxinputs - epsilon*e;
r2=  m_outputs - s_outputs + s_maxinputs - epsilon*e;
r3= -m_outputs - s_outputs + s_maxinputs - epsilon*e;
r4= -m_outputs + s_outputs - s_maxinputs - epsilon*e;

r = [r1;r2;r3;r4];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

iter = 0;

% oldu is a state to start comparison with newu
oldu = 10*ones(4*m_tuples,1);
```

```
newu = ones(4*m_tuples,1); % newu= [1 1 1 1 ..]'


while(iter<maxIter&&norm(oldu-newu)>tol)

    oldu = newu;
    newu = Q_inv*(r+max((Q*oldu-r)-alpha*oldu,0));
    iter = iter+1;
end

%%%%%% We divide newu into 4 parts to find u1, u2, u3, and u4

[u_tuples, u_columns] = size(newu);

u1_length = u_tuples/4;
u1 = newu( 1:u1_length,1);

u2_length = u1_length+u_tuples/4;
u2 = newu( u1_length+1:u2_length,1);

u3_length = u2_length+u_tuples/4;
u3 = newu( u2_length+1:u3_length,1);

u4_length = u3_length+u_tuples/4;
u4 = newu( u3_length+1:u4_length,1);


u = [u1 u2 u3 u4];
```

File 3: K.m

```
%%%%%%       Kernel Function       %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [K_output]= K(A, B, mu)

%In case of training: A = G , B = G
%In case of testing: A = G , B = test_inputs

for i = 1:size(A,1)

    for j = 1:size(B,1)

        K_output(i,j) = exp(-mu * norm(A(i,:)-B(j,:)) );

    end
end
```

File 4: testfsvr.m

```
function [error_rate_my, error_rate_sy, new_distance] = testfsvr(u,
test_inputs, train_inputs, test_outputs, mu, file1)

error_counter_my =0;
error_counter_sy =0;

distance = 0;

count = 0;

[nof_tuples, no_columns] = size(train_inputs);
[nof_testtuples, no_testcolumns] = size(test_outputs);

K(train_inputs(:,1), test_inputs(:,1), mu)


predicted_mx = K(train_inputs(:,1), test_inputs(:,1), mu)' * (
u(:,1)+u(:,2)-u(:,3)-u(:,4) );

predicted_mx

predicted_sx = test_inputs(:, 2); % maximum(sx)

for i=1 : size(test_outputs)


    fprintf(file1, 'Predicted(%g , %g) ', predicted_mx(i),
predicted_sx(i));
    fprintf(file1, '- Exact(%g , %g)\n', test_outputs(i,1),
test_outputs(i,2));

    error_counter_my = error_counter_my + (test_outputs(i, 1)-
predicted_mx(i))*(test_outputs(i, 1)-predicted_mx(i));
    error_counter_sy = error_counter_sy + (test_outputs(i, 2)-
predicted_sx(i))*(test_outputs(i, 2)-predicted_sx(i));

    new_distance = (test_inputs(i,1)-test_inputs(i,2))-
(predicted_mx(i)-predicted_sx(i));

    if(distance < new_distance)
        distance = new_distance;

end


error_rate_my = sqrt( error_counter_my/nof_tuples );
error_rate_sy = sqrt( error_counter_sy/nof_tuples );

end
```

References:

1. Cristianini, N., & Shawe-Taylor, J., 2000, "*An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*". Cambridge University Press.
2. Gunn, S. R., 1998, "*Support Vector Machines for Classification and Regression*", University of Southampton.
3. Mangasarian O.L. and Musicant D.R., 2001, "*Lagrangian Support Vector Machines*", Journal of Machine Learning Research, vol.1, pp.161-77.
4. Vapnik, V. N., 1995. "*The Nature of Statistical Learning Theory*". Springer, New York.
5. (Electronic Version): StatSoft, Inc., 201, "*Electronic Statistics Textbook*", Tulsa, OK: StatSoft. WEB: http://www.statsoft.com/textbook/support-vector-machines/, (Printed Version): Hill, T. & Lewicki, P.,2007, "*STATISTICS Methods and Applications*", StatSoft, Tulsa, OK.
6. Yan H.S., and Xu D., 2007. "*An approach to estimating product design time based on fuzzy v-support vector machine*", IEEE Transactions on Neural Networks, 2007, 18(3), pp. 721-731.
7. MATLAB 7.0.1., 2004, "*User's Guide*". The Math Works, Inc., Natik, MA 01769,
8. Vapnik V. N., 2000, "*The Nature of Statistical Learning*", Springer, New York.