WP 5.6: Support Vector Machine Imputation Description of Algorithms

H. Mallinson, A. Gammerman

Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX
{hugh,alex}@cs.rhul.ac.uk

Contents

1 Introduction **3**

2 The Support Vector Machine Imputation Harness **4**
2.1 Training data must be fully observed 4
2.2 Training Data Heuristics 5
2.3 Test data treatment 5
2.4 Iteration 6

3 Overview of Support Vector Machines **6**

4 Kernel Functions **7**
4.1 Commonly used kernels 9

5 Optimisation Issues **9**
5.1 Commercial Toolboxes 10
5.2 Chunking, Decomposition and SMO 11

A Formal Derivation of Support Vector Machines **11**
A.1 SVM for Classification 12
A.2 SVM for Regression Estimation 14

B Mercer's Theorm **16**

1 Introduction

In this document we present a description of the Support Vector Machine algorithm for imputation. We choose to solve the 'imputation' problem by extracting a number of standard classification and regression problems from the data set that must be completed. In other words, we choose to manipulate the problem so that it fits the algorithm. Section 2 gives details of this approach.

In section 3 we give an overview of the algorithm in a 'no-frills' classification formulation. This reveals more clearly the SVM's three interlinked elements: 1) a linear algorithm having a *dual form*, 2) a projection of the training data to a high-dimensional space prior to application of the linear algorithm, and 3) a capacity measure or regularisation term. An appendix contains a formal derivation of the SVM, firstly for classification and secondly for regression.

The fourth section describes kernel functions. These supply the non-linear projection of the data. We explain how this projection can be left implicit. A second appendix contains Mercer's Theorem, which gives conditions that must be satisfied for a function to be kernel.

Section 5 discusses Optimisation Theory. The training of the SVM requires the solution of a quadratic programme.

The bibliography lists publications that go into full detail. Vapnik's key work [4] is listed, and also a short introductory text by Cristianini and Shawe-Taylor [7]. Two shorter tutorials are [12] for SVM regression and [16] for classification. The web-site, **www.kernel-machines.org** contains a large repository of documents concerning Support Vector Machines and related algorithms.

2 The Support Vector Machine Imputation Harness

The application of the SVM algorithm to imputation is achieved through manipulation of the problem. No change to the core algorithm is made. For a data set $A^{n\times m}$, we produce p [1] interlinked prediction 'models'. The problems are interlinked in the following sense; values on variable X_j that have been predicted by one SVM may subsequently be used as training data by another SVM trained to predict variable X_k. To aid in the presentation, we will refer to an SVM trained to impute the j^{th} variable as SVM_j.

Any model of the form $y = f(x_1, ...x_m; \theta)$ for the conditional expectation, $E(Y|X_1, ...X_m)$. can be applied using the same 'prediction-approach'. The method presently exploits no special feature of the algorithm used[2].

The steps that are described in this section are only required for problems in which more than one variable is missing. When a single variable is missing the problem will be treated straightforwardly as a classification or regression task.

2.1 Training data must be fully observed

SVMs can be trained to predict only one variable X_j at a time. For the sake of clarity we will use the subscript j to denote the variable currently being imputed, known as the *target* variable. Other variables, known as *input* variables will be denoted $\mathbf{X_k}$ together, or $X_{k_i:k_i\in\{1...n\}\setminus j}$ individually.

Once X_j is chosen, a dependency is learnt by training on fully observed pairs, $(\mathbf{x_k}, \mathrm{x_j})$. When the parameters of the machine, SVM_j have been 'learned', the imputations are obtained by applying the function to those units lacking X_j, $SVM_j(\mathbf{x_k}) = \hat{\mathrm{x}}_\mathrm{j}$.

Other variables X_{k_i} may be incomplete in a unit however, not just X_j. We must consider how to handle this additional missingness when it occurs in either training or testing units. Our decision on how to handle the former will affect how we handle the latter.

[1] for a dataset with p missing variables.

[2] apart from its superior predictive accuracy.

2.2 Training Data Heuristics

We assemble those units which are complete in X_j, we will denote this set $trainset_j$. We can choose to estimate, or discard missing input variables from this set. An alternative is to discard those units that are missing input values. In the test phase however all units lacking X_j must be treated.

We investigate three 'dirty' methods. The first heuristic: **heuristic1** assumes that a sufficiently large subset of the data exists which is complete. This subset alone is used for training, incomplete units are discarded.

This approach makes the assumption that the missing data patterns are 'missing completely at random'(MCAR).

A second approach **heuristic2** completes those X_{k_i} variables that are missing. No data or variables are discarded. We use the feature mean or mode $\hat{X_{k_i}} = \mu_{k_i}$. This approach assumes that the input variables are themselves correlated. Perturbing one variable by introducing its mean or mode hence does not distort the relationship with the target variable.

We note that training data can be weighted variably in the SVM, through setting a different C_i for each unit. Lower values would signify a unit with more 'pseudo' values.

A third approach **heuristc3** discards any missing training variables, X_{k_i}, using a subset of the $m-1$ measurements on each unit, to build the model. These variables are also discarded from $testset_j$.

In summary, the optimal heuristic must be ascertained for each data set and for each missing variable within the dataset, by validation.

2.3 Test data treatment

Once SVM_j has been trained we assemble all units with variable X_j missing, denoted $testset_j$. This set may exhibit missingness on other variables also. We order our approaches according to the approach chosen for $trainset_j$.

Testing for heuristics 1 and 2 If the model was built using method 1 or 2, any missing X_{k_i} in the $testset_j$ will need to be estimated for SVM_j to be applied. We use a feature mean or mode, depending on the type of X_{k_i}.

Testing for heuristic 3 If heuristic3 was used for $trainset_j$, application of the model will require deletion of the same variables. If the $testset_j$ exhibits missingness also in other variables, these must be estimated (before the support vector machine can give predictions.

2.4 Iteration

If heuristic2 was applied to $trainset_j$ we may be able to improve the quality of imputations by iterating. When X_{k_i} is subsequently imputed we may use this value in $trainset_j$ to reestimate SVM_j. This approach resembles EM in some respects.

3 Overview of Support Vector Machines

The support vector machines [4] is a new tool for prediction and function estimation. Given a training set of input-output pairs, $\{(x_1, y_1), (x_2, y_2)$ $\ldots (x_n, y_n)\} \in \mathbb{R}^n \times \pm 1$, the SVM algorithm estimates a function (f) such that, for (x, y) drawn according to the same distribution, $P(X, Y)$ as the training set, $f(x) = y$. The function describes a non-linear decision surface that separates the two classes of data, denoted by +1 or -1 (the label y).

The SVM can be adapted to perform multi-class classification $(y \in 1, 2, ..N)$ and regression $(y \in \mathbb{R})$. The SVM can be most clearly understood, in its classification form, as an extension of Rosenblatt's perceptron algorithm. The perceptron learns a linear discriminant function, $f(x) = sign(\langle \mathbf{w} \cdot x_i \rangle + b)$. $(\mathbf{w}, b)$ are the parameters that are to be estimated.

The SVM extends this algorithm in two respects. It introduces non-linear decision surfaces, and a means of avoiding overfitting. The first is achieved through a non-linear projection of the data into a higher dimensional feature space prior to estimation of the linear discriminant. The linear model learned in this space is equivalent to a non-linear model in the input space. For example, a point $\mathbf{x} \in \mathbb{R}^3$ with position, $\mathbf{x} = (x_1, x_2, x_3)$ could be projected to the new space $\mathbb{R}^5$ with coordinates $\mathbf{x}' = (x_1, x_2, x_3, c_1 x_1^2, c_2 x_2^2)$. The new features are more 'exotic' functions of the original attributes. The SVM algorithm finding a linear discriminant function in this feature space is equivalent to the estimation of a polynomial discriminant in the original space $\mathbb{R}^3$.

$$f'(\mathbf{x}') = \langle \mathbf{w}' \cdot \mathbf{x}' \rangle + b = w_1 x_1 + w_2 x_2 + ...w_4 c x_1^2 + w_5 c_2 x_2^2 + b$$

The second extension of the perceptron algorithm concerns capacity control or regularisation. The problem of overfitting is well understood; if the projection introduce enough new features, we would learn the noise on the data, and not the underlying dependency. The SVM achieves good generalisation by choosing a discriminant function that maximally separates the

two classes in the feature space. The Euclidean distance between the closest point and the decision surface is known as the margin. This maximisation of the margin acts as a form of regularisation. This is due to constants c_i that are associated with the added dimensions. (c_1 and c_2 in the example above). Their effect is to penalise discriminants that exploit the new features.

This SVM algorithm can be formulated in such a way that it only requires the calculation of the dot product $\langle \cdot \rangle$, between training points to find $\mathbf{w}$. Moreover this parameter vector is always expressable as a weighted sum of training points, $\mathbf{w} = \sum \alpha_\mathbf{i} \mathbf{x_i}$. Hence in test or prediction phase, test points also only occur as dot products: $f(\mathbf{z}) = sign(\sum \alpha_i \langle \mathbf{x}_i \cdot \mathbf{z} \rangle + b)$. An algorithm with this feature is known as having a *dual form*. The SVM algorithm exploits the dual form by finding functions that perform the non linear projection described above, and the dot product in one step. These 'kernel functions' k(x,y) equate $\langle \phi(x), \phi(y) \rangle$. The positions of the points in the feature space are in fact never calculated. There are many choices of kernel function, some of which have implicit feature spaces of infinite dimension. Such feature spaces provide a large number of models. Maximising the margin however is able to effectively choose the model with lowest capacity.

Support vector machines are motivated by bounds on the generalisation error. They can be understood as linear algorithms working in an implicit high-dimensional feature space, with an objective function to be minimised that is quadratic and convex. Overfitting is avoided by controlling a dimension-independent parameter, the margin, which can be calculated using only the dot products of the training set data. The solution of the convex quadratic programme results in a hypothesis that is often *sparse*. I.e. $f(z) = \sum_{i=1}^{n} \alpha_i k(x_i, z)$, and many α are zero.

4 Kernel Functions

The concept of a kernel function was introduced in the overview in section 2. There we considered expanding the representation of each data point by adding features (quadratic functions) of the original attributes. We view this new representation as a 'projection'; $x \rightarrow \phi(x)$ of the input vector into a new space.

The kernel function projects two points into the new space and calculates the dot product in that space, *in one step*. The new position of each data

point remains implicit.

$$K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle$$

If the projection had been into a much higher-dimensional space, this 'trick' can provide great computational efficiency.

To make the concept clear we give an example of a polynomial kernel function and calculate the (usually) implicit feature vectors, $\phi(x)$. Mercer's Theorem, given in the appendix specifies the general conditions that must hold for a function to be a kernel.

We will consider a projection of two data points $\mathbf{x}, \mathbf{y}$,from a two to a 5 dimensional space. (In fact all the points exist in a four-dimensional subspace, as the fifth feature is constant).

$$\mathbf{x} = (x_1, x_2)' \rightarrow \phi(\mathbf{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, 1)'$$

$$\mathbf{y} = (y_1, y_2)' \rightarrow \phi(\mathbf{y}) = (\sqrt{2}y_1, \sqrt{2}y_2, y_1^2, y_2^2, 1)'$$

The dot product in this feature space is,

$$\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{y}) \rangle = 2x_1y_1 + 2x_2y_2 + x_1^2y_1^2 + x_2^2y_2^2 + 1$$

$$= (x_1y_1 + x_2y_2 + 1)^2 = (\langle \mathbf{x} \cdot \mathbf{y} \rangle + 1)^2$$

If our algorithm only requires dot products of the data points (a dual algorithm), the projection may remain implicit, as we need only calculate this last term. We also note that the number of tunable parameters remains unchanged whichever kernel we chose, unlike a neural network where the number of parameters is a function of the architecture.

All functions of the form, $k(x, y) = (\langle x \cdot x \rangle + 1)^d$ where d is specified by the user, can be used as kernels. A theorem from functional analysis provides a general characterisation of all functions that can do service as a kernel. This theorem, known as Mercer's Theorem, is given in an appendix. It allows us to use kernels, for example the 'RBF' kernel, with an implicit feature space of infinite dimension.

4.1 Commonly used kernels

A full list of kernels may be found at **http//:www.clrc.svm.rhul.ac.uk**. We describe those used so far in experiments for the Euredit project.

- The simplest kernel, supplying a linear solution, is the dot product for the input space:-

$$k(x, y) = \langle x \cdot y \rangle$$

- Polynomial kernels are of the form:-

$$k(x, y) = (\langle x \cdot x \rangle + 1)^d$$

  where d is user defined. As d gets larger, the SVM is able to supply higher capacity models.

- Radial basis function kernels are of the form:-

$$k(x, y) = exp(-\frac{\|x - y\|^2}{\sigma^2})$$

  where σ is user defined. This kernel has an infinite dimensional feature space. As σ gets larger the capacity gets lower.

5 Optimisation Issues

Both classification and regression SVMs are estimated by solving a particular class of optimisation problem known as a *quadratic programme* (QP). These comprise of a convex quadratic objective function which must be maximised subject to linear constraints.

QPs are solved by defining a function (known as a Lagrangian) which incorporates information about both the objective function and the constraints, and whose stationarity can be used to detect solutions of the constrained problem. The Lagrangian is the sum of the objective function and a weighted some of the constraints:-

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) + \sum_{i=1}^{m} \alpha_i h_i(\mathbf{w})$$

Here $f(\mathbf{w})$ is the objective function, the constraints are $h_i(\mathbf{w})$ and the α are known as Lagrange multipliers. We seek the values of the α at the minimum of L. For an SVM,

$$L(\mathbf{w}, \alpha) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1]$$

The constraints of a Lagrangian function can sometimes by simplified by transposing to a *Dual Form*. We impose stationarity, and observe that, at the extremum,

$$\frac{\delta L}{\delta \mathbf{w}} = \mathbf{w} - \sum y_i \alpha_i \mathbf{x}_i = 0$$

$$\frac{\delta L}{\delta b} = y_i \alpha_i = 0$$

Hence $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$. The weight vector can be written as a linear combination of the training points. Substituting back into the Lagrangian function;

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle - \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + \sum_{i=1}^{n} \alpha_i$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

These α coefficients are the parameters of the model we seek.

5.1 Commercial Toolboxes

QPs are relatively well understood problems and a number of commercial packages exist for solving them. The Royal Holloway implementation allows the user to choose between the LOQO [10] , MINOS [11] and BOTTOU routines. The first two are commercially available general-purpose QP solvers. The last is a routine specifically created for the SVM at Royal Holloway. However, it should be noted that features of the SVM problem can be exploited by specialist software increasing the speed considerably, particularly in the case of regression.

We briefly recount descriptions of these packages given in [12]
"MINOS uses a reduced gradient algorithm in conjunction with a quasi-Newton algorithm. The constraints are handled by an active set strategy and

feasibility is maintained throughout the process. The variables are classified as basic, superbasic, and nonbasic; at the solution, the basic and superbasic variables are away from their bounds. The null space is spanned by a matrix that is constructed from the coefficient matrix of the basic variables by using a sparse factorization. On the active constraint manifold ,a quasi-Newton appprosimation to the reduced Hessian is maintained.' More briefly:
"LOQO uses a primal-dual logarithmic barrier algorithm with a predictor-corrector step."

5.2 Chunking, Decomposition and SMO

SVMs are limited by the size of the *Hessian matrix* $K^{n \times n} = k(x_i, x_j)$, which must be held in memory. This is the matrix containing the dot products of all training points, in the feature space. Problems of more than a 1000 data points become unmanageable on most systems. An approach that has been proposed for handling larger datasets called *chunking* takes advantage of 'active set' methods in optimisation. These simplify the problem by temporarily discarding inactive constraints. An arbitrary subset of the data is selected and the SVM is trained. The support vectors are retained and added to them are a set of M points from the unsampled data that most violate the Karush-Kuhn Tucker conditions.

This procedure is iterated, initialising α for each new sub-problem with the values output from the previous stage, halting when some stopping criterion is satisfied.

SMO, (Sequential Minimal Optimisation) was proposed by Platt [14] in 1999. This algorithm selects subsets of just 2 data points and optimises the target function with respect to them. It has been reported to be several orders of magnitude faster and exhibit better scaling properties than classical chunking.

A Formal Derivation of Support Vector Machines

Suppose we are given a set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_T$ and we have a supervisor which gives us a label y_t, $t = 1, \ldots, T$, for each of the given vectors. Our

problem is to construct a learning machine which minimises some measure of discrepancy between its prediction $\hat{y}$ and the label y of a new example x.

In the case of classification the label has only two values: $y_t \in \{-1, 1\}$; that is, each vector x_t belongs to one of two classes. The loss function in this case measures the number of incorrectly classified vectors: the loss suffered at trial t is

$$L(y_t, \hat{y}_t) = \begin{cases} 0, & \text{if } y_t = \hat{y}_t; \\ 1, & \text{otherwise.} \end{cases}$$

In the case of regression estimation the label y_t is a real value: $y_t \in \mathbb{R}$. Naturally, the loss function L is defined differently. For example, sometimes it is useful to define it as the cumulative square loss,

$$L(y_t, \hat{y}_t) = (y_t - \hat{y}_t)^2,$$

and sometimes as cumulative absolute loss,

$$L(y_t, \hat{y}_t) = |y_t - \hat{y}_t|,$$

where y_t is the supervisor's assignment and $\hat{y}_t$ is the predicted value.

A.1 SVM for Classification

Let us suppose that the training data consists of

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T),$$

where $\mathbf{x}_t \in \mathbb{R}^n$ and $y_t \in \{-1, 1\}$, $t = 1, \ldots, T$. We can try to separate the data by a hyperplane

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0, \tag{1}$$

where $\mathbf{w}$ are weights and b is a coefficient.

The intuition behind the SVM approach is that a separating hyperplane is **optimal**, if it minimizes the number of errors and maximizes the distance between the hyperplane and the closest (to the hyperplane) vectors.

Formally, it can be expressed as the following quadratic optimization problem: minimize the quadratic form

$$L(\mathbf{w}, \xi) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \left(\sum_{t=1}^{T} \xi_t \right) \tag{2}$$

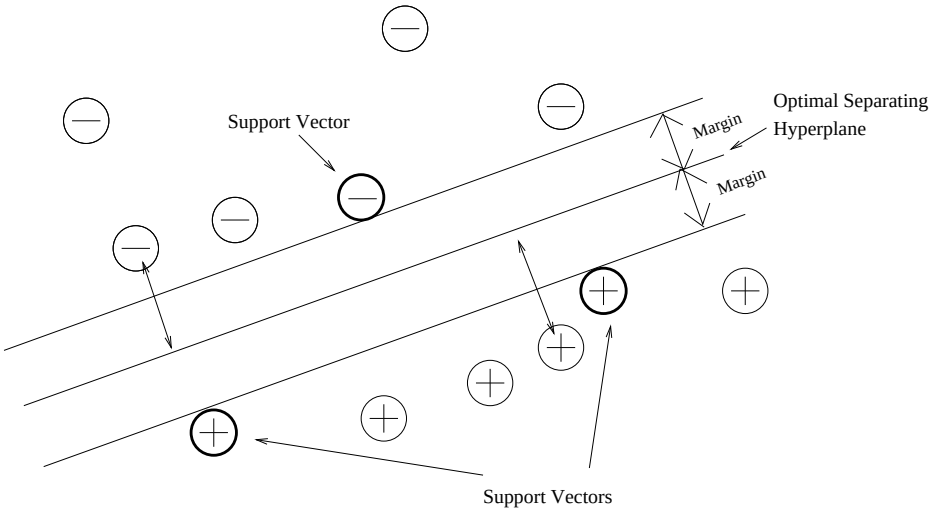

Figure 1: classification

under the linear constraints

$$y_t\left(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b\right) \geq 1 - \xi_t,\ t = 1, \ldots, T, \tag{3}$$

$$\xi_t \geq 0,\ t = 1, \ldots, T. \tag{4}$$

Here the "slack variables" ξ_t represent the magnitude of error.

The first term in (2) maximizes the margin of the separating hyperplane, while the second term controls, using a coefficient C, the number and magnitude of errors in the training set that we are prepared to ignore. The higher the value of C, the fewer errors are accepted; when $C = \infty$ the data must be linearly separated (otherwise, the loss will be infinite).

Figure 1 illustrates the SVM approach to classification.

The Dual Representation of Classification

Using Lagrange multipliers the original setting of the problem can be replaced by the "dual" setting: maximize the quadratic form

$$\sum_{t=1}^{T} \alpha_t - \frac{1}{2} \sum_{t,s=1}^{T} y_t y_s \alpha_t \alpha_s K(\mathbf{x}_t, \mathbf{x}_s) \tag{5}$$

under the "box" constraints

$$0 \leq \alpha_t \leq C, \;\; t = 1, 2, \ldots, T. \tag{6}$$

Here, K is a 'kernel function' and the values α_t, $t = 1, \ldots, T$, are Lagrange multipliers, which play the role of the weights for each vector. For each non-zero weight α_t the corresponding vector $\mathbf{x}_t$ is called a support vector. Using kernels K makes it possible to construct *nonlinear* separating surfaces.

In this dual form the coefficients $\mathbf{w}$ of the hyperplane are not used explicitly; they are replaced by a linear combination of hyperplanes. The dual approach allows the construction of hyperplanes in the high dimensional space.

If $\mathbf{x}$ is a new vector, then the prediction $\hat{y}$ will be:

$$\hat{y} = \text{sign}\left(\sum_{t=1}^{T} \alpha_t y_t K(\mathbf{x}_t, \mathbf{x}) + b\right). \tag{7}$$

(We omit the details of computing the coefficient b in the dual setting.)

After the problem is transformed to a quadratic optimization problem it can be solved using standard software packages.

A.2 SVM for Regression Estimation

The problem is to find a regression function

$$y = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

that would have the best fit for the new examples. Let us fix some tolerance limit (or "insensitivity zone" $\varepsilon > 0$) so that errors of less than ε will not be punished. The following loss function can be used:

$$L(y_t, \hat{y}_t) = |y_t - \hat{y}_t|_\epsilon = \begin{cases} 0, \text{ if } |y_t - \hat{y}_t| \leq \epsilon, \\ |y_t - \hat{y}_t| - \epsilon, \text{ otherwise.} \end{cases}$$

The regression estimation problem can now be formulated in the following way: find the minimum of the objective function

$$\frac{1}{2}\langle w \cdot w \rangle + C\left(\sum_{t=1}^{T}(\xi_t^* + \xi_t)\right) \tag{8}$$

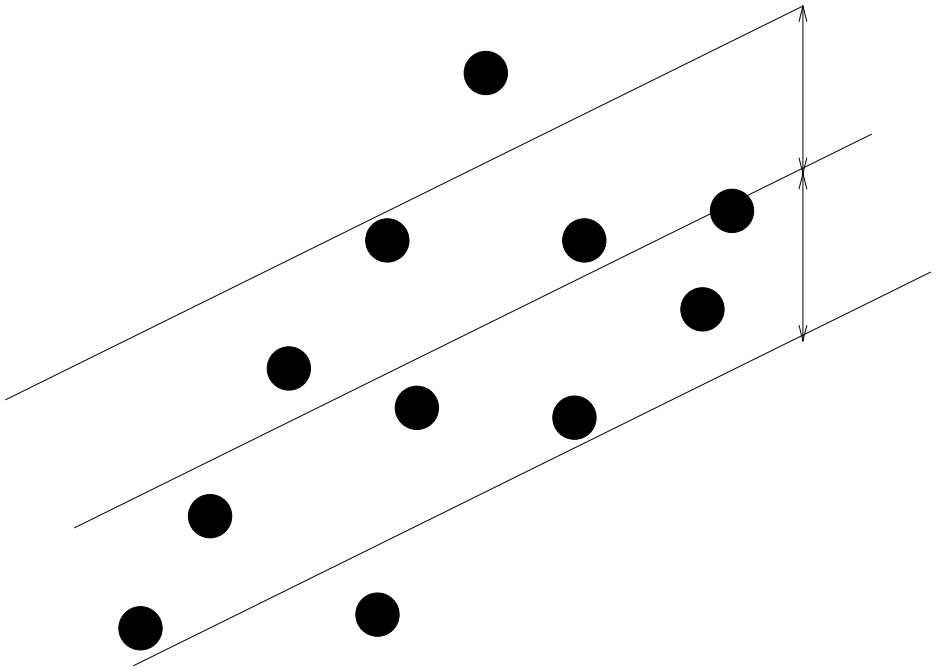

Figure 2: Regression estimation

subject to the constraints

$$y_t - \langle w \cdot \mathbf{x}_t \rangle - b \leq \epsilon + \xi_t^*, \; t = 1, \ldots, T, \tag{9}$$

$$\langle w \cdot \mathbf{x}_t \rangle + b - y_t \leq \epsilon + \xi_t, \; t = 1, \ldots, T, \tag{10}$$

$$\xi_t^* \geq 0, \; t = 1, \ldots, T, \tag{11}$$

$$\xi_t \geq 0, \; t = 1, \ldots, T. \tag{12}$$

Roughly, the algorithm finds the flattest function (by minimizing the norm of w) which passes within ε distance of the training examples.

Figure 2 illustrates this approach.

The Dual Representation of Regression Estimation

As in the case of classification the problem can be represented using Lagrange multipliers. The problem becomes: find α and α^* such that the expression

$$-\epsilon\sum_{t=1}^{T}(\alpha_t^*+\alpha_t)+\sum_{t=1}^{T}y_t(\alpha_t^*-\alpha_t)-\frac{1}{2}\sum_{t,s=1}^{T}y_ty_s(\alpha_t^*-\alpha_t)(\alpha_s^*-\alpha_s)K(\mathbf{x}_t,\mathbf{x}_s) \quad (13)$$

is maximized subject to the box constraints

$$0 \le \alpha_t^* \le C, \;\; t=1,2,\ldots,T, \quad (14)$$

$$0 \le \alpha_t \le C, \;\; t=1,2,\ldots,T, \quad (15)$$

and the constraint

$$\sum_{t=1}^{T}(\alpha_t^*-\alpha_t)=0.$$

The solution to this dual problem gives the following prediction $\hat{y}$ on a new example $\mathbf{x}$:

$$\hat{y}=\sum_{t=1}^{T}(\alpha_t^*-\alpha_t)K(\mathbf{x}_t,\mathbf{x})+b. \quad (16)$$

Similarly to the case of classification, K enables us to perform *nonlinear* regression estimation.

The desired regression function should pass through (or near) the training points (at least if we assume C to be large). When the free parameter ε is small, the regression function is not very smooth; when $\varepsilon = 0$, the function must pass through all training examples. When ε increases the regression function becomes smoother, and the number of support vectors decreases. Thus, the tradeoff between the accuracy of approximation and the complexity of approximation is controlled.

Kernel methods share the following elements: they transform the original problem into a high dimensional space using the kernel technique, and control dimension-independent capacity measures to avoid overfitting.

B Mercer's Theorm

We mention this theorem briefly, which gives the conditions for a function to be a kernel. Let X be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous

symmetric function such that the integral operator $T_K : L_2(X) \rightarrow L_2(X)$,

$$(T_(f))(\cdot) = \int_X K(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

is positive, that is

$$\int_{X \times X} K(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0$$

for all $f \in L_2(X)$. Then we can expand $K(\mathbf{x}, \mathbf{z})$ in a uniformly convergent series in terms of $T'_K s$ eigen-functions $\phi_j \in L_2(X)$, normalised in such a way that $\|\phi_j\|_{L_2} = 1$, and positive assoicated eigenvalues $\lambda_j > 0$,

$$K(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\inf} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z})$$

A corollary of this theorem is that , for any finite subset of X, the corresponding matrix $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, must be positive semi-definite.

References

[1] Bankier. Nearest neighbour imputation methodology. Technical report, Canadian Office of Statistics, 1998.

[2] R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines: Estimating the bayes point in kernel space, 1999.

[3] J. L. Schafer. *Analysis of Incomplete Multivariate Data*. Number 72 in Monographs on Statistics and Applied Probability. Chapman and Hall, London, 1997. ISBN: 0412040611.

[4] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[5] V. Gammerman, A. J. Vovk and V. Vapnik. Learning by transduction. In G. Cooper and S. Moral, editors, *Uncertainty in Artificial Intelligence Proceedings of the 14th Conference*, pages 148–155, 1998.

[6] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall, 1997.

[7] N. Cristianini and Shawe-Taylor J. *An Introduction to Support Vector Machines*. CUP, 2000.

[8] Volker Tresp, Subutai Ahmad, and Ralph Neuneier. Training neural networks with deficient data. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 128–135. Morgan Kaufmann Publishers, Inc., 1994.

[9] Zoubin Ghahramani and Michael I. Jordan. Learning from incomplete data. Technical Report AIM-1509, 1994.

[10] R. J. Vanderbei. Loqo: An interior point code for quadratic programming. Technical Report SOR-94-15, Statistics and Perations Research, Princeton University, 1994.

[11] B. A. Murtagh and M. A. Saunders. Minos 5.1 user's guide. Technical Report SOL-83-20R, Stanford University, CA, USA, 1983.

[12] Alex J. Smola and Bernhard Scholkopf. A tutorial on support vector regression. Technical Report NC2-TR-1998-030, GMD, 1998.

[13] Y. Bengio and F. Gingras. Recurrent neural networks for missing or asynchronous data. *Y. Bengio and F. Gingras, Recurrent neural networks for missing or asynchronous data, in Advances in Neural Information Processing Systems (M. Mozer, D. Touretzky, and M. Perrone, eds.), vol. 8, The MIT Press, 1996.*, 1996.

[14] J. Platt. Fast training of support vector machines using sequential minimal optimzation. *Advances in Kernel Methods - Support Vector Learning*, 1996.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39:1 – 38, 1977.

[16] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Knowlege Discovery and Data Mining*, pages 100 – 130, 1998.

[17] G. E. Box and G. C. Tiao. *Bayesian Inference in Statistical Analysis*. J. Wiley and Sons, New York, wiley classics library edition edition, 1992.

[18] D. B. Rubin. http://www.statsol.ie. SOLAS 2.0.

[19] D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. Wiley Series in Probability and Mathematical Statistics. Wiley, New York, 1987. ISSN: 0271-6232.

[20] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. Wiley, New York, 1987.