Support Vector Machines

Chapter 6 (and 5)

April 10, 2003

T.P. Runarsson (tpr@hi.is) and S. Sigurdsson (sven@hi.is)

Introduction

Support Vector Machines (SVM) are:

- efficiently trained *linear learning machines* introduced in chapter 2,
- in *kernel induced feature spaces*, described in chapter 3,
- while respecting the insights provided by the *generalization theory* in chapter 4.

The support vector machine:

- is efficient in the sense that it can deal with sample sizes of the order $100\ 000$, and
- generates "good" hyperplanes in the sense of optimizing the generalization bounds described in chapter 4. Different bounds motivate different algorithms:
  1. minimize the number of support vectors,
  2. the margin distribution,
  3. optimize the maximal margin.

Let's start with an algorithm motivated by finding the maximum margin hyperplane.

Finding the maximum margin hyperplane

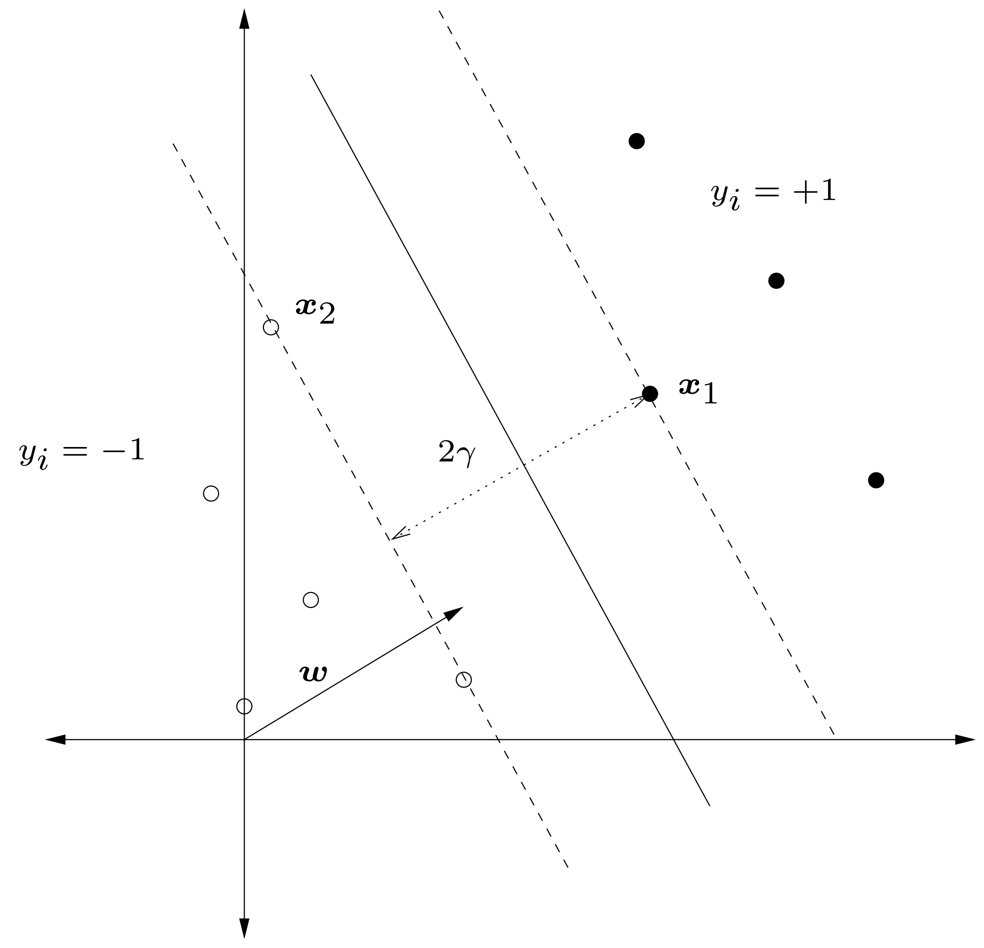
Let us start with the binary classification of linearly separable data *in feature space*.

> **Important:** for the remainder of these lecture notes we will be using $\boldsymbol{x}$ instead of the usual $\boldsymbol{\phi}(\boldsymbol{x})$ notation for features. This is for convenience and because the book does this. When you see $\left\langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \right\rangle$ just think of $K(\boldsymbol{x}_j, \boldsymbol{x}_j)$ or if you like $\left\langle \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}_j) \right\rangle$.

Let $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_\ell, y_\ell)\}$ denote a linearly separable training sample of data points $\boldsymbol{x} \in \mathbb{R}^n$ in an n-dimensional feature space[1], that are classified either as $+$ points of $-$ points. Here

$$y_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \text{ is a } + \text{ point} \\ -1 & \text{if } \boldsymbol{x}_i \text{ is a } - \text{ point} \end{cases}$$

[1] Again think of n as equivalent to N, the dimension of the feature space.



Consider the two sample points $(\boldsymbol{x}_1, +1)$ and $(\boldsymbol{x}_2, -1)$ on the margin, where for some $\gamma' > 0$,

$$\big\langle \boldsymbol{w} \cdot \boldsymbol{x}_1 \big\rangle + b = +\gamma'$$
$$\big\langle \boldsymbol{w} \cdot \boldsymbol{x}_2 \big\rangle + b = -\gamma'$$

where the geometric margin $\gamma = \gamma'/\|\boldsymbol{w}\|$. Since $\boldsymbol{w}$ and b can be scaled arbitrarily we choose[2] the margin to be $\gamma = 1/\|\boldsymbol{w}\|$.

[2] i.e. set $\gamma' = 1$.

The fact that the set is linearly separable means that there exists a hyperplane $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = 0$ in feature space such that :

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \begin{cases} > 0 & \text{if } y_i = 1 \\ < 0 & \text{if } y_i = -1 \end{cases}$$

We are now interested in determining the parameters, $\boldsymbol{w}$ and b, in such a way that:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \begin{cases} \geq 1 & \text{if } y_i = 1 \\ \leq 1 & \text{if } y_i = -1 \end{cases} \tag{1}$$

and the margin $1/\|w\|$ as large as possible.

Sine the geometric distance from a point $\boldsymbol{x}_i$ to the plane is given by

$$\frac{1}{\|w\|}(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b),$$

where the distance is > 0 if $\boldsymbol{x}_i$ is in the half-space that $\boldsymbol{w}$ points out from, this means that the absolute distance of all points in S from the half plane will be at least $1/\|\boldsymbol{w}\|$ and the smallest distance will be as large as possible. The smallest distance being the *margin* of the plane, and this plane is called a

maximal margin classifier.

The quadratic programming problem

The problem of determining $\boldsymbol{w}$ and b such that (1) is satisfied may be re-expressed as follows:

Find $(\boldsymbol{w}, b)$ that minimizes:

$$\tfrac{1}{2}\big\langle \boldsymbol{w} \cdot \boldsymbol{w} \big\rangle \tag{2}$$

subject to: $y_i\big(\big\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \big\rangle + b\big) \geq 1$, $i = 1, 2, \ldots, \ell$
(cf. proposition 6.1 p. 95).

This is in fact a so called *quadratic programming problem* which in general is expressed as follows. Find $\boldsymbol{x} \in \mathbb{R}^n$ that minimizes:

$$\tfrac{1}{2}\boldsymbol{z}'\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}'\boldsymbol{z} \tag{3}$$

subject to: $\boldsymbol{A}\boldsymbol{z} \leq c$.

where $\boldsymbol{H}$ is a $p \times p$ (semi) positive definite matrix, $\boldsymbol{f}$ is an p-column vector, $\boldsymbol{A}$ is an $m \times p$ matrix, and $\boldsymbol{c}$ is an m-column vector.

Problem (2) becomes (3) if we set $p = n + 1$ and $m = \ell$, where $\boldsymbol{z} = [\boldsymbol{w} \;\; b]'$, $\boldsymbol{f} = [0 \ldots 0]'$, $\boldsymbol{c} = -[1 \ldots 1]'$ and

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & 1 & 0 \\ 0 & \cdots & 0 & 0 \end{bmatrix}, \; A = -\begin{bmatrix} y_1 & & 0 \\ & \ddots & \\ 0 & & y_\ell \end{bmatrix} \begin{bmatrix} \boldsymbol{x}'_1 & 1 \\ \vdots & \vdots \\ \boldsymbol{x}'_\ell & 1 \end{bmatrix}$$

MATLAB example

```
>> X=[3 7;4 6;5 6;7 7;8 5;4 5;5 5;6 3;7 4;9 4]
X =
     3     7
     4     6
     5     6
     7     7
     8     5
     4     5
     5     5
     6     3
     7     4
     9     4
>> y=[1 1 1 1 1 -1 -1 -1 -1 -1]'
y =
     1
     1
     1
     1
     1
    -1
    -1
    -1
    -1
    -1
>> n=size(X,2)
n =
     2
>> ell=size(X,1)
ell =
    10
>> H=diag([ones(1,n) 0])
H =
     1     0     0
     0     1     0
     0     0     0
>> f=zeros(1,n+1)
```

```
f =
     0     0     0
>> A=-diag(y)*[X ones(ell,1)]
A =
    -3    -7    -1
    -4    -6    -1
    -5    -6    -1
    -7    -7    -1
    -8    -5    -1
     4     5     1
     5     5     1
     6     3     1
     7     4     1
     9     4     1
>> c=-ones(ell,1)
c =
    -1
    -1
    -1
    -1
    -1
    -1
    -1
    -1
    -1
    -1
>> z=quadprog(H,f,A,c)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.



> In /usr/local/matlab/toolbox/optim/quadprog.m at line 213
Optimization terminated successfully.
z =
    0.6667
    2.6667
  -17.6667
```

```
>> w=z(1:n)
w =
    0.6667
    2.6667
>> b=z(n+1)
b =
  -17.6667
>> X*w+b*ones(ell,1)
ans =
    3.0000
    1.0000
    1.6667
    5.6667
    1.0000
   -1.6667
   -1.0000
   -5.6667
   -2.3333
   -1.0000
>> margin=1/norm(w)
margin =
    0.3638
```

The kernel form

Since we are working in feature space we will in general not know the data (feature) vectors $\boldsymbol{x}_i$, $i = 1, \ldots, \ell$ explicitly but only the kernel values $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle$.

To overcome this problem it is possible to introduce ℓ-vector $\boldsymbol{\alpha}$ s.t. $\boldsymbol{w} = \sum_{i=1}^{\ell} \alpha_i \boldsymbol{x}_i = \boldsymbol{X}'\boldsymbol{\alpha}$, where $\boldsymbol{X}' = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_\ell]$. Then (2) is transformed into the following problem: Find $(\boldsymbol{\alpha}, b)$ that maximizes

$$\tfrac{1}{2}\boldsymbol{\alpha}'\boldsymbol{G}\boldsymbol{\alpha} \tag{4}$$

subject to: $y_i\big(\sum_{j=1}^{\ell} \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b\big) \geq 1$, $i = 1, \ldots \ell$, where $\boldsymbol{G}$ is the $\ell \times \ell$ Gram matrix whose (i, j)-th element is $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

The MATLAB example continued:

```
>>G=X*X';
>>H=zeros(ell+1,ell+1);
>>H(1:ell,1:ell)=G;
>>f=zeros(1,ell+1);
>>A=-diag(y)*[G ones(ell,1)];
>>c=-ones(ell,1);
>>z=quadprog(H,f,A,c);
>>alpha=z(1:ell);
>>w=X'*alpha;
>>b=z(ell+1);
>>margin=1/sqrt(alpha'*G*alpha)
>>margin =
    0.3638
```

Remarks

It should be noted that the $\boldsymbol{\alpha}$-vector that solves problem (4) need not be unique if $\ell > n$, i.e. the number of data points is larger than the dimension of the feature space.

`missing discussion on sparse` α`s`

An alternative approach that results directly in sparse $\boldsymbol{\alpha}$-vectors, and only involves information on the kernel values, is to solve the so-called dual form of problem (2).

We shall see that the constraints turn out to be much simpler in the dual formulation than in their primal counterpart given by (2) or (4).

The dual formulation

Lets start by expressing the problem (2) in a somewhat more general way: Find $(\boldsymbol{w}, b)$ that minimizes

$$f(\boldsymbol{w}, b) \tag{5}$$

subject to: $g_i(\boldsymbol{w}, b) \leq 0$, $i = 1, \ldots, \ell$. Thus in (2): $f(\boldsymbol{w}, b) = \frac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle$ and $g_i(\boldsymbol{w}, b) = 1 - y_i\big(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b\big)$.

We now introduce the so-called *Lagrangian function*:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = f(\boldsymbol{w}, b) + \sum_{i=1}^{\ell} \alpha_i g_i(\boldsymbol{w}, b) \tag{6}$$

and call the coefficients α_i, $i = 1, 2, \ldots, \ell$, in this context the *Lagrangian multipliers*.

We then introduce the function

$$\Theta(\boldsymbol{\alpha}) = \min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) \tag{7}$$

Thus for problem (2)

$$\Theta(\boldsymbol{\alpha}) = \inf_{\boldsymbol{w}, b} \left(\tfrac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + \sum_{i=1}^{\ell} \alpha_i \Big(1 - y_i\big(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b\big)\Big) \right)$$

But now at the minimum point we know, since (7) is an unconstrained minimization problem, that:

$$\frac{\partial L}{\partial \boldsymbol{w}} = \mathbf{0} = \boldsymbol{w} + \sum_{i=1}^{\ell} \alpha_i(-y_i \boldsymbol{x}_i) \text{ i.e. } \boldsymbol{w} = \sum_{i=1}^{\ell} \alpha_i y_i \boldsymbol{x}_i \tag{8}$$

and

$$\frac{\partial L}{\partial b} = 0 = \sum_{i=1}^{\ell} \alpha_i y_i \tag{9}$$

Thus we may re-express

$$\begin{aligned} \Theta(\boldsymbol{\alpha}) &= \tfrac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j \big\langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \big\rangle + \sum_{i=1}^{\ell} \alpha_i \\ &\quad -\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j \big\langle \boldsymbol{x}_j \cdot \boldsymbol{x}_i \big\rangle - b\sum_{i=1}^{\ell} \alpha_i y_i \\ &= \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \end{aligned} \tag{10}$$

where $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ and note that the double sum may be expressed as $\tilde{\boldsymbol{\alpha}}'\boldsymbol{G}\tilde{\boldsymbol{\alpha}} = \boldsymbol{\alpha}'\tilde{\boldsymbol{G}}\boldsymbol{\alpha}$, where $\tilde{\boldsymbol{\alpha}} = [\alpha_1 y_1 \ldots \alpha_\ell y_\ell]$ and the (i,j)-the element of $\tilde{\boldsymbol{G}}$ is $y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Returning to the more general formulation (5) (which we call in this context the *primal problem*) the *dual problem* is to: Find $\boldsymbol{\alpha}$ that maximizes

$$\Theta(\boldsymbol{\alpha}) \tag{11}$$

subject to $\alpha_i \geq 0$, $i = 1, \ldots, \ell$, and $\sum_{i=1}^{\ell} y_i \alpha_i = 0$.
cf. definition 5.14, p. 85.

A *feasible* solution to the dual problem is *any* $\boldsymbol{\alpha} \in \mathbb{R}^{\ell}$ that satisfies the constraints $\alpha_i \geq 0$, and the *optimal* solution denoted by $\boldsymbol{\alpha}^*$ is the feasible solution that maximizes $\Theta(\boldsymbol{\alpha})$.

Similarly, a feasible solution to the primal problem is any $\boldsymbol{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ such that $g_i(\boldsymbol{w}, b) \leq 0$ and the optimal solution $(\boldsymbol{w}^*, b^*)$ is the feasible solution that minimizes $f(\boldsymbol{w}, b)$.

Now if $\hat{\boldsymbol{\alpha}}$ is any feasible solution to the dual problem and $(\hat{\boldsymbol{w}}, \hat{b})$ is any feasible solution to the primal problem then

$$\begin{aligned}
\Theta(\hat{\boldsymbol{\alpha}}) &= \min_{\boldsymbol{w}, b} L(\boldsymbol{w}, b, \hat{\boldsymbol{\alpha}}) \leq L(\hat{\boldsymbol{w}}, \hat{b}, \hat{\boldsymbol{\alpha}}) \\
&\leq f(\hat{\boldsymbol{w}}, \hat{b}) + \sum_{i=1}^{\ell} \hat{\alpha}_i g_i(\hat{\boldsymbol{w}}, \hat{b}) \leq f(\hat{\boldsymbol{w}}, \hat{b})
\end{aligned}$$

since $\hat{\alpha}_i \geq 0$ and $g_i(\hat{\boldsymbol{w}}, \hat{b}) \leq 0$, $i = 1, \ldots, \ell$.
cf. theorem 5.15, p. 85.

This implies in particular that $\Theta(\boldsymbol{\alpha}^*) \leq f(\boldsymbol{w}^*, b^*)$.

In many cases we have in fact that $\Theta(\boldsymbol{\alpha}^*) = f(\boldsymbol{w}^*, b^*)$. This will eg. be the case if the set of possible $(\boldsymbol{w}, b)$ values form a *convex set*, the function $f(\boldsymbol{w}, b)$ is a *convex function* and the constraint functions $g_i(\boldsymbol{w}, b)$ are *affine function* (see definitions on p. 81), cf. theorem 5.20, p. 86.

In this case there are also special conditions, so-called Kuhn-Tucker conditions, that characterize the relationship between $\boldsymbol{\alpha}^*$, $\boldsymbol{w}^*$ and b^*:

$$\frac{\partial}{\partial \boldsymbol{w}} f(\boldsymbol{w}^*, b^*) + \sum_{i=1}^{\ell} \alpha_i^* \frac{\partial}{\partial \boldsymbol{w}} g_i(\boldsymbol{w}^*, b^*) = \mathbf{0} \tag{12}$$

$$\frac{\partial}{\partial b} f(\boldsymbol{w}^*, b^*) + \sum_{i=1}^{\ell} \alpha_i^* \frac{\partial}{\partial b} g_i(\boldsymbol{w}^*, b^*) = \mathbf{0} \tag{13}$$

$$\text{Either } \alpha_i^* = 0 \text{ or } g_i(\boldsymbol{w}^*, b^*) = 0 \text{ for } i = 1, \ldots, \ell \tag{14}$$

If we add to these conditions the constraints that $\alpha^* \geq 0$, $g_i(\boldsymbol{w}^*, b^*) \leq 0, \quad i = 1, \ldots, \ell$, these conditions in fact determine $\boldsymbol{\alpha}^*$, $\boldsymbol{w}^*$, and b^* of theorem 5.21, p. 87.

We say that the constraint $g_i(\boldsymbol{w}^*, b^*) \leq 0$ is *active* at the optimal point if $g_i(\boldsymbol{w}^*, b^*) = 0$, but *passive* if $g_i(\boldsymbol{w}^*, b^*) < 0$. Condition (12) thus states that the Lagrangian coefficients of all passive constraints must be zero!

`missing 3 pictures and description`

Returning again to the primal problem (2) we have from (10) and (11) that the corresponding dual problem is:
Find $\boldsymbol{\alpha}$ that maximizes

$$\sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15}$$

subject to $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ and $\alpha_i \geq 0$, $i = 1, \ldots, \ell$.

This can yet again be re-expressed as a quadratic programming problem of type (3) but it is advantageous to make use of the following alternative formulation:
Find $\boldsymbol{z} \in \mathbb{R}^m$ that maximizes $\frac{1}{2}\boldsymbol{z}'\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}'\boldsymbol{z}$
subject to: $\boldsymbol{A}\boldsymbol{z} \leq \boldsymbol{c}$, $\boldsymbol{A}_{eq}\boldsymbol{z} = c_{eq}$, and $\boldsymbol{z}_l \leq \boldsymbol{z} \leq \boldsymbol{z}_u$.

Changing maximization to minimization by reversing the sign of the cost function we set: $\boldsymbol{z} = \boldsymbol{\alpha}$, $\boldsymbol{H} = \tilde{\boldsymbol{G}}$ (where the (i, j)-th element is $y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i, j = 1, \ldots, \ell$, and $\boldsymbol{f} = -[1, \ldots, 1]'$, $A = [0, \ldots, 0]$, $c = 0$ (a dummy inequality constraint), $\boldsymbol{A}_{eq} = [y_1, \ldots, y_\ell]$, $c_{eq} = 0$, $\boldsymbol{z}_l = [0, \ldots, 0]$, and finally $\boldsymbol{z}_u = [\infty, \ldots, \infty]$.

MATLAB example (continued)

Note that we get three non-zero α-values that must correspond to active constraints, i.e. data points $\boldsymbol{x}_i$ whose geometric distance from the maximal margin classifying plane is the same as the margin, or for which it holds that $y_i\big(\big\langle \boldsymbol{w}^* \cdot \cdot \cdot \boldsymbol{x}_i\big\rangle + b\big) = 1$ Note that when $n = 2$, i.e. the data points lie in a two-dimensional plane, there must in general always be three such points. Data points that correspond to active constraints are called **support vectors**.

```
>> H=(y*y').*(X*X')
H =
    58    54    57    70    59   -47   -50   -39   -49   -55
    54    52    56    70    62   -46   -50   -42   -52   -60
    57    56    61    77    70   -50   -55   -48   -59   -69
    70    70    77    98    91   -63   -70   -63   -77   -91
    59    62    70    91    89   -57   -65   -63   -76   -92
   -47   -46   -50   -63   -57    41    45    39    48    56
   -50   -50   -55   -70   -65    45    50    45    55    65
   -39   -42   -48   -63   -63    39    45    45    54    66
   -49   -52   -59   -77   -76    48    55    54    65    79
   -55   -60   -69   -91   -92    56    65    66    79    97
>> f=-ones(1,ell)
f =
    -1    -1    -1    -1    -1    -1    -1    -1    -1    -1
>> A=zeros(1,ell)
A =
     0     0     0     0     0     0     0     0     0     0
>> c=0
c =
     0
>> Aeq=y'
Aeq =
     1     1     1     1     1    -1    -1    -1    -1    -1
```

```
>> ceq=0
ceq =
     0
>> LB=zeros(ell,1)
LB =
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0
>> UB=inf*ones(ell,1)
UB =
   Inf
   Inf
   Inf
   Inf
   Inf
   Inf
   Inf
   Inf
   Inf
   Inf
>> alpha=quadprog(H,f,A,c,Aeq,ceq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.
> In /usr/local/matlab/toolbox/optim/quadprog.m at line 213
Exiting: The solution is unbounded and at infinity;
         the constraints are not restrictive enough.
alpha =
   1.0e+16 *
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
>> UB=1000*ones(ell,1)
UB =
        1000
        1000
        1000
        1000
        1000
        1000
        1000
        1000
        1000
        1000
>> alpha=quadprog(H,f,A,c,Aeq,ceq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.
> In /usr/local/matlab/toolbox/optim/quadprog.m at line 213
Optimization terminated successfully.
alpha =
    0.0000
    2.6667
    0.0000
   -0.0000
    1.1111
    0.0000
    3.7778
    0.0000
    0.0000
   -0.0000
```

In order to obtain $\boldsymbol{w}^*$ and b^* from $\boldsymbol{\alpha}^*$ we make use of the Kuhn-Tucker conditions: It follows from (12) that

$$\frac{\partial}{\partial \boldsymbol{w}}\left[\tfrac{1}{2}\left\langle \boldsymbol{w}^* \cdot \boldsymbol{w}^* \right\rangle + \sum_{i=1}^{\ell} \alpha_i^* \Big(1 - y_i(\left\langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \right\rangle + b^*)) \Big)\right]$$

$$= \boldsymbol{w} - \sum_{i=1}^{\ell} \alpha_i^* y_i \boldsymbol{x}_i = 0$$

i.e.

$$\boldsymbol{w}^* = \sum_{i=1}^{\ell} \alpha_i^* y_i \boldsymbol{x}_i \tag{16}$$

If follows from (14) that is constraint i is an active constraint characterized by the fact that $\alpha_i > 0$, then we can calculate b^* from $1 - y_i\big(\left\langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \right\rangle + b^*\big) = 0$, i.e.

$$b^* = 1 - \left\langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \right\rangle \tag{17}$$

Since we are working in feature space we do not in general know $\boldsymbol{x}_i$ but only the kernel values: This making us of (16) and (17) we would first calculate

$$b^* = 1 - \sum_{j=1}^{\ell} \alpha_j^* K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{18}$$

where constraint i is such that $\boldsymbol{\alpha}_i > 0$ and $y_i = 1$.
cf. remark 6.4 p. 96 in book.

Our classification function then becomes:

$$F(\boldsymbol{x}) = \big\langle \boldsymbol{w}^* \cdot \boldsymbol{x} \big\rangle + b^* = \sum_{j=1}^{\ell} \alpha_j^* y_j K(\boldsymbol{x}_j, \boldsymbol{x}) + b^*$$

where $F(\boldsymbol{x})$ is such that

$$f(\boldsymbol{x}_i) = \begin{cases} \geq 1 & \text{if } y_i = 1 \\ \leq 1 & \text{if } y_i = -1 \end{cases}$$

cf. p. 97.

Finally, note that since

$$y_i\big(\big\langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \big\rangle + b^*\big) = y_i\big(\sum_{j=1}^{\ell} \alpha_j^* y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b^*\big) = 1$$

if i is such that $\boldsymbol{x}_i$ is a support vector and $\alpha_i^* = 0$ otherwise then

$$\begin{aligned}\langle \boldsymbol{w}^* \cdot \boldsymbol{w}^* \rangle &= \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i^* y_i \alpha_j^* y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= \sum_{i \in \mathsf{SV}} \alpha_i^* y_i \sum_{j \in \mathsf{SV}} \alpha_j^* y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= \sum_{i \in \mathsf{SV}} \alpha_i^* (1 - y_i b^*) \\ &= \sum_{i \in \mathsf{SV}} \alpha_i^*\end{aligned}$$

Since it follows from the Kuhn-Tucker condition (13) that

$$\frac{\partial}{\partial b}\Big[\tfrac{1}{2}\langle \boldsymbol{w}^* \cdot \boldsymbol{w}^* \rangle + \sum_{i=1}^{\ell} \alpha_i^* \Big(1 - y_i(\langle \boldsymbol{w}^* \cdot \boldsymbol{x}_i \rangle + b^*)\Big)\Big]$$

$$= -\sum_{i=1}^{\ell} \alpha_i^* y_i = 0$$

Thus the maximum geometric margin can also be calculated as:

$$1/\Big(\sum_{i \in \mathsf{SV}} \alpha_i^*\Big)^{1/2} = 1/\Big(\sum_{i=1}^{\ell} \alpha_i^*\Big)^{1/2} = 1/\|\boldsymbol{\alpha}^*\|_1^{1/2}$$

MATLAB example (contd.)

```
>> w=X'*(alpha.*y)
w =
    0.6667
    2.6667
>> i=min(find((alpha>0.1)&(y==1)))
i =
     2
>> b=1-G(i,:)*(alpha.*y)
b =
  -17.6667
>> F=G*(alpha.*y)+b*ones(ell,1)
F =
    3.0000
    1.0000
    1.6667
    5.6667
    1.0000
   -1.6667
   -1.0000
   -5.6667
   -2.3333
   -1.0000
>> margin=1/sqrt(sum(abs(alpha)))
margin =
    0.3638
```

Linear Programming SVM

The linear programming formulation finds the sparsest separating hyperplane (reduce the number of support vectors) and the bounds on generalization are directly in terms of $\sum_{i=1}^{\ell} \alpha_i$.

The dual formulation suggest the following classification model based on kernel formulation and *linear programming*:

$$\min \sum_{i=1}^{\ell} y_i \alpha_i \quad , \quad \text{s.t.:} \begin{cases} y_i\big(\sum_{j=1}^{\ell} K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b\big) \geq 1 \\ y_i \alpha_i \geq 0, \quad i = 1, \ldots, \ell \end{cases}$$

Note that in the previous model margin $1/\sqrt{\boldsymbol{\alpha}'\boldsymbol{G}\boldsymbol{\alpha}}$ was maximized, this is not the case here.

This model can be solved with `LINPROG(f,A,c,Aeq,ceq,LB,UB)` in MATLAB by setting:

```
>> f = [y;0];
>> A = -diag(y)*[G ones(ell,1)];
>> c =-ones(ell,1);
>> Aeq = zeros(1,ell+1);
>> ceq = 0;
>> LB = [-L*(y<0);-inf];
>> UB = [L*(y>0);inf];
>> R = linprog(f,A,c,Aeq,ceq,LB,UB);
>> alpha = R(1:ell);
>> margin = 1/sqrt(alpha'*G*alpha)
margin =
    0.3638                              % <-------
>> b = R(ell+1);
>> G*alpha + b; % the decision function value
```

where `L` above is some large number (say 1000, but not `Inf` since `Inf*0=NaN`).

We show the result of applying this approach to our test problem which results in the same margin. However, this is not always the case as seen by our examples when we apply both approaches to the same data using a polynomial kernel.

```
>> G2 = (1+G).^2;
>> H = (y*y').*G2;
>> f = -ones(1,ell);
>> A = zeros(1,ell);
>> c = 0;
>> Aeq = y';
>> ceq = 0;
>> LB = zeros(ell,1);
>> UB = 10000*ones(ell,1);
>> alpha = quadprog(H,f,A,c,Aeq,ceq,LB,UB)
alpha =
  -0.00000000000048
   0.01789950609325
  -0.00000000000005
  -0.00000000000054
   0.00461343208147
   0.00000000000000
   0.02241482643331
  -0.00000000000018
   0.00000000000038
   0.00009811172955
>> i = min(find(alpha>0.00001 & 1==y))
i =
     2
>> margin = 1/sqrt((y.*alpha)'*G2*(alpha.*y))
margin =
    4.7127
>> margin = 1/sqrt(sum(abs(alpha)))
margin =
    4.7127
```

```
%%% THE SAME VALUE BUT
%%% WHAT ABOUT LINPROG?

>> f = [y;0];
>> A = -diag(y)*[G2 ones(ell,1)];
>> Aeq = zeros(1,ell+1);
>> c = -ones(ell,1);
>> ceq = 0;
>> LB = [-1000*(y<0);-inf];
>> UB = [1000*(y>0);inf];
>> R = linprog(f,A,c,Aeq,ceq,LB,UB)
Optimization terminated successfully.
R =
   0.00401508496963
   0.00000009322276
   0.00000009713393
   0.00000017202486
   0.00000001136271
  -0.00000000719558
  -0.00000001881483
  -0.00000009523865
  -0.00000022317090
  -0.00046847406827
  -9.40306191028867
>> alpha = R(1:ell);
>> margin = 1/sqrt(alpha'*G2*alpha)
margin =
   0.40612638198888
>> margin=1/sqrt(sum(abs(alpha)))
margin =
  14.93323069613136
%%% YOU SEE THE LAST FORMULA IS NOT VALID WHEN USING LINPROG
```

Soft margin optimization

- When the data in noisy, there will in general be no linear separation in feature space (unless you want to overfit your data with powerful kernels).
- In the case of overfitting, the corresponding "outliers" will have large Lagrange multipliers.
- One could "clean" the data by ranking the training data according to how difficult it is to classify.

When the training data is not linearly separable in feature space, the optimization problem cannot be solved since there exists no feasible solution. We must therefore introduce the *slack variables* to our optimization problem:

$$\min_{\boldsymbol{w},b} \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle$$

subject to: $y_i\big(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b\big) \geq 1 - \xi_i$, $i = 1, \ldots, \ell$.

From Theorem 4.22 (chapter 4), bounds on the generalization error in terms of the 2-norm of the margin slack vector is:

$$\text{err}_{\mathcal{D}}(f) \leq \frac{c}{\ell}\left(\frac{R^2 + \|\boldsymbol{\xi}\|_1^2}{\gamma^2} \log^2 \ell + \log \frac{1}{\delta}\right)$$

where $\|w\| = 1$.

Having fixed ℓ, δ and c, the generalization depends on the following factor:

$$\frac{R^2 + \frac{\|\boldsymbol{\xi}\|_2^2}{\|\boldsymbol{w}\|_2^2}}{\gamma^2} = \frac{R^2 + \frac{\|\boldsymbol{\xi}\|_2^2}{\|\boldsymbol{w}\|_2^2}}{1/\|\boldsymbol{w}\|_2^2} = \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle R^2 + \langle \boldsymbol{\xi} \cdot \boldsymbol{\xi} \rangle$$

We therefore propose the following 2-norm soft margin optimization problem:

$$\min_{\boldsymbol{w},b} \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{\ell} \xi_i^2$$

subject to: $y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi \geq 0$, $i = 1, \ldots, \ell$. The factor above suggest that $C \approx R^{-2}$, but in practice this parameter needs to be tuned[3] either using cross validation or a modified radius margin bound.

The approach can be adopted in the 1-norm case (does not match theorem 4.22 however):

$$\min_{\boldsymbol{w},b} \langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{\ell} \xi_i$$

subject to: $y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) \geq 1 - \xi_i$ and $\xi \geq 0$, $i = 1, \ldots, \ell$.

[3]In `LIBSVM` C is set with the `-c` option.

2-norm soft margin (weighting the diag) L2-SVC

The primal Lagrangian for the 2-norm soft margin is:

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \tfrac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + \tfrac{C}{2} \sum_{i=1}^{\ell} \xi_i^2 - \sum_{i=1}^{\ell} \alpha_i [y_i(\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b) - 1 + \xi_i]$$

where $\alpha_i \geq 0$ are the Lagrange multipliers.

As before the corresponding dual is found by differentiating with respect to $\boldsymbol{w}$, b and now also $\boldsymbol{\xi}$, and imposing stationarity. Then the new relations are substituted into the primal above. The dual problem is then:
Find $\boldsymbol{\alpha}$ that maximizes

$$\sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j \left(K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \tfrac{1}{C}\delta_{ij} \right)$$

subject to $\sum_{i=1}^{\ell} \alpha_i y_i = 0$ and $\alpha_i \geq 0$, $i = 1, \ldots, \ell$. Here δ_{ij} is the Kronecker δ defined to be 1 if $i = j$ else 0.

The bias b^* is computed as before from (14) (Karush-Kuhn-Tucker complementary conditions), that is for $\alpha_j^* \neq 0$:

$$y_j\left(\sum_{i=1}^{\ell} y_i \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}_j) + b^*\right) = 1 - \xi_j = 1 - \alpha_j^*/C$$

or

$$b^* = 1/y_j - \sum_{i=1}^{\ell} y_i \alpha_i^* K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \alpha_j^*/(C y_j)$$

recall that $y \in [-1, 1]$ (binary classification).

Furthermore, the margin is (see derivation on page 106 in book):

$$\gamma = \frac{1}{\|\boldsymbol{w}\|} = \left(\sum_{i \in \mathsf{SV}} \alpha_i^* - \frac{1}{C}\langle \boldsymbol{\alpha}^* \cdot \boldsymbol{\alpha}^* \rangle\right)^{-1/2}$$

Note that the dual 2-norm soft margin optimization is still just a quadratic programming problem. The only change is that we add a factor of $1/C$ to the diagonal of our kernel (Gram) matrix.

MATLAB example revisited

```
>> G = X*X';
>> C = 1
C =
     1
>> H = (y*y').*(G + (1/C)*eye(ell))
H =
    59    54    57    70    59   -47   -50   -39   -49   -55
    54    53    56    70    62   -46   -50   -42   -52   -60
    57    56    62    77    70   -50   -55   -48   -59   -69
    70    70    77    99    91   -63   -70   -63   -77   -91
    59    62    70    91    90   -57   -65   -63   -76   -92
   -47   -46   -50   -63   -57    42    45    39    48    56
   -50   -50   -55   -70   -65    45    51    45    55    65
   -39   -42   -48   -63   -63    39    45    46    54    66
   -49   -52   -59   -77   -76    48    55    54    66    79
   -55   -60   -69   -91   -92    56    65    66    79    98
>> f = -ones(1,ell);
>> A = zeros(1,ell);
>> c = 0;
>> Aeq = y';
>> ceq = 0;
>> LB = zeros(ell,1);
>> UB = inf*ones(ell,1);
>> alpha=quadprog(H,f,A,c,Aeq,ceq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
alpha =
   -0.0000
    0.4826
    0.2442
   -0.0000
    0.6744
    0.3721
    0.6105
    0.0000
    0.0000
    0.4186
>> SV = find(alpha>0.0001)
SV =
     2
     3
     5
     6
     7
    10
>> margin = 1/sqrt(sum(alpha(SV)) + (1/C)*alpha'*alpha)
margin =
    0.4859
>> C = 100
C =
   100
>> H = (y*y').*(G + (1/C)*eye(ell));
>> alpha=quadprog(H,f,A,c,Aeq,ceq,LB,UB);
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
>> SV = find(alpha>0.0001)
SV =
     2
     5
     7
    10
>> margin = 1/sqrt(sum(alpha(SV)) + (1/C)*alpha'*alpha)
margin =
    0.3639
```

1-norm soft margin (the box constraint) L1-SVC

The corresponding Lagrangian for the 1-norm soft margin optimization problem is:

$$L(\boldsymbol{w}, b\boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{r}) = \tfrac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{\ell} \xi_i$$

$$- \sum_{i=1}^{\ell} \alpha_i \big[y_i(\langle \boldsymbol{x}_i \cdot \boldsymbol{w} \rangle + b) - 1 + \xi_i \big] - \sum_{i=1}^{\ell} r_i \xi_i$$

with $\alpha_i \geq 0$ and $r_i \geq 0$ (Lagrangian multipliers for $\xi_i \geq 0$). Again differentiating with respect to $\boldsymbol{w}$, b and $\boldsymbol{\xi}$, and imposing stationarity, $\partial L / \partial \boldsymbol{w} = \boldsymbol{w} - \sum_{i=1}^{\ell} y_i \alpha_i \boldsymbol{x}_i = \mathbf{0}$, $\partial L / \partial \xi_i = C - \alpha_i - r_i = 0$, $\partial L / \partial b = \sum_{i=1}^{\ell} y_i \alpha_i = 0$, the relations are substituted into the primal above. The dual problem is then:
Find $\boldsymbol{\alpha}$ that maximizes

$$\sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i y_i \alpha_j y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

which is identical to that for the maximal margin. The only difference is that the constraint $C - \alpha_i - r_i = 0$, together with $r_i \geq 0$, enforces $\alpha_i \leq C$, while $\xi_i \neq 0$ only if $r_i = 0$ and therefore $\alpha_i = C$.

The Karush-Kuhn-Tucker complementary conditions for when $\alpha_i^* \neq 0$:

$$y_i(\langle \boldsymbol{x}_i \cdot \boldsymbol{w}^* \rangle + b^*) - 1 + \xi_i^* = 0$$

and when $\xi_i^* \neq 0$

$$\alpha_i^* - C = 0$$

which implies that non-zero slack variables can only occur when $\alpha_i^* = C$ (this also implies that example i is misclassified). The points with non-zero slack variables have a geometric margin less than $1/\|\boldsymbol{w}^*\|$. Points for which $0 < \alpha_i^* < C$ lie at the target distance of $1/\|\boldsymbol{w}^*\|$ from the hyperplane. Because when $\alpha_i^* = C$ then $\xi_i = 0$ we can only compute the bias b^* from points corresponding to $0 < \alpha_i^* < C$ using the first condition above: $y_j(\langle \boldsymbol{x}_j \cdot \boldsymbol{w}^* \rangle + b^*) - 1 = 0$, or:

$$b^* = 1/y_j - \sum_{i=1}^{\ell} \alpha_i^* y_i K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

for $0 < \alpha_j^* < C$. Furthermore, the margin is computed as follows:

$$\gamma = \left(\sum_{i,j \in SV} y_i y_j \alpha_i^* \alpha_j^* K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right)^{-1/2}$$

MATLAB example continued

The MATLAB example is then the same as for the *hard margin*:

```
>> UB = C*ones(ell,1);
>> alpha = quadprog(H,f,A,c,Aeq,ceq,LB,UB)
alpha =
    0.0000
    2.6667
    0.0000
   -0.0000
    1.1111
    0.0000
    3.7778
    0.0000
    0.0000
   -0.0000
>> margin = 1/sqrt((alpha)'*H*(alpha))
margin =
    0.3638
>> SV=find(alpha>0.0001&alpha<(C-0.0001))
SV = 2 5 7
>> b = ( 1./y(SV) -  G(SV,:)*(alpha.*y) )
b =
  -17.6667
  -17.6667
  -17.6667
>> b=mean(b)
b =
  -17.6667
```

```
>> F = G*(alpha.*y) + b*ones(ell,1)
F =
    3.0000
    1.0000
    1.6667
    5.6667
    1.0000
   -1.6667
   -1.0000
   -5.6667
   -2.3333
   -1.0000

>> C = 1
C =
     1
>> UB = C*ones(ell,1);
>> alpha=quadprog(H,f,A,c,Aeq,ceq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213


Optimization terminated successfully.
alpha =
    0.0000
    1.0000
    0.1111
   -0.0000
    1.0000
    0.5556
    1.0000
   -0.0000
   -0.0000
    0.5556
>> SV=find(alpha>0.0001&alpha<(C-0.0001))
SV = 3 6 10
```

```
>> margin = 1/sqrt((alpha)'*H*(alpha))
margin =
    0.5883
>> b = ( 1./y(SV) -  G(SV,:)*(alpha.*y) )
b =
  -10.6667
  -10.6667
  -10.6667
>> b=mean(b)
b =
  -10.6667
>> F = G*(alpha.*y) + b*ones(ell,1)
F =
    2.0000
    0.6667
    1.0000
    3.3333
    0.3333
   -1.0000
   -0.6667
   -3.6667
   -1.6667
   -1.0000
```

Missing detailed example based on assignment 9

ν – support vector classification

It has been shown that the L1-C-SVC is equivalent to minimizing:

$$W(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to:

$$\begin{aligned} \sum_{i=1}^{\ell} y_i \alpha_i &= 0 \\ \sum_{i=1}^{\ell} \alpha_i &\geq \nu \\ 1/\ell \geq \alpha_i \geq 0 \quad &, \quad i = 1, \ldots, \ell \end{aligned} \tag{19}$$

Unlike the parameter C which depends on the choice of feature space, the new parameter ν is more transparent. The parameter $\nu \in (0, 1]$ is an upper bound on the fraction of training errors and a lower bound on the fraction of support vectors.

ν does not depend on the scaling of the feature space but rather the noise level in the data.

The primal form of ν-SVC is given by:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad \frac{1}{2}\big\langle \boldsymbol{w} \cdot \boldsymbol{w} \big\rangle - \nu\rho + \frac{1}{\ell}\sum_{i=1}^{\ell} \xi_i$$

subject to:

$$y_i(\big\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \big\rangle + b) \geq \rho - \xi_i$$

where $\xi \geq 0, \rho \geq 0, i = 1, \ldots, \ell$.

In LIBSVM they use property that $\sum_{i=1}^{\ell} \alpha = \nu$, therefore the constraints in the dual version (19) become: $\sum_{i=1}^{\ell} \alpha_i y_i = 0$, $\sum_{i=1}^{\ell} \alpha_i = \nu\ell$, and $0 \leq \alpha_i \leq 1$, $i = 1, \ldots, \ell$.

They then output α_i/ρ so the computed decision function is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} y_i(\alpha_i/\rho)\big(K(\boldsymbol{x}_i, \boldsymbol{x}) + b\big)$$

and the two margins are

$$y_i\big(\big\langle \boldsymbol{w} \cdot \boldsymbol{x} \big\rangle + b\big) = \pm 1$$

which is the same as for the L1-C-SVC.

One-class SVM

The one-class classifier function returns $+1$ in a "small" region capturing most of the data points, and -1 elsewhere. The aim is to separate the data from the origin with maximum margin. This is solved as follows, in primal form:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad \frac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle - \rho + \frac{1}{\nu\ell}\sum_{i=1}^{\ell} \xi_i$$

subject to:

$$\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b \geq \rho - \xi_i$$

where $\xi \geq 0, i = 1, \ldots, \ell$ and $b = 0$. The behaviour of parameter $\nu \in (0, 1)$ is similar to that in ν–SVC.

The dual is:

$$\min_{\boldsymbol{\alpha}} = \frac{1}{2}\sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$\sum_{i=1}^{\ell} \alpha_i = 1$, $0 \leq \alpha_i \leq 1/(\nu\ell)$, $i = 1 \ldots, \ell$.
(or equivalently $\sum_{i=1}^{\ell} \alpha_i = \nu\ell$, $0 \leq \alpha_i \leq 1$) The decision function is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) - \rho$$

where ρ can be recovered like the bias b in C-SVC for any $0 < \alpha_i < 1/(\nu\ell)$: $\rho = \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle = \sum_j \alpha_j K(\boldsymbol{x}_j, \boldsymbol{x}_i)$.

Missing two figures for one class classification

One-class demo using UPSP data

We create a data set containing 500 zeros only, data set called zeros.dat. Then we create a random sample of mixed samples also containing 500 samples.

```
>> load usps_digits_training.mat
>> I = find(y==0);
>> mat2libsvm(X(I(1:500),:),y(I(1:500)),'zeros.dat');
>> X(I(1:500),:) = []; y(I(1:500)) = []; % remove these samples from the set
>> I = randperm(length(y));  % random index
>> mat2libsvm(X(I(1:500),:),y(I(1:500)),'testset.dat');
```

We now use the RBF kernel with $\gamma = 1E-5$ and set $\nu = 0.05$, that is we don't want more mistakes than 5%.

```
>> !svmtrain -s 2 -t 2 -g 0.00001 -n 0.05 zeros.dat model.dat
obj = 311.566388, rho = 24.926440 nSV = 32, nBSV = 19
>> !svmpredict zeros.dat model.dat predict.dat
>> load predict.dat
>> sum(predict == 1)/length(predict)
ans =
   0.9560
>> !svmpredict testset.dat model.dat predict.dat
>> load predict.dat
>> param = libsvm2mat('testset.dat');
>> I = find(param.y == 0);
>> sum(predict(I) == 1)/length(I)
ans =
   0.9091
>> I=find(param.y ~= 0);
>> sum(predict(I) == -1)/length(I)
ans =
   0.9079
>> fraction_of_support_vectors = 32/500*100
fraction_of_support_vectors =
    6.4000
```

Support vector regression

We have ℓ data points $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, \ell$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ and may be in feature space and $y_i \in \mathbb{R}$. When x_i is in feature space we do not know it explicitly but only the kernel values

$$K(\boldsymbol{x}, \boldsymbol{x}_j), i, j = 1, \ldots, \ell$$

Before we look at support vector regression let us revisit:

- Classical least squares regression
- Ridge regression

Classical least squares regression

$$\text{minimize} \quad \sum_{i=1}^{\ell} \xi_i^2$$

subject to $\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b - y_i = \xi_i$, $i = 1, 2, \ldots, \ell$.

We can augment b to $\boldsymbol{w}$ such that $\hat{\boldsymbol{w}} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$ by extending the $\boldsymbol{x}_i$-vector such that $\hat{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix}$. The constraints then become:

$$y_i - \langle \hat{\boldsymbol{w}} \cdot \hat{\boldsymbol{x}}_i \rangle = \xi_i, \quad i = 1, \ldots, \ell$$

Such a solution is based on the assumption that there is an underlying linear model that relates y to $\boldsymbol{x}$ of the form $f(\boldsymbol{x}) = \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b$. If the “observed” y values are corrupted with Gaussian noise, then solving the regression problem will results in the *most likely* value of the model parameters $\boldsymbol{w}$ and b based on the observed data.

A solution of the minimization problem amounts to solving the over-determined linear system:

$$\hat{\boldsymbol{X}}\hat{\boldsymbol{w}} \approx \boldsymbol{y} \text{ where } \hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_i \; \hat{\boldsymbol{x}}_i \ldots \hat{\boldsymbol{x}}_\ell]' \text{ and } \boldsymbol{y} = [y_1 \ldots y_\ell]'$$

This can be done directly in MATLAB, or equivalently we can solve the normal equations:

$$\hat{\boldsymbol{X}}'\hat{\boldsymbol{X}}\hat{\boldsymbol{w}} = \hat{\boldsymbol{X}}\boldsymbol{y}$$

c.f. p. 21 in chapter 2 in notes.

When we only know the kernel values we can try to introduce an ℓ-vector $\hat{\boldsymbol{\alpha}}$, s.t. $\hat{\boldsymbol{w}} = \hat{\boldsymbol{X}}'\hat{\boldsymbol{\alpha}}$. Multiplying through the normal equations with $\hat{\boldsymbol{X}}$ we then get that

$$GG\hat{\boldsymbol{\alpha}} = G\boldsymbol{y}$$

where G is the $\ell \times \ell$ Gram matrix whose (i, j)-th element is $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ but the problem is that G will be singular if $\ell > n$.

Ridge regression (cf 6.2.2 p. 118-19 in book)

$$\text{minimize } \lambda\left\langle \boldsymbol{w} \cdot \boldsymbol{w} \right\rangle + \sum_{i=1}^{\ell} \xi_i^2$$

subject to $y_i - \left\langle \boldsymbol{w} \cdot \boldsymbol{x}_i \right\rangle - b = \xi_i$ for some $\lambda > 0$. When $b = 0$ the solution to this problem amounts to solving the over-determined linear system:

$$\tilde{\boldsymbol{X}}\boldsymbol{w} \approx \boldsymbol{y} \text{ where } \tilde{\boldsymbol{X}} = \left[\boldsymbol{x}_1\ \boldsymbol{x}_2 \ldots \boldsymbol{x}_\ell\ \mu\boldsymbol{I}_n\right]' \text{ and } \mu = \sqrt{\lambda}$$

and when $b \neq 0$ it amounts to solving the over-determined system:

$$\hat{\boldsymbol{X}}\hat{\boldsymbol{w}} \approx \boldsymbol{y} \text{ where } \hat{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_\ell & \mu\boldsymbol{I}_n \\ 1 & 1 & \ldots & 1 & 0\ldots0 \end{bmatrix}'$$

c.f. p. 33, chapter 2, in notes, but there we are in fact

$$\text{minimizing } \lambda(\left\langle \boldsymbol{w} \cdot \boldsymbol{w} \right\rangle + b^2) + \sum_{i=1}^{\ell} \xi_i^2$$

When we only know the kernel values we can resort to the dual formulation where we consider the Lagrangian:

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \lambda\left\langle \boldsymbol{w}\cdot\boldsymbol{w} \right\rangle + \sum_{i=1}^{\ell} \xi_i^2 + \sum_{i=1}^{\ell} \alpha_i(y_i - \left\langle \boldsymbol{w}\cdot\boldsymbol{x}_i \right\rangle - b - \xi_i)$$

At the minimum point of this Lagrangian, for a fixed $\boldsymbol{\alpha}$

$$\frac{\partial L}{\partial \boldsymbol{w}} = \mathbf{0} \quad \Rightarrow \quad \boldsymbol{w} = \frac{1}{2\lambda} \sum_{i=1}^{\ell} \alpha_i \boldsymbol{x}_i \tag{20}$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{\ell} \alpha_i = 0 \tag{21}$$

$$\frac{\partial L}{\partial \boldsymbol{\xi}} = \mathbf{0} \quad \Rightarrow \quad \xi_i = \tfrac{1}{2}\alpha_i, \quad i = 1, \ldots, \ell \tag{22}$$

inserting this into the Lagrangian the dual problem becomes to find $\boldsymbol{\alpha}$ that maximizes

$$\sum_{i=1}^{\ell} y_i \alpha_i - \tfrac{1}{4\lambda} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j \langle \boldsymbol{x}_i \cdot \boldsymbol{x}_j \rangle - \tfrac{1}{4} \sum_{i=1}^{\ell} \alpha_i^2$$

i.e.:

$$\max \quad \boldsymbol{y}'\boldsymbol{\alpha} - \tfrac{1}{4\lambda}\boldsymbol{\alpha}' G \boldsymbol{\alpha} - \tfrac{1}{4\lambda}\boldsymbol{\alpha}'\boldsymbol{\alpha} \tag{23}$$

subject to $\sum_{i=1}^{\ell} \alpha_i = 0$ (because $b \neq 0$) and since we had equality constraints rather that inequality constraints in the primal problem we do **not** require that $\boldsymbol{\alpha} > 0$ cf. p. 85 in book.

In the case when $b = 0$ (which is the case considered on p. 118-19 in book) we do not have the constraint $\sum_{i=1}^{\ell} \alpha_i = 0$. This implies that at the maximal point of (23)

$$\boldsymbol{y} - \frac{1}{2\lambda}\boldsymbol{G}\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha} = \boldsymbol{0}$$

so we can calculate $\boldsymbol{\alpha}$ by solving the linear system

$$\frac{1}{2\lambda}(\boldsymbol{G} + \lambda\boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{y}$$

Since we also have that $\boldsymbol{w} = \frac{1}{2\lambda}\boldsymbol{X}'\boldsymbol{\alpha}$ the model $f(\boldsymbol{x}) = \left\langle \boldsymbol{w} \cdot \boldsymbol{x} \right\rangle = \boldsymbol{w}'\boldsymbol{x}$ can be expressed in terms or kernel values as:

$$\frac{1}{2\lambda}\boldsymbol{\alpha}'\boldsymbol{X}\boldsymbol{x} = \frac{1}{2\lambda}\boldsymbol{\alpha}'\boldsymbol{k}(\boldsymbol{x}) = \boldsymbol{y}'(\boldsymbol{G} + \lambda\boldsymbol{I})^{-1}\boldsymbol{k}(\boldsymbol{x})$$

where $k_i(\boldsymbol{x}) = K(\boldsymbol{x}_i, \boldsymbol{x})$.

In the case when $b \neq 0$ we can obtain $\boldsymbol{\alpha}$ by quadratic programming using `QUADPROG` to minimize:

$$\frac{1}{2}\boldsymbol{\alpha}'\boldsymbol{H}\boldsymbol{\alpha} + \boldsymbol{f}'\boldsymbol{\alpha} = \frac{1}{2}\boldsymbol{\alpha}'(\frac{1}{2\lambda}\boldsymbol{G} + \frac{1}{2}\boldsymbol{I}_{\ell})\boldsymbol{\alpha} + (-\boldsymbol{y}')\boldsymbol{\alpha}$$

subject to $[1 \ldots 1]\boldsymbol{\alpha} = 0$.

Subsequently we can evaluate b from any of the equality constraints of the primal problem, eg. 1st sample

$$b = y_1 - \langle \boldsymbol{w} \cdot \boldsymbol{x}_1 \rangle - \xi_1 = y_1 - \tfrac{1}{2\lambda} \sum_{i=1}^{\ell} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_1) - \tfrac{1}{2}\alpha_1$$

making use of (20) and (22). Finally we van evaluate the model as:

$$f(\boldsymbol{x}) = \langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b = \tfrac{1}{2\lambda} \boldsymbol{\alpha}' \boldsymbol{k}(\boldsymbol{x}) + b$$

Note that as $\lambda \to 0$ this model tends to the classical regression model. Thus one way of dealing with that model when we only have the kernel values is to solve the ridge regression problem for "sufficiently small" λ. However, in this case it is simpler to consider the problem where we minimize $\lambda(\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + b^2) + \sum_{i=1}^{\ell} \xi_i^2$ because then the model simply becomes:

$$f(\boldsymbol{x}) = \boldsymbol{y}'(\hat{\boldsymbol{G}} + \lambda \boldsymbol{I})^{-1} \hat{\boldsymbol{k}}(\boldsymbol{x})$$

where the (i,j)-th element of $\hat{\boldsymbol{G}}$ is $K(\boldsymbol{x}_i, \boldsymbol{x}_j) + 1$ and the i-th element of $\boldsymbol{k}(\boldsymbol{x})$ is $K(\boldsymbol{x}_i, \boldsymbol{x}) + 1$, cf. p. 37 chapter 2 in notes.

MATLAB example: ridge regression

```
>> X=[3 7;4 6;5 6;7 7;8 5;4 5;5 5;6 3;7 4;9 4];
>> ell=size(X,1)
ell =
    10
>> n=size(X,2)
n =
     2
>> y=[6 5 5 7 5 4 4 2 3 4]';
>> lambda = 10;
% Ridge regression with no bias and lambda=10, primal solution
>> Xr=[X;sqrt(lambda)*eye(n)];
>> yr = [y;zeros(n,1)];
>> w = Xr\yr
w =
    0.1195
    0.7271
>> y=X*w
y =
    5.4483
    4.8407
    4.9602
    5.9263
    4.5916
    4.1136
    4.2331
    2.8983
    3.7450
    3.9840
% Ridge regression with bias and lambda=10, primal solution
>> Xrh = [Xr [ones(ell,1);zeros(n,1)]];
>> wh = Xrh\yr
wh =
    0.0669
    0.6527
    0.7176
>> w = wh(1:n);
>> b = wh(n+1);
>> yest=X*w+b
yest =
    5.4874
    4.9017
    4.9686
    5.7552
    4.5167
    4.2490
    4.3159
    3.0774
    3.7971
    3.9310
% Ridge regression with no bias and lambda=10, dual kernel solution
>> G=X*X';
>> alpha = (G + lambda*eye(ell))\(2*lambda*y)
alpha =
    1.1034
    0.3186
    0.0796
    2.1474
    0.8169
   -0.2271
   -0.4661
   -1.7967
   -1.4899
    0.0321
>> yest = 1/(2*lambda)*G*alpha
yest =
    5.4483
    4.8407
    4.9602
    5.9263
    4.5916
    4.1136
    4.2331
    2.8983
    3.7450
    3.9840
>> wk = (1/(2*lambda))*X'*alpha
wk =
    0.1195
    0.7271
% Ridge regression with bias and lambda=10, dual kernel solution
>> H=0.5*((1/lambda)*G+eye(ell));
>> f=-y;
>> A=zeros(1,ell);
>> b=0;
>> Aeq=ones(1,ell);
>> beq=0;
>> alpha=quadprog(H,f,A,b,Aeq,beq)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
alpha =
    1.0251
    0.1967
    0.0628
    2.4895
    0.9665
   -0.4979
   -0.6318
   -2.1548
   -1.5941
    0.1381
>> b = y(1)-(1/(2*lambda))*G(1,:)*alpha-0.5*alpha(1)
b =
    0.7176
```

```
>> yest=1/(2*lambda)*G*alpha+b
yest =
    5.4874
    4.9017
    4.9686
    5.7552
    4.5167
    4.2490
    4.3159
    3.0774
    3.7971
    3.9310
>> wkh=(1/(2*lambda))*X'*alpha
wkh =
    0.0669
    0.6527
```

Support-vector regression with quadratic ε-insensitive loss

Consider now the problem of minimizing:

$$\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{\ell} ({\xi_i^+}^2 + {\xi_i^-}^2)$$

subject to:

$$\begin{aligned} \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b - y_i &\leq \varepsilon + \xi_i^- \\ y_i - \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle - b &\leq \varepsilon + \xi_i^+ \\ \xi_i^+, \xi_i^- \geq 0 & \qquad i = 1, 2, \ldots, \ell \end{aligned}$$

Note the this corresponds to ridge regression if $\varepsilon = 0$ and $C = 1/\lambda$ and

$$\xi_i = \begin{cases} \xi_i^+ & \text{if } \xi_i \geq 0 \\ \\ \xi_i^- & \text{if } \xi_i \leq 0 \end{cases} \qquad \text{i.e. } \xi_i = \xi_i^+ - \xi_i^-$$

and the difference is that ξ_i does not affect the cost function that we are minimizing if $|\xi_i| \leq \varepsilon$. When $|\xi_i| > \varepsilon$ we are only considering the error beyond ε.

The statistical background to this formulation is that if we solve this problem with $\varepsilon = \theta - \gamma$ where $\gamma \leq \theta$ then we are assured with probability $1 - \delta$ that if $(\boldsymbol{x}, y)$ are distributed according to any fixed probability distribution $\mathcal{D}$, with support in a region in $\mathcal{X} \times \mathbb{R}$ bounded by a sphere of radius R, and we have determined $\boldsymbol{w}$ and b from ℓ randomly chosen data-points from $\mathcal{D}$, and we subsequently choose randomly a new $(\boldsymbol{x}, y)$ and evaluate y_{est} from $\boldsymbol{x}$ according to the model $\langle \boldsymbol{w} \cdot \boldsymbol{x} \rangle + b$, then the probability that $|y - y_{est}| > \theta$ is bounded by:

$$\frac{c}{\ell}\left(\frac{\|\boldsymbol{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_2^2}{\gamma^2} \log^2 \ell + \log \tfrac{1}{\delta} \right)$$

where c is a constant independent of $\mathcal{D}$ and ℓ, cf. Theorem 4.28 p. 72 in book.

When we know the data vectors $\boldsymbol{x}_i$, $i = 1, 2, \ldots, \ell$ we can solve the minimization problem directly using QUADPROG by minimizing

$$\boldsymbol{z}' \boldsymbol{H} \boldsymbol{z}$$

where $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{\xi}^+ \\ \boldsymbol{\xi}^- \end{bmatrix}$ $\boldsymbol{H} = \text{diag}([\text{ones}(1, n)\ 0\ C \times \text{ones}(1, 2\ell)])$

and LB $= [-\inf\ \ 0 \ldots 0]'$ although the lower bounds on ξ_i^+ and ξ_i^- are in fact superfluous.

When we only know the kernel-values we can consider a dual problem in a similar fashion to ridge regression. This problem is to maximize[4]:

$$\begin{aligned}&\textstyle\sum_{i=1}^{\ell} y_i(\alpha_i^+ - \alpha_i^-) - \varepsilon \sum_{i=1}^{\ell}(\alpha_i^+ + \alpha_i^-)\\ -&\tfrac{1}{2}\textstyle\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}(\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-)(K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \frac{1}{C}\delta_{ij})\end{aligned}$$

subject to: $\sum_{i=1}^{\ell}(\alpha_i^+ - \alpha_i^-) = 0$ and $\alpha_i^+, \alpha_i^- \geq 0$.

This problem can again be solved using `QUADPROG` by minimizing:

$$\tfrac{1}{2}\boldsymbol{z}'\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}'\boldsymbol{z}$$

where $\boldsymbol{z} = \begin{bmatrix}\boldsymbol{\alpha}^+\\ \boldsymbol{\alpha}^-\end{bmatrix}$ $\boldsymbol{H} = \begin{bmatrix}\boldsymbol{G} + \frac{1}{C}\boldsymbol{I} & -G - \frac{1}{C}\boldsymbol{I}\\ -\boldsymbol{G} - \frac{1}{C}\boldsymbol{I} & G + \frac{1}{C}\boldsymbol{I}\end{bmatrix}$
$\boldsymbol{f} = \varepsilon \begin{bmatrix}1\\ \vdots\\ 1\end{bmatrix} - \begin{bmatrix}\boldsymbol{y}\\ -\boldsymbol{y}\end{bmatrix}$ subject to $[1 \ldots 1 - 1 \ldots - 1]\boldsymbol{z} = 0$
and LB $= [0 \ldots 0]'$.

[4]We find the notation confusing in the book, they use α where we use α^-, they use $\hat{\alpha}$ where we use α^+, and they use β and then later α for β while we simply use α consistently.

Having obtained $\begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix}$ we can calculate $\boldsymbol{\alpha} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$ and then

$$b = y_i - \sum_{j=1}^{\ell} K(\boldsymbol{x}_i, \boldsymbol{x}_j)\alpha_j - \varepsilon - \alpha_i / C$$

using the active constraint from the primal problem characterized by the fact that $\alpha_i > 0$ and that we now have the relationships $\boldsymbol{w} = \sum_{i=1}^{\ell} \alpha_i \boldsymbol{x}_i$ and $\xi_i^+ = \alpha_i^+ / C$, $\xi_i^- = \alpha_i^- / C$. Finally, the model becomes:

$$f(\boldsymbol{x}) = \boldsymbol{\alpha}' \boldsymbol{k}(\boldsymbol{x}) + b$$

Note that it is only the term $\varepsilon \sum_{i=1}^{\ell} (\alpha_i^+ - \alpha_i^-) = \varepsilon \sum_{i=1}^{\ell} |\alpha_i|$ that forces us to introduce α_i^+ and α_i^- instead of working with just α_i. In particular when $\varepsilon = 0$ this is not necessary and the formulation reduces to that of the ridge regression problem.

MATLAB example: quadratic ε-insensitive loss

```
% Quadratic epsilon-insensitive loss with eps=0.5 and C=0.1, primal solution
>> C = 1/10;
>> epsilon = 0.5;
>> H = diag([ones(1,n) 0 C*ones(1,2*ell)]);
>> f = zeros(n+1+2*ell,1);
>> A = [X ones(ell,1) zeros(ell) -eye(ell);
       -X -ones(ell,1) -eye(ell) zeros(ell)];
>> b = [epsilon + y; epsilon - y];
>> Aeq = zeros(1,n+1+2*ell);
>> beq = 0;
>> LB = [-inf*ones(n+1,1);zeros(2*ell,1)];
>> wbxipxim = quadprog(H,f,A,b,Aeq,beq,LB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
The equality constraints are dependent.
The system of equality constraints is consistent. Removing
the following dependent constraints before continuing:
     1
Optimization terminated successfully.
wbxipxim =
    0.0215
    0.4874
    1.8321
    0.1917
         0
         0
    1.1055
    0.0587
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.9235
    0.4324
         0
>> w = wbxipxim(1:n);
>> b = wbxipxim(n+1);
>> yest = X*w + b
yest =
    5.3083
    4.8425
    4.8640
    5.3945
    4.4413
    4.3551
    4.3767
    3.4235
    3.9324
    3.9755
>> y-yest
ans =
    0.6917
    0.1575
    0.1360
    1.6055
    0.5587
   -0.3551
   -0.3767
   -1.4235
   -0.9324
    0.0245
% Quadratic epsilon-insensitive loss with epsilon=0.5 and C=0.1, dual solution
```

```
>> G=X*X';
>> GC = G+(1/C)*eye(ell);
>> H = [GC -GC; -GC GC];
>> f = [epsilon - y; epsilon + y];
>> A = zeros(1,2*ell);
>> b = 0;
>> Aeq = [ones(1,ell) -ones(1,ell)];
>> beq = 0;
>> LB = zeros(2*ell,1);
>> alphapalpham = quadprog(H,f,A,b,Aeq,beq,LB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
alphapalpham =
    0.0192
   -0.0000
    0.0000
    0.1105
    0.0059
    0.0000
   -0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
   -0.0000
    0.0000
    0.0923
    0.0432
   -0.0000
>> alpha = alphapalpham(1:ell)-alphapalpham(ell+1:2*ell);
>> i = min(find(alpha>0.00000001))
```

```
i =
     1
>> b = y(i)-G(i,:)*alpha-epsilon-alpha(i)/C
b =
    1.8321
>> yest = G*alpha + b
yest =
    5.3083
    4.8425
    4.8640
    5.3945
    4.4413
    4.3551
    4.3767
    3.4235
    3.9324
    3.9755
>> w = X'*alpha
w =
    0.0215
    0.4874
```

Support-vector regression with linear ε-insensitive loss

Now we minimize:

$$\tfrac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C \sum_{i=1}^{\ell} (\xi_i^+ + \xi_i^-)$$

subject to:

$$\begin{array}{rcl} \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b - y_i & \leq & \varepsilon + \xi_i^- \\ y_i - \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle - b & \leq & \varepsilon + \xi_i^+ \\ \xi_i^+, \xi_i^- \geq 0 & & i = 1, 2, \ldots, \ell \end{array}$$

The statistical background to this formulation is analogous to the quadratic case and is expressed by Theorem 4.30, p. 74 in the book.

When we know the data vectors $\boldsymbol{x}_i$, $i = 1, 2 \ldots, \ell$ we can solve this directly using `QUADPROG` as in the quadratic case except now we minimize:

$$\tfrac{1}{2}\boldsymbol{z}'\boldsymbol{H}\boldsymbol{z} + \boldsymbol{f}'\boldsymbol{z}$$

where $\boldsymbol{z}$ is as before but
$\boldsymbol{H} = \text{diag}([\text{ones}(1, n)\ 0\ C \times \text{zeros}(1, 2\ell)])$ and
$\boldsymbol{f}' = [\text{zeros}(1, n)\ 0\ C \times \text{ones}(1, 2\ell)]$. The constraints are the same and the lower bounds on ξ_i^+ and ξ_i^- are now essential.

When we only know the kernel-value we can again consider a dual problem which is the same as that of the quadratic problem except we do not have any $\frac{1}{C}\delta_{ij}$ term in the cost function and we now have both lower and upper bounds on α_i^+ and α_i^-, $0 \le \alpha_i^+, \alpha_i^- \le C$.

This can be solved by `QUADPROG` in a similar fashion as before. Having obtained $\begin{bmatrix} \boldsymbol{\alpha}^+ \\ \boldsymbol{\alpha}^- \end{bmatrix}$ we can calculate $\boldsymbol{\alpha} = \boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-$ as before but now

$$b = y_i - \sum_{j=1}^{\ell} K(\boldsymbol{x}_i, \boldsymbol{x}_j)\alpha_j - \varepsilon$$

from an active constraint that is now characterized by the fact that $0 < \alpha_i < C$ (and hence $\xi_i = 0$). Finally, the model becomes:

$$f(\boldsymbol{x}) = \boldsymbol{\alpha}'\boldsymbol{k}(\boldsymbol{x}) + b$$

as before.

MATLAB example: linear ε-insensitive loss

```
% Linear epsilon-insensitive loss with epsilon=0.5 and C=0.1, primal solution
>> H = diag([ones(1,n) 0 C*zeros(1,2*ell)]);
>> f = [zeros(n+1,1);C*ones(2*ell,1)];
>> A = [X ones(ell,1) zeros(ell) -eye(ell);
-X -ones(ell,1) -eye(ell) zeros(ell)];
>> b = [epsilon + y; epsilon - y];
>> Aeq = zeros(1,n + 1 + 2*ell);
>> beq = 0;
>> LB = [-inf*ones(n+1,1);zeros(2*ell,1)];
>> wbxipxim = quadprog(H,f,A,b,Aeq,beq,LB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
The equality constraints are dependent.
The system of equality constraints is consistent. Removing
the following dependent constraints before continuing:
     1
Optimization terminated successfully.
wbxipxim =
    0.0276
    0.5690
    1.4345
   -0.0000
         0
         0
    0.8897
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.8069
    0.4034
         0
>> w = wbxipxim(1:n);
>> b = wbxipxim(n+1);
>> yest = X*w + b
yest =
    5.5000
    4.9586
    4.9862
    5.6103
    4.5000
    4.3897
    4.4172
    3.3069
    3.9034
    3.9586
>> y - yest
ans =
    0.5000
    0.0414
    0.0138
    1.3897
    0.5000
   -0.3897
   -0.4172
   -1.3069
   -0.9034
    0.0414
% Linear epsilon-insensitive loss with epsilon=0.5 and C=0.1, dual solution
>> H = [G -G; -G G];
>> f = [epsilon - y; epsilon + y];
>> A = zeros(1,2*ell);
>> b = 0;
>> Aeq = [ones(1,ell) -ones(1,ell)];
>> LB = zeros(2*ell,1);
>> UB = C*ones(2*ell,1);
>> allphapalpham = quadprog(H,f,A,b,Aeq,beq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
allphapalpham =
    0.0345
    0.0000
    0.0000
    0.1000
    0.0655
    0.0000
    0.0000
    0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
   -0.0000
   -0.0000
    0.1000
    0.1000
   -0.0000
>>alpha = alphapalpham(1:ell)-alphapalpham(ell+1:2*ell)
>> i = find(alpha>0.0000001 & alpha<C-0.0000001)
i =
     1
     5
>> b = y(i)-G(i,:)*alpha-epsilon
```

```
b =
    1.4345
    1.4345
>> b = mean(b)
b =
    1.4345
>> yest = G*alpha + b
yest =
    5.5000
    4.9586
    4.9862
    5.6103
    4.5000
    4.3897
    4.4172
    3.3069
    3.9034
    3.9586
>> w = X'*alpha
w =
    0.0276
    0.5690
```

ν – support vector regression

The parameter used in ε-insensitive loss is not always known to a specific level of accuracy a priori. The following modification, called ν-SVR, automatically computes ε.

The size of ε is traded off against model complexity and slack variables ξ_i via a constant $\nu \geq 0$, by minimizing:

$$\tfrac{1}{2}\langle \boldsymbol{w} \cdot \boldsymbol{w} \rangle + C\left(\nu\varepsilon + \frac{1}{\ell}\sum_{i=1}^{\ell}(\xi_i^- + \xi^+) \right)$$

subject to:

$$\begin{array}{lcl} \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle + b - y_i & \leq & \varepsilon + \xi_i^- \\ y_i - \langle \boldsymbol{w} \cdot \boldsymbol{x}_i \rangle - b & \leq & \varepsilon + \xi_i^+ \\ \xi_i^+, \xi_i^- \geq 0, \varepsilon \geq 0 & & i = 1, 2, \ldots, \ell \end{array}$$

Following the usual procedure of finding first the saddle point of the Lagrangian and inserting back into the Lagrangian the resulting equations, the dual formulation is, minimize:

$$\sum_{i=1}^{\ell}(\alpha_i^+ - \alpha_i^-)y_i - \tfrac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}(\alpha_i^+ - \alpha_i^-)(\alpha_j^+ - \alpha_j^-)K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to $\sum_{i=1}^{\ell}(\alpha_i^- - \alpha_i^+) = 0$, $0 \leq \alpha_i^+, \alpha_i^- \leq C/\ell$ and $\sum_{i=1}^{\ell}(\alpha_i^+ + \alpha^-) \leq C\nu$, for $i = 1, 2, \ldots, \ell$.

In LIBSVM the equivalent problem is solved:

$$\sum_{i=1}^{\ell} \alpha_i y_i - \tfrac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

subject to $\sum_{i=1}^{\ell}(\alpha_i^- - \alpha_i^+) = 0$, $0 \leq \alpha_i^+, \alpha_i^- \leq C$ and $\sum_{i=1}^{\ell}(\alpha_i^+ + \alpha^-) = C\ell\nu$, for $i = 1, 2, \ldots, \ell$.

The bias b is computed as in the linear ε-insensitive case for $0 < \alpha_i < C$. The same way we can compute ε. The regression estimate takes as before the form:

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b = \boldsymbol{\alpha}'\boldsymbol{k}(\boldsymbol{x}) + b$$

If ν-SVR is applied to some data set and the resulting ε is nonzero, then:

- ν is an upper bound on the fraction of errors,
- ν is a lower bound on the fraction of SVs.

MATLAB example using ν-SVR

```
% Linear eps-insensitive loss, nu-version with nu=0.5 and C=1.0, primal solution
>> C = 1.0;
>> nu = 0.5;
>> H = diag([ones(1,n) zeros(1,1+2*ell+1)]);
>> f = [zeros(n+1,1);(C/ell)*ones(2*ell,1);C*nu];
>> A = [ X ones(ell,1) zeros(ell) -eye(ell) -ones(ell,1);
        -X -ones(ell,1) -eye(ell) zeros(ell) -ones(ell,1)];
>> b = [y; -y];
>> Aeq = zeros(1,n+1+2*ell+1);
>> beq = 0;
>> LB = [-inf*ones(n+1,1);zeros(2*ell+1,1)];
>> wbxipximeps = quadprog(H,f,A,b,Aeq,beq,LB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
The equality constraints are dependent.
The system of equality constraints is consistent. Removing
the following dependent constraints before continuing:
     1
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 8.726386e-017.
> In C:\MATLAB\toolbox\optim\private\compdir.m at line 25
  In C:\MATLAB\toolbox\optim\private\qpsub.m at line 694
  In C:\MATLAB\toolbox\optim\quadprog.m at line 247
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 8.726386e-017.
> In C:\MATLAB\toolbox\optim\private\compdir.m at line 25
  In C:\MATLAB\toolbox\optim\private\qpsub.m at line 694
  In C:\MATLAB\toolbox\optim\quadprog.m at line 247
Optimization terminated successfully.
wbxipximeps =
    0.0483
    0.6207
    1.0828
   -0.0000
         0
         0
    0.8069
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.8069
    0.4759
         0
    0.4276
>> w = wbxipximeps(1:n)
w =
    0.0483
    0.6207
>> b = wbxipximeps(n+1)
b =
    1.0828
>> yest = X*w + b
yest =
    5.5724
    5.0000
    5.0483
    5.7655
    4.5724
    4.3793
    4.4276
    3.2345
    3.9034
    4.0000
% Linear eps-insensitive loss, nu-version with nu=0.5 and C=1.0, dual solution
>> H = [G -G;-G G];
>> f = [-y; y];
>> A = ones(1,2*ell);
>> b = C*nu;
>> Aeq = [ones(1,ell) -ones(1,ell)];
>> beq = 0;
>> LB = zeros(2*ell,1);
>> UB = (C/ell)*ones(2*ell,1);
>> alphapalpham = quadprog(H,f,A,b,Aeq,beq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
alphapalpham =
    0.0603
   -0.0000
         0
    0.1000
    0.0897
    0.0000
    0.0000
    0.0000
    0.0000
   -0.0000
    0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
    0.0500
    0.1000
    0.1000
```

```
         0
>> alpha = alphapalpham(1:ell)-alphapalpham(ell+1:2*ell);
>> [sum(alphapalpham) C*nu] % active constraint
ans =
    0.5000    0.5000
>> i = find(alpha>0.0000001 & alpha <C/ell-0.00000001)
i =
     1
     5
>> j = find(alpha<-0.0000001 & alpha>-C/ell+0.00000001)
j =
     7
>> beps = [1 1;-1 1]\[y(i(1))-G(i(1),:)*alpha;G(j(1),:)*alpha-y(j(1))]
beps =
    1.0828
    0.4276
>> b = beps(1)
b =
    1.0828
>> eps = beps(2)
eps =
    0.4276
>> yest = G*alpha+b
yest =
    5.5724
    5.0000
    5.0483
    5.7655
    4.5724
    4.3793
    4.4276
    3.2345
    3.9034
    4.0000
>> w = X'*alpha
w =
    0.0483
    0.6207
```

```
% Linear eps-indensitive loss, nu-version with nu=0.5 and C=1.0, LIBSVM dual
>> C = 1.0/ell
>> A = zeros(1,2*ell);
>> b = 0;
>> Aeq = [ones(1,ell) -ones(1,ell); ones(1,ell) ones(1,ell)];
>> beq = [0;C*nu];
>> LB = zeros(2*ell,1);
>> UB = (C/ell)*ones(2*ell,1);
>> alphapalpham = quadprog(H,f,A,b,Aeq,beq,LB,UB)
Warning: Large-scale method does not currently solve this problem formulation,
switching to medium-scale method.

> In C:\MATLAB\toolbox\optim\quadprog.m at line 213
Optimization terminated successfully.
alphapalpham =
    0.0603
         0
    0.0000
    0.1000
    0.0897
    0.0000
   -0.0000
   -0.0000
   -0.0000
    0.0000
    0.0000
   -0.0000
    0.0000
    0.0000
    0.0000
         0
    0.0500
    0.1000
    0.1000
    0.0000
>> alpha = alphapalpham(1:ell)-alphapalpham(ell+1:2*ell);
>> [sum(alphapalpham) C*nu]
ans =
    0.5000    0.5000
>> i = find(alpha>0.0000001 & alpha < C/ell-0.00000001)
i =
     1
     5
>> j = find(alpha<-0.0000001 & alpha>-C/ell+0.00000001)
j =
     7
>> beps = [1 1;-1 1]\[y(i(1))-G(i(1),:)*alpha;G(j(1),:)*alpha-y(j(1))]
beps =
    1.0828
    0.4276
>> b = beps(1)
b =
    1.0828
>> eps = beps(2)
eps =
    0.4276
>> yest = G*alpha+b
yest =
    5.5724
    5.0000
    5.0483
    5.7655
    4.5724
    4.3793
    4.4276
    3.2345
    3.9034
    4.0000
>> w = X'*alpha
w =
    0.0483
    0.6207
% and the slack is computed from the iequalities
>>xip = -(G(i,:)*alpha + b) + y(i) - eps
xip =
1.0e-14 *
```

```
         0
    0.2665
>>xim = (G(j,:)*alpha + b) - y(j) - eps
xim =
         0
>> margin = theta - eps
?? Undefined function or variable 'theta'.
% Yep, what is theta?
>> let's say theta = 0.5 (is eps is 0 then margin = theta)
>> theta = 0.5
>> margin = theta - eps
margin =
    0.0724
% the radius would be computed by
>> R = max(sqrt(diag(G)+y.^2))
R =
   12.1244
% criteria to minimize (model selection) ??
>>citeria = (R/margin).^2 + sum([xip;xim])
citeria =
   2.8033e+04
```

LIBSVM ν-SVR example: time series prediction

You are given a data set called `mg17.dat`. This data series was generated by the Mackey-Glass delay differential equation:

$$\frac{dx(t)}{dt} = -0.1x(t) + \frac{0.2x(t-t_\Delta)}{1+x(t-t_\Delta)^{10}}$$

with delay $t_\Delta = 17$. This equation was originally introduced as a model of blood cell regulation (Mackey and Glass, 1977) and became common as an artificial forecasting benchmark.

Let $\{x(t)\}$, $t = 1, \ldots, T$, be a time series that was generated by a dynamical system, such as the one above. We assume that $\{x(t)\}$ is a projection of a dynamics operating in a high-dimensional state space, where the state space vector is denoted as:

$$\boldsymbol{x}_t = (x(t), x(t-\tau), \ldots, x(t-n-1)\tau)),$$

with time delay τ and embedding dimension n. If we assume τ and n are known, then ℓ training points for a time-series may be generated as follows:

```
load mg17.dat; mg17 = mg17';
% create training set matrix
n = 6; tau = 6; ell = 1000;
for i = 1:ell,
  X(i,:) = mg17(i : tau : i + n * tau);
  y(i,1) = mg17(i + n * tau + 1);
end
% write to LIBSVM style file
mat2libsvm(X,y,'mg17libsvm.dat');
% create test set (the remaining time-series)
mg17testing = mg17((ell-tau*n):end);
```

Now select parameters for SVMTRAIN and train!

```
sigma = sqrt(.75);          % kernel parameter
libsvmgamma = 1/(2*sigma^2) % used by libsvm
nu = 0.05                   % the nu parameter
epsilon = 0.1;              % used with the C-SVR only
C = 15                      % The C parameter
eval(['!svmtrain -s 3 -t 2' ... % use -s 3 or -s 4
        ' -g ' num2str(libsvmgamma) ...
        ' -n ' num2str(nu) ....
        ' -c ' num2str(C) ...
        ' -p ' num2str(epsilon) ...
        ' mg17libsvm.dat model.dat']);
```

We now load the model and compute some properties:

```
% the model
param = libsvm2mat('model.dat');
% the bias
b = -param.rho;
% the Lagrangian multipliers
alpha = param.alpha;
% the support vectors
Xsv = param.SV;

% the Kernel matrix (based on SVs)
```

```
for i = 1:size(Xsv,1),
  for j = 1:size(Xsv,1),
    Gsv(i,j) = exp(-sum((Xsv(i,:)-Xsv(j,:)).^2)/(2*sigma^2));
  end,
end

% the Kernel matrix (based on Xs)
for i = 1:size(Xsv,1),
  for j = 1:size(Xsv,1),
    G(i,j) = exp(-sum((X(i,:)-X(j,:)).^2)/(2*sigma^2));
  end,
end


% the radius
R = max(diag(G) + y.^2)
% the margin

% for now see assigment 12 and last MATLAB example
```