Next Up Previous
Next: Multi-Class Classification **Up:** Support Vector machine **Previous:** Soft Margin SVM

Sequential Minimal Optimization (SMO) Algorithm

The sequential minimal optimization (SMO, due to John Platt 1998, also see notes here) is a more efficient algorithm for solving the SVM problem, compared with the generic QP algorithms such as the internal-point method. The SMO algorithm can be considered as a method of decomposition, by which an optimization problem of multiple variables is decomposed into a series of subproblems each optimizing an objective function of a small number of variables, typically only one, while all other variables are treated as constants that remain unchanged in the subproblem. For example, the coordinate descent algorithm is just such a decomposition method, which solves a problem in a multi-dimensional space by converting it into a sequence of subproblems each in a one-dimensional space.

When applying the decomposition method to the soft margin SVM problem, we could consider optimizing only one variable α_i at a time, while treating all remaining variables $\alpha_1, \cdots, \alpha_{i-1}, \alpha_{i+1}, \cdots, \alpha_N$ as constants. However, due to the constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$, α_i is a linear combination of $N-1$ constants and is therefore also a constant. We therefore need to consider two variables at a time, here assumed to be α_i and α_j, while treating the remaining $N-2$ variables as constants. The objective function of such a subproblem can be obtained by dropping all constant terms independent of the two selected variables α_i and α_j in the original objective function in Eq. (132):

$$\begin{aligned}
\text{maximize:} \quad & L(\alpha_i, \alpha_j) = \alpha_i + \alpha_j - \frac{1}{2}\left(\alpha_i^2 \mathbf{x}_i^T \mathbf{x}_i + \alpha_j^2 \mathbf{x}_j^T \mathbf{x}_j + 2\alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\right) \\
& -\alpha_i y_i \left(\sum_{n \neq i} \alpha_n y_n \mathbf{x}_n^T\right) \mathbf{x}_i - \alpha_j y_j \left(\sum_{n \neq j} \alpha_n y_n \mathbf{x}_n^T\right) \mathbf{x}_j \\
= & \alpha_i + \alpha_j - \frac{1}{2}\left(\alpha_1^2 K_{ii} + \alpha_2^2 K_{jj} + 2\alpha_i \alpha_j y_i y_j K_{ij}\right) \\
& -\alpha_i y_i \sum_{n \neq i,j} \alpha_n y_n K_{ni} - \alpha_j y_j \sum_{n \neq i,,j} \alpha_n y_n K_{nj} \\
\text{subject to:} \quad & 0 \leq \alpha_i, \alpha_j \leq C, \qquad \sum_{n=1}^{N} \alpha_n y_n = 0
\end{aligned} \tag{136}$$

where, to use the kernel method, all inner products $\mathbf{x}_m^T \mathbf{x}_n$ are replaced by the corresponding kernel function $K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n) = \mathbf{z}_m^T \mathbf{z}_n$ with $\mathbf{z}_n = \phi(\mathbf{x}_n)$.

This maximization problem can be solved iteratively. Out of all previous values α_n^{old} $(n = 1, \cdots, N)$, the two selected variables are updated to their correspoinding new values α_i^{new} and α_j^{new} that maximize $L(\alpha_i, \alpha_j)$, subject to the two constraints.

We rewrite the second constraint as

$$\alpha_i y_i + \alpha_j y_j = -\sum_{n \neq i,j} \alpha_n y_n \tag{137}$$

and multiply both sides by y_i to get

$$y_i^2 \alpha_i + y_i y_j \alpha_j = \alpha_i + s\alpha_j = \left(-\sum_{n \neq i,j} \alpha_n y_n\right) y_i = \delta, \quad \text{i.e.} \quad \alpha_i = \delta - s\alpha_j \tag{138}$$

where we have defined $s = y_i y_j$ and $\delta = -\sum_{n \neq i,j} \alpha_n y_n y_i$. As δ is independent of α_i and α_j, it remains the same before and after they are updated, and we have

$$\alpha_i^{new} + s\alpha_j^{new} = \alpha_i^{old} + s\alpha_j^{old} = \delta \tag{139}$$

i.e.,

$$\Delta\alpha_i = \alpha_i^{new} - \alpha_i^{old} = -s(\alpha_j^{new} - \alpha_j^{old}) = -s\Delta\alpha_j \tag{140}$$

We now consider a closed-form solution for updating α_i and α_j in each iteration of this two-variable optimization problem. We first rewrite Eq. (87) as

$$\sum_{n \neq i,j} \alpha_n y_n \mathbf{x}_n = \mathbf{w} - \alpha_i y_i \mathbf{x}_i - \alpha_j y_j \mathbf{x}_j \tag{141}$$

so that the two summations in $L(\alpha_i, \alpha_j)$, now denoted by v_k $(k = i, j)$, can be written as

$$v_k =$$

$$\begin{aligned}\sum_{n\neq i,j}\alpha_n y_n K_{nk} &= \sum_{n=1}^{N}\alpha_n y_n K_{nk} - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk}\\ &= \left(\sum_{n=1}^{N}\alpha_n y_n K_{nk} + b\right) - b - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk}\\ &= u_k - b - \alpha_i y_i K_{ik} - \alpha_j y_j K_{jk}\end{aligned} \tag{142}$$

where

$$u_k = f(\mathbf{z}_k) = \sum_{n=1}^{N}\alpha_n y_n K_{nk} + b \tag{143}$$

is the output in Eq. (115) corresponding to input $\mathbf{x}_k$. Now the objective function above can be written as:

$$L(\alpha_i,\alpha_j) = \alpha_i + \alpha_j - \frac{1}{2}(\alpha_i^2 K_{ii} + \alpha_j^2 K_{jj} + 2s\alpha_i\alpha_j K_{ij}) - \alpha_i y_i v_i - \alpha_j y_j v_j \tag{144}$$

Substituting $\alpha_i = \delta - s\alpha_j$ (Eq. (140)) into $L(\alpha_i,\alpha_j)$, we can rewrite it as a function of a single variable α_j alone:

$$\begin{aligned}L(\alpha_j) &= \delta + (1-s)\alpha_j - \frac{1}{2}\left[(\delta - s\alpha_j)^2 K_{ii} + \alpha_j^2 K_{jj} + 2s(\delta - s\alpha_j)\alpha_j K_{ij}\right]\\ &\quad -(\delta - s\alpha_j)y_i v_i - \alpha_j y_j v_j\\ &= \frac{1}{2}(2K_{ij} - K_{ii} - K_{jj})\alpha_j^2 + [1 - s + s\delta(K_{ii} - K_{ij}) + y_j(v_i - v_j)]\alpha_j + \text{Const.}\end{aligned} \tag{145}$$

where $s^2 = (y_i y_j)^2 = 1$, and scalar Const. represents all constant terms independent of α_i and α_j and can therefore be dropped. Now the objective function becomes a quadratic function of the single variable α_j. Consider the coefficients of this quadratic function:

- The coefficient of α_j^2:

$$\frac{1}{2}(2K_{ij} - K_{ii} - K_{jj}) = \frac{1}{2}\eta \tag{146}$$

  where η is defined as

$$\begin{aligned}\eta &= 2K_{ij} - K_{ii} - K_{jj} = 2\mathbf{z}_i^T\mathbf{z}_j - \mathbf{z}_i^T\mathbf{z}_i - \mathbf{z}_j^T\mathbf{z}_j = -(\mathbf{z}_i - \mathbf{z}_j)^T(\mathbf{z}_i - \mathbf{z}_j)\\ &= -||\mathbf{z}_i - \mathbf{z}_j||^2 \leq 0\end{aligned} \tag{147}$$

- The coefficient of α_j:

$$\begin{aligned}&1 - s + s\delta(K_{ii} - K_{ij}) + y_j(v_i - v_j)\\ &= 1 - s + (s\alpha_i + \alpha_j)(K_{ii} - K_{ij})\\ &\quad + y_j[(u_i - b - \alpha_i y_i K_{ii} - \alpha_j y_j K_{ij}) - (u_j - b - \alpha_i y_i K_{ij} - \alpha_j y_j K_{jj})]\\ &= 1 - s + s\alpha_1(K_{ii} - K_{ij}) + \alpha_2(K_{ii} - K_{ij})\\ &\quad + y_j(u_i - u_j) - \alpha_i s K_{ii} - \alpha_j K_{ij} + \alpha_i s K_{ij} + \alpha_j K_{jj}\\ &= y_j^2 - s - \alpha_j(2K_{ij} - K_{ii} - 2K_{jj}) + y_j(u_i - u_j)\\ &= y_j(u_i - y_j - u_j + y_j) - \alpha_j(2K_{ij} - K_{ii} - K_{jj})\\ &= y_j(E_i - E_j) - \alpha_j\eta\end{aligned} \tag{148}$$

  where we have defined

$$E_k = u_k - y_k = \sum_{n=1}^{N}\alpha_n^{old} y_n K_{nk} + b - y_k \qquad (k = 1, 2) \tag{149}$$

  as the difference between the desired output y_k and the actual output based on the previous values of the variables α_n^{old} $(n = 1, \cdots, N)$.

Now the quadratic objective function can be rewritten as:

$$L(\alpha_j) = \frac{1}{2}\eta\alpha_j^2 + \left[y_j(E_i - E_j) - \alpha_j^{old}\eta\right]\alpha_j \tag{150}$$

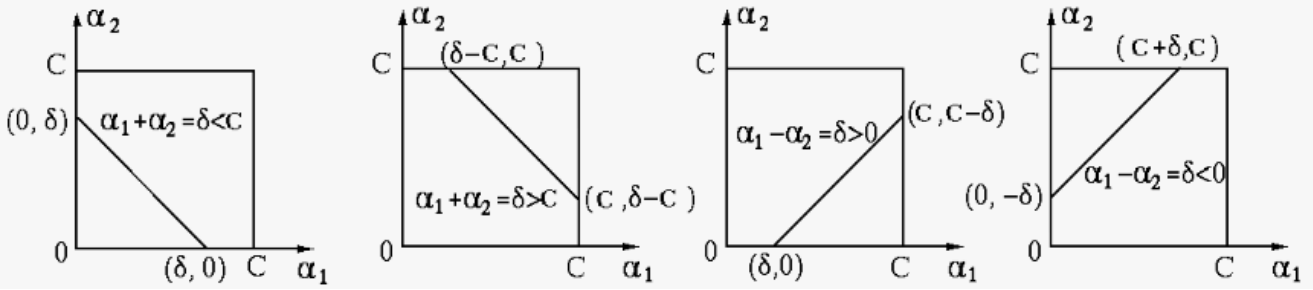
Given the previous values α_n^{old} $(n = 1, \cdots, N)$ in the coefficients, we will find α_j^{new} that maximizes $L(\alpha_j)$. Consider its first and second order derivatives:

$$\begin{aligned}\frac{d}{d\alpha_j}L(\alpha_j) &= \eta\alpha_j + y_j(E_i - E_j) - \alpha_j^{old}\eta\\ \frac{d^2}{d\alpha_j^2}L(\alpha_j) &= \eta \leq 0\end{aligned} \tag{151}$$

As the second order derivative of $L(\alpha_j)$ is $\eta \leq 0$, it has a maximum. Solving the equation $d\,L(\alpha_j)/d\alpha_j = 0$, we get α_j^{new} that maximizes $L(\alpha_j)$ based on the old value α_j^{old}:

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_j - E_i)}{\eta} = \alpha_j^{old} + \Delta\alpha_j, \qquad \Delta\alpha_j = \alpha_j^{new} - \alpha_j^{old} = \frac{y_j(E_j - E_i)}{\eta}, \tag{152}$$

Note that this is an unconstrained solution, which needs to be modified if the constraints in Eq. (136) are violated. As shown in the figure below, α_i and α_j are inside the square of size C, due to the first constraint $0 \le \alpha_i,\ \alpha_j \le C$; also, due to the second constraint $\alpha_i + s\alpha_j = \delta$ (Eq. (138)), they have to be on the diagonal line segment inside the square.



The four panels correspond to the four possible cases, depending on whether $s = 1$ ($y_i = y_j = \pm 1$, panels 1 and 2), or $s = -1$ ($y_i = -y_j = \pm 1$, panels 3 and 4), and whether $\delta < C$ (panels 1 and 3) or $\delta > C$ (panels 2 and 4), which can be further summarized below in terms of the lower bound L and upper bound H of α_2:

- If $s = 1$, then $\alpha_i + s\alpha_j = \alpha_i + \alpha_j = \delta$,
  - if $\delta < C$, then $\min(\alpha_j) = 0,\ \max(\alpha_j) = \delta$ (first panel)
  - if $\delta > C$, then $\min(\alpha_j) = \delta - C,\ \max(\alpha_j) = C$ (second panel)

  Combining these two cases, we have

$$\begin{cases} L = \max(0,\ \delta - C) = \max(0,\ \alpha_i + \alpha_j - C) \\ H = \min(\delta,\ C) = \min(\alpha_i + \alpha_j,\ C) \end{cases} \tag{153}$$

- If $s = -1$, then $\alpha_i + s\alpha_j = \alpha_i - \alpha_j = \delta$,
  - if $\delta > 0$, then $\min(\alpha_j) = 0,\ \max(\alpha_j) = C - \delta$ (third panel)
  - if $\delta < 0$, then $\min(\alpha_j) = -\delta,\ \max(\alpha_j) = C$ (fourth panel)

  Combining these two cases, we have

$$\begin{cases} L = \max(0,\ -\delta) = \max(0,\ \alpha_j - \alpha_i) \\ H = \min(C - \delta,\ C) = \min(C + \alpha_j - \alpha_i,\ C) \end{cases} \tag{154}$$

Now we apply the constraints in terms of L and H to the optimial solution α_j^{new} obtained previously to get α_j that is feasible as well as optmal:

$$\alpha_j^{new} \Longleftarrow \begin{cases} H & \text{if } \alpha_j^{new} \ge H \\ \alpha_j^{new} & \text{if } L < \alpha_2^{new} < H \\ L & \text{if } \alpha_j^{new} \le L \end{cases} \tag{155}$$

We can further find α_i based on Eq. (140):

$$\alpha_i^{new} = \alpha_i^{old} - s\Delta\alpha_j = \alpha_i^{old} - s(\alpha_j^{new} - \alpha_j^{old}) \tag{156}$$

and update the weight vector based on Eq. (119):

$$\begin{aligned} \mathbf{w}^{new} &= \sum_{n \ne i,j} y_n \alpha_n^{old} \mathbf{x}_n + y_i \alpha_i^{new} \mathbf{x}_i + y_j \alpha_j^{new} \mathbf{x}_j \\ &= \mathbf{w}^{old} - y_i \alpha_i^{old} \mathbf{x}_i - y_j \alpha_j^{old} \mathbf{x}_j + y_i \alpha_i^{new} \mathbf{x}_i + y_j \alpha_j^{new} \mathbf{x}_j \\ &= \mathbf{w}^{old} + y_i \Delta\alpha_i \mathbf{x}_i + y_j \Delta\alpha_j \mathbf{x}_j = \mathbf{w}^{old} + \Delta\mathbf{w} \end{aligned} \tag{157}$$

where $\Delta\mathbf{w} = \mathbf{w}^{new} - \mathbf{w}^{old} = y_i \Delta\alpha_i \mathbf{x}_i + y_j \Delta\alpha_j \mathbf{x}_j$. To update b, consider:

$$\begin{aligned} \Delta E_k &= E_k^{new} - E_k^{old} = u_k^{new} - u_k^{old} \\ &= \sum_{n=1}^{N} \alpha_n^{new} y_n K_{nk} + b^{new} - \sum_{n=1}^{N} \alpha_n^{old} y_n K_{nk} - b^{old} \\ &= y_i \Delta\alpha_i K_{ik} + y_j \Delta\alpha_j K_{jk} + b^{new} - b^{old} \qquad (k = 1, 2) \end{aligned} \tag{158}$$

Consider the following cases:

- If $0 < \alpha_k < C$ for either $k = i$ or $k = j$, then $y_k E_k = 0$ according to Eq. (131). We let $E_k^{new} = 0$ and solve the equation above to get:

$$b_k^{new} = b_k^{old} - (E_k^{old} + y_i \Delta\alpha_i K_{ik} + y_j \Delta\alpha_j K_{jk}) \qquad (k = 1 \text{ or } 2) \tag{159}$$

- If $0 < \alpha_k < C$ for both $k = i$ and $k = j$, then $b_i^{new} = b_j^{new}$.
- If $\alpha_k = 0$ or $\alpha_k = C$ for both $k = i$ and $k = 2$, then $E_k \neq 0$, and $b_i^{new} \neq b_j^{new}$, their average $b^{new} = (b_i^{new} + b_j^{new})/2$ can be used.

The computation above for maximizing $L(\alpha_i, \alpha_j)$ is carried out iteratively each time when a pair of variables α_i and α_j is selected to be updated. Specifically, we select a variable α_i that violates any of the KKT conditions given in Eq. (131) in the outer loop of the SMO algorithm, and then pick a second variable α_j in the inner loop of the algorithm, both to be optimized. The selection of these variables can be either random (e.g., by following their order), or guided by some heuristics, such as always choosing the variable with maximum step size $\Delta\alpha_n = \alpha_n^{new} - \alpha_n^{old}$. This iterative process is repeated until convergence when the KKT conditions are satisfied by all $\alpha_1, \cdots, \alpha_N$ variables, i.e., no more variables need to be updated. All data points corresponding to $\alpha_n \neq 0$ are the support vectors, based on which we can find the offset b by Eq. (134) and classify any unlabeled point $\mathbf{x}$ by Eq. (135).

The Matlab code for the essential part of the SMO algorithm is listed below, based mostly on a simplified SMO algorithm and the related code with some modifications.

In particular, the if-statement

```
if (alpha(i)<C & yE<-tol) | (alpha(i)>0 & yE>tol)
```

checks the three KKT conditions in Eq. (131), where $yE = y_n E_n$ and tol is a small constant. The first half checks if the first two of the three conditions are violated, while the second half checks if the second two of the three conditions are violated.

```
[X y]=getData;                      % get training data
[m n]=size(X);                      % size of data
alpha=zeros(1,n);                   % alpha variables
bias=0;                             % initial bias
it=0;                               % iteration index
while (it<maxit)                    % number of iterations less than maximum
    it=it+1;
    changed_alphas=0;               % number of changed alphas
    N=length(y);                    % number of support vectors
    for i=1:N                       % for each alpha_i
        Ei=sum(alpha.*y.*K(X,X(:,i)))+bias-y(i);
        yE=Ei*y(i);
        if (alpha(i)<C & yE<-tol) | (alpha(i)>0 & yE>tol)   % KKT violation
            for j=[1:i-1,i+1:N]        % for each alpha_j not equal alpha_i
                Ej=sum(alpha.*y.*K(X,X(:,j)))+bias-y(j);
                ai=alpha(i);        % alpha_i old
                aj=alpha(j);        % alpha_j old
                if y(i)==y(j)       % s=y_i y_j=1
                    L=max(0,alpha(i)+alpha(j)-C);
                    H=min(C,alpha(i)+alpha(j));
                else                % s=y_i y_j=-1
                    L=max(0,alpha(j)-alpha(i));
                    H=min(C,C+alpha(j)-alpha(i));
                end
                if L==H             % skip when L==H
                    continue
                end
                eta=2*K(X(:,j),X(:,i))-K(X(:,i),X(:,i))-K(X(:,j),X(:,j));
                alpha(j)=alpha(j)+y(j)*(Ej-Ei)/eta;   % update alpha_j
                if alpha(j) > H
                    alpha(j) = H;
                elseif alpha(j) < L
                    alpha(j) = L;
                end
                if norm(alpha(j)-aj) < tol       % skip if no change
                    continue
                end
                alpha(i)=alpha(i)-y(i)*y(j)*(alpha(j)-aj);   % find alpha_i
                bi = bias - Ei - y(i)*(alpha(i)-ai)*K(X(:,i),X(:,i))...
                    -y(j)*(alpha(j)-aj)*K(X(:,j),X(:,i));
                bj = bias - Ej - y(i)*(alpha(i)-ai)*K(X(:,i),X(:,j))...
                    -y(j)*(alpha(j)-aj)*K(X(:,j),X(:,j));
                if 0<alpha(i) & alpha(i)<C
                    bias=bi;
                elseif 0<alpha(j) & alpha(j)<C
                    bias=bj;
                else
                    bias=(bi+bj)/2;
                end
                changed_alphas=changed_alphas+1;  % one more alpha changed
            end
        end
    end
    if changed_alphas==0            % no more changed alpha, quit
        break
    end
    I=find(alpha~=0);               % indecies of non-zero alphas
    alpha=alpha(I);                 % find non-zero alphas
    Xsv=X(:,I);                     % find support vectors
    ysv=y(I);                       % their corresponding indecies
```

```
    end                                 % end of iteration
    nsv=length(ysv);                    % number of support vectors
```

where `K(X,x)` is a function that takes an $m \times n$ matrix $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$ and an n-D vector $\mathbf{x}$ and produces the kernel functions $K(\mathbf{x}_i, \mathbf{x}),\ (i = 1, \cdots, n)$ as output.

```
function ker=K(X,x)
    global kfunction
    global gm                               % parameter for Gaussian kernel
    [m n]=size(X);
    ker=zeros(1,n);
    if kfunction=='l'
        for i=1:n
            ker(i)=X(:,i).'*x;              % linear kernel
        end
    elseif kfunction=='g'
        for i=1:n
            ker(i)=exp(-gm*norm(X(:,i)-x)); % Gaussian kernel
        end
    elseif kfunction=='p'
        for i=1:n
            ker(i)=(X(:,i).'*x).^3;         % polynomial kernel
        end
    end
end
```

Having found all non-zero $\alpha_n \neq 0$ and the corresponding support vectors, we further find the offset or bias term:

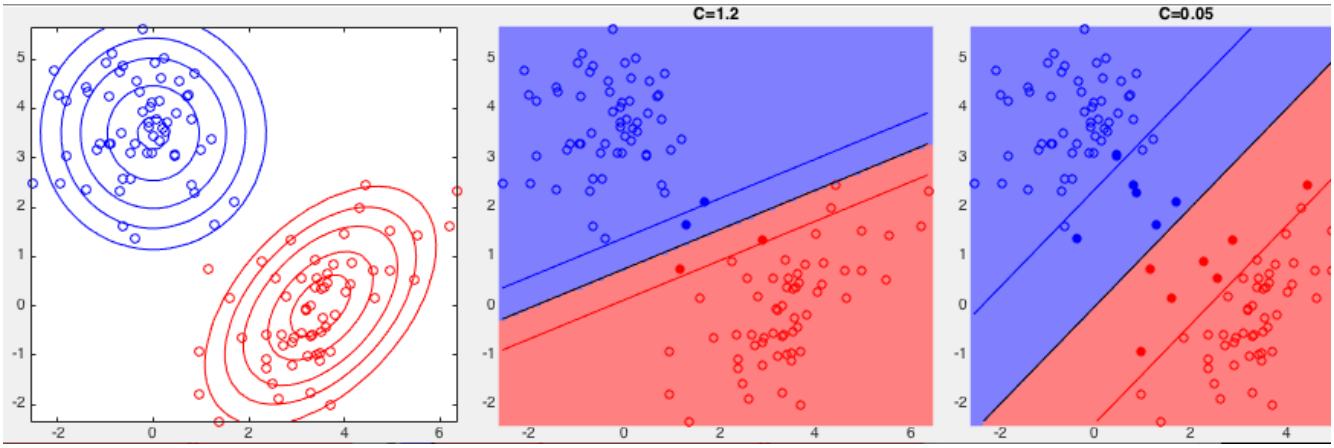
```
     bias=0;
     for i=1:nsv
        bias=bias+(ysv(i)-sum(ysv.*alpha.*K(Xsv,Xsv(:,i))));
    end
    bias=bias/nsv;
```

Any unlabeled data point $\mathbf{x}$ is classified into class C_+ or C_-, depending on whether the following expression is greater or smaller than zero:

```
    y=sum(alpha.*ysv.*K(Xsv,x))+bias;
```

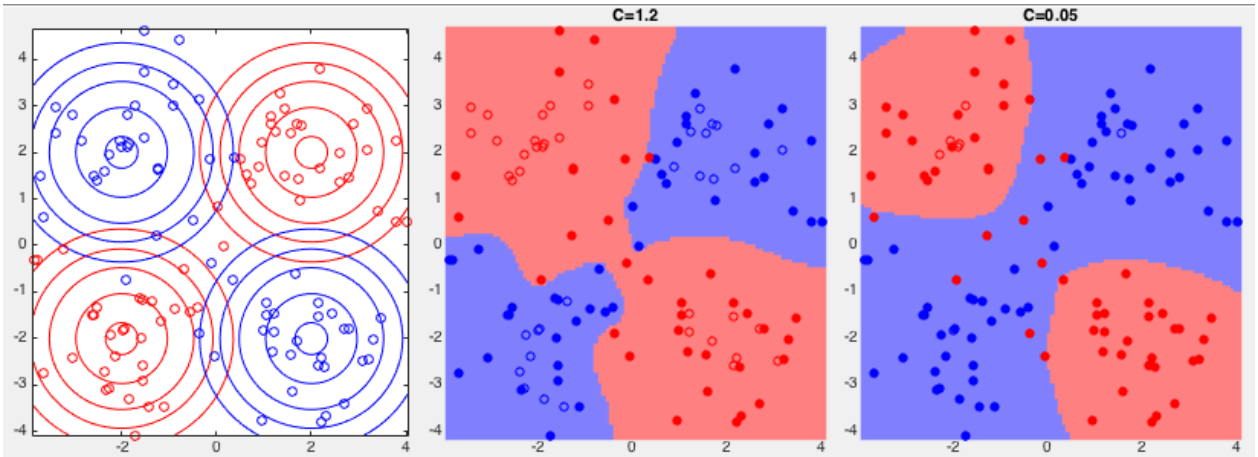
Example 1: The same training set used previously to test the SVM with a hard margin is used again to test the SVM with a soft margin. Two results are shown below, corresponding to two different values $C = 1.2$ and $C = 0.05$ for the weight of the error term $\sum_{n=1}^{N} \xi_n$ in the objective function. We see that when $C = 1.2$ is large, the number of support vectors (the four solid dots) is small due to the small errors ξ_n allowed, and the decision boundary determined by the these support vectors independent of the rest of the data set (circles) is mostly dictated by the local points close to the boundary, not necessarily a good reflection of the global distribution of the data points. This result is similar to the previous one generated by the hard-margin SVM.

On the other hand, when $C = 0.1$ is small, a greater number of data points become support vectors due to the larger (the 13 solid dots) error ξ_n allowed, and the resulting linear decision line determined by these support vectors reflects global distribution of the data points, and it better separates the two classes in terms of their mean positions.

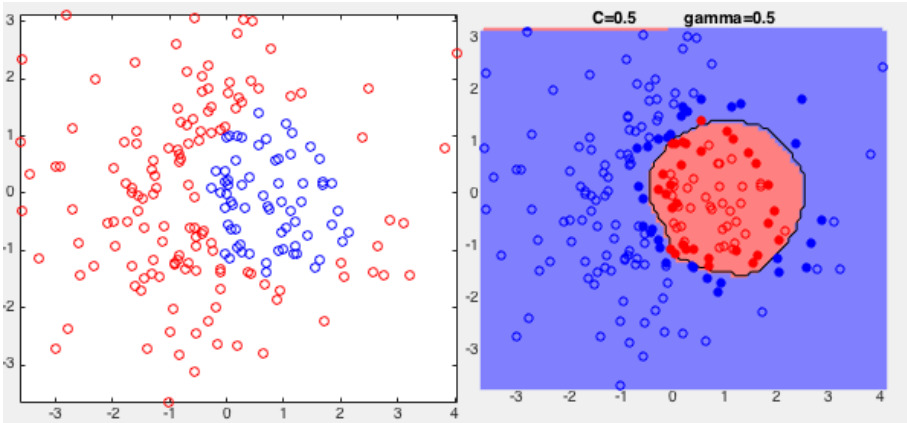


Example 2: The classification results for the XOR data set are shown below with $C = 1.2$ and $C = 0.05$. As the two classes in the XOR data set are not linearly separable, a Gaussian or radial basis function (RBF) kernel is used to map the data from 2-D space to an infinite dimensional space, which is partitioned into two regions for the two classes. When $C = 1.2$ is large, the decision boundary is dictated by a relatively small number of support vectors (solid dots, with small error ξ_n) and all data points are classified correctly, but it is more prone to the overfitting problem, if some of the small number of support vectors are outliers due to noise.

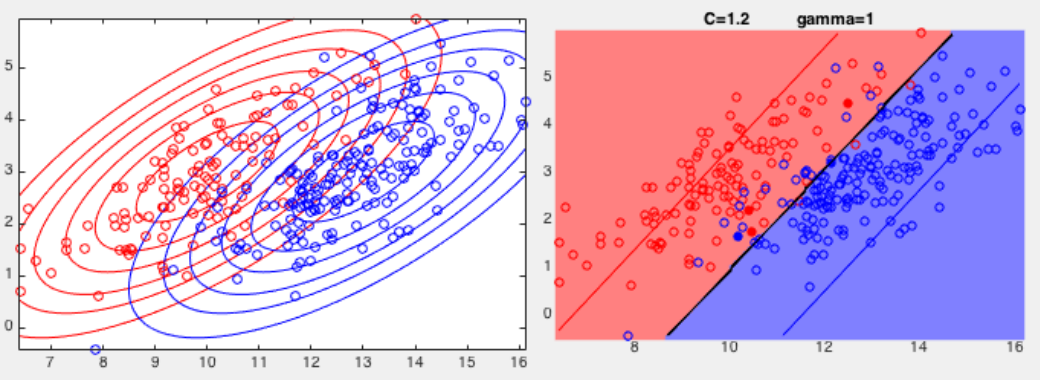
One the other hand, when $C = 0.05$ is small, the decision boundary is determined by a greater number of support vectors, better reflecting the global distribution of the data points. However, as the local points close to the boundary are not relatively deemphasized, some miss-classification occurs, some nine red dots are classified into the blue region incorrectly.



Example 3:



Example 4:



Next Up Previous
Next: Multi-Class Classification **Up:** Support Vector machine **Previous:** Soft Margin SVM