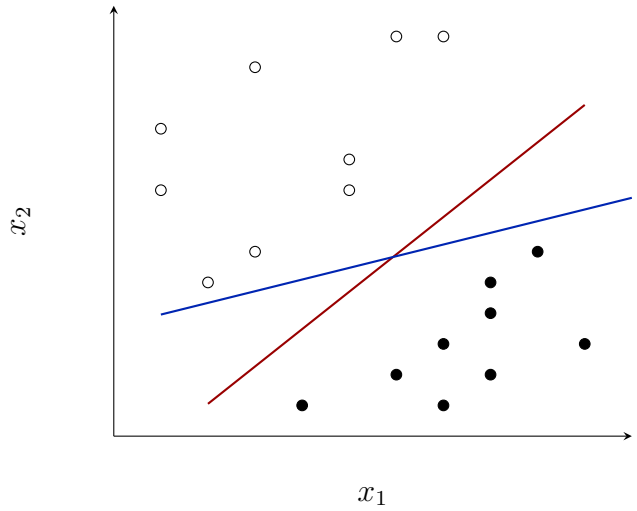
Lecture 6

1. Support vector machines: intuition
2. Primal/dual optimization problem (KKT)
3. SVM dual
4. Kernels
5. Soft margin
6. SMO algorithm

1 Support vector machines: intuition

In this section we develop another nonlinear algorithm. There are two intuitions behind it:

- Logistic regression computes $\theta^T x$ and predict 1 if $\theta^T x > 0$, 0 - otherwise. If $\theta^T x >> 0$, then the algorithm is very "confident" that $y = 1$. If $\theta^T x << 0$, the algorithm is very "confident" that $y = 0$. Our aim is to obtain the algorithm that gives $\theta^T x^{(i)} >> 0$ for any i such that $y^{(i)} = 1$, and $\theta^T x^{(i)} << 0$ for any i such that $y^{(i)} = 0$.
- If we assume that classes are linearly separable, then we prefer the red line as our decision boundary (the intuition is that blue line is not good decision boundary):



For this algorithm we should use slightly different notations. First of all, we assume that $y \in \{-1, 1\}$ which means that output values could be 1 or -1. Second of all, we are using the hypothesis

$$h_{w,b}(x) = \text{sign}(w^T x + b),$$

where

$$\text{sign}(z) = \begin{cases} 1 & \text{if } z \geqslant 0, \\ -1 & \text{otherwise} \end{cases}$$

We removed the convention $x_0 = 1$ and θ_0 is replaced by b, θ is replaced by vector w.

Definition. Functional margin of a hyperplane $w^T x + b = 0$ with respect to $(x^{(i)}, y^{(i)})$ is:

$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b).$$

Notice that if $y^{(i)} = 1$ then the algorithm should give $w^T x^{(i)} + b >> 0$, and if $y^{(i)} = -1$ the algorithm should give $w^T x^{(i)} + b << 0$. In both cases $\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b) > 0$. Our aim is to build the algorithm that gives the functional margin positive for all training examples.

Definition. Minimal functional margin is

$$\hat{\gamma} = \min_i \hat{\gamma}^{(i)}.$$

Based on our intuition we should claim from the algorithm that the worst functional margin should be large.

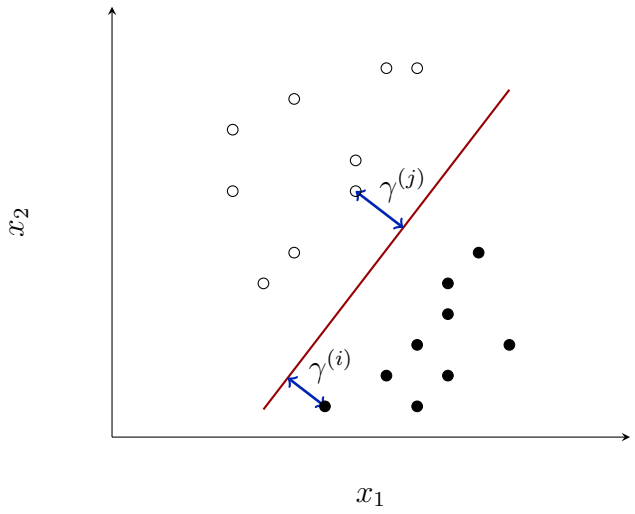
Definition. A **geometric margin** $\gamma^{(i)}$ is the distance between training example $x^{(i)}$ to the decision boundary $w^T x + b = 0$:

$$\gamma^{(i)} = y^{(i)} \left[\left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||} \right].$$

Notice that the formula for the distance between point and hyperplane from Calculus is:

$$d = \left(\frac{w}{||w||} \right)^T x^{(i)} + \frac{b}{||w||},$$

and d has different signs for points on different sides of the plane. We multiply this distance by $y^{(i)}$ to have positive value for all training examples (remember that $y^{(i)} \in \{-1, 1\}$).



Definition. Minimal geometric margin is

$$\gamma = \min_i \gamma^{(i)}.$$

It is easy to see that functional and geometric margins are connected by $\gamma^{(i)} = \dfrac{\hat{\gamma}^{(i)}}{||w||}$. If $||w|| = 1$, then $\hat{\gamma}^{(i)} = \gamma^{(i)}$. The disadvantage of the functional margin is that it is not normalized: for example, if we double w and b, then we double functional margin, but the hyperplane $w^T x + b = 0$ does not change. To simplify our following calculations we can assume that $||w|| = 1$ or $|w_1| = 1$.

We can formulate two equivalent optimization problems (**optimal margin classifier**):

1. $\max\limits_{\gamma,w,b} \gamma$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant \gamma,\ i = 1 \dots m,$$
$$||w|| = 1.$$

2. $\max\limits_{\hat{\gamma},w,b} \dfrac{\hat{\gamma}}{||w||}$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant \hat{\gamma},\ i = 1 \dots m.$$

Notice that for the first formulation the functional and geometric margins are the same. Also this is an example of non-convex optimization.

Consider the second optimization problem, as we mentioned before functional margin can be increasing, but the decision boundary stays the same. To avoid this situation we impose additional constraint on $\hat{\gamma}$:

$$\hat{\gamma} = 1 \text{ (scaling constraint)}.$$

With this constraint the second optimization problem transforms to the **Support Vector Machine (SVM) classifier** problem

$$\min_{w,b} \frac{||w||^2}{2} \text{ (the same as } \max_{w,b} \frac{1}{||w||})$$
$$\text{subject to } y^{(i)}(w^T x^{(i)} + b) \geqslant 1,\ i = 1 \dots m.$$

This is the quadratic programming problem, because our objective function is a quadratic function and all our constraints are linear. Recall that after we solve this optimization problem the prediction for the test point x is calculated as

$$\hat{y} = \text{sign}(w^T x + b).$$

Example. For the following dataset

x_1	x_2	y
-1	-2	1
1	-1	1
0	2	-1
1	3	-1

the SVM classifier optimization problem is formulated as:

$$\min_{w,b} \frac{1}{2}(w_1^2 + w_2^2)$$

subject to

$$\begin{aligned} &-w_1 - 2w_2 + b \geqslant 1, \\ &w_1 - w_2 + b \geqslant 1, \\ &-2w_2 - b \geqslant 1, \\ &-w_1 - 3w_2 - b \geqslant 1. \end{aligned}$$

2 Primal/dual optimization problem

To introduce the Support Vector Machine algorithm for the problems with nonlinearly separable classes we should recall the notion of primal and dual optimization problems. The method of Lagrange multipliers in the Multidimensional Calculus helps to solve the problem of optimization with additional constraints:

$$\min_w f(w)$$

subject to $h_i(w) = 0, i = 1 \dots l$, or $h(w) = \begin{bmatrix} h_1(w) \\ h_2(w) \\ \vdots \\ h_l(w) \end{bmatrix} = \vec{0}$.

The Lagrangian is defined as

$$L(w, \beta) = f(w) + \sum_i \beta_i h_i(w),$$

where β are Lagrange multipliers. The solution of the original optimization problem can be found by solving the system of equations

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

with respect to w and β.

The primal problem by tradition is formulated in more general form:

$$\min_w f(w)$$

subject to

$$\begin{aligned} &g_i \leqslant 0, \ i = 1, \dots, k, \\ &h_i(w) = 0, \ i = 1, \dots, l, \end{aligned}$$

or with vector notations:

$$\begin{aligned} &g(w) \leqslant \vec{0}, \\ &h(w) = \vec{0} \end{aligned}$$

For this problem the Lagrangian is defined by

$$L(w,\alpha,\beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

and by definition

$$\theta_P(w) = \max_{\alpha,\beta,\alpha_i \geqslant 0} L(w,\alpha,\beta) = \begin{cases} f(w), \text{ if conditions for } g,h \text{ satisfies} \\ \infty, \text{ otherwise.} \end{cases}$$

Notice that if $g_i(w) > 0$, then $\theta_P(w) = \infty$; if $h_i(w) \neq 0$, then $\theta_P(w) = \infty$; otherwise, $\theta_P(w) = f(w)$.

With this definition the original problem is transformed to the **primal problem**:

$$p^* = \min_w \theta_P(w) = \min_w \max_{\alpha,\beta,\alpha_i \geqslant 0} L(w,\alpha,\beta).$$

The natural way to modify this problem is to switch max and min and formulate the **dual problem**:

$$d^* = \max_{\alpha \geqslant 0,\beta} \theta_D(\alpha,\beta) = \max_{\alpha \geqslant 0,\beta} \min_w L(w,\alpha,\beta),$$

where by definition

$$\theta_D(\alpha,\beta) = \min_w L(w,\alpha,\beta).$$

It is easy to show that $d^* < p^*$, because $\max\min \leqslant \min\max$.

Example.

$$\max_{y\in\{0,1\}} \min_{x\in\{0,1\}} \mathbb{1}\{x=y\} \leqslant \min_{x\in\{0,1\}} \max_{y\in\{0,1\}} \mathbb{1}\{x=y\}$$

Notice that $\min\limits_{x\in\{0,1\}} \mathbb{1}\{x=y\} = 0$ and $\max\limits_{y\in\{0,1\}} \mathbb{1}\{x=y\} = 1$.

The important theorem from optimization theory tells that under certain conditions: $d^* = p^*$ and we can solve dual problem instead of primal problem.

Theorem. Let

1) f is convex (if function f convex and hessian H exists then $H \geqslant 0$);
2) h_i is affine ($h_i(w) = a_i^T w + b_i$);
3) constraints g_i are strictly feasible (there exist w such that for any i $g_i(w) < 0$).

Then

1) there exists w^*, α^* and β^* such that w^* solves primal problem and α^*, β^* solve the dual problem and $p^* = d^* = L(w^*,\alpha^*,\beta^*)$;
2) $\dfrac{\partial L}{\partial w}(w^*,\alpha^*,\beta^*) = 0$, $\dfrac{\partial L}{\partial \beta}(w^*,\alpha^*,\beta^*) = 0$;
3) $\alpha_i^* g_i(w^*) = 0$ (Karush-Kuhn-Tucker (KKT) complementarity condition).

Moreover, by definition $\alpha_i^* \geqslant 0$ and from the initial conditions $g_i(w^*) \leqslant 0$, which means that if $\alpha_i^* > 0$, then KKT condition implies $g_i(w^*) = 0$. In most cases,

$$\alpha_i^* > 0 \Leftrightarrow g_i(w^*) = 0$$

($g_i(w)$ is an **active constraint**).

3 SVM dual problem

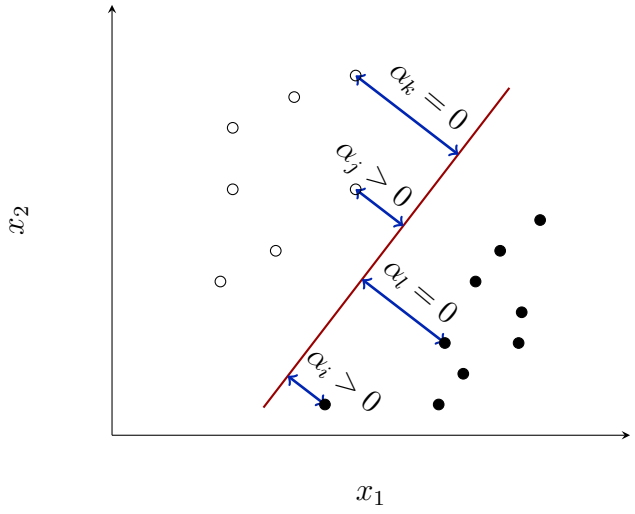
In this section we apply the idea of Lagrange multipliers to the SVM optimization problem and formulate a SVM dual problem. The SVM optimization problem has been formulated in the previous lecture as

$$\min_{w,b} \frac{||w||^2}{2},$$

subject to

$$y^{(i)}(w^T x^{(i)} + b) \geqslant 1, \ i = 1, \dots, m.$$

We define $g_i(w, b) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leqslant 0$. Notice that we do not have coefficients β as there are no constraints for h. If $\alpha_i > 0$, then $g_i(w, b) = 0$ (active constraint) and implies that the training example $(x^{(i)}, y^{(i)})$ has a functional margin equals to 1.



The Lagrangian has the form

$$L(w, b, \alpha) = \frac{||w||^2}{2} - \sum_{i=1}^{m} \alpha_i (y^{(i)}(w^T x^{(i)} + b) - 1)$$

and dual problem is

$$\theta_D(\alpha) = \min_{w,b} L(w, b, \alpha).$$

In order to minimize the Lagrangian we find derivatives and set them to zero:

$$\nabla_w L(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \Rightarrow w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \tag{1}$$

and

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{m} y^{(i)} \alpha_i = 0.$$

Substitute these conditions back to the Lagrangian:

$$\begin{aligned}
&L(w,b,\alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{m}\alpha_i(y^{(i)}(w^T x^{(i)}+b)-1) = \\
&= \frac{1}{2}\left(\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\right)^T\left(\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\right) - \sum_{i=1}^{m}\alpha_i(y^{(i)}(w^T x^{(i)}+b)-1) = \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle - \sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle + \sum_{i=1}^{m}\alpha_i = \\
&= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle = W(\alpha),
\end{aligned}$$

where $\langle\cdot,\cdot\rangle$ is a notation for the dot product of two vectors.

Finally, the **SVM dual problem** is to find

$$\max_{\alpha} W(\alpha) = \max_{\alpha}\left(\sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle\right)$$

subject to

$$\begin{aligned}
&\alpha_i \geqslant 0,\\
&\sum_i y^{(i)}\alpha_i = 0.
\end{aligned}$$

Notice that if $\sum_i y^{(i)}\alpha_i \neq 0$, then $\theta_D(\alpha) = -\infty$, otherwise, $\theta_D(\alpha) = W(\alpha)$.

Example. The SVM dual problem for the dataset (see the previous sections):

x_1	x_2	y
-1	-2	1
1	-1	1
0	2	-1
1	3	-1

is formulated as:

$$\begin{aligned}
&\max_{\alpha}(\alpha_1+\alpha_2+\alpha_3+\alpha_4 - \frac{5}{2}\alpha_1^2 - \alpha_2^2 - 2\alpha_3^2 - 5\alpha_4^2\\
&-\alpha_1\alpha_2 - 4\alpha_1\alpha_3 - 7\alpha_1\alpha_4 - 2\alpha_2\alpha_3 - 3\alpha_2\alpha_4 - 6\alpha_3\alpha_4)
\end{aligned}$$

subject to

$$\begin{aligned}
&\alpha_i \geqslant 0,\\
&\alpha_1+\alpha_2-\alpha_3-\alpha_4 = 0.
\end{aligned}$$

After we find the solution α^*, the coefficients can be found as

$$w = \sum_{i=1}^{m}\alpha_i^* y^{(i)}x^{(i)} \tag{2}$$

and we use the worst positive and negative training examples to find b:

$$b = \frac{\max\limits_{i:y^{(i)}=-1} w^T x^{(i)} + \min\limits_{i:y^{(i)}=1} w^T x^{(i)}}{2}.$$

With the equation (1) the hypothesis for the new test point x is expressed in terms of dot products:

$$h_{w,b} = \text{sign}(w^T x + b) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} \left\langle x^{(i)}, x \right\rangle + b\right) \tag{3}$$

4 Kernels

The idea of kernels is that often features are high dimensional ($x^{(i)} \in \mathbb{R}^m$) but instead of feature representations it is enough to find dot products.

Example. Assuming we have the problem with one feature $x \in \mathbb{R}$ only, the polynomial regression of the fourth order can be represented as a linear regression with the following list of features:

$$\varphi(x) : x \to \begin{bmatrix} x \\ x^2 \\ x^3 \\ x^4 \end{bmatrix}$$

For the SVM optimization problem the hypothesis (3) is replaced by

$$h_{w,b} = \text{sign}(w^T \varphi(x) + b) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} \left\langle \varphi(x)^{(i)}, \varphi(x) \right\rangle + b\right),$$

and in all following calculations we should replace the dot product $\left\langle x^{(i)}, x^{(j)} \right\rangle$ by $\left\langle \varphi(x^{(i)}), \varphi(x^{(j)}) \right\rangle$.

There are no any restrictions for the mapping $\varphi(x)$, in fact it is possible to have infinite dimensional $\varphi(x) \in \mathbb{R}^\infty$. Fortunately, for many different φ we can specify the function (**kernel**) that defines the dot product:

$$K(x^{(i)}, x^{(j)}) = \left\langle \varphi(x^{(i)}), \varphi(x^{(j)}) \right\rangle.$$

In such situations we do not need to compute $\varphi(x)$ explicitly, but we should compute the kernel $K(x, z)$ (which is less computationally expensive than computing $\varphi(x)$).

4.1 Kernel examples

1. $K(x, z) = (x^T z)^2$, where $x, z \in \mathbb{R}^n$. We try to transform this kernel to the exact form of dot product:

$$K(x, z) = (x^T z)^2 = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} (x_i x_j)(z_i z_j),$$

that can be interpreted as a dot product of vectors that contains all possible combinations of x and z components. For example, if $n = 3$, then

$$\varphi(x) : x \to \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

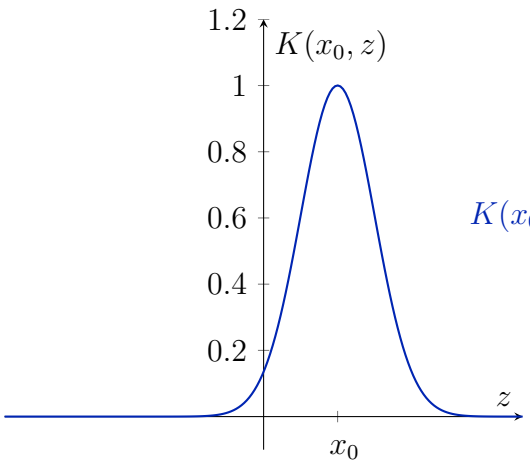
To compute the dot product $\langle \varphi(x), \varphi(z) \rangle$ for two training examples we need $O(2n^2+n)$ operations. If we use kernel for that we need $O(n)$ operations only (because we just calculate dot product of two vectors $x^T z$ and take square of it).

2. $K(x, z) = (x^T z + c)^2$ corresponds to

$$\varphi(x) : x \to \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c} \cdot x_1 \\ \sqrt{2c} \cdot x_2 \\ \sqrt{2c} \cdot x_3 \\ c \end{bmatrix}$$

3. $K(x, z) = (x^T z + c)^d$ corresponds to $\begin{pmatrix} n+d \\ d \end{pmatrix}$ features of all monomials up to degree d.

4. $K(x, z) = \exp\left(-\dfrac{||x - z||^2}{2\sigma^2}\right)$ (**radial basis function (RBF) kernel**) corresponds to the transformation of feature space into an infinite dimensional Hilbert space. The intuition of this kernel is that if x and z are very similar than they will be pointing to the same direction and dot product should be large. In contrast if x and z are very different, then the dot product should be very small. If we fix x and consider $K(x, z)$ as a function of z the graph of this function is a bell shaped function:



4.2 Kernel testing

Assuming that we have chosen some function $K(x, z)$ as a kernel. The main question is: does there exist some $\varphi(x)$ such that $K(x, z) = \langle \varphi(x), \varphi(z) \rangle$?

Definition. For the given set of points $\{x^{(i)}, \dots, x^{(m)}\}$ a **kernel matrix** $\mathbf{K} \in \mathbb{R}^{m \times m}$ is defined by

$$\mathbf{K}_{ij} = K(x^{(i)}, x^{(j)}), \tag{4}$$

where K is a kernel function.

Theorem (Mercer). Let $K(x, z)$ be given. Then K is a valid (Mercer) kernel (i.e. there exists φ such that $K(x, z) = \langle \varphi(x), \varphi(z) \rangle$) if and only if for all $\{x^{(i)}, \dots, x^{(m)}\}$ the kernel matrix $\mathbf{K} \in \mathbb{R}^{m \times m}$ is symmetric positive semi-definite.

Indeed, for any vectors $x, z \in \mathbb{R}^n$

$$\begin{aligned} z^T \mathbf{K} z &= \sum_i \sum_j z_i \mathbf{K}_{ij} z_j = \sum_i \sum_j z_i \varphi(x^{(i)})^T \varphi(x^{(j)}) z_j = \\ &= \sum_i \sum_j z_i \sum_k (\varphi(x^{(i)}))_k (\varphi(x^{(j)}))_k \, z_j = \sum_k \sum_i \sum_j z_i \, (\varphi(x^{(i)}))_k (\varphi(x^{(j)}))_k \, z_j = \\ &= \sum_k \left(\sum_i z_i \varphi(x^{(i)})_k \right)^2 \geqslant 0. \end{aligned}$$

Here we used a fact that $a^T b = \sum_k a_k b_k$.

Example. $K(x, z) = -1$ is not a valid kernel function.

4.3 SVM with kernels

We can reformulate the SVM dual problem from the previous section as follows:

$$\max_{\alpha} W(\alpha) = \max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K\left(x^{(i)}, x^{(j)}\right) \right)$$

subject to

$$\alpha_i \geqslant 0,$$
$$\sum_i y_i \alpha_i = 0,$$

with the prediction for the new test point x

$$h_{w,b} = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b \right), \tag{5}$$

where K is a chosen kernel function.

The last remark is that the kernel idea is more general than SVM and we can formulate many algorithms in terms of dot products.

5 Soft margin

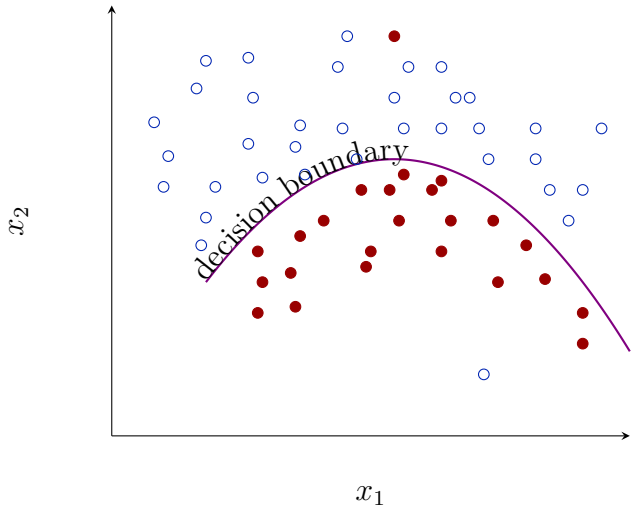
In case of non linear decision boundaries the SVM algorithm is called L_1 **norm soft margin SVM** and formulated as follows:

$$\min_{w} \frac{||w||^2}{2} + C \sum_{i=1}^{m} \xi_i$$

subject to

$$y^{(i)}(w^T x^{(i)} + b) \geqslant 1 - \xi_i,\ \xi_i \geqslant 0,\ i = 1, \ldots, m.$$

Such formulation is useful for non linear separable datasets, for example, in the next picture we cannot find the hyperplane that separates two classes.



Remember that if $y^{(i)}(w^T x^{(i)} + b) > 0$, then the example is classified correctly. With the above formulation we allow the algorithm to misclassify something (because of the term

$1-\xi_i$), but we encourage the algorithm not to do it, because it will increase the objective function by $\sum_{i=1}^{m}\xi_i$. Notice that this is also convex optimization problem.

As before we find the derivatives of Lagrangian

$$L(w,b,\xi,\alpha,r)=\frac{1}{2}||w||^2+C\sum_i\xi_i-\sum_{i=1}^{m}\alpha_i(y^{(i)}(w^Tx^{(i)}+b)-1+\xi_i)-\sum_{i=1}^{m}r_i\xi_i$$

and equate them to zero:

$$\begin{aligned}
&\nabla_w L(w,b,\xi,\alpha,r)=w-\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}=0\Rightarrow w=\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)},\\
&\frac{\partial L}{\partial b}=-\sum_{i=1}^{m}\alpha_i y^{(i)}=0,\\
&\frac{\partial L}{\partial \xi_i}=C-\alpha_i-r_i=0.
\end{aligned}$$

We can also add the KKT conditions:

$$\begin{aligned}
&\alpha_i(y^{(i)}(w^Tx^{(i)}+b)-1+\xi_i)=0,\\
&r_i\xi_i=0.
\end{aligned}$$

Taking into consideration all these conditions we derive

$$\alpha_i=0\Rightarrow r_i=C>0\Rightarrow\xi_i=0\Rightarrow y^{(i)}(w^Tx^{(i)}+b)\geqslant 1 \tag{6}$$

$$\alpha_i=C\Rightarrow y^{(i)}(w^Tx^{(i)}+b)=1-\xi_i\Rightarrow y^{(i)}(w^Tx^{(i)}+b)\leqslant 1 \tag{7}$$

$$0<\alpha_i<C\Rightarrow r_i>0\Rightarrow\xi_i=0\Rightarrow y^{(i)}(w^Tx^{(i)}+b)=1 \tag{8}$$

To obtain the dual problem we substitute all these conditions to the Lagrangian:

$$\begin{aligned}
&L(w,b,\xi,\alpha,r)=\frac{1}{2}w^Tw+C\sum_i\xi_i-\sum_{i=1}^{m}\alpha_i(y^{(i)}(w^Tx^{(i)}+b)-1+\xi_i)-\sum_{i=1}^{m}r_i\xi_i=\\
&=\frac{1}{2}\left(\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\right)^T\left(\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\right)-\sum_{i=1}^{m}\alpha_i(y^{(i)}(w^Tx^{(i)}+b)-1)-\sum_i(C-\alpha_i-r_i)\xi_i=\\
&=\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle-\sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle+\sum_{i=1}^{m}\alpha_i=\\
&=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)},x^{(j)}\right\rangle=W(\alpha).
\end{aligned}$$

Finally, the dual optimization problem with the kernel idea is stated as

$$\max_\alpha W(\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_jK(x^{(i)},x^{(j)}) \tag{9}$$

subject to

$$\begin{aligned}
&\sum_{i=1}^{m}y^{(i)}\alpha_i=0,\\
&0\leqslant\alpha_i\leqslant C,\ i=1,\dots,m.
\end{aligned} \tag{10}$$

6 SMO algorithm

In this section we come up with an efficient algorithm that solves the SVM optimization problem. First, consider the problem

$$\max_{\alpha} W(\alpha_1, \ldots, \alpha_m)$$

without constraint on α's.

Algorithm 1 Coordinate ascent algorithm

1: **repeat**
2: **for** $i = 1$ **to** m **do**
3: $\alpha_i = \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$ (freeze all variables except α_i)
4: **until** convergence

Compared to the gradient descent this algorithm takes much more steps, but for many optimization problems it is very easy to make step by one parameter.

We apply it for our SVM dual optimization problem. Unfortunately, this algorithm does not work in a straight way, because of the condition $\sum_{i=1}^{m} y^{(i)}\alpha_i = 0$ (if we fix all α's except one, then we can find the last α explicitly). That's why we try to optimize two α's per step.

Algorithm 2 Sequential Minimal Optimization (SMO) algorithm

1: define the way how to choose pairs of α's for the following loop
2: **repeat**
3: **for** pairs α_i and α_j **do**
4: $\alpha_i, \alpha_j = \arg\max_{\hat{\alpha}_i, \hat{\alpha}_j} W(\alpha_1, \ldots, \hat{\alpha}_i, \ldots, \hat{\alpha}_j, \ldots, \alpha_m)$ (freeze all variables except α_i and α_j)
5: **until** convergence

We elaborate more about the implementation of this algorithm. Without loss of generality we update α_1 and α_2. The general case could be obtained in the same manner by replacing α_1 by α_i and α_2 by α_j. We assume that we have α_i^{old} from the previous step of the SMO algorithm, for which conditions (10) hold:

$$\sum_{i=1}^{m} y^{(i)}\alpha_i^{old} = 0,$$
$$0 \leqslant \alpha_i^{old} \leqslant C.$$

The first equation can be transformed to

$$\alpha_1^{old}y^{(1)} + \alpha_2^{old}y^{(2)} = -\sum_{i=3}^{m} \alpha_i^{old}y^{(i)} \Rightarrow \alpha_1^{old} + \alpha_2^{old}y^{(2)}y^{(1)} = -y^{(1)}\sum_{i=3}^{m} \alpha_i^{old}y^{(i)},$$

and the right-hand side of the last equation will be denoted by ζ, then

$$\alpha_1^{old} + s\alpha_2^{old} = \alpha_1 + s\alpha_2 = \zeta,$$
$$\text{where } s = y^{(1)}y^{(2)}.$$

Notice that ζ does not change after one step of the SMO algorithm.

We introduce the following notations (using (4)):

$$h(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b,$$
$$v_j = \sum_{i=3}^{m} y^{(i)}\alpha_i \mathbf{K}_{ij} = h(x^{(j)}) - b - \alpha_1 y^{(1)}\mathbf{K}_{1j} - \alpha_2 y^{(2)}\mathbf{K}_{2j}.$$

Then

$$W(\alpha_1, \alpha_2, \ldots, \ldots) = \alpha_1 + \alpha_2 - \frac{1}{2}(\alpha_1)^2\mathbf{K}_{11} - \frac{1}{2}(\alpha_2)^2\mathbf{K}_{22} - y^{(1)}y^{(2)}\alpha_1\alpha_2\mathbf{K}_{12}$$
$$-\alpha_1 y^{(1)}\sum_{i=3}^{m} y^{(i)}\alpha_i\mathbf{K}_{i1} - \alpha_2 y^{(2)}\sum_{i=3}^{m} y^{(i)}\alpha_i\mathbf{K}_{i2} + V(\alpha_3, \ldots, \alpha_m) =$$
$$= \alpha_1 + \alpha_2 - s\alpha_1\alpha_2\mathbf{K}_{12} - \frac{1}{2}(\alpha_1)^2\mathbf{K}_{11} - \frac{1}{2}(\alpha_2)^2\mathbf{K}_{22}$$
$$-\alpha_1 y^{(1)}v_1 - \alpha_2 y^{(2)}v_2 + V(\alpha_3, \ldots, \alpha_m) =$$

substitute the expression $\alpha_1 = \zeta - s\alpha_2$:

$$= \zeta - s\alpha_2 + \alpha_2 - s(\zeta - s\alpha_2)\alpha_2\mathbf{K}_{12} - \frac{1}{2}(\zeta - s\alpha_2)^2\mathbf{K}_{11} - \frac{1}{2}(\alpha_2)^2\mathbf{K}_{22}$$
$$-(\zeta - s\alpha_2)y^{(1)}v_1 - \alpha_2 y^{(2)}v_2 + V(\alpha_3, \ldots, \alpha_m).$$

We have obtained the quadratic function with respect to α_2. To find the maximum we find the derivative and equate it to zero:

$$W'_{\alpha_2} = -s + 1 - s\zeta\mathbf{K}_{12} + 2\alpha_2\mathbf{K}_{12}$$
$$+\zeta s\mathbf{K}_{11} - \alpha_2\mathbf{K}_{11} - \alpha_2\mathbf{K}_{22} + y^{(2)}v_1 - y^{(2)}v_2 = 0.$$

Then

$$2\alpha_2\mathbf{K}_{12} - \alpha_2\mathbf{K}_{11} - \alpha_2\mathbf{K}_{22} = s - 1 + s\zeta\mathbf{K}_{12} - \zeta s\mathbf{K}_{11} - y^{(2)}v_1 + y^{(2)}v_2.$$

Substitute the expressions for v_1, v_2 and $\zeta = \alpha_1^{old} + s\alpha_2^{old}$:

$$\alpha_2(2\mathbf{K}_{12} - \mathbf{K}_{11} - \mathbf{K}_{22}) = s - 1 + s(\alpha_1^{old} + s\alpha_2^{old})(\mathbf{K}_{12} - \mathbf{K}_{11})$$
$$-y^{(2)}(h(x^{(1)}) - b - \alpha_1^{old}y^{(1)}\mathbf{K}_{11} - \alpha_2^{old}y^{(2)}\mathbf{K}_{12})$$
$$+y^{(2)}(h(x^{(2)}) - b - \alpha_1^{old}y^{(1)}\mathbf{K}_{12} - \alpha_2^{old}y^{(2)}\mathbf{K}_{22}) =$$
$$= s - 1 + \alpha_2^{old}(2\mathbf{K}_{12} - \mathbf{K}_{11} - \mathbf{K}_{22}) - y^{(2)}h(x^{(1)}) + y^{(2)}h(x^{(2)})$$

and finally, using $s - 1 = y^{(2)}(y^{(1)} - y^{(2)})$:

$$\alpha_2 = \alpha_2^{old} - y^{(2)}\frac{(h(x^{(1)}) - y^{(1)}) - (h(x^{(2)}) - y^{(2)})}{2\mathbf{K}_{12} - \mathbf{K}_{11} - \mathbf{K}_{22}} \tag{11}$$

and

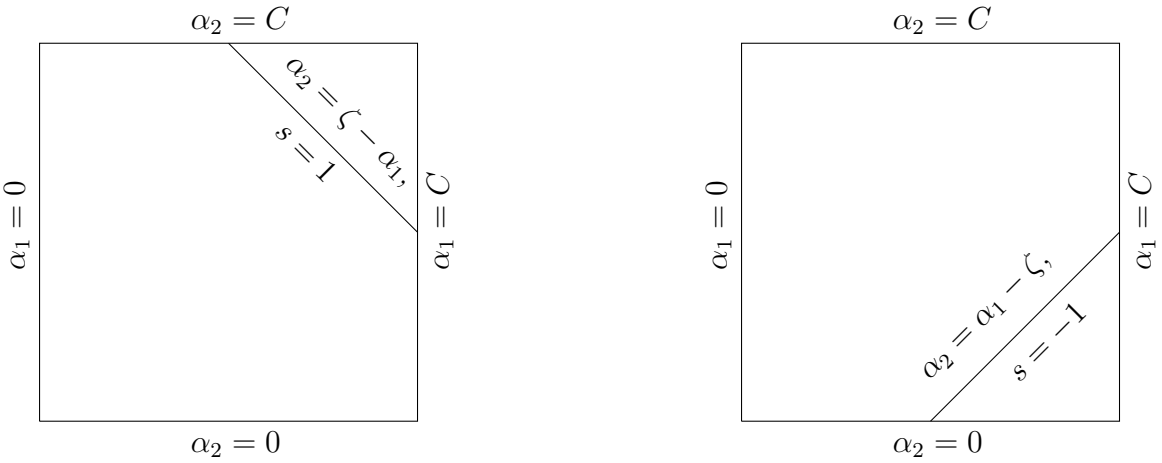
$$\alpha_1 = \zeta - s\alpha_2 = \alpha_1^{old} + s(\alpha_2^{old} - \alpha_2) \tag{12}$$

Notice that h in the formula (11) is calculated for old values of α's.

We have not used the conditions $0 \leqslant \alpha_i \leqslant C$ yet. Remember that ζ is a constant during each step of the SMO algorithm, which means that we can consider the equation

$$\alpha_1 + s\alpha_2 = \zeta \Rightarrow \alpha_2 = s\zeta - s\alpha_1$$

as an equation of the straight line.



There are two possible cases (see the figures):

- If $s = 1$, then $\alpha_2 = \zeta - \alpha_1$. Then we will have the following chain of implications:

$$0 \leqslant \alpha_1 \leqslant C \Rightarrow \zeta - C \leqslant \zeta - \alpha_1 \leqslant \zeta \Rightarrow \zeta - C \leqslant \alpha_2 \leqslant \zeta,$$

  which means that $\max(0, \zeta - C) \leqslant \alpha_2 \leqslant \min(\zeta, C)$, or using $\zeta = \alpha_1^{old} + s\alpha_2^{old}$ $(s = 1)$:

$$\max(0, \alpha_1^{old} + \alpha_2^{old} - C) \leqslant \alpha_2 \leqslant \min(\alpha_1^{old} + \alpha_2^{old}, C) \tag{13}$$

- If $s = -1$, then $\alpha_2 = \alpha_1 - \zeta$. In this case:

$$0 \leqslant \alpha_1 \leqslant C \Rightarrow -\zeta \leqslant \alpha_1 - \zeta \leqslant C - \zeta \Rightarrow -\zeta \leqslant \alpha_2 \leqslant C - \zeta,$$

  which means that $\max(0, -\zeta) \leqslant \alpha_2 \leqslant \min(C - \zeta, C)$, or using $\zeta = \alpha_1^{old} + s\alpha_2^{old}$ $(s = -1)$:

$$\max(0, \alpha_2^{old} - \alpha_1^{old}) \leqslant \alpha_2 \leqslant \min(C + \alpha_2^{old} - \alpha_1^{old}, C) \tag{14}$$

The KKT conditions also give the formula to calculate b. Assuming that after one step of the SMO algorithm we got $0 < \alpha_2 < C$, then

$$y^{(2)}(w^T x^{(2)} + b) = 1 \Rightarrow w^T x^{(2)} + b = y^{(2)},$$

implies

$$\begin{aligned} b_2 &= y^{(2)} - w^T x^{(2)} = y^{(2)} - \sum_{i=1}^{m} y^{(i)} \alpha_i \mathbf{K}_{i2} = \\ &= y^{(2)} - y^{(1)} \alpha_1 \mathbf{K}_{12} - y^{(2)} \alpha_2 \mathbf{K}_{22} - \sum_{i=3}^{m} y^{(i)} \alpha_i \mathbf{K}_{i2} = \\ &= y^{(2)} - y^{(1)} \alpha_1 \mathbf{K}_{12} - y^{(2)} \alpha_2 \mathbf{K}_{22} - (h(x^{(2)}) - b - \alpha_1^{old} y^{(1)} \mathbf{K}_{12} - \alpha_2^{old} y^{(2)} \mathbf{K}_{22}) = \\ &= b^{old} - (h(x^{(2)}) - y^{(2)}) - y^{(2)} \mathbf{K}_{22} (\alpha_2 - \alpha_2^{old}) - y^{(1)} \mathbf{K}_{12} (\alpha_1 - \alpha_1^{old}) \end{aligned} \tag{15}$$

Similarly, if $0 < \alpha_1 < C$:

$$b_1 = b^{old} - (h(x^{(1)}) - y^{(1)}) - y^{(2)}\mathbf{K}_{12}(\alpha_2 - \alpha_2^{old}) - y^{(1)}\mathbf{K}_{11}(\alpha_1 - \alpha_1^{old}) \tag{16}$$

If none of the conditions $0 < \alpha_1 < C$ and $0 < \alpha_2 < C$ is true, then we can take the average $\frac{b_1 + b_2}{2}$ (any b between b_1 and b_2 satisfies to the KKT conditions).

When we switch to the general case and take any pair of α_i, α_j, first we choose α_j such that it does not satisfy the KKT condition (8) (with some tolerance γ):

$$0 < \alpha_j < C \Rightarrow y^{(j)}(w^T x^{(j)} + b) = 1 \Rightarrow y^{(j)}(h(x^{(j)}) - y^{(j)}) = 0 \tag{17}$$

Also notice that if $\alpha_j = C$, then we could have $y^{(j)}(h(x^{(j)}) - y^{(j)}) < 0$ and if $\alpha_j = 0$, then we could have $y^{(j)}(h(x^{(j)}) - y^{(j)}) > 0$. The following algorithm summarizes all our calculations with references to the formulas.

Algorithm 3 SMO algorithm for Support Vector Machine

set C, γ, initial values $\alpha_i = 0$, $b = 0$
repeat
  for $j = 1$ **to** m **do**
    evaluate $E_j = h(x^{(j)}) - y^{(j)}$
    if $(y^{(j)}E_j < -\gamma$ and $\alpha_j < C)$ or $(y^{(j)}E_j > \gamma$ and $\alpha_j > 0)$ **then** ▷ (17)
      repeat
        choose $\alpha_i, i \neq j$, randomly
        evaluate $E_i = h(x^{(i)}) - y^{(i)}$
        if $y^{(i)} \cdot y^{(j)} > 0$ **then**
          $L = \max(0, \alpha_i + \alpha_j - C)$ ▷ (13)
          $H = \min(\alpha_i + \alpha_j, C)$ ▷ (13)
        else
          $L = \max(0, \alpha_j - \alpha_i)$ ▷ (14)
          $H = \min(C + \alpha_j - \alpha_i, C)$ ▷ (14)
        if $L == H$ **then continue**
        evaluate $\eta = 2\mathbf{K}_{ij} - \mathbf{K}_{ii} - \mathbf{K}_{jj}$
        if $\eta == 0$ **then continue**
        evaluate $\alpha_j^{new} = \min(\max(\alpha_j - y^{(j)}\frac{E_i - E_j}{\eta}, L), H)$ ▷ (11)
        if $|\alpha_j^{new} - \alpha_j| < 10^{-5}$ **then continue**
        evaluate $\alpha_i^{new} = \alpha_i + y^{(j)}y^{(i)}(\alpha_j - \alpha_j^{new})$ ▷ (12)
        if $(\alpha_j^{new} > 0$ and $\alpha_j^{new} < C)$ **then**
          $b = b - E_j - y^{(j)}\mathbf{K}_{jj}(\alpha_j^{new} - \alpha_j) - y^{(i)}\mathbf{K}_{ij}(\alpha_i^{new} - \alpha_i)$ ▷ (15)
        else
          if $(\alpha_i^{new} > 0$ and $\alpha_i^{new} < C)$ **then**
            $b = b - E_i - y^{(j)}\mathbf{K}_{ij}(\alpha_j^{new} - \alpha_j) - y^{(i)}\mathbf{K}_{ii}(\alpha_i^{new} - \alpha_i)$ ▷ (16)
          else

$$b = b - 0.5 \cdot (E_i + E_j + y^{(j)}(\mathbf{K}_{jj} + \mathbf{K}_{ij})(\alpha_j^{new} - \alpha_j) + y^{(i)}(\mathbf{K}_{ij} + \mathbf{K}_{ii})(\alpha_i^{new} - \alpha_i))$$

▷ (15), (16)

      until False
until convergence
return α, b

6.1 Python implementation

Loading necessary libraries:

```
In [1]: import numpy as np
        import random
        import math
```

```
import sklearn.datasets as ds
from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

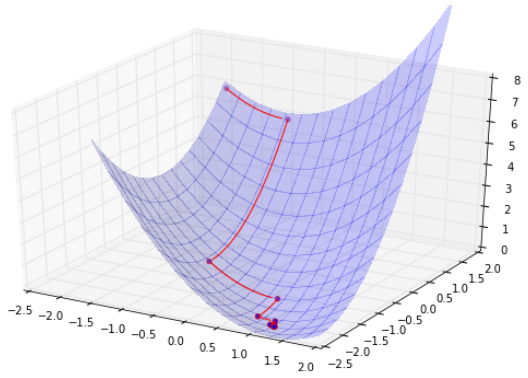
The coordinate ascent algorithm for the function

$$z = x^2 + y^2 + 2x + xy$$

can be implemented as follows:

```
In [2]: def fun(a,b):
            return a**2 + b**2 + 2*b + a*b

        def dfun(a,b,var):
            if var == 0:
                return -b/2.0
            else:
                return - 1.0 - 0.5*a

        def coordinate_ascent(theta0, iters):
            history = [np.copy(theta0)] # to store all thetas
            theta = theta0 # initial values for thetas
            for i in range(iters): # number of iterations
                for j in range(2): # number of variables
                    theta[j] = dfun(theta[0], theta[1], j)
                    history.append(np.copy(theta))
            return history

        history = coordinate_ascent(theta0 = [-1.8, 1.6], iters = 7)

        fig = plt.figure(figsize=(10, 6))
        ax = fig.gca(projection='3d')
        plt.hold(True)
        a = np.arange(-1.85, 1.85, 0.25)
        b = np.arange(-1.85, 1.85, 0.25)
        a, b = np.meshgrid(a, b)
        c = fun(a,b)
        surf = ax.plot_surface(a, b, c, rstride=1, cstride=1, alpha=0.2,
                               linewidth=0, antialiased=False)
        ax.set_zlim(-0.01, 8.01)

        a = np.array([x[0] for x in history])
        b = np.array([x[1] for x in history])
        c = fun(a,b)
        ax.scatter(a, b, c, color="r");
```

```
for i in range(len(history)-1):
    if history[i][0] == history[i+1][0]:
        b = np.arange(history[i][1], history[i+1][1],
                      np.sign(history[i+1][1] - history[i][1])*0.1)
        a = np.ones(len(b)) * history[i][0]
        c = fun(a,b)
        ax.plot(a, b, c, color="r")
    else:
        a = np.arange(history[i][0], history[i+1][0],
                      np.sign(history[i+1][0] - history[i][0])*0.1)
        b = np.ones(len(a)) * history[i][1]
        c = fun(a,b)
        ax.plot(a, b, c, color="r")

plt.show()
```



There are several implementations of SVM in the `scikit-learn` package: SVC (for classification) and SVR (for regression).

```
In [3]: from sklearn.svm import SVC
```

As in the previous lecture we try to solve the XOR problem, but this time we are going to use Support Vector Machine for it.

$\mathbf{x_1}$	$\mathbf{x_2}$	XOR $(\mathbf{x_1}, \mathbf{x_2})$
1	1	1
1	0	0
0	1	0
0	0	1

The XOR function detects if x_1 and x_2 are the same. Define our dataset: 4 training examples with XOR as a target variable y.

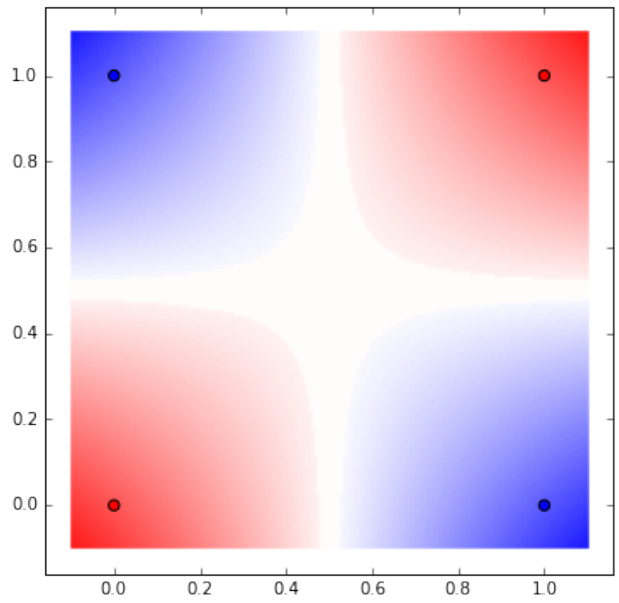
```
In [4]: X_train = np.array([[1, 1], [1, 0], [0, 1], [0, 0]]).astype(np.float64)
        y_train = np.array([1, -1, -1, 1])
```

Train the SVC model:

```
In [5]: clf = SVC(probability=True, C=100.0, gamma=0.1, kernel='rbf')
        clf.fit(X_train, y_train)
```

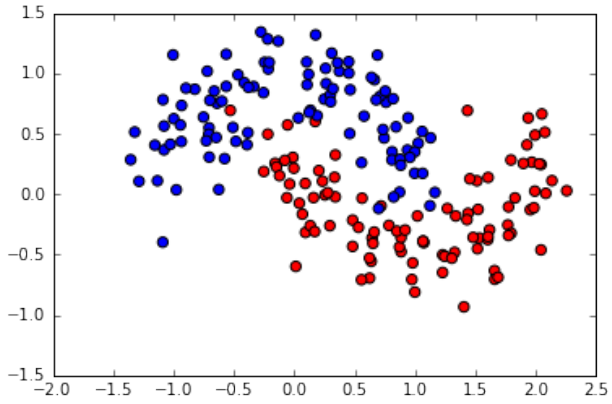
We are going to use the function `plot_decision_boundary()` to draw the decision boundary.

```
In [6]: def plot_decision_boundary(X, y, model,
                                   xmin=-0.1, xmax=1.1,
                                   ymin=-0.1, ymax=1.1):
            fig = plt.figure(figsize=(6,6))
            x1, x2 = np.meshgrid(np.linspace(xmin, xmax, 200),
                                 np.linspace(ymin, ymax, 200))
            ypred = model.predict_proba(np.c_[x1.ravel(), x2.ravel()])[:,0]
            ypred = ypred.reshape(x1.shape)
            extent = xmin, xmax, ymin, ymax

            plt.imshow(ypred, cmap=cm.bwr, alpha=.9, interpolation='bilinear',
                       extent = extent, origin='lower')
            plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=cm.bwr)

In [7]: plot_decision_boundary(X_train, y_train, clf)
```



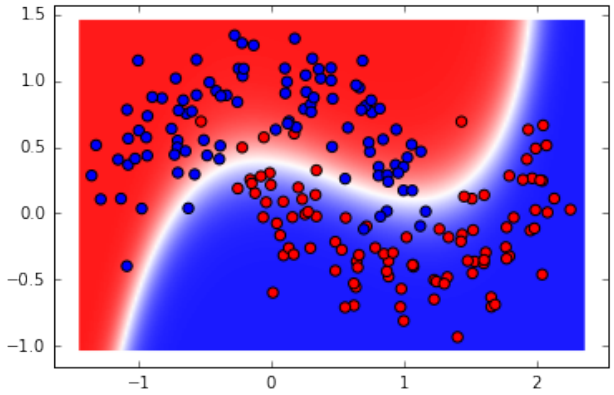
For the `make_moons` benchmark from the `scikit-learn`:

```
In [8]: np.random.seed(0)
        X_train, y_train = ds.make_moons(200, noise=0.20)
        plt.scatter(X_train[:,0], X_train[:,1], s=40, c=y_train, cmap=cm.bwr)
        plt.show()
```




```
In [9]: clf = SVC(probability=True, C=100.0, gamma=0.1, kernel='rbf')
        clf.fit(X_train, y_train)
```

```
In [10]: plot_decision_boundary(X_train, y_train, clf,
                                 xmin=np.min(X_train[:,0])-0.1,
                                 xmax=np.max(X_train[:,0])+0.1,
                                 ymin=np.min(X_train[:,1])-0.1,
                                 ymax=np.max(X_train[:,1])+0.1)
```



We can get some useful information from the model, for example, number of support vectors (number of i such that $\alpha_i \neq 0$):

```
In [11]: print "Number of support vectors: " + str(len(clf.support_))
```

```
Number of support vectors: 51
```

We have also implemented the simplified version of SMO algorithm. First we define the function that calculates a kernel. We are going to use RBF kernel, which means that $\mathbf{K}_{ii} = 1$ for any i.

```
In [12]: def kernel(x1, x2):
             return np.exp(-np.sum(np.power([i-j for i,j in zip(x1,x2)], 2))/2.0)
```

When we predict the output for the test dataset, we should calculate the kernel function for each pair (test example, support vector). It becomes very computationally expensive if you use loops for that. I have written the function that use some vectorized operations from `numpy`:

```
In [13]: def h(X, y, alpha, b, Xtest):
             ix = [i for i,a in enumerate(alpha) if a!=0]
             X_magnitude = 0.5*np.sum(np.power(X[ix,:], 2), axis=1)
             Xtest_magnitude = 0.5*np.sum(np.power(Xtest, 2), axis=1)
             dists = np.multiply(np.exp(np.dot(Xtest, X[ix,:].T) -
                                        np.add.outer(Xtest_magnitude,
                                                     X_magnitude)),
                                 y[ix])
             return np.sum(dists, axis=1) + b
```

The main function is `smo_step()` that takes X, y, α, b and index j as inputs and returns new values α and new value b. Notice that if this function fails it returns previous values for α and b.

```
In [14]: def smo_step(X, y, alpha, b, j):
             Ej = h(X, y, alpha, b, X[j,:].reshape((1,-1)))[0] - y[j]
             if ((y[j]*Ej < -gamma and alpha[j]<C) or
                 (y[j]*Ej > gamma and alpha[j]>0)):
                 ilist = [x for x in range(X.shape[0]) if x != j]
                 while True:
                     if len(ilist) == 0:
                         break
                     i = random.sample(ilist, 1)[0]
                     ilist.remove(i)
                     Ei = h(X, y, alpha, b, X[i,:].reshape((1,-1)))[0] - y[i]
                     if y[i]*y[j] > 0:
                         L = max(0.0, alpha[i] + alpha[j] - C)
                         H = min(alpha[i] + alpha[j], C)
                     else:
                         L = max(0.0, alpha[j] - alpha[j])
                         H = min(C + alpha[j] - alpha[i], C)
                     if L == H: continue
                     Kij = kernel(X[i,:], X[j,:])
                     eta = 2.0*Kij - 2.0
                     if eta == 0: continue
                     alpha_j = np.clip(alpha[j] - y[j]*(Ei - Ej)/eta, L, H)
                     if np.abs(alpha_j - alpha[j]) < 0.00001: continue
```

```
                    alpha_i = alpha[i] + y[i]*y[j]*(alpha[j] - alpha_j)
                    if (alpha_j > 0 and alpha_j < C):
                        b = b - Ej - y[j]*(alpha_j - alpha[j])
                                   - y[i]*Kij*(alpha_i - alpha[i])
                    else:
                        if (alpha_i > 0 and alpha_i < C):
                            b = b - Ei - y[j]*Kij*(alpha_j - alpha[j])
                                       - y[i]*(alpha_i - alpha[i])
                        else:
                            b = b - 0.5*(Ei + Ej
                                         + y[j]*(1.0 + Kij)*(alpha_j - alpha[j])
                                         + y[i]*(1.0 + Kij)*(alpha_i - alpha[i]))
                    alpha[i] = alpha_i
                    alpha[j] = alpha_j
                    break
            return alpha, b
```

One more auxiliary function `plot_decision_boundary()`. The main difference is that now we are using function h for the prediction.

```
In [15]: def plot_decision_boundary(X, y, alpha, b,
                                    xmin=-0.1, xmax=1.1,
                                    ymin=-0.1, ymax=1.1):
             x1, x2 = np.meshgrid(np.linspace(xmin, xmax, 200),
                                  np.linspace(ymin, ymax, 200))
             ypred = np.array(h(X, y, alpha, b, np.c_[x1.ravel(), x2.ravel()]))
             ypred = ypred.reshape(x1.shape)
             extent = xmin, xmax, ymin, ymax

             plt.imshow(ypred, cmap=cm.bwr, alpha=.9, interpolation='bilinear',
                        extent = extent, origin='lower')
             plt.scatter(X[:,0], X[:,1], s=40, c=y, cmap=cm.bwr)
```

As an example I have implemented the simple function that returns the number of support vectors:

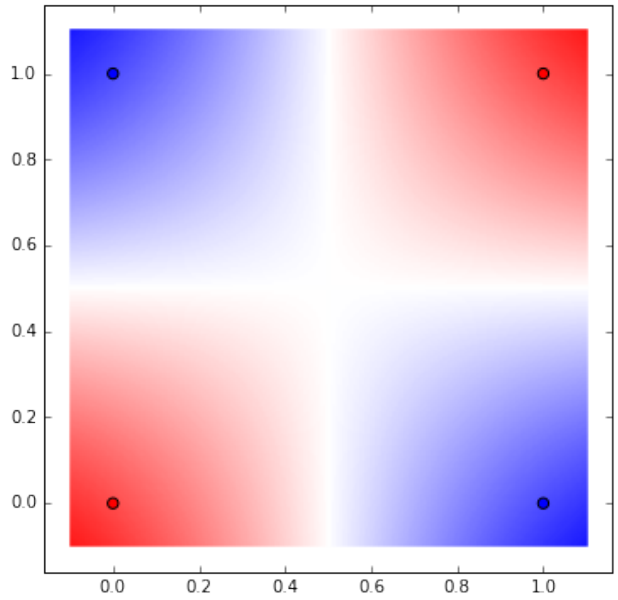
```
In [16]: def support_vector_count(alpha):
             return len([i for i,a in enumerate(alpha) if a!=0])
```

We solve the XOR problem using our implemented SVM.

```
In [17]: X_train = np.array([[1, 1], [1, 0], [0, 1], [0, 0]]).astype(np.float64)
         y_train = np.array([1, -1, -1, 1])
```

```
In [18]: ### define the parameters and initialize values for alpha and b
         C = 10.0
```

```
gamma = 0.1
b = 0.0
alpha = np.zeros(X_train.shape[0])
niter = 10

### parameters for plotting
fig = plt.figure(figsize=(6,6))

### our main loop
for it in range(niter):
    for j in range(X_train.shape[0]):
        alpha, b = smo_step(X_train, y_train, alpha, b, j)
plot_decision_boundary(X_train, y_train, alpha, b,
                       xmin=np.min(X_train[:,0])-0.1,
                       xmax=np.max(X_train[:,0])+0.1,
                       ymin=np.min(X_train[:,1])-0.1,
                       ymax=np.max(X_train[:,1])+0.1)
plt.show()
```



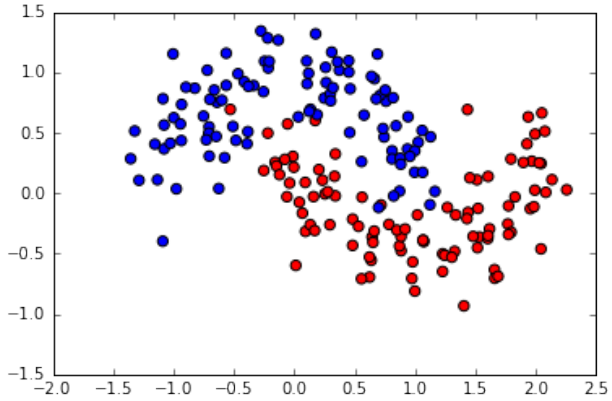

```
In [19]: print "alpha: " + str(alpha)
         print "b: " + str(b)
```

```
alpha: [ 10.  10.  10.  10.]
b: -0.368625049269
```

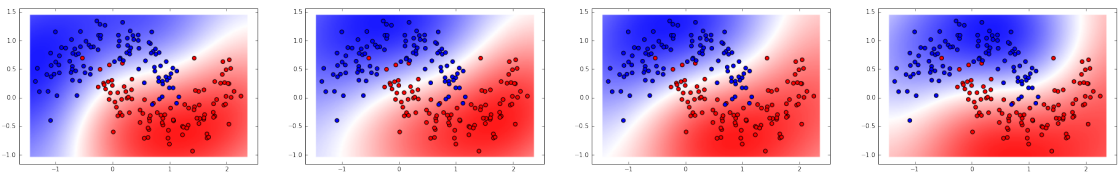
We check our implementation for `make_moons` data:

```
In [20]: np.random.seed(0)
         X_train, y_train = ds.make_moons(200, noise=0.20)
```

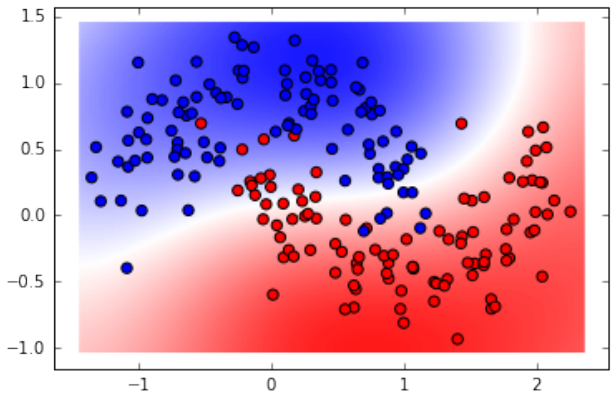
```
y_train[y_train==0] = -1
plt.scatter(X_train[:,0], X_train[:,1], s=40, c=y_train, cmap=cm.bwr)
plt.show()
```



In [21]:
```
### define the parameters and initialize values for alpha and b
C = 1000.0
gamma = 0.001
b = 0.0
alpha = np.zeros(X_train.shape[0])
niter = 600

### parameters for plotting
iter_to_plot = [1, niter/3, 2*niter/3, niter-1]
count_plot = 0
fig = plt.figure(figsize=(32,32))

### our main loop
for it in range(niter):
    for j in range(X_train.shape[0]):
        alpha, b = smo_step(X_train, y_train, alpha, b, j)
    if it in iter_to_plot:
        count_plot = count_plot + 1
        fig.add_subplot(1,len(iter_to_plot),count_plot)
        plot_decision_boundary(X_train, y_train, alpha, b,
                               xmin=np.min(X_train[:,0])-0.1,
                               xmax=np.max(X_train[:,0])+0.1,
                               ymin=np.min(X_train[:,1])-0.1,
                               ymax=np.max(X_train[:,1])+0.1)
plt.show()
```



```
In [22]: fig = plt.figure(figsize=(6,6))
         plot_decision_boundary(X_train, y_train, alpha, b,
                                xmin=np.min(X_train[:,0])-0.1,
                                xmax=np.max(X_train[:,0])+0.1,
                                ymin=np.min(X_train[:,1])-0.1,
                                ymax=np.max(X_train[:,1])+0.1)
         plt.show()
```




```
In [23]: print "Number of support vectors: " + str(support_vector_count(alpha))
```

```
Number of support vectors: 88
```

The final remark is that built-in algorithm uses a lot of optimization techniques (for example, it does not choose pair of α_i, α_j randomly but utilizes some heuristic for that). As we can see even without these techniques our algorithm gives pretty good result.

References

[1] J. Platt. "Fast Training of Support Vector Machines using Sequential Minimal Optimization", *in Advances in Kernel Methods - Support Vector Learning, B. Scholkopf, C. Burges, A. Smola, eds., MIT Press*, 1998.