Machine Learning: Foundations **Fall Semester, 2012**

Lecture 11: January 6, 2013 (SVM)

Lecturer: Yishay Mansour *Scribe: Aviv Eisenschtat, Yackov Lubarsky, Shimi Salant*

11.1 Kernels

Here we will see the advatage of the dual program, of using a number of variables proportioned to the number of examples rather than to the number of features.

11.1.1 The Idea

Say we want to use SVM on data which is not linearly separable. If we could map from our initial dimension to a larger dimension, maybe in this new space we can find a linear seperation for our data.

11.1.2 Example

Consider the linear separator

$$sign(\sum_{i=1}^{n} x_i \alpha_i).$$

Instead we can use a quadratic function

$$sign(\sum_{i,j} x_i x_j \alpha_{i,j}).$$

Our transformation:

$$(x_1 \dots x_n) \longmapsto (x_1 \dots x_n, x_1^2, \dots, x_n^2, x_1 x_2, \dots, x_{n-1} x_n).$$

So we moved from n-dimensional space to $n + n^2$ dimensional space. We could do the same for degree 3, etc.

11.1.3 The problem and its solution

Once we move to a higher dimension, all the calculations become more complex. In fact, the complexity increases exponentially. Our main goal would be to get the benefits of the higher dimension without paying the penalty.

Definition We define a kernel as the inner products between two $\phi(\cdot)$:

$$K(x, z) = \phi(x) \cdot \phi(z)$$

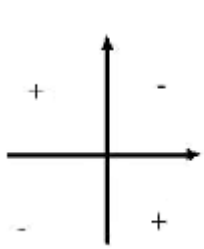

Figure 11.1: XOR

The idea is that in some cases we can calculate $K(x, z)$ without calculating $\phi(x)$, thus avoiding the added comlplexity.

11.1.4 Example 1 - Polynomials

Assume $n = 2$.

$$K(x,z) = (x \cdot z)^2 = (x_1z_1 + x_2z_2)^2 = x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 = \underbrace{(x_1^2, x_2^2, \sqrt{2}x_1x_2)}_{\phi(x)} \cdot \underbrace{(z_1^2, z_2^2, \sqrt{2}z_1z_2)}_{\phi(z)}$$

We can see that it is possible to calculate $K(x, z)$ without actually calculating $\phi(x)$ or $\phi(z)$, and that makes the difference in complexity ($O(n)$ instead of $O(n^2)$).

Similarly, $K_d(x, z) = (x \cdot z + c)^d$ is mapped into $\phi(\cdot)$ with $\binom{n+d}{d} = O(n^d)$ entries, but calculating K_d is only $O(n)$.

11.1.5 Example 2 - XOR

XOR of two variables is obviously not linearly separablse. Let's look at a general degree-2 kernel:

$$K_2(x, y) = (x \cdot y + c)^2 = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ c \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}y_1y_2 \\ \sqrt{2c}y_1 \\ \sqrt{2c}y_2 \\ c \end{pmatrix}$$

If $c = 0$ we are left with only the top three components:

$$K_2(x, y) = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}y_1y_2 \end{pmatrix}$$

XOR representation will be:

$$\phi(x) \cdot \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{w} = \sqrt{2}x_1x_2$$

Where

$$sign(\phi(x) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}) = sign(x_1x_2) \geq 0 \leftrightarrow XOR(x) = 0$$

So we have the weight vector w in the higher dimension, and we want to build it from examples in the lower dimension.

label	x_1	x_2	x_1^2	x_2^2	x_1x_2
$+1$	$+1$	$+1$	$+1$	$+1$	$+1$
-1	$+1$	-1	$+1$	$+1$	-1
-1	-1	$+1$	$+1$	$+1$	-1
$+1$	-1	-1	$+1$	$+1$	$+1$

$$w = +1 \cdot \phi(+1, +1) - 1 \cdot \phi(+1, -1) - 1 \cdot \phi(-1, +1) + 1 \cdot \phi(-1, -1)$$

$$\begin{aligned} h(x) &= sign\left(K_2\left(x, (+1, +1)\right) + K_2\left(x, (-1, -1)\right) - K_2\left(x, (+1, -1)\right) - K_2\left(x, (-1, +1)\right)\right) \\ &= sign\left(\phi\left(x\right) \cdot \begin{pmatrix} 0 \\ 0 \\ 4\sqrt{2} \end{pmatrix}\right) = sign\left(\begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 4\sqrt{2} \end{pmatrix}\right) \\ &= sign\left(8x_1x_2\right) \end{aligned}$$

11.1.6 Example 3 - Gaussian Kernel

Another example is the gaussian kernel, which maps the space into an infinite new space, but is also computable in linear comlexity.

$$K_G(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) = \sum_{n=0}^{\infty}\frac{(x\cdot z)^n}{\sigma^n n!}$$

A classifier (given w):

$$h(x) = sign(-\frac{1}{e} + K_G(x,w))$$

11.1.7 To sum it up

The vector $\phi(x)$ can be thought of as the features that we learn in the new space. If the learning algorithm (SVM in this case) can be written in a way that uses only the inner products of the data, then it is possible to "kernelize" it.

11.1.8 Closure Properties of Kernel Functions

Given the benefits of kernels, we would like to create new kernels from existing ones.

1. Closure under the following actions:

   (a) Adding a constant:
   $$K(x,z) = K_1(x,z) + c,$$
   for a given kernel K_1 with $\phi_1(x)$, we can construct the resulting kernel's $\phi(x)$ by adding another entry to $\phi_1(x)$ with value $\sqrt{c}$.

   (b) Multiplication by constant:
   $$K(x,z) = c\cdot K_1(x,z),$$
   by setting $\phi(x) = \sqrt{c}\cdot\phi_1(x)$.

   (c) Addition of two kernel functions:
   $$K(x,z) = K_1(x,z) + K_2(x,z),$$
   by setting $\phi(x) = \begin{pmatrix}\phi_1(x)\\ \phi_2(x)\end{pmatrix}$.

(d) Multiplication of two kernel functions:

$$K(x,z) = K_1(x,z) \cdot K_2(x,z)$$

To see this, let l_1 be the length of ϕ_1 and l_2 be the length of ϕ_2. Define $\phi(x)$ of length $l_1 l_2$, such that the (i,j) entry in $\phi(x)$ is $\phi_{1,i}(x)\,\phi_{2,j}(x)$ where $\phi_{1,i}(x)$ is the i-th entry of $\phi_1(x)$.

2. Let K be a legitimate kernel function. Then it is possible to define a kernel matrix as follows: $K_{ij} = K\left(x^{(i)}, x^{(j)}\right)$. The result matrix K will be symmetric. For any z :

$$\begin{aligned} z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\ &= \sum_i \sum_j z_i \phi\left(x^{(i)}\right)^T \cdot \phi\left(x^{(j)}\right) z_j \\ &= \sum_i \sum_j z_i \sum_k \phi_k\left(x^{(i)}\right) \phi_k\left(x^{(j)}\right) z_j \\ &= \sum_k \sum_i \sum_j z_i \phi_k\left(x^{(i)}\right) \phi_k\left(x^{(j)}\right) z_j \\ &= \sum_k \left(\sum_i z_i \phi_k\left(x^{(i)}\right) \right)^2 \geq 0 \end{aligned}$$

This means that if K is a kernel then its matrix representation is positive-semi-definite (PSD). We have not shown the other direction, but it is also true:

Theorem 11.1 (Mercer) *K is a legitimite kernel function $\leftrightarrow$ K is PSD.*

(Note: For our needs it is not required to calculate $\phi(\cdot)$. It's enough that it exists.) The following theorem characterizes the solution to an optimization problem when we have kernel functions.

Theorem 11.2 (The Representaion Theorem) *Let K be a legitimate kernel function. H is the function space (RKHS). G is monotone increasing function. Then the solution to the maximization problem*

$$arg\min_{h \in H} F(h) = arg\min_{h \in H} G\left(\|h\|_H^2\right) + l\left(h(x_1) \ldots h(x_m)\right)$$

is of the form:

$$h^*(x) = \sum_{i=1}^{m} \alpha_i K(x_i, x)$$

Theorem 11.3 *(Reminder about SVM)*
The solution to the maximization problem

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

where $\alpha_i \geq 0$ *and* $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$*, is of the following form:*

$$h(x) = sign\left(\sum_{i=1}^{m} \alpha_i y^{(i)} K\left(x^{(i)}, x\right) + b\right).$$

11.2 Dealing with non linearly-separable data

11.2.1 The Idea

What happens if the data that we have is not linearly separable? (which is a common event)

Every hyperplane (w, b) has a point $x^{(i)}$ such that $y^{(i)}(w \cdot x^{(i)} + b) < 0$ (otherwise the data would be linearly separable). The idea is "to soften" the hard constraint ("hard margin") and turn it into a soft constraint ("soft margin"). We would want to minimize the violations of the constraints, so we'll change the constraint such that $y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \psi^{(i)}$ where $\psi^{(i)} \geq 0$. Now there is always a solution. So what exactly do we want to minimize?

$$\min_{w,b,\psi} \underbrace{\frac{1}{2}\|w\|^2}_{margin} + \underbrace{C}_{cost} \cdot \underbrace{F\left(\psi^{(1)}, \ldots, \psi^{(m)}\right)}_{accuracy},$$

where F is a measurement of accuracy, and C is an important parameter which determines a tradeoff between accuracy and margin size. In other words - a tradeoff between generalization capability (small $\|w\|$) and the classification error on the training data.

A natural choice for F is to calculate accuracy individually for each point and to sum everything up:

$$F\left(\psi^{(1)}, \ldots, \psi^{(m)}\right) = \sum_{i=1}^{m} F\left(\psi^{(i)}\right).$$

Ideally:

$$F_{01}(\psi) = \begin{cases} 1 & \psi > 0 \\ 0 & \psi = 0 \end{cases},$$

this will count the exact number of points that are not classified correctly by our classifier.

11.2.2 Optimization

This is a hard optimization problem (even with no margin). From a complexity point of view, even if it is guaranteed that a hyperplane exists such that it classifies 99% of the points correctly, finding a hyperplane which classifies 51% of the points correctly is an NP-complete problem (also note that F is not convex).

For this reason we write the following convex program:

$$\min_{w,b,\psi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \psi^{(i)}$$

Such that

$$y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \psi^{(i)}$$
$$\psi^{(i)} \geq 0.$$

For this convex program we get the following Lagrangian (α and β are the lagrange multipliers):

$$L\left(w, b, \psi, \alpha, \beta\right) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \psi^{(i)} - \sum_{i=1}^{m} \alpha_i \left(y^{(i)}(w \cdot x^{(i)} + b) - 1 + \psi^{(i)}\right) - \sum_{i=1}^{m} \beta_i \psi^{(i)}$$

Now we'll check the KKT conditions:

1. $\nabla L = 0$
   (a) $\nabla_w L = w - \sum \alpha_i y^{(i)} x^{(i)} = 0 \leftrightarrow w^* = \sum \alpha_i y^{(i)} x^{(i)}$
   (b) $\nabla_b L = -\sum \alpha_i y^{(i)} = 0 \leftrightarrow \sum \alpha_i y^{(i)} = 0$
   (c) $\nabla_{\psi^{(i)}} L = C - \alpha_i - \beta_i = 0 \leftrightarrow \alpha_i + \beta_i = C$
2. Complementary
   (a) $\alpha_i \left[y^{(i)}(w \cdot x^{(i)} + b) - 1 + \psi^{(i)}\right] = 0 \rightarrow \alpha_i = 0$ or $y^{(i)}(w \cdot x^{(i)} + b) = 1 - \psi^{(i)}$ (when $y^{(i)}(w \cdot x^{(i)} + b) = 1 - \psi^{(i)}$ and $\alpha_i > 0$ this is a support vector, it can be classified incorrectly or be exactly on the margin)
   (b) $\beta_i \psi^{(i)} = 0 \rightarrow \beta_i = 0$ or $\psi^{(i)} = 0$
3. Non negativity
   (a) $\alpha_i \geq 0$
   (b) $\beta_i \geq 0$

(c) $\psi^{(i)} \geq 0$

Substituting these conditions back to the Lagrangian we get:

$$\frac{1}{2}\|w\|^2 = \frac{1}{2}\sum \alpha_i \alpha_j y^{(i)} y^{(j)} \left(x^{(i)} \cdot x^{(j)}\right)$$

$$C\sum \psi^{(i)} - \sum \alpha_i \psi^{(i)} - \sum \beta_i \psi^{(i)} = 0$$

$$\sum \alpha_i \left[y^{(i)}(w \cdot x^{(i)} + b) - 1\right] = \underbrace{\sum \alpha_i y^{(i)} \left(w^* \cdot x^{(i)}\right)}_{\sum_{i,j} \alpha_i \alpha_j y^{(i)} y^{(j)} \left(x^{(i)} \cdot x^{(j)}\right)} + \underbrace{\sum \alpha_i y^{(i)} b}_{0} - \sum \alpha_i$$

We have the following

$$\boxed{L = -\frac{1}{2}\sum \alpha_i \alpha_j y^{(i)} y^{(j)} \left(x^{(i)} \cdot x^{(j)}\right) + \sum \alpha_i} \tag{11.1}$$

We need to add the constraints. For the constraints $\beta_i \geq 0$ and $C = \beta_i + \alpha_i$ we add the constraint: $C \geq \alpha_i \geq 0$. We have in addition $\sum \alpha_i y^{(j)} = 0$. The solution:

$$h\left(x\right) = sign\left[\sum \alpha_i y^{(i)} \left(x^{(i)} \cdot x\right) + b\right],$$

where b is defined for the vector that adhere to $C > \alpha_i > 0$:

$$b = y^{(i)} - \sum_{j=1}^{m} \alpha_j y^{(j)} \left(x^{(i)} \cdot x^{(j)}\right).$$

If for point i we get $\psi^{(i)} > 0$, from the complementry we have $\beta_i \psi^{(i)} = 0$ so this means $\beta_i = 0$. In the optimal solution from the KKT conditions we have $C = \alpha_i + \beta_i$ meaning we get $\alpha_i = C$. Every point that is classified wrongly (either the margin is too small or negative), gets $\alpha_i = C$, so its effect on the decision hyperplane is limited to C. If a point is classified correctly, we get $\psi^{(i)} = 0$, then $C \geq \beta_i > 0$ and this means that $C > \alpha_i > 0$. For points that are classified correctly **and** are not support vectors we get $\alpha_i = 0$.

So again, large C means avoiding errors on the training data. Small C means more freedom in the hyperplane, and a better generalization ability.

11.3 SMO - Sequential Minimization Optimization

We shall now see a practical algorithm for solving the optimization problem (11.1). For practical Implementations, we need to add an accuracy parameter, say 0.001.

11.3.1 Single Variable Optimization

In order to optimize (11.1) we will start with a feasible but not optimal solution (e.g. $\alpha_i = 0 \ \forall i$) and look for local improvements (For the dual problem). We cannot change only one variable because, given the values of $\alpha_j \, \forall j \neq i$ we have:

$$\begin{aligned} \alpha_i y_i &= -\sum_{j \neq i} \alpha_j y_j \\ \alpha_i &= -y_i \sum_{j \neq i} \alpha_j y_j \end{aligned}$$

(Since $y_i \in \{\pm 1\}$)

11.3.2 Two Variables Optimization

So we would like to change the values of two variables, WLOG α_1, α_2. Thus, we need to maximize:

$$\begin{aligned} L(\alpha_1, \alpha_2) &= \left(1 - y_1 \sum_{j \notin \{1,2\}} y_j \alpha_j K(x_1, x_j)\right) \alpha_1 + \left(1 - y_2 \sum_{j \notin \{1,2\}} y_j \alpha_j K(x_2, x_j)\right) \alpha_2 \\ &\quad - \left(\frac{1}{2} y_1^2 K(x_1, x_2)\right) \alpha_1^2 - (y_1 y_2 K(x_1, x_2)) \alpha_1 \alpha_2 - \left(\frac{1}{2} y_2^2 K(x_1, x_2)\right) \alpha_2^2 \\ &\quad + const(\alpha_3 ... \alpha_n) \end{aligned}$$

$$\begin{aligned} s.t. \qquad & 0 \leq \alpha_1, \alpha_2 \leq C, \\ & \alpha_1 y_1 + \alpha_2 y_2 = Z = -\textstyle\sum_{j \notin \{1,2\}} \alpha_j y_j \end{aligned}$$

Substitute $\alpha_1 = y_1 (Z - \alpha_2 y_2)$ to get a quadratic:

$$L(\alpha_2) = a\alpha_2^2 + b\alpha_2 + c$$

We then need to find the value α_2^* that maximizes the L. If the value we get fits the constraints (i.e. $y_1(Z - \alpha_2^* y_2)$, $\alpha_2^* \in [0, C]$) we can use it. Otherwise, we use the best extreme values.

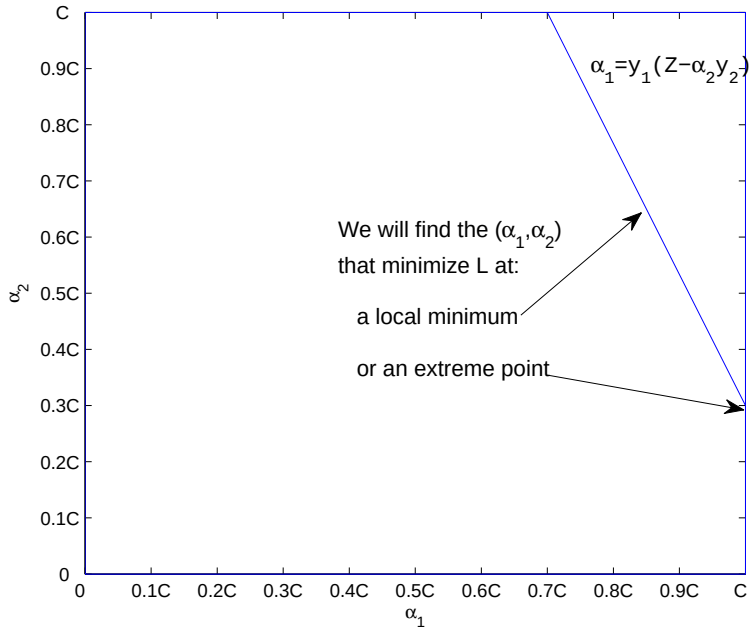

Figure 11.2: Constraints on(α_1, α_2)

11.3.3 Calculating the maximum of a quadratic on a finite interval

To find the α_2^* that maximizes $L(\alpha_2)$ given the constraints, we need to calculate the constraints in terms of an upper bound UB and a lower bound LB to α_2. Let $\left(\alpha_1^{old}, \alpha_2^{old}\right)$ be the values before optimization, and let $(\alpha_1^{new}, \alpha_2^{new})$ be the values that maximize $L(\alpha_1, \alpha_2)$ without the constraints.

We have from construction of α_2^*:

$$\begin{aligned} \alpha_2^* &= y_2(Z - \alpha_1^* y_1) \\ &= y_2 Z - y_1 y_2 \alpha_1^* \\ &= y_2(y_1 \alpha_1^{old} + y_2 \alpha_2^{old}) - y_1 y_2 \alpha_1^* \\ &= \alpha_2^{old} + y_1 y_2 \alpha_1^{old} - y_1 y_2 \alpha_1^* \end{aligned}$$

For $y_1 = y_2 = 1$; or $y_1 = y_2 = -1$:

$$\alpha_2^* = \alpha_1^* + \alpha_2^{old} + \alpha_1^{old} \Rightarrow \alpha_1^* = \alpha_2^{old} + \alpha_1^{old} - \alpha_2^*$$

From $C \geq \alpha_1^{new} \geq 0$ we have:

$$0 \leq \alpha_2^{old} + \alpha_1^{old} - \alpha_2^* \leq C$$

From $C \geq \alpha_2^{new} \geq 0$ we have:

$$max\left\{\alpha_1^{old} + \alpha_2^{old} - C,\, 0\right\} \;\; \leq \;\; \alpha_2^* \leq min\left\{C,\, \alpha_1^{old} + \alpha_2^{old}\right\}$$

So we set:

$$\begin{aligned} LB &= \max\left\{C,\, \alpha_1^{old} + \alpha_2^{old}\right\} \\ UB &= \min\left\{\alpha_1^{old} + \alpha_2^{old} - C,\, 0\right\} \end{aligned}$$

If $y_1 = 1$ and $y_2 = -1$; or $y_1 = -1$ and $y_2 = 1$:

$$\alpha_2^* = \alpha_1^* + \alpha_2^{old} - \alpha_1^{old} \Rightarrow \alpha_1^* = -\left(\alpha_2^{old} - \alpha_1^{old}\right) + \alpha_2^*$$

From $C \geq \alpha_1^{new} \geq 0$ we have:

$$0 \leq -\left(\alpha_2^{old} - \alpha_1^{old}\right) + \alpha_2^* \leq C$$

$$\alpha_2^{old} - \alpha_1^{old} \leq \alpha_2^* \leq \alpha_2^{old} - \alpha_1^{old} + C$$

From $C \geq \alpha_2^{new} \geq 0$ we have:

$$max\left\{\alpha_2^{old} - \alpha_1^{old},\, 0\right\} \;\; \leq \;\; \alpha_2^* \leq min\left\{C,\, \alpha_2^{old} - \alpha_1^{old} + C\right\}$$

So we set:

$$\begin{aligned} LB &= \max\left\{\alpha_2^{old} - \alpha_1^{old},\, 0\right\} \\ UB &= \min\left\{C,\, \alpha_2^{old} - \alpha_1^{old} + C\right\} \end{aligned}$$

To get:

$$\alpha_2^* = \begin{cases} UB & \alpha_2^{new} \geq UB \\ \alpha_2^{new} & LB < \alpha_2^{new} < UB \\ LB & \alpha_2^{new} \leq LB \end{cases}$$

11.3.4 Choosing α_1, α_2

A heuristic for choosing which α_i, α_j to update. We look at the KKT conditions:

$$\begin{aligned} \alpha_i = 0 &\implies h(x_i)y_i \geq 1 \\ 0 < \alpha_i < C &\implies h(x_i)y_i = 1 \\ \alpha_i = C &\implies h(x_i)y_i \leq 1 \end{aligned}$$

Recall that for an optimal solution, all the KKT conditions are satisified. In practice, we need to introduce an accuracy parameter, say $\epsilon = 0.001$. We will only expect the KKT conditions to be satisfied to within ϵ. To increase the speed of convergence, we will prefer to update α_i's which are significantly far (i.e. ϵ-far) from satisfying the KKT conditions.

11.3.5 Calculating b

At each iteration, to calculate b we shall use:

$$\begin{aligned} b_1 &= y_1 - \sum \alpha_j y_j K(x_j, x_1) \\ b_2 &= y_2 - \sum \alpha_j y_j K(x_j, x_2) \\ b &= \frac{b_1 + b_2}{2} \end{aligned}$$