Introduction to Machine Learning	**Fall Semester**
Lecture 7	
Lecturer: Lior Wolf	*Scribe: Yishay Mansour*

7.1 Kernels for SVM classifiers

Today we continue with *Support Vector Machine (SVM)* and introduce the idea of a *kernel*. Using a kernel we will be able to have non-linear classifiers, but maintain the computational efficiency.

We will cover today the following topics:

- Following discussions from last class
  - How many support vectors are there anyhow?
  - Positive definite matrices
- Support Vector machine (SVM) classifier and Kernels
  - The kernel trick
  - Which kernels to use
  - Constructing kernels
  - SVD and kernel SVD

7.2 Review of Some Topics from Last Class

7.2.1 Number of Support Vectors

When the data is linearly separable, the set of support vectors is at most $d+1$. (We can have degenerate cases, where many points are at the same distance, but in most "real" cases this will not happen.) However, you might have less than $d+1$ support vectors. (See Figures 7.2 and 7.1 for an example on the plane.)

To see why having $d+1$ support vectors is sufficient, consider the SVM convex program,

$$\begin{aligned} \min_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad & \frac{1}{2} w^t w \\ \text{s.t.} \quad & y_n(w^t x_n + b) \geq 1 \quad \forall n \in [1, N] \end{aligned}$$

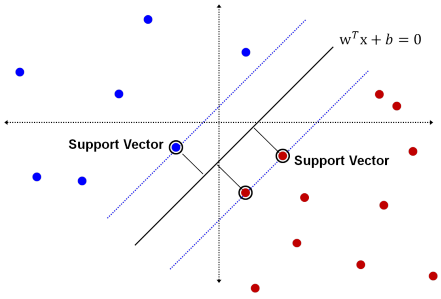

Figure 7.1: three support vectors

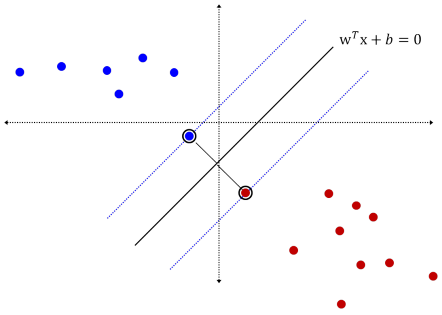

Figure 7.2: Two support vectors.

The support vectors maintain the inequality as an identity. Since we have $d+1$ variables (d for w and one for b), having $d+1$ linearly independent examples will already force the solution. Having more support vectors, means that some of them are dependent on the others, and therefore are not "essential". (This is not exactly a proof, since we are ignoring the non-linear objective.)

7.2.2 Positive Definite (PD) and Positive Semi-definite (PSD) Matrices

A symmetric $d \times d$ matrix A is *positive definite* (alternatively, *positive semi-definite*) if for any $y \in \mathbb{R}^d$ such that $y \neq 0$ we have $y^t A y > 0$ (alternative, $y^t A y \geq 0$).

An alternative second definition is that a symmetric matrix A is positive definite if there is a matrix P such that $A = PP^t$ and $det(P) \neq 0$. (Alternatively, a matrix A is positive semi-definite if we drop the requirement that $det(P) \neq 0$.)

If there is such a matrix P, then $y^t A y = y^t P P^t y = \|P^t y\|^2 \geq 0$, and if $det(P) \neq 0$ then $\|P^t y\|^2 > 0$.

The other direction is not shown here.

There are non-symmetric matrices, that can have the property that for any $y \neq 0$ we have $y^t A y > 0$, for example $A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$. One can verify that $(y_1, y_2)^t A (y_1, y_2) = y_1^2 + y_2^2 > 0$. The test for a non-symmetric matrix A whether for any y we have $y^t A y > 0$ is by testing if $A + A^t$ is PSD. Note that $A + A^t$ is symmetric. Also, for any y, we have $y^t A y = (y^t A y)^t = y^t A^t y$, since it is a scalar. Therefore, $2y^t A y = y^t A y + y^t A^t y = y^t (A + A^t) y$, and A has the property iff $A + A^t$ is PSD.

An alternative third definition is based on the eigenvalues of a positive definite matrix. Recall that a $d \times d$ symmetric real-valued matrix A has d eigenvalues λ_i and eigenvectors x_i

such that

$$Ax_i = \lambda_i x_i$$

and

$$[x_1 \cdots x_d]^t A [x_1 \cdots x_d] = diag(\lambda_1, \ldots, \lambda_d)$$

1. If $\lambda_i > 0$ for $i \in [1, d]$ then A is positive definite.
2. If $\lambda_i \geq 0$ for $i \in [1, d]$ then A is positive semi-definite.

Let $U = [x_1 \cdots x_d]$ where x_i are orthonormal eigenvectors. Since $UU^t = I$ for any $y \in \mathbb{R}^d$ we have

$$y = UU^t y = U \begin{bmatrix} x_1^t y \\ \vdots \\ x_d^t y \end{bmatrix} = \sum_{i=1}^{d} \alpha_i x_i$$

Then

$$y^t A y = y^t (\sum_{i=1}^{d} \alpha_i A x_i) = y^t (\sum_{i=1}^{d} \alpha_i \lambda_i x_i) =< (\sum_{i=1}^{d} \alpha_i x_i), (\sum_{i=1}^{d} \alpha_i \lambda_i x_i) >= \sum_{i=1}^{d} \alpha_i^2 \lambda_i$$

The last expression is strictly positive if $\lambda_i > 0$ and non-negative if $\lambda_i \geq 0$. This shows that if the eigenvalues are positive (non-negative) the matrix A is positive definite (positive semi-definite).

Similarly, for the other direction. If we assume that for any y we have $\sum_{i=1}^{d} \alpha_i^2 \lambda_i$ positive (non-negative), then by selecting $y = x_i$ (setting $\alpha_i = 1$) we show that $\lambda_i > 0$ ($\lambda_i \geq 0$).

7.2.3 The dual SVM formulation and PSD

Recall that the dual formulation is

$$\begin{aligned} \min_{\alpha} \quad & \alpha^t \underbrace{\begin{bmatrix} y_1 y_1 x_1^t x_1 & y_1 y_2 x_1^t x_2 & \cdots & y_1 y_N x_1^t x_N \\ y_2 y_1 x_2^t x_1 & y_2 y_2 x_2^t x_2 & \cdots & y_2 y_N x_2^t x_N \\ \vdots & \vdots & \vdots & \vdots \\ y_N y_1 x_1^t x_N & y_N y_2 x_N^t x_2 & \cdots & y_N y_N x_N^t x_N \end{bmatrix}}_{M} \alpha - 1^t \alpha \\ \text{s.t.} \quad & y^t \alpha = 0 \\ & 0 \leq \alpha \leq C \end{aligned}$$

We can write the matrix M as PP^t where $P^t = [y_1 x_1 \cdots y_N x_N]$. For this reason the matrix M is positive semi-definite. This guarantees that the optimization would have a unique local minima (which is the global minima) and that the computation can be done efficiently.

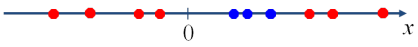

Figure 7.3: An interval of blue between two red areas

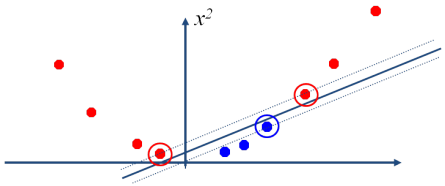

Figure 7.4: Lifting the interval to 2-D.

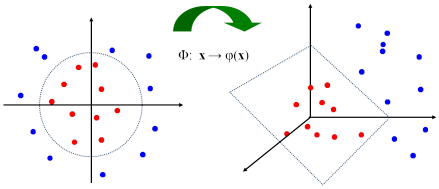

Figure 7.5: mapping from a circle to 3D

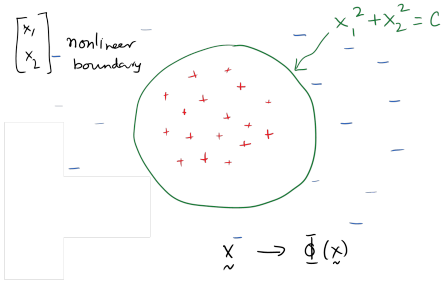

Figure 7.6: circle in 2-D.

7.2.4 Non-linear SVM

The basic SVM that we presented has a linear decision boundary. For example, in 1-dimension ($d = 1$) the decision boundary is simply a threshold. Consider Figure 7.3. Any threshold that we would select would have a high error rate. For this reason, introducing slack variables and using hinge loss, would not solve the problem. At the end, we will be using a threshold on the line, and any threshold is bad. An interesting solution is to map the points to a higher dimension, in this case 2-D. We can map each point x to (x, x^2), namely map the line to a parabola. In figure 7.4 we show this for our example. Now there is a linear separator that can separate the blues from the reds.

> In general, we can always add enough dimensions such that the points are linearly separable, but in some extreme cases this might require a number of dimensions which is proportional to the number of points. In such a case the linear separator is very likely to overfit, unless it is done carefully as in the case of a Gaussian kernel.

Another example is in figures 7.6 and 7.5. In 2D there is a circle which perfectly classifies the points. Namely, for each (x_1, x_2) the decision is based on $x_1^2 + x_2^2 \leq C$. We can get around this problem in a few ways. One solution is to move to polar coordinates, namely, map (x_1, x_2) to (r, θ), where $r = \sqrt{x_1^2 + x_2^2}$ and $\tan(\theta) = x_2/x_1$.

An alternative solution, is to map (x_1, x_2) to (x_1^2, x_2^2). This will solve our problem, but would seems a rather adhoc solution. A better solution, which could handle any

quadratic mapping, is to map (x_1, x_2) to $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$ and then we can implement any quadratic function.

We can now discuss a general mapping of x to $\phi(x)$. Let us consider the basic primal optimization.

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^t w \\ \text{s.t.} \quad & y_n(w^t \phi(x_n) + b) \geq 1 \quad \forall n \in [1, N] \end{aligned}$$

The first question is *what is the dimension of the weights w?* Note that this dimension depends on the dimension of $\phi(\cdot)$. Since we would like to handle arbitrary large dimensions, this might be a computational problem. However, the rest of the math is not influenced. If we have a very large feature vector $\phi(\cdot)$ then the computational problem is twofold, first the inner products become an expensive operation, and second we need to compute and store a very large weight vector. Let us consider the dual program.

$$\begin{aligned} \min_{\alpha} \quad & \alpha^t \underbrace{\begin{bmatrix} y_1y_1\phi(x_1)^t\phi(x_1) & y_1y_2\phi(x_1)^t\phi(x_2) & \cdots & y_1y_N\phi(x_1)^t\phi(x_N) \\ y_2y_1\phi(x_2)^t\phi(x_1) & y_2y_2\phi(x_2)^t\phi(x_2) & \cdots & y_2y_N\phi(x_2)^t\phi(x_N) \\ \vdots & \vdots & \vdots & \vdots \\ y_Ny_1\phi(x_N)^t\phi(x_1) & y_Ny_2\phi(x_N)^t\phi(x_2) & \cdots & y_Ny_N\phi(x_N)^t\phi(x_N) \end{bmatrix}}_{M} \alpha - 1^t\alpha \\ \text{s.t.} \quad & y^t\alpha = 0 \\ & 0 \leq \alpha \leq C \end{aligned}$$

The good news here is that the new features $\phi(\cdot)$ do not influence the constraint, and in the matrix M they appear only when computing an inner product of two generated by $\phi(\cdot)$. Let us consider the hypothesis h that we build. Recall that $w = \sum_{i=1}^m \alpha_i y_i \phi(x_i)$ and then

$$h(x) = w^t\phi(x) + b = \sum_{i=1}^m \alpha_i y_i \phi(x_i)^t \phi(x) + b$$

again, we need to consider only inner products of $\phi(\cdot)$.

Finally, we compute b by considering any support vector (x, y) and setting

$$b = y - \sum_{i=1}^m \alpha_i y_i \phi(x_i)^t \phi(x)$$

again, we need to consider only inner products of $\phi(\cdot)$.

This suggests that if we can compute the inner product of feature vectors $\phi(\cdot)$ efficiently, we can perform the entire mapping efficiently. This is the entire idea behind the *kernel trick*.

Rather than computing the inner product explicitly, we can compute it implicitly. For a mapping $\phi(\cdot)$ we define a kernel K such that

$$K(x', x'') = \phi(x')^t \phi(x'')$$

and then the hypothesis becomes

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

Also in the matrix M we can replace any $\phi(x_i)^t \phi(x_j)$ by $K(x_i, x_j)$.

Going back to the example of figure 7.4. We can set $\phi(x) = (x, x^2)$ and then

$$K(x, y) = \langle \phi(x), \phi(y) \rangle = \left\langle (x, x^2), (y, y^2) \right\rangle = xy + x^2 y^2$$

In this example we did not gain much by not doing the inner product explicitly using ϕ. However, in higher dimensions there will be a significant benefit.

For the quadratic polynomial kernel, we can first describe the kernel itself, and only then show that indeed it is an inner product of some ϕ. The quadratic polynomial K_2 is,

$$K_2(x, x') = (1 + x^t x')^2$$

clearly this kernel can be computed in time $O(d)$ for $x, x' \in \mathbb{R}^d$. We still need to show that there are functions ϕ such that $K(x, x') = \phi(x)^t \phi(x')$. We can compute the kernel explicitly

$$\begin{aligned}
K_2(x, x') &= (1 + x^t x')^2 = (1 + x_1 x'_1 + x_2 x'_2)^2 \\
&= 1 + x_1^2 (x'_1)^2 + x_2^2 (x'_2)^2 + 2 x_1 x_2 x'_1 x'_2 + 2 x_1 x'_1 + 2 x_2 x'_2 \\
&= \begin{pmatrix} 1 \\ \sqrt{2} x_1 \\ \sqrt{2} x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2} x'_1 \\ \sqrt{2} x'_2 \\ x'^2_1 \\ x'^2_2 \\ \sqrt{2} x'_1 x'_2 \end{pmatrix} \\
&= \phi(x)^t \phi(x')
\end{aligned}$$

In a similar why this can be extended to $x \in \mathbb{R}^d$. The main benefit is that the computation is $O(d)$ while the length of ϕ is $O(d^2)$.

There are a few classes of popular kernels. The first is the *linear kernel* which is the basic SVM. We have the degree r kernel K_r, where,

$$K_r(x, x') = (1 + x^t x')^r$$

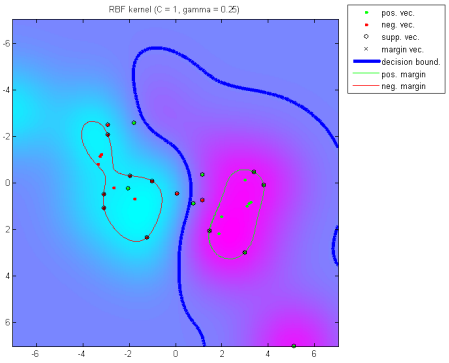

Figure 7.7: Gaussian kernel with large σ

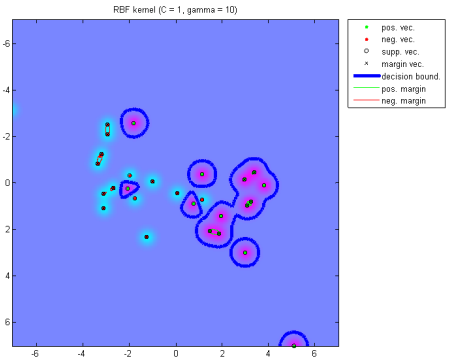

Figure 7.8: Gaussian kernel with small σ.

The degree r kernel uses a feature vector of length $O(d^r)$ and the computation is only $O(d)$. The Gaussian kernel K_g has a parameter σ and

$$K_g(x, x') = e^{-\|x-x'\|^2/(2\sigma^2)}$$

The feature vector of a Gaussian kernel is of *infinite* size. Still the computation of a Gaussian kernel is $O(d)$.

To better understand the role of a kernel we can define a kernel matrix $\overline{\overline{K}}$ where $\overline{\overline{K}}_{i,j} = \phi(x_i)^t\phi(x_j)$. Note that this definition depends on the points $x_1, \ldots x_n$ and therefore we denote the kernel by $K[x_1, \ldots, x_N]$.

With a Gaussian kernel we can select the points x_i such that $K(x_i, x_j) \leq \epsilon$ for $i \neq j$, and for sufficiently small ϵ, the matrix $\overline{\overline{K}}$ is full rank. This implies that the feature vector grows with the number of points, and hence is infinite.

> To see the connection between the rank of $\overline{\overline{K}}$ and the dimension of $\phi(x)$ observe the following. Assume that $\phi(x) \in \mathbb{R}^d$, i.e., has dimension d. Recall that $\overline{\overline{K}}[i, j] = \langle\phi(x_i), \phi(x_j)\rangle = \sum_{r=1}^{d} \phi_r(x_i)\phi_r(x_j)$, where $\phi_r(x)$ is the r-dimension of $\phi(x)$. Then we can write $\overline{\overline{K}}$ as a sum of d matrices M_r, where $M_r[i, j] = \phi_r(x_i)\phi_r(x_j)$. The rank of each matrix M_r is one, since row j is equal to the first row times $\phi_r(x_j)/\phi_r(x_1)$. This implies that the rank of $\overline{\overline{K}}$ is at most d, the dimension of $\phi(x)$. Therefore, if the dimension of $\overline{\overline{K}}$ is unbounded, then the dimension of $\phi(x)$ is unbounded.

The Gaussian kernels are very flexible. Many times they are parameterized by $\gamma = 1/(2\sigma)$ rather than σ. For small values of γ the decision line are elliptic (see figure 7.7). For large

values of γ they are essentially a nearest-neighbor since the influence falls exponentially with the distance (see figure 7.8).

7.2.5 Proper Kernels

We now define *Proper kernels* which require that

1. Symmetric: $K(x_i, x_j) = K(x_j, x_i)$.
2. Positive semi-definite kernel. For any N and any $x_1, \ldots, x_N$ we have that for any y it holds that $y^t \overline{\overline{K}} y \geq 0$, where $\overline{\overline{K}}_{i,j} = K(x_i, x_j)$.

One can slightly relax the second condition. In practice we can run the SVM even if the matrix $\overline{\overline{K}}$ is not positive semi-definite. This implies that we are not guaranteed that a local optimum is a global one, and we are not guaranteed that the algorithm would even converge. However, we can always evaluate the solution it finds.

We can compose proper kernels from existing ones. Here are two simple examples:

1. Addition $K_3(x, x') = K_1(x, x') + K_2(x, x')$. If ϕ_1 and ϕ_2 are the feature vectors of K_1 and K_2, we can create $\phi_3(x) = [\phi_1(x); \phi_2(x)]$, namely concatenating the two feature vectors.
2. Multiplication: $K_3(x, x') = K_1(x, x') \cdot K_2(x, x')$. If ϕ_1 and ϕ_2 are the feature vectors of K_1 and K_2 we can do the following:

$$\begin{aligned} K_3(x, x') &= (\phi_1(x)^t \phi_1(x'))(\phi_2(x')^t \phi_2(x)) \\ &= tr(\phi_1(x)^t \phi_1(x') \phi_2(x')^t \phi_2(x)) \\ &= tr(\phi_2(x) \phi_1(x)^t \phi_1(x') \phi_2(x')^t) \\ &= \left\langle VEC(\phi(x_2)\phi_1(x)^t), VEC(\phi_1(x')\phi_2(x')^t) \right\rangle, \end{aligned}$$

   where we used the fact that $tr(ABCD) = tr(DABC)$ and $VEC(\cdot)$ transforms a matrix to a vector.

Suppose we want to do Nearest Neighbor in feature space, how can we use kernels. Recall that

$$\|a - b\|^2 = (a - b)^t (a - b) = a^t a - 2a^t b + b^t b$$

Similarly, in feature space we have

$$\|\phi(a) - \phi(b)\|^2 = K(a, a) - 2K(a, b) + K(b, b)$$

We can create a kernel from almost any proper distance metric. One way of doing it is using the *generalized Gaussian Kernels*, where

$$K(h_1, h_2) = e^{-D^2(h_1, h_2)/\beta}$$

There are a few ways of setting the distance $D(h_1, h_2)$:

- L_1 distance: $D(h_1, h_2) = \sum_{i=1}^{d} |h_1(i) - h_2(i)|$
- L_2 distance: $D^2(h_1, h_2) = \sum_{i=1}^{d} (h_1(i) - h_2(i))^2$
- L_∞ distance: $D(h_1, h_2) = \max_{i=1}^{d} |h_1(i) - h_2(i)|$
- χ^2 distance: $D^2(h_1, h_2) = \sum_{i=1}^{d} \frac{(h_1(i)-h_2(i))^2}{h_1(i)+h_2(i)}$ which is very applicable to histograms.
- Hellinger distance: $D^2(h_1, h_2) = \sum_{i=1}^{d} (\sqrt{h_1(i)} - \sqrt{h_2(i)})^2$ which is very applicable to distributions.
- Mahalanobis distance: $D^2(h_1, h_2) = (h_1 - h_2)^t S^{-1} (h_1 - h_2)$ which allows to learn the metric, which is parametrizes by a positive definite matrix S.

7.2.6 Intersection kernels

For histograms we can apply the *intersection kernel* which is defined as follows.

$$K(h_1, h_2) = \sum_{i=1}^{d} \min(h_1(i), h_2(i))$$

when the K value is small the histograms are different and when the K is large the histograms are similar. (Note that implicitly we assume that both histograms have the same number of elements, otherwise we would need to normalize for size.)

We can show that the kernel is positive definite. We will show it for the case where the entries are integers and there is a maximum value ℓ. We can encode each entry of $h(i)$ in unary. For example, assume $\ell = 7$, then

$$\min(3, 5) =< (1, 1, 1, 0, 0, 0, 0), (1, 1, 1, 1, 1, 0, 0) >= 3$$

For the intersection kernel we can make the predictions faster, depending logarithmically on the number of support vectors s. Let $x_1 ... x_s$ be the support vectors, with coordinates $x_{j,i}$, $i = 1..d$. The decision function is

$$\begin{aligned} h(x) &= \sum_{j=1}^{s} \alpha_j \left(\sum_{i=1}^{d} \min(x_i, x_{j,i}) \right) + b \\ &= \sum_{i=1}^{d} \left(\sum_{j=1}^{s} \alpha_j \min(x_i, x_{j,i}) \right) + b \\ &= \sum_{i=1}^{d} h_i(x) + b \end{aligned}$$

where $h_i(x) = \sum_{j=1}^{s} \alpha_j \min(x_i, x_{j,i}) = \sum_{x_{j,i}<x_i} \alpha_j x_{j,i} + \sum_{x_{j,i}\geq x_i} \alpha_j x_i$.

We can precompute the following for each coordinate $i \in [1, d]$. First, sort the points $x_{j,i}$, for $j \in [1, s]$, and let the sorted values be $z_1 \leq z_2 \leq \cdots \leq z_s$. Then, for each one of the $s+1$ prefixes compute both $\beta_{1,\ell} = \sum_{j=1}^{\ell} \alpha_j z_j$ and $\beta_{2,\ell} = \sum_{j=\ell+1}^{s} \alpha_j$. When we need to compute $h_i(x_i)$, we simply test how many values $x_{j,i}$ (i.e. z_j) are smaller than x_i. We can do this in time $O(\log s)$ using binary search. Suppose that the number of smaller values is ℓ. We can use the precomputed values and return $h_i(x) = \beta_{1,\ell} + \beta_{2,\ell} x_i$.

7.3 SVD

See slides.