Lecture 6

Lecturer: Amir Globerson *Scribe: Yishay Mansour*

6.1 Kernels for SVM classifiers

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

We will cover the following topics:

- How many support vectors are there?
- Review of positive definite matrices
- Non linear classification and the kernel trick
- Characterization and construction of kernels.
- Optimization for SVM and stochastic gradient descent (details in next scribe).

6.1.1 Number of Support Vectors

When the data is linearly separable, we will expect at most $d+1$ support vectors. There are degenerate cases where data exhibits very specific dependencies and more SVs are possible, but we generally do not expect this to happen. An example of a degenerate cases is when we only have two unique points (one per class) that are each repeated $\frac{m}{2}$ times. In this case all m points will be support vectors.

Note however, you might have less than $d+1$ support vectors (See Figures 6.2 and 6.1 for an example on the plane).

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

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^d, b\in\mathbb{R}} \quad & \tfrac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geq 1 \end{aligned}$$

The support vectors satisfy the constraint as an equality. This means that $[\boldsymbol{w}, b]\cdot[\boldsymbol{x}_i, 1] = 1$ for all support vectors. This is a linear equation in $d+1$ dimensions. If we have more than $d+1$ such equations we generally do not expect a solution (unless there is some degenaracy as above).

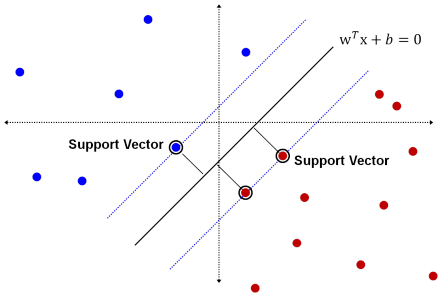

Figure 6.1: three support vectors

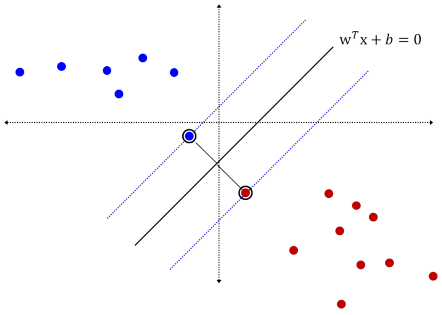

Figure 6.2: Two support vectors.

6.1.2 Positive Semi-definite (PSD) Matrices

A symmetric $d \times d$ matrix X is *positive semi-definite* (or PSD) if for any $\boldsymbol{v} \in \mathbb{R}^d$ such that $\boldsymbol{v} \neq \mathbf{0}$ we have $\boldsymbol{v}^T X \boldsymbol{v} \geq 0$. If strict inequality is required, the matrix is called positive definite (the $\boldsymbol{v} = 0$ case is excluded). We focus on PSD in what follows.

It can be shown (see slides) that a symmetric matrix X is positive definite if and only if there is a matrix B such that $X = BB^T$.

Given the above property, PSD matrices generalize the notion of a non-negative scalar, in the sense that they have a square root.

There are non-symmetric matrices, that can have the property that for any $\boldsymbol{v} \neq 0$ we have $\boldsymbol{v}^T X \boldsymbol{v} > 0$, for example $X = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$. One can verify that $\boldsymbol{v}^T X \boldsymbol{v} = v_1^2 + v_2^2 > 0$. The test for a non-symmetric matrix X whether for any $\boldsymbol{v}$ we have $\boldsymbol{v}^T X \boldsymbol{v} > 0$ is by testing if $X + X^T$ is PSD. Note that $X + X^T$ is symmetric. Also, for any $\boldsymbol{v}$, we have $\boldsymbol{v}^T X \boldsymbol{v} == \boldsymbol{v}^T X^T \boldsymbol{v}$, since it is a scalar. Therefore, $2\boldsymbol{v}^T X \boldsymbol{v} = \boldsymbol{v}^T (X + X^T) \boldsymbol{v}$, and X has the property iff $X + X^T$ is PSD.

It can also be shown that a matrix is PSD if and only if all its eigenvalues are non-negative (see slides).

6.1.3 The dual SVM formulation and PSD

Recall that the dual formulation of SVM is (when written as a minimization problem):

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T \underbrace{\begin{bmatrix} y_1 y_1 \boldsymbol{x}_1 \cdot \boldsymbol{x}_1 & y_1 y_2 \boldsymbol{x}_1 \cdot \boldsymbol{x}_2 & \cdots & y_1 y_N \boldsymbol{x}_1 \cdot \boldsymbol{x}_N \\ y_2 y_1 \boldsymbol{x}_2 \cdot \boldsymbol{x}_1 & y_2 y_2 \boldsymbol{x}_2 \cdot \boldsymbol{x}_2 & \cdots & y_2 y_N \boldsymbol{x}_2 \cdot \boldsymbol{x}_N \\ \vdots & \vdots & \vdots & \vdots \\ y_N y_1 \boldsymbol{x}_1 \cdot \boldsymbol{x}_N & y_N y_2 \boldsymbol{x}_N \cdot \boldsymbol{x}_2 & \cdots & y_N y_N \boldsymbol{x}_N \cdot \boldsymbol{x}_N \end{bmatrix}}_{M} \boldsymbol{\alpha} - \mathbf{1}^T \boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{y}^T \boldsymbol{\alpha} = 0 \\ & 0 \leq \boldsymbol{\alpha} \leq C \end{aligned} \tag{6.1}$$

We can write the matrix M as PP^T where $P^T = [y_1 \boldsymbol{x}_1 \cdots y_m \boldsymbol{x}_m]$. For this reason the matrix M is positive semi-definite. This guarantees that the optimization is a convex quadratic program and can therefore be globally optimized efficiently.

6.1.4 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. There are many cases when such a classifier isn't good enough. For example, consider Figure 6.3. 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. Any classifier we'll end up choosing 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 6.4 we show this for our example. Now there is a linear separator that can separate the data.

> 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 RBF kernel. In terms of the generalization bounds we discussed, we are saying that an arbitrary mapping into n dimensional space will separate, but with a small margin, and therefore bad generalization error.

Another example is in figures 6.6 and 6.5. 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 $\tan(\theta) = x_2/x_1$.

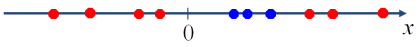

Figure 6.3: An interval of blue between two red areas

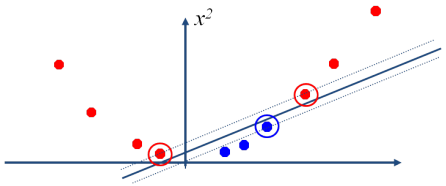

Figure 6.4: Lifting the interval to 2-D.

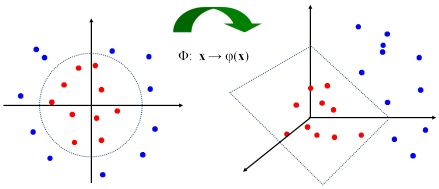

Figure 6.5: mapping from a circle to 3D

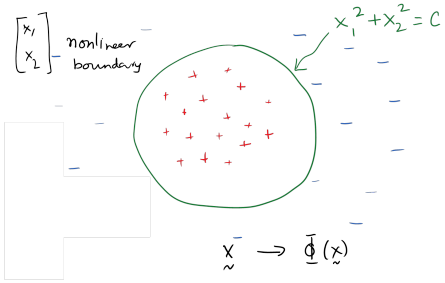

Figure 6.6: circle in 2-D.

An alternative solution, is to map (x_1, x_2) to (x_1^2, x_2^2). This will solve our problem, but seems like a rather ad hoc 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 $\boldsymbol{x}$ to $\boldsymbol{\phi}(\boldsymbol{x})$. Let us consider the basic primal optimization.

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 \quad i = 1, \ldots, m \end{aligned}$$

The first question is *what is the dimension of the weights w?* Note that this dimension depends on the dimension of $\boldsymbol{\phi}(\boldsymbol{x})$. Since we would like to handle arbitrary large dimensions, this might be a computational problem. However, the rest of the math remains the same. If we have a very high dimensional feature vector $\boldsymbol{\phi}(\boldsymbol{x})$ 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 in Equation 6.1. The good news here is that the new features $\boldsymbol{\phi}(\boldsymbol{x})$ do not influence the constraint, and in the matrix M they appear only when computing an inner product of two generated by $\boldsymbol{\phi}(\boldsymbol{x})$. Let us consider the hypothesis h

that we build. Recall that $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\phi}(\boldsymbol{x}_i)$ and then

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

again, we need to consider only inner products of $\boldsymbol{\phi}(\boldsymbol{x})$.

Finally, we compute b by considering any support vector $(\boldsymbol{x}, y)$ and setting

$$b = y - \sum_{i=1}^{m} \alpha_i y_i \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x})$$

again, we need to consider only inner products of $\boldsymbol{\phi}(\boldsymbol{x})$.

This suggests that if we can compute the inner product of feature vectors $\boldsymbol{\phi}(\boldsymbol{x})$ efficiently, we can perform the entire mapping efficiently. This is precisely the idea behind the *kernel trick*. Rather than computing the inner product explicitly, we can compute it implicitly. For a mapping $\boldsymbol{\phi}(\boldsymbol{x})$ we define a kernel K such that

$$K(\boldsymbol{x}', \boldsymbol{x}'') = \boldsymbol{\phi}(\boldsymbol{x}')^T \boldsymbol{\phi}(\boldsymbol{x}'')$$

and then the hypothesis becomes

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

Also in the matrix M we can replace any $\boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(x_j)$ by $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Going back to the example of figure 6.4. We can set $\boldsymbol{\phi}(\boldsymbol{x}) = (x, x^2)$ and then

$$K(x, z) = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(z) = xz + x^2 z^2$$

In this example we did not gain much by not doing the inner product explicitly using $\boldsymbol{\phi}$. 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 $\boldsymbol{\phi}$. The quadratic polynomial K_2 is,

$$K_2(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x} \cdot \boldsymbol{x}')^2$$

clearly this kernel can be computed in time $O(d)$ for $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$. We still need to show that there are functions $\boldsymbol{\phi}$ such that $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}')$. We can compute the kernel

explicitly. Indeed for $d = 2$ we have:

$$\begin{aligned} K_2(\boldsymbol{x}, \boldsymbol{x}') &= (1 + \boldsymbol{x} \cdot \boldsymbol{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2x_1 x_2 x_1' x_2' + 2x_1 x_1' + 2x_2 x_2' \\ &= \begin{pmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ x_1^2 \\ x_2^2 \\ \sqrt{2}x_1 x_2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}x_1' \\ \sqrt{2}x_2' \\ x_1'^2 \\ x_2'^2 \\ \sqrt{2}x_1' x_2' \end{pmatrix} \\ &= \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}') \end{aligned}$$

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

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

$$K_r(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x} \cdot \boldsymbol{x}')^r$$

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

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

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

To better understand the role of a kernel we can define a kernel matrix K_S where $[K_S]_{i,j} = \boldsymbol{\phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\phi}(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Note that the matrix depends on the points $\boldsymbol{x}_1, \ldots \boldsymbol{x}_n$.

With the RBF kernel we can select the points $\boldsymbol{x}_i$ such that $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \epsilon$ for $i \neq j$, and for sufficiently small ϵ, the matrix K_S is full rank. This implies that the feature vector grows with the number of points, and hence is infinite.

The RBF Radial Basis Function (RBF) kernels are very flexible. They are often parameterized by $\gamma = 1/(2\sigma)$ rather than σ. For large values of γ they are essentially a nearest-neighbor computation since the influence falls exponentially with the distance (see Figures 6.8 and 6.7).

6.1.5 Characterizing Kernels

How do we find a function $K(\boldsymbol{x}, \boldsymbol{x}')$ that corresponds to a dot product in *some* feature space $\boldsymbol{\phi}(\boldsymbol{x})$? To formalize the result, we first define *Positive Definite Kernels*. These are functions $K(\boldsymbol{x}, \boldsymbol{x}')$ that satisfy:

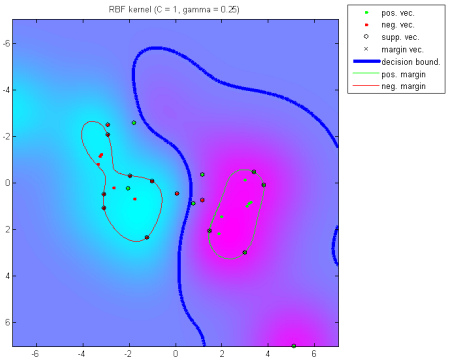

Figure 6.7: RBF kernel with large σ

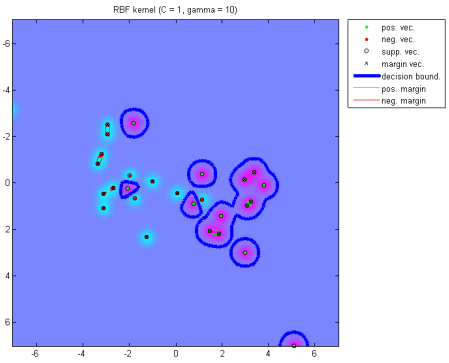

Figure 6.8: RBF kernel with small σ.

1. Symmetry: $K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}, \boldsymbol{x}')$.

2. For any sample $S = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m]$ the matrix K_S is PSD.

It can be shown that a kernel is a dot product in feature space if and only if it is Positive Definite Kernel. This result is known as Mercer's theorem (Mercer's result is typically stated for an equivalent notion of positive definiteness).

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

1. Addition $K_3(\boldsymbol{x}, \boldsymbol{x}') = K_1(\boldsymbol{x}, \boldsymbol{x}') + K_2(\boldsymbol{x}, \boldsymbol{x}')$. If $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are the feature vectors of K_1 and K_2, we can create $\boldsymbol{\phi}_3(\boldsymbol{x}) = [\boldsymbol{\phi}_1(\boldsymbol{x}); \boldsymbol{\phi}_2(\boldsymbol{x})]$, namely concatenating the two feature vectors.

2. Multiplication: $K_3(\boldsymbol{x}, \boldsymbol{x}') = K_1(\boldsymbol{x}, \boldsymbol{x}') \cdot K_2(\boldsymbol{x}, \boldsymbol{x}')$. If $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are the feature vectors of K_1 and K_2 we can do the following:

$$\begin{aligned} K_3(\boldsymbol{x}, \boldsymbol{x}') &= (\boldsymbol{\phi}_1(\boldsymbol{x})^T \boldsymbol{\phi}_1(x'))(\boldsymbol{\phi}_2(x')^T \boldsymbol{\phi}_2(\boldsymbol{x})) \\ &= tr(\boldsymbol{\phi}_1(\boldsymbol{x})^T \boldsymbol{\phi}_1(x') \boldsymbol{\phi}_2(x')^T \boldsymbol{\phi}_2(\boldsymbol{x})) \\ &= tr(\boldsymbol{\phi}_2(\boldsymbol{x}) \boldsymbol{\phi}_1(\boldsymbol{x})^T \boldsymbol{\phi}_1(x') \boldsymbol{\phi}_2(x')^T) \\ &= \left\langle VEC(\boldsymbol{\phi}(x_2) \boldsymbol{\phi}_1(\boldsymbol{x})^T), VEC(\boldsymbol{\phi}_1(x') \boldsymbol{\phi}_2(x')^T) \right\rangle, \end{aligned}$$

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

$$\|a-b\|^2 = (a-b)^T(a-b) = a^Ta - 2a^Tb + b^Tb$$

Similarly, in feature space we have

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

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