CS229 Note Lecture 08

$$\begin{aligned} \min \ & \frac{1}{2}\|w\|^2 \\ \text{s.t. } & y^{(i)}(w^T x^{(i)} + b) \geq 1 \end{aligned}$$

All this is assuming that the data is linearly separable, which is an assumption that we'll fix later.

Dual problem:

$$\begin{aligned} \max \ & \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t. } & \alpha_i \geq 0 \\ & \sum_i \alpha_i y^{(i)} = 0 \end{aligned}$$

We also worked out

$$w = \sum_i \alpha_i y^{(i)} x^{(i)}$$

Therefore, when we need to make a prediction of classification time, we need compute

$$\begin{aligned} h_{w,b}(x) &= g(w^T x + b) \\ &= g\Big(\sum_i \alpha_i y^{(i)} \langle x^{(i)}, x^{(j)} \rangle + b \Big) \end{aligned}$$

Kernels

Suppose we have $x \in \mathbb{R}$, we'll map it to a richer set of features, for example

$$x \to \begin{bmatrix} x \\ x^2 \\ x^3 \\ x^4 \end{bmatrix} = \phi(x)$$

we call this mapping $\phi(x)$, so we'll let $\phi(x)$ denote the mapping from original features to some high-dimensional features. If we want to use $\phi(x)$, what we should do is go back to our

algorithm and everywhere we see $\langle x^{(i)}, x^{(j)} \rangle$, we'll replace the inner product with $\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$.

Sometimes $\phi(x)$ will be very high-dimensional(even infinite dimensional vector), but we can compute a kernel between $x^{(i)}, x^{(j)}$. Define **Kernel Function**

$$K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$$

so the idea of the SVM is that we're going to replace inner product with kernel function, and that let the algorithm works in feature spaces $\phi(x)$, even if $\phi(x)$ are very high dimensional.

Let's say we have two inputs $x, z \in \mathbb{R}^n$, so the kernel is

$$\begin{aligned} K(x, z) &= (x^T z)^2 \\ &= \Big(\sum_{i=1}^{n} x_i z_i \Big) \Big(\sum_{j=1}^{n} x_j z_j \Big) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

so this kernel corresponds to the feature mapping(suppose $n = 2$)

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix} \quad \phi(z) = \begin{bmatrix} z_1 z_1 \\ z_1 z_2 \\ z_2 z_1 \\ z_2 z_2 \end{bmatrix}$$

so we get

$$K(x, z) = (\phi(x))^T (\phi(z))$$

In order to computer $\phi(x)$, we need $O(n^2)$ to compute $\phi(x)$ where $n = dim(x) = dim(z)$. But we only need $O(n)$ to compute $K(x, z)$

Generalization

Linear Kernel

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

This corresponds to the feature mapping(again shown $n = 2$)

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_2x_1 \\ x_2x_2 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ c \end{bmatrix} \quad \phi(z) = \begin{bmatrix} z_1z_1 \\ z_1z_2 \\ z_2z_1 \\ z_2z_2 \\ \sqrt{2c}z_1 \\ \sqrt{2c}z_2 \\ c \end{bmatrix}$$

This takes only $O(n)$ time.

Polynomial Kernel

$$K(x, z) = (x^T z + c)^d$$

This corresponds

$$\begin{pmatrix} h+d \\ d \end{pmatrix} \text{feature of all monomial up to degree } d$$

This still takes only $O(n)$ time.

RBF(Gaussian Radio Basis Function) Kernel

$$K(x, z) = \exp\left(- \frac{||x - z||^2}{2\sigma^2} \right)$$

Is this a valid kernel? Consider our definition of kernel, the question becomes

$$\exists \phi \text{ s.t. } K(x, z) = \langle \phi(x), \phi(z) \rangle$$

Suppose K is a valid kernel, then let any set of points $\{x_1, \dots, x_m\}$ be given, define a matrix $K \in \mathbb{R}^{m \times m}$ such that

$$K_{ij} = K(x^{(i)}, x^{(j)})$$

It turns out that for any vector $z \in \mathbb{R}^m$, consider $z^T K z$, by definition of matrix multiplication

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

$K \geq 0$ when the matrix K is symmetric **Positive-semidefinite Matrix**.

To summarize, if K is a valid kernel, then the kernel matrix must be positive-semidefinite matrix. It turns out that the converse holds true and so this gives us a test for whether a function K is a valid kernel.

Theorem(Mercer): Let $K(x, z)$ be given, then K is a valid kernel(i.e., $\exists \phi$ s.t. $K(x, z) = \phi(x)^T \phi(z)$) if and only $\forall \{x^{(1)}, \dots, x^{(m)}\} (m < \infty)$, the kernel matrix $K \in \mathbb{R}^{m \times m}$ is symmetric positive-semidefinite.

A concrete example of something is not a valid kernel would be if we find a input x such that $K(x, x) = -1 \neq \phi(x)^T \phi(x)$.

How To Create Kernel

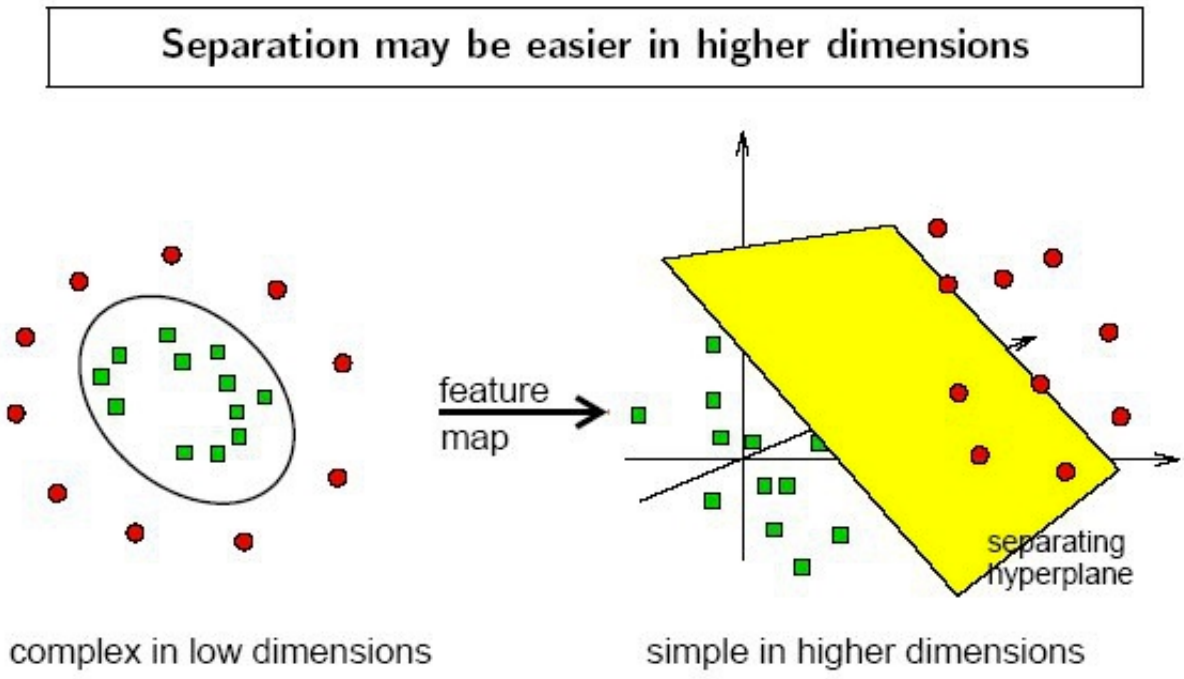
Given a set of x and z, we want $x \mapsto \phi(x)$ and $z \mapsto \phi(z)$, so the kernel is computing inner product
$\langle \phi(x), \phi(z) \rangle$. If x and z are very similar, then $\phi(x)$ and $\phi(z)$ will be pointing in the same direction, and therefore the inner product would be large. Whereas in contrast, if x and z are very dissimilar, then $\phi(x)$ and $\phi(z)$ maybe pointing different directions, and so the inner product may be small.

The way to find a kernel is try to come up with a function
$K(x, z)$ that is large if we want to learn the algorithm to think of x and z as similar and small if x and z are dissimilar.

Using a SVM with a kernel, what we gonna do is choose some function $K(x, z)$ such as $K(x, z) = \exp(-\|x - z\|^2 / 2\sigma^2)$, and replace every $\langle x^{(i)}, x^{(j)} \rangle$ with $K(x^{(i)}, x^{(j)})$.

Why Kernel Is A Good Idea

If we have 1-dimensional data, what a kernel does is takes our original input data and maps it to a very high-dimensional feature space.



Soft Margin

L1 Norm Soft Margin SVM

Change the primal problem as

$$\begin{aligned} \min_{w,b,\xi} \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t. } & y^{(i)}(w^T x + b) \geq 1 - \xi_i \\ & \xi \geq 0 \end{aligned}$$

Lagrangian

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

Do same math process, we'll get

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

$$\text{s.t.} \ \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

$$0 \leq \alpha_i \leq C, \ i = 1, \dots, m$$

We can actually derive conversions conditions by use KKT condition

$$\begin{aligned} \alpha_i = 0 \ &\Rightarrow \ y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \ &\Rightarrow \ y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C \ &\Rightarrow \ y^{(i)}(w^T x^{(i)} + b) = 1 \end{aligned}$$

SMO(Sequential Minimal Optimization) Algorithm

Coordinate Assent Algorithm

Consider the problem

$$\max W(\alpha_1, \dots, \alpha_m) \quad \text{(no constraints on } \alpha_i)$$

This is the coordinate assent algorithm

Repeat {

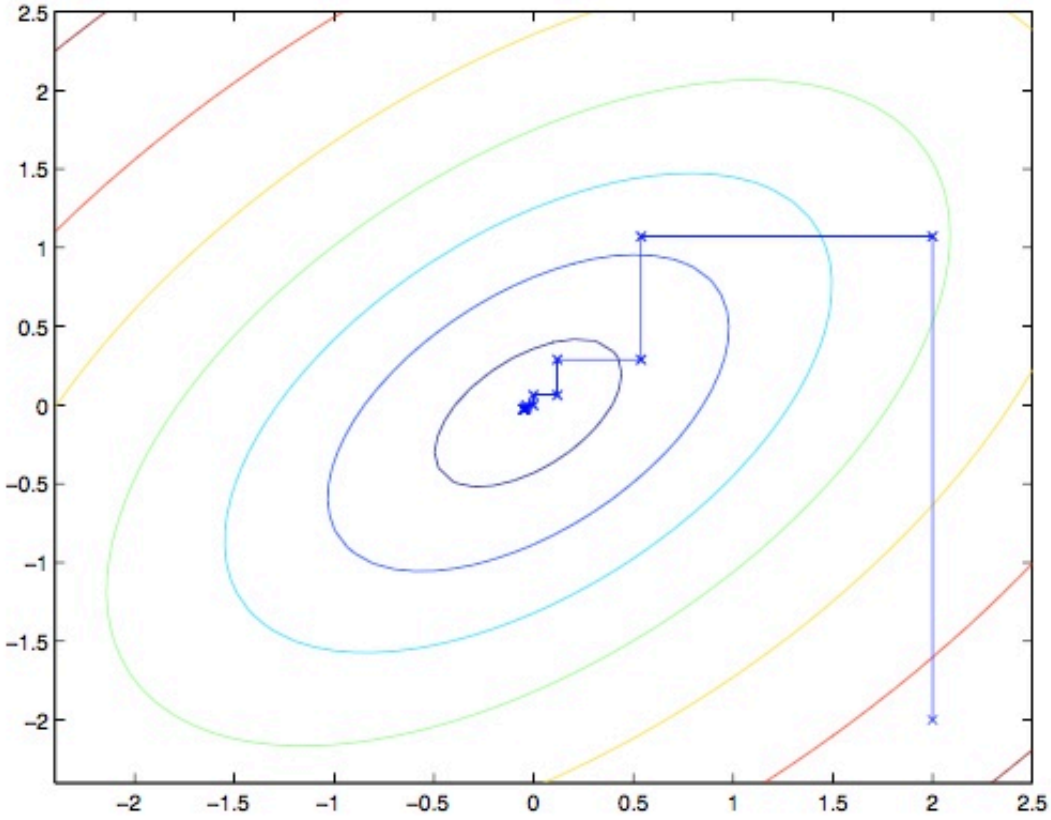
For $i = 1$ to m {

$$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i=1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$$

}

}

hold everything except α_i. Here is a picture of coordinate assent in action



It turns out that coordinate assent in its basic form does not work because we have constraints on the α_i's.

In SMO algorithm, we'll try to change two α's at a time.

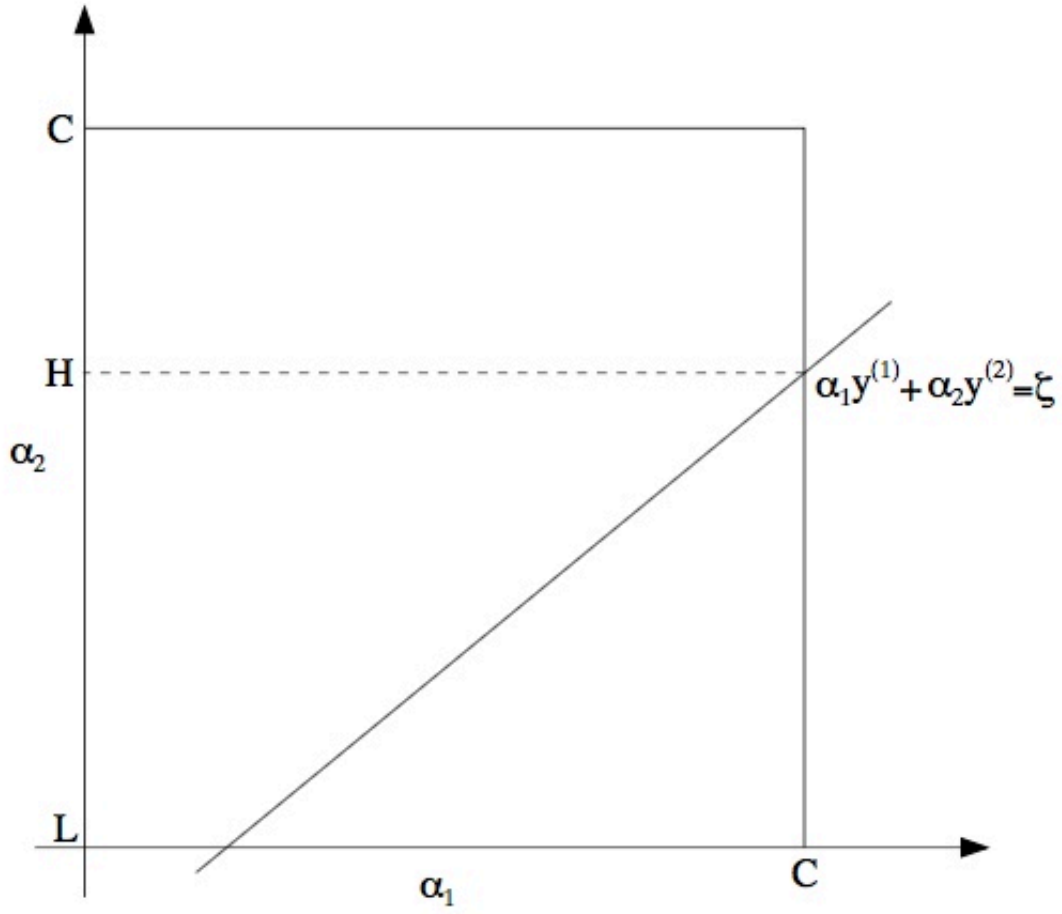
SMO algorithm:

1. Select some pair α_i and α_j to update next(using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Holds all α's fixed except α_i and α_j.
3. Optimize $W(\alpha)$ with respect α_i and α_j s.t. all constraints.

Suppose we've decide to hold $\alpha_3, \ldots, \alpha_m$ fixed and want to optimize $W(\alpha)$ with respect to α_1 and α_2. One every step of the algorithm with respect to constraint $\sum_i \alpha_i y^{(i)} = 0$, we get

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=1}^{m} \alpha_i y^{(i)} = \xi \quad \text{s.t. } 0 \leq \alpha_i \leq C$$

This constraint also called **the Box Constraint**, we can picture the constraints on α_1 and α_2 as follow:



It implies that

$$\alpha_1 = \frac{\xi - \alpha_2 y^{(2)}}{y^{(1)}}$$

then the $W(\alpha)$ can be written as

$$\begin{aligned} W(\alpha) &= W\left(\frac{\xi - \alpha_2 y^{(2)}}{y^{(1)}}, \alpha_2, \dots, \alpha_m\right) \\ &= a\alpha_2^2 + b\alpha_2 + c \end{aligned}$$

There'll be some sort of quadratic function over the line, and so if we minimize the quadratic function, maybe we get a value that lies in the box. Or we may end up with a value outside, we should clip our solution just to map it back inside the box if that happen.

And then we can solved α_1.

A Couple of Examples of Application of SVM

Handler's Integer Recognition

We can apply an SVM with the polynomial kernel $K(x, y) = (x^T y)^d$ or Gaussian kernel

$K(x, y) = \exp(-\|x - y\|^2/2\sigma^2)$ works well.

Classify Esoteric Objects

Suppose we want classify protein sequences $BAJTSIAIBAJTAU$ into different classes depending on what type of protein it is. How could we construct our feature vector $\phi(x)$?

We are going to write down all possible combinations of four alphabets $AAAA, \dots, AAAZ, \dots, UUUU$ as a vector, and scan through the sequence of amino acids and count how often each of these subsequence occur. For example, $BAJT$ occurs twice, so the feature vector is

$$\phi(x) = \begin{matrix} AAAA \\ AAAB \\ \vdots \\ BAJT \\ \vdots \\ UUUU \end{matrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 2 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{(20^4)} = \mathbb{R}^{160000}$$

This representation applies no matter how long our protein sequence is.

Now we have a 160000-dimensional feature vector, which is reasonably large even by modern computer standards. Clearly we don't want to explicitly represent these high dimensional feature vectors. It turns out that there is an efficient dynamic programming algorithm that can efficiently compute inner products.