

ECE 5424: Introduction to Machine Learning

Topics:

- SVM
 - SVM dual & kernels

Readings: Barber 17.5

Stefan Lee
Virginia Tech

Lagrangian Duality

- On paper

Dual SVM derivation (1) – the linearly separable case

$$\begin{aligned} &\text{minimize}_{\mathbf{w}, b} \quad \frac{1}{2} \mathbf{w} \cdot \mathbf{w} \\ &\left(\mathbf{w} \cdot \mathbf{x}_j + b \right) y_j \geq 1, \quad \forall j \end{aligned}$$

Dual SVM derivation (1) – the linearly separable case

$$L(\mathbf{w}, \alpha) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \sum_j \alpha_j \left[(\mathbf{w} \cdot \mathbf{x}_j + b) y_j - 1 \right]$$
$$\alpha_j \geq 0, \quad \forall j$$

$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Dual SVM formulation – the linearly separable case

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k$$

for any k where $\alpha_k > 0$

Dual SVM formulation – the non-separable case

$$\begin{aligned} \text{minimize}_{\mathbf{w}, b} \quad & \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_j \xi_j \\ \left(\mathbf{w} \cdot \mathbf{x}_j + b \right) y_j \geq & 1 - \xi_j, \quad \forall j \\ \xi_j \geq & 0, \quad \forall j \end{aligned}$$

Dual SVM formulation – the non-separable case

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

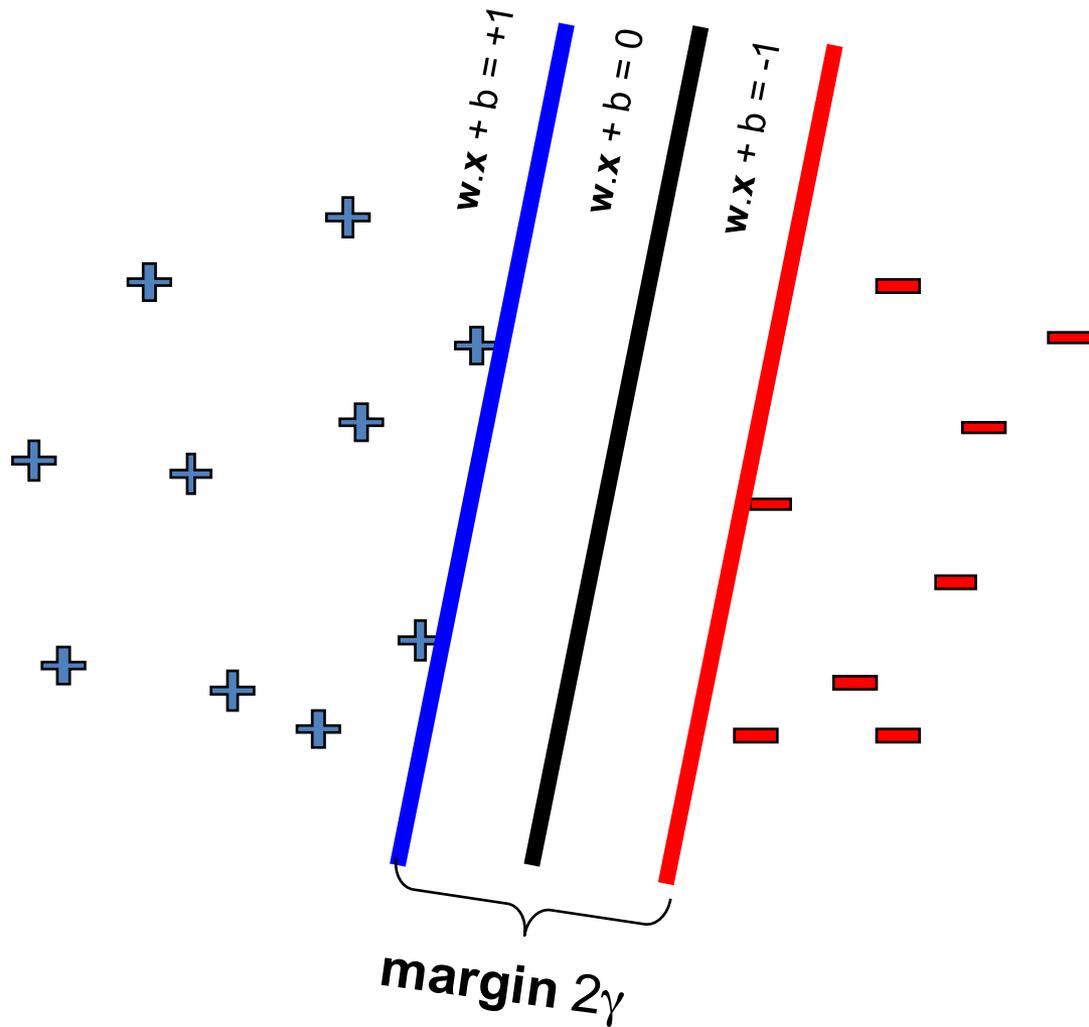
$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

Why did we learn about the dual SVM?

- Builds character!
- Exposes structure about the problem
- There are some quadratic programming algorithms that can solve the dual faster than the primal
- The “**kernel trick**”!!!

Dual SVM interpretation: Sparsity



$$w = \sum_j \alpha_j y_j x_j$$

Dual formulation only depends on dot-products, not on \mathbf{w} !

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

Dot-product of polynomials

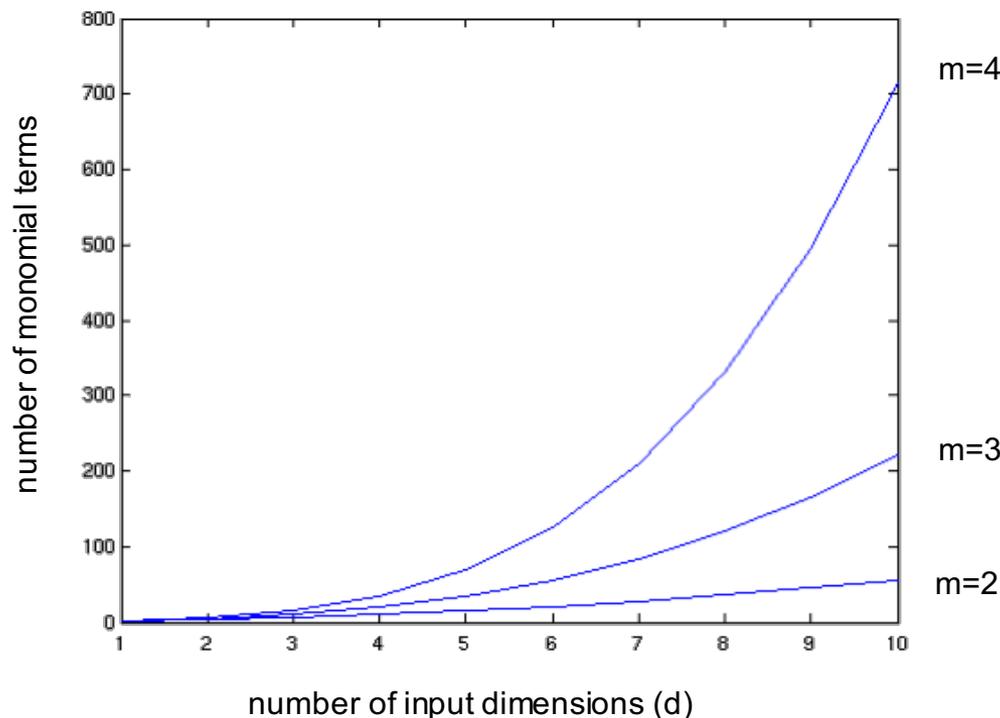
$\Phi(\mathbf{u}) =$ Vector of Monomials of degree m

Higher order polynomials

$$\# \text{terms} = D = \binom{m + d - 1}{m} = \frac{(m + d - 1)!}{m!(d - 1)!}$$

d – input features

m – degree of polynomial



grows fast!

$$m = 6, d = 100$$

$D = \text{about } 1.6 \text{ billion terms}$

Common kernels

- Polynomials of degree d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian kernel / Radial Basis Function

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

Kernel Demo

- Demo
 - <http://www.eee.metu.edu.tr/~alatan/Courses/Demo/AppletSV M.html>

What is a kernel?

- $k: X \times X \rightarrow \mathbb{R}$
- Any measure of “similarity” between two inputs
- Mercer Kernel / Positive Semi-Definite Kernel
 - Often just called “kernel”

How to Check if a Function is a Kernel

Problem:

- Checking if a given $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ fulfills the conditions for a kernel is *difficult*:
- We need to prove or disprove

$$\sum_{i,j=1}^n t_i k(x_i, x_j) t_j \geq 0.$$

for *any set* $x_1, \dots, x_n \in \mathcal{X}$ and *any* $t \in \mathbb{R}^n$ for *any* $n \in \mathbb{N}$.

Workaround:

- It is easy to *construct* functions k that are positive definite kernels.

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \rightarrow \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \rightarrow \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a *distance function*, i.e.
 - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') = 0$ only for $x = x'$,
 - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \rightarrow \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a *distance function*, i.e.
 - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') = 0$ only for $x = x'$,
 - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
 - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Finally: the “kernel trick”!

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

- Never represent features explicitly
 - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features
- Very interesting theory – Reproducing Kernel Hilbert Spaces

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w} \cdot \mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

Kernels in Computer Vision

- Features x = histogram (of color, texture, etc)
- Common Kernels
 - Intersection Kernel
 - Chi-square Kernel

$$K_{\text{intersect}}(\mathbf{u}, \mathbf{v}) = \sum_i \min(u_i, v_i)$$

$$K_{\chi^2}(\mathbf{u}, \mathbf{v}) = \sum_i \frac{2u_i v_i}{u_i + v_i}$$

What about at classification time

- For a new input \mathbf{x} , if we need to represent $\Phi(\mathbf{x})$, we are in trouble!
- Recall classifier: $\text{sign}(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)$
- Using kernels we are fine!

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

Kernels in logistic regression

$$P(Y = 1 | x, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)}}$$

- Define weights in terms of support vectors:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i)$$

$$\begin{aligned} P(Y = 1 | x, \mathbf{w}) &= \frac{1}{1 + e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b)}} \\ &= \frac{1}{1 + e^{-(\sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b)}} \end{aligned}$$

- Derive simple gradient descent rule on α_i

Kernels

- Kernel Logistic Regression
- Kernel Least Squares
- Kernel PCA ...

