Support Vector Machines & Kernels
Lecture 5

David Sontag
New York University

Slides adapted from Luke Zettlemoyer and Carlos Guestrin

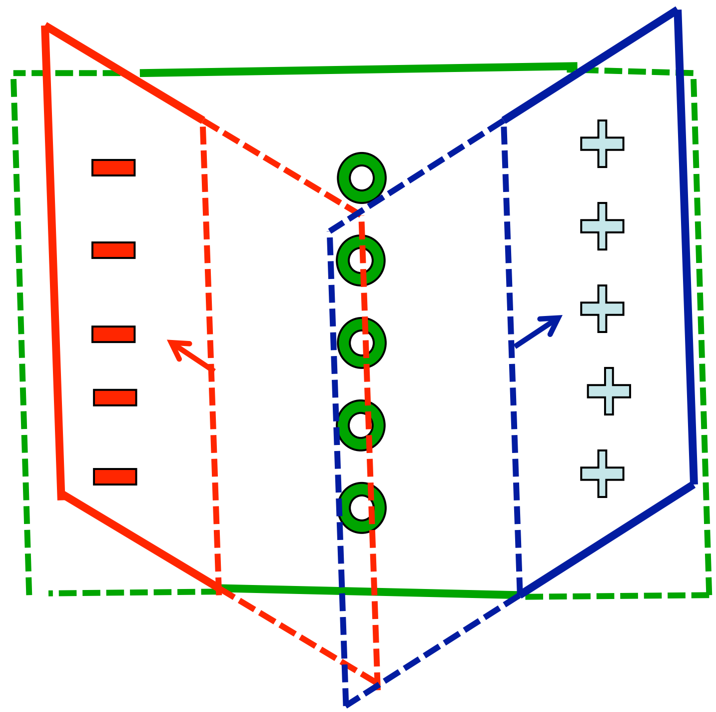
Multi-class SVM

As for the SVM, we introduce slack variables and maximize margin:

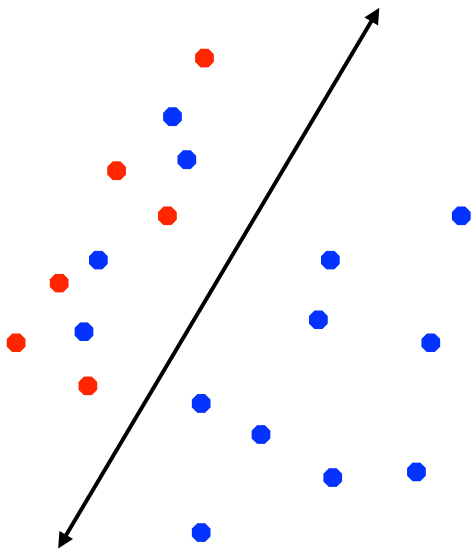
$$\text{minimize}_{\mathbf{w},b} \quad \textstyle\sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C\sum_j \xi_j$$
$$\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1 - \xi_j, \ \forall y' \neq y_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

To predict, we use:
$$\hat{y} \leftarrow \arg\max_k \ w_k \cdot x + b_k$$

Now can we learn it? →



How to deal with imbalanced data?



- In many practical applications we may have **imbalanced** data sets
- We may want errors to be equally distributed between the positive and negative classes
- A slight modification to the SVM objective does the trick!

$$\min_{w,b} \ \frac{1}{2}||w||_2^2 + \frac{C}{N_+}\sum_{j\,:\,y_j=+1}\xi_j + \frac{C}{N_-}\sum_{j\,:\,y_j=-1}\xi_j$$

Class-specific weighting of the slack variables

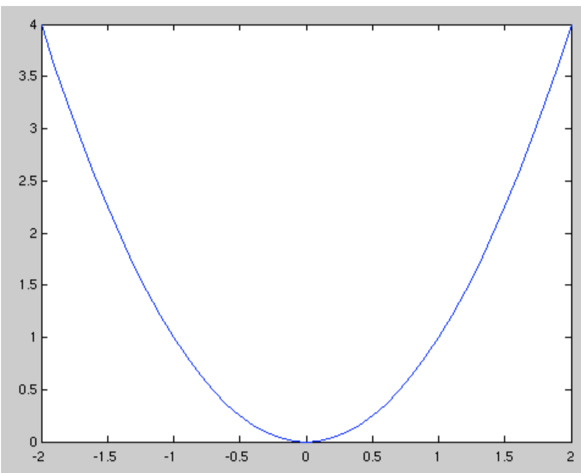
What's Next!

- Learn one of the most interesting and exciting recent advancements in machine learning
  - The "kernel trick"
  - High dimensional feature spaces at no extra cost!
- But first, a detour
  - Constrained optimization!

Constrained optimization

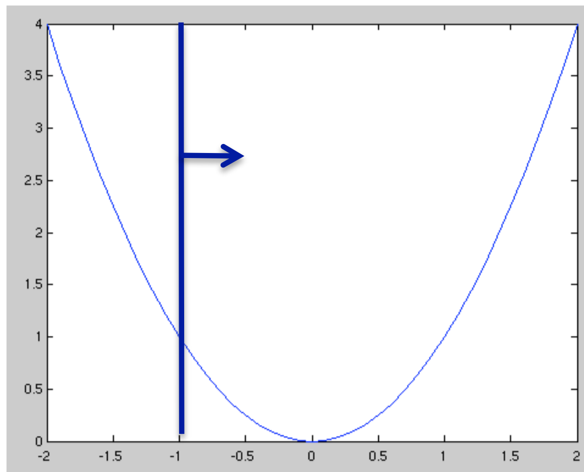
$$\boxed{\begin{array}{ll}\text{min}_x & x^2 \\ \text{s.t.} & x \geq b\end{array}}$$

No Constraint



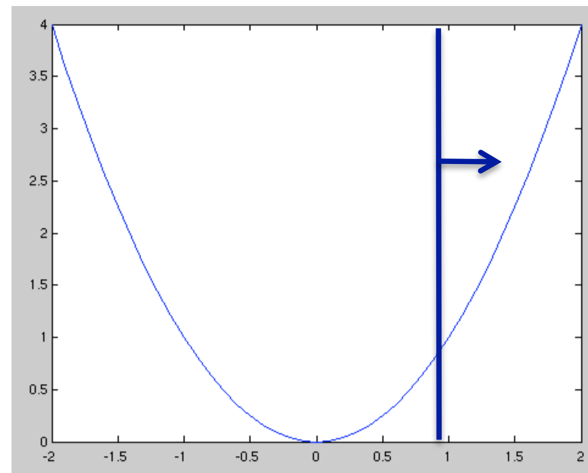
$x^*=0$

x ≥ -1



$x^*=0$

x ≥ 1

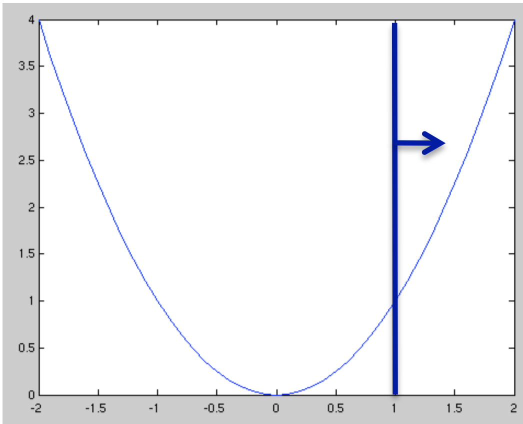


$x^*=1$

How do we solve with constraints?

→ Lagrange Multipliers!!!

Lagrange multipliers – Dual variables



$$\min_x \;\; x^2$$
$$\text{s.t.} \quad x \geq b$$

Add Lagrange multiplier

Rewrite Constraint

Introduce Lagrangian (objective):

$$L(x,\alpha) = x^2 - \alpha(x-b)$$

We will solve:

$$\min_x \max_\alpha \;\; L(x,\alpha)$$
$$\text{s.t.} \quad \alpha \geq 0$$

Add new constraint

Why is this equivalent?

- min is fighting max!

$x<b \rightarrow (x-b)<0 \rightarrow \max_\alpha -\alpha(x-b) = \infty$

- min won't let this happen!

$x>b,\ \alpha \geq 0 \rightarrow (x-b)>0 \rightarrow \max_\alpha -\alpha(x-b) = 0,\ \alpha^*=0$

- min is cool with 0, and $L(x, \alpha)=x^2$ (original objective)

$x=b \rightarrow \alpha$ can be anything, and $L(x, \alpha)=x^2$ (original objective)

The *min* on the outside forces *max* to behave, so constraints will be satisfied.

Dual SVM derivation (1) – the linearly separable case

Original optimization problem:

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

Rewrite constraints

One Lagrange multiplier per example

Lagrangian:

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

Our goal now is to solve: $\min_{\vec{w},b} \ \max_{\vec{\alpha}\geq 0} \ L(\vec{w}, \vec{\alpha})$

Dual SVM derivation (2) – the linearly separable case

(Primal) $$\min_{\vec{w},b} \max_{\vec{\alpha}\geq 0} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w} \cdot \vec{x}_j + b)\, y_j - 1 \right]$$

Swap min and max

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w} \cdot \vec{x}_j + b)\, y_j - 1 \right]$$

Slater's condition from convex optimization guarantees that these two optimization problems are equivalent!

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha}\geq 0,\, \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} \underbrace{y_i y_j \alpha_i \alpha_j}_{\text{scalars}} \underbrace{(\vec{x}_i \cdot \vec{x}_j)}_{\text{dot product}}$$

Sums over all training examples

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1 \right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha}\geq 0,\ \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j\right)$$

So, in dual formulation we will solve for α directly!

- **w** and b are computed from α (if needed)

Dual SVM derivation (3) – the linearly separable case

Lagrangian:

$$L(\mathbf{w}, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

$$\alpha_j \geq 0, \ \ \forall j$$

$\alpha_j > 0$ for some *j* implies constraint is tight. We use this to obtain *b*:

$$y_j \left(\vec{w} \cdot \vec{x}_j + b\right) = 1 \quad (1)$$

$$y_j y_j \left(\vec{w} \cdot \vec{x}_j + b\right) = y_j \quad (2)$$

$$\left(\vec{w} \cdot \vec{x}_j + b\right) = y_j \quad (3)$$

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

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

for any k where $\alpha_k > 0$

Dual for the non-separable case – same basic story (we will skip details)

Primal:

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

Solve for w,b, α:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

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

Dual:

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

What changed?

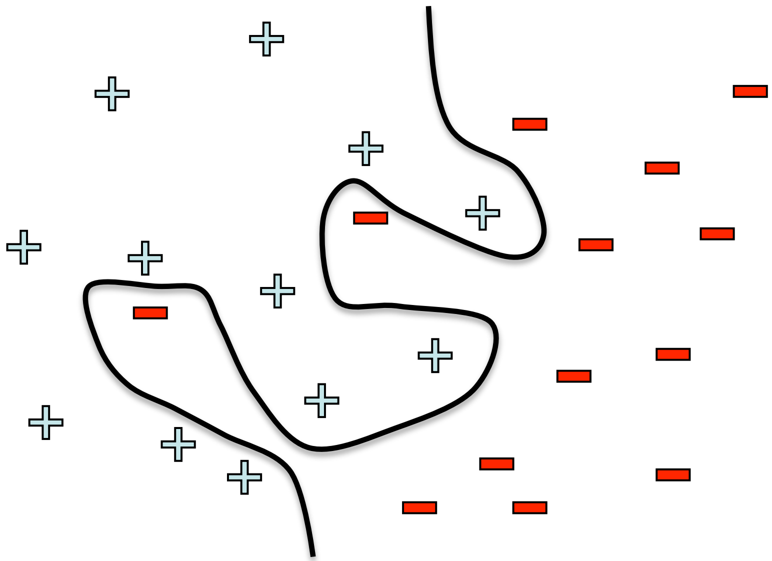
- Added upper bound of C on α_i!
- Intuitive explanation:
  - Without slack, $\alpha_i \rightarrow \infty$ when constraints are violated (points misclassified)
  - Upper bound of C limits the α_i, so misclassifications are allowed

Wait a minute: why did we learn about the dual SVM?

- There are some quadratic programming algorithms that can solve the dual faster than the primal
  - At least for small datasets

- But, more importantly, the "**kernel trick**"!!!

Reminder: What if the data is not linearly separable?



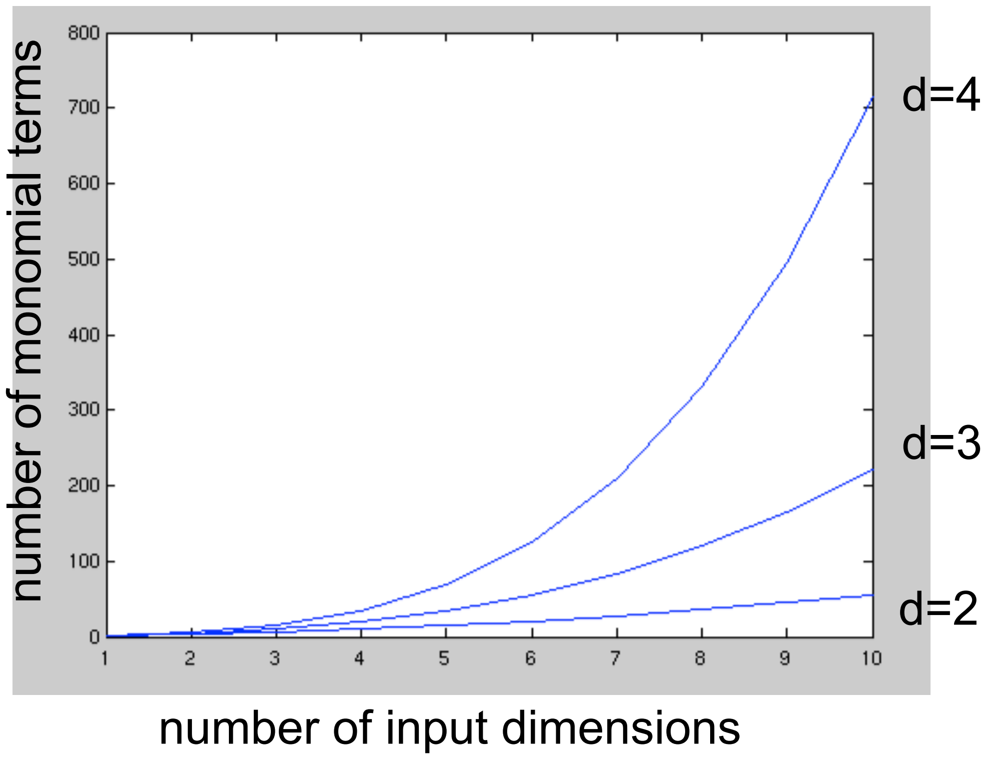
Use features of features of features of features….

$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \ldots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \ldots \\ e^{x^{(1)}} \\ \ldots \end{pmatrix}$$

Feature space can get really large really quickly!

Higher order polynomials

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



m – input features
d – degree of polynomial

grows fast!
d = 6, m = 100
about 1.6 billion terms

Dual formulation only depends on dot-products, not on **w**!

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

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Remember the examples x only appear in one dot product

First, we introduce features:

$$\mathbf{x}_i\mathbf{x}_j \quad \rightarrow \quad \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$$

Next, replace the dot product with a Kernel:

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

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Why is this useful???

Efficient dot-product of polynomials

Polynomials of degree exactly *d*

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} \begin{aligned} &= u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u.v)^2 \end{aligned}$$

For any *d (we will skip proof):*

$$\phi(u).\phi(v) = (u.v)^d$$

- Cool! Taking a dot product and exponentiating gives same results as mapping into high dimensional space and then taking dot produce

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 compute features explicitly!!!
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features
- But, $O(n^2)$ time in size of dataset to compute objective
  - Naïve implements slow
  - much work on speeding up

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

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

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

Common kernels

- Polynomials of degree exactly *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 kernels

$$K(\vec{u},\vec{v}) = \exp\left(-\frac{||\vec{u}-\vec{v}||_2^2}{2\sigma^2}\right)$$

- Sigmoid

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

- And many others: very active area of research!