CSE546: SVMs, Dual Formulation, and Kernels

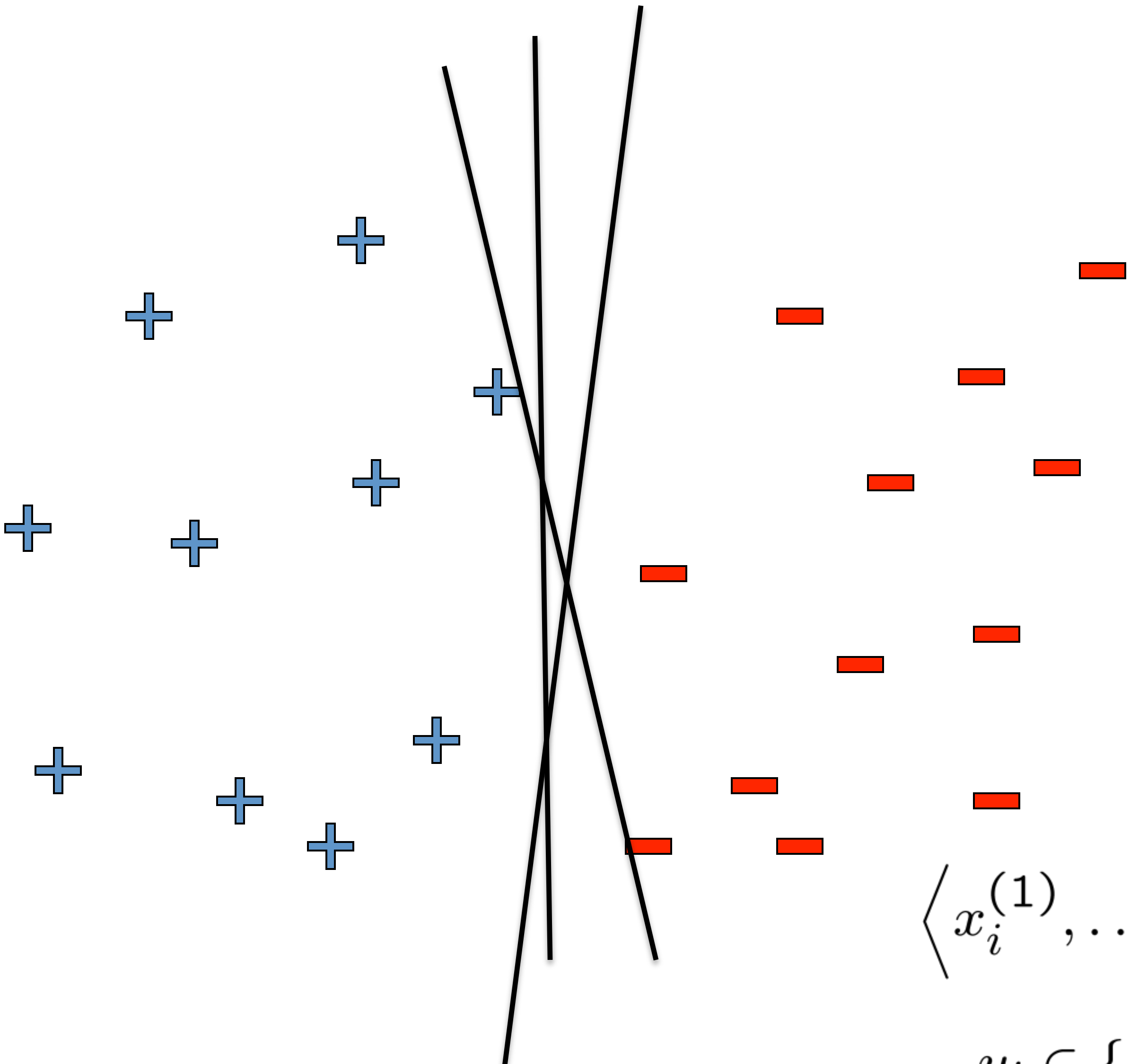
Winter 2012

Luke Zettlemoyer

Slides adapted from Carlos Guestrin

Linear classifiers – Which line is better?

$\mathbf{w}. = \sum_j w^{(j)} x^{(j)}$



Data

$$\left\langle x_1^{(1)}, \ldots, x_1^{(m)}, y_1 \right\rangle$$

$$\vdots$$

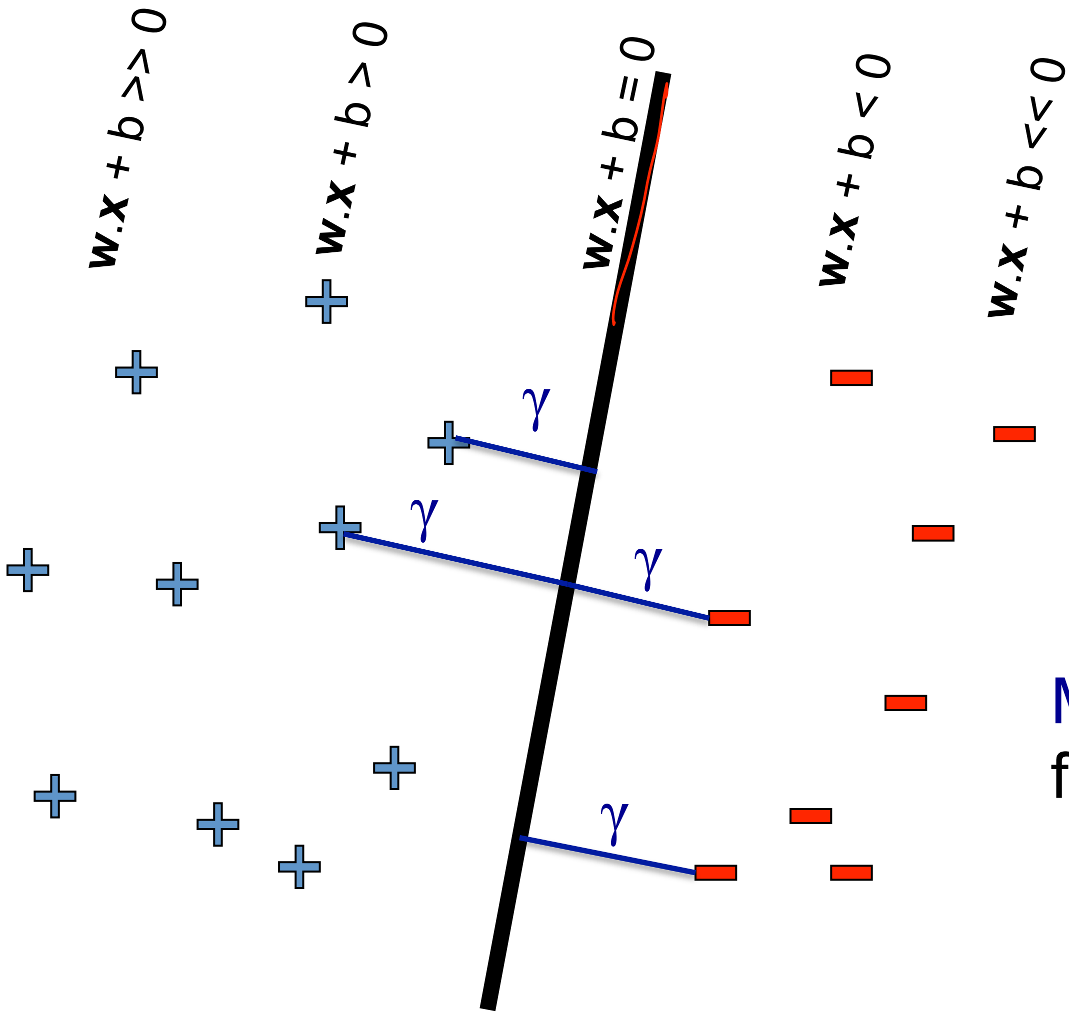
$$\left\langle x_n^{(1)}, \ldots, x_n^{(m)}, y_n \right\rangle$$

Example i

$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle$ — m features

$y_i \in \{-1, +1\}$ — class

Pick the one with the largest margin!



Margin: measures height of w.x+b plane at each point, increases with distance

$$\gamma_j = (w.x_j + b)y_j$$

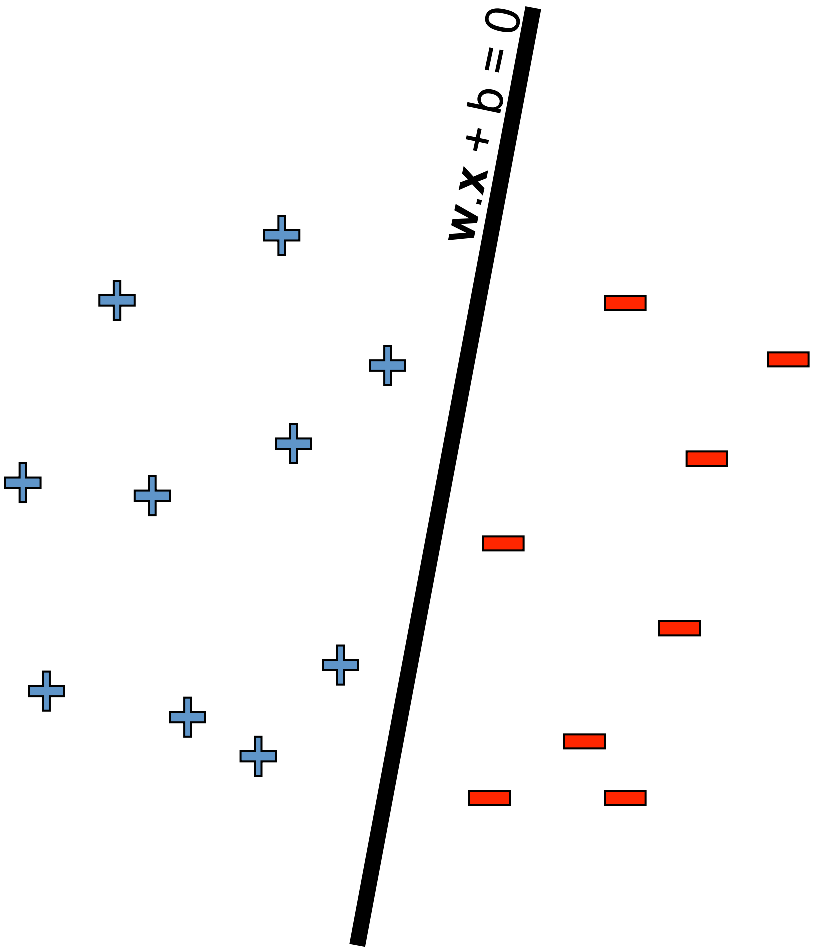
Max Margin: two equivalent forms

(1) $$\max_{w,b} \min_j \gamma_j$$

(2) $$\max_{\gamma,w,b} \gamma$$
$$\forall j \quad (w.x_j + b)y_j > \gamma$$

$\mathbf{w.x} = \sum_j w^{(j)}\, x^{(j)}$

How many possible solutions?



$$\max_{\gamma,w,b} \gamma$$

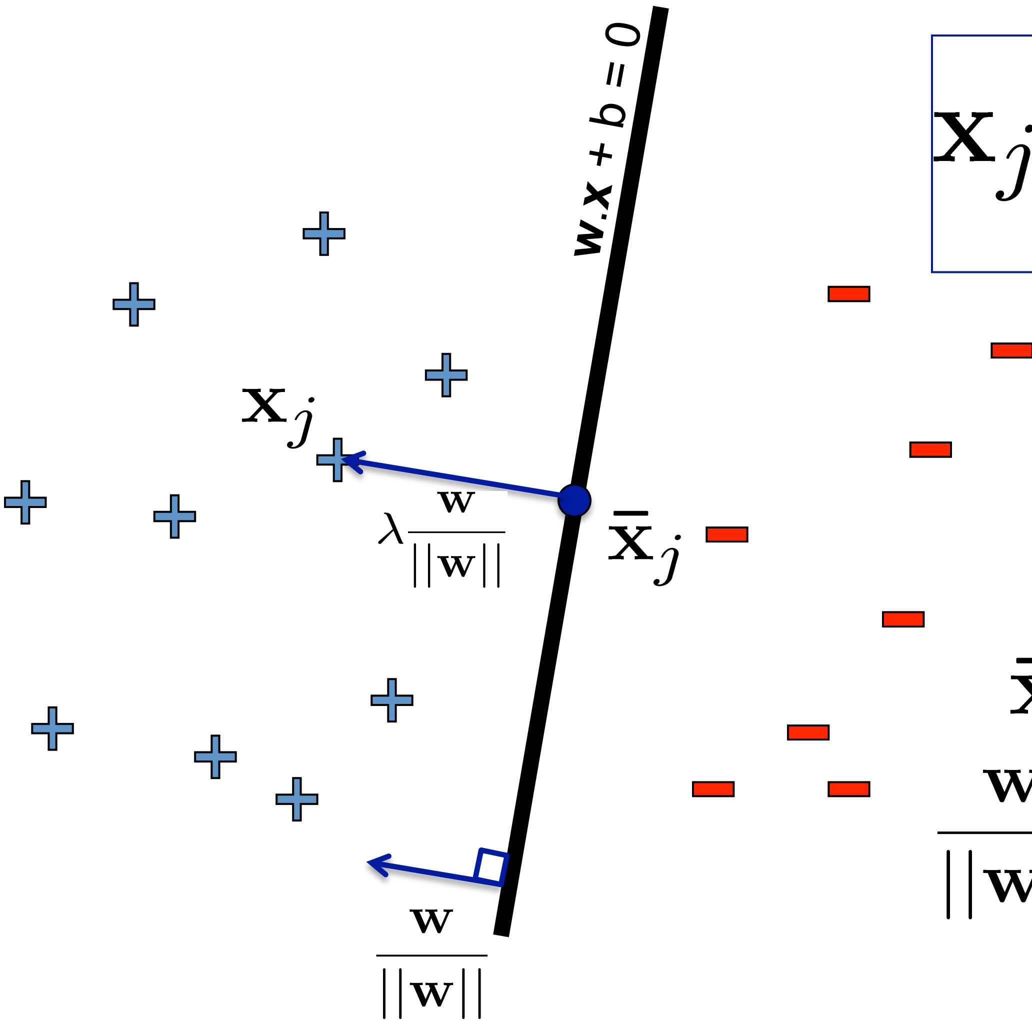
$$\forall j \quad (w.x_j + b)y_j > \gamma$$

Any other ways of writing the same dividing line?

- **w.x** + b = 0
- 2**w.x** + 2b = 0
- 1000**w.x** + 1000b = 0
- ....
- Any constant scaling has the same intersection with z=0 plane, so same dividing line!

Do we really want to max $_{\gamma,w,b}$?

Review: Normal to a plane



$$\boxed{\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}}$$

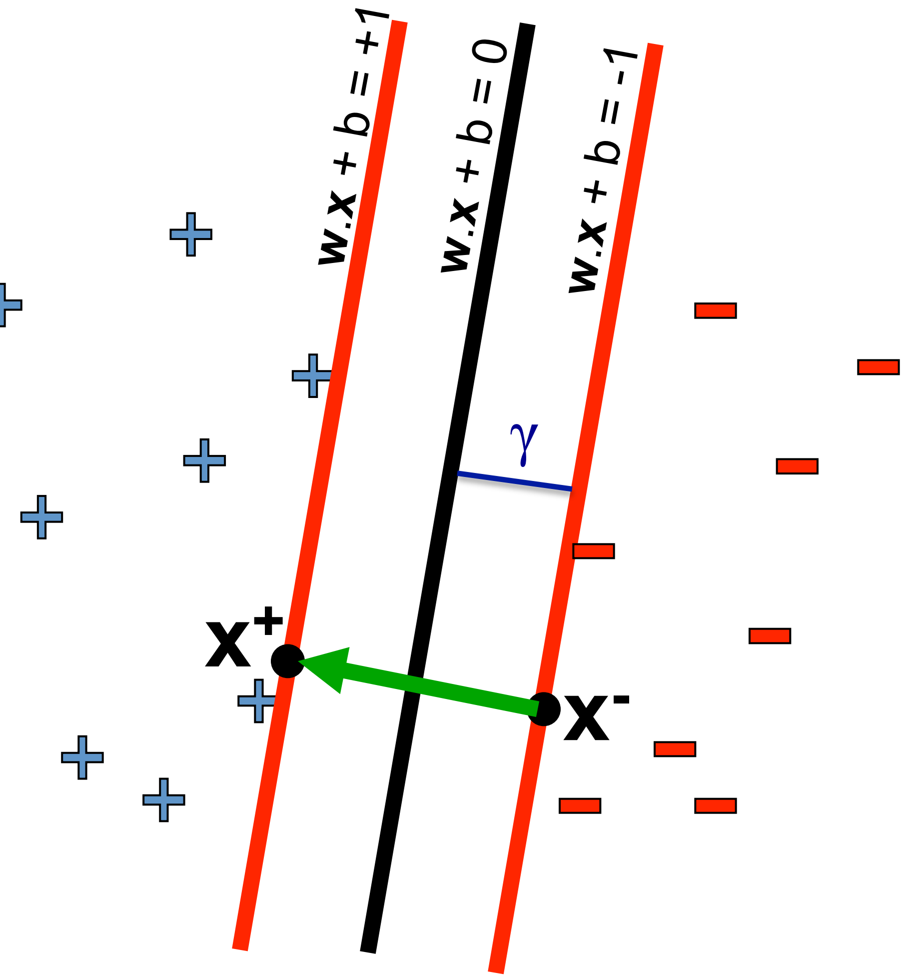
Key Terms

$\bar{\mathbf{x}}_j$ -- projection of x_j onto w

$\frac{\mathbf{w}}{||\mathbf{w}||}$ -- unit vector normal to w

Idea: *constrained* margin

$$\mathbf{x}_j = \bar{\mathbf{x}}_j + \lambda \frac{\mathbf{w}}{||\mathbf{w}||}$$



Generally:

$$x^+ = x^- + 2\gamma \frac{w}{||w||}$$

Assume: x^+ on positive line, x^- on negative

$$w.x^+ + b = 1$$

$$w.\left(x^- + 2\gamma \frac{w}{||w||}\right) + b = 1$$

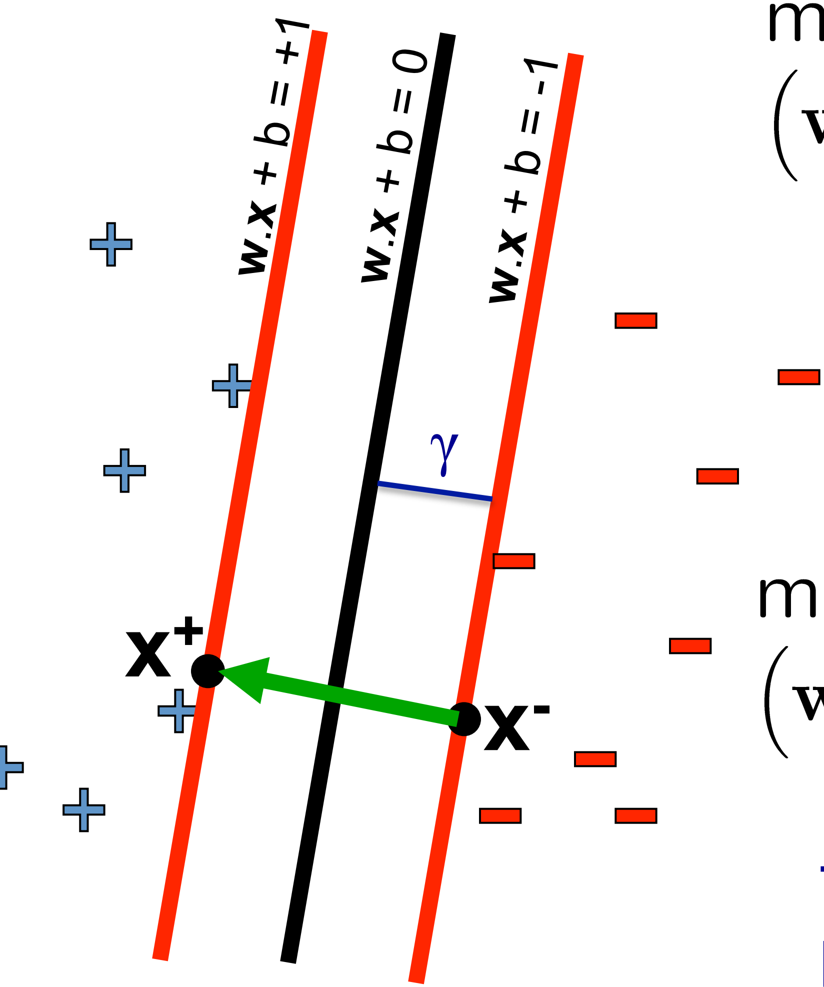
$$w.x^- + b + 2\gamma \frac{w.w}{||w||} = 1$$

$$\gamma \frac{w.w}{||w||} = 1$$

$$\boxed{\gamma = \frac{||w||}{w.w} = \frac{1}{\sqrt{w.w}}}$$

Final result: can maximize constrained margin by minimizing $||w||_2$!!!

Max margin using canonical hyperplanes



$$\text{maximize}_{\gamma,\mathbf{w},b} \quad \gamma$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq \gamma, \ \forall j \in \text{Dataset}$$

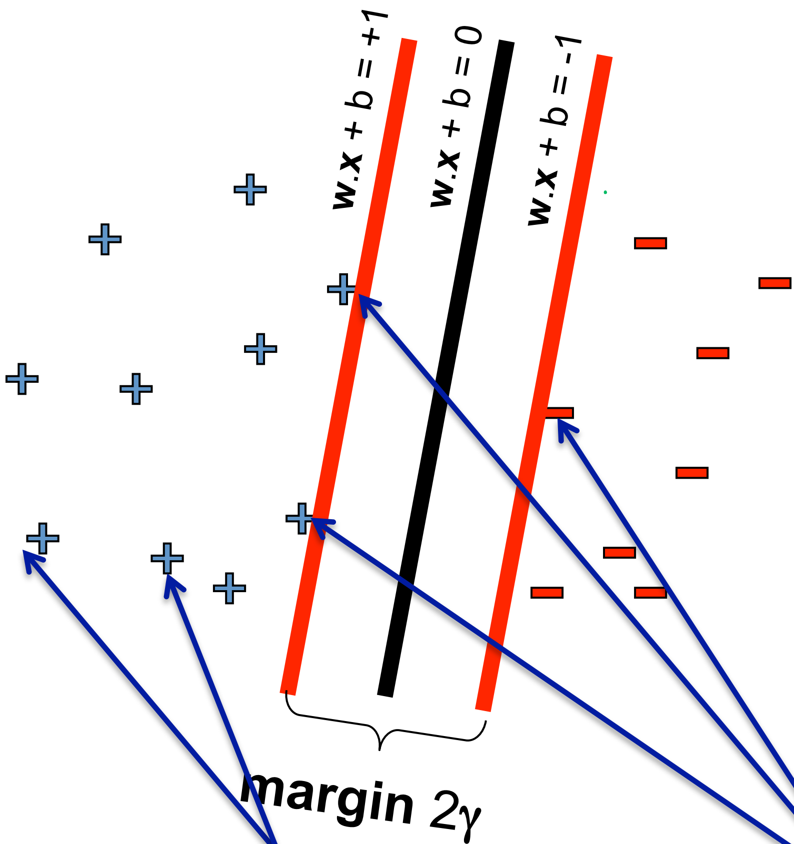
$$\gamma = \frac{1}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

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

The assumption of canonical hyperplanes (at +1 and -1) changes the objective and the constraints!

Support vector machines (SVMs)

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



- Solve efficiently by quadratic programming (QP)
  - Well-studied solution algorithms
  - Not simple gradient ascent, but close
- Hyperplane defined by support vectors
  - Could use them as a lower-dimension basis to write down line, although we haven't seen how yet
  - More on this later

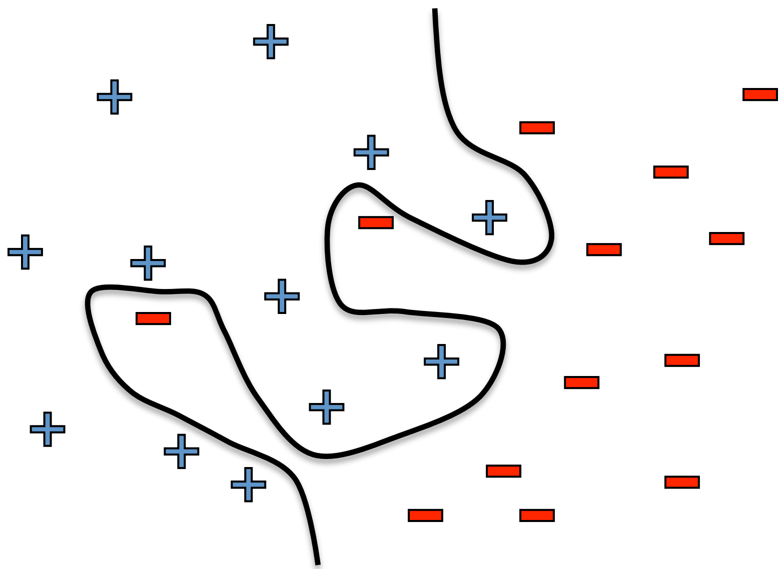
Non-support Vectors:
- everything else
- moving them will not change w

Support Vectors:
- data points on the canonical lines

What if the data is not linearly separable?

$$\left\langle x_i^{(1)}, \ldots, x_i^{(m)} \right\rangle \text{ — } m \text{ features}$$

$$y_i \in \{-1, +1\} \text{ — class}$$



Add More Features!!!

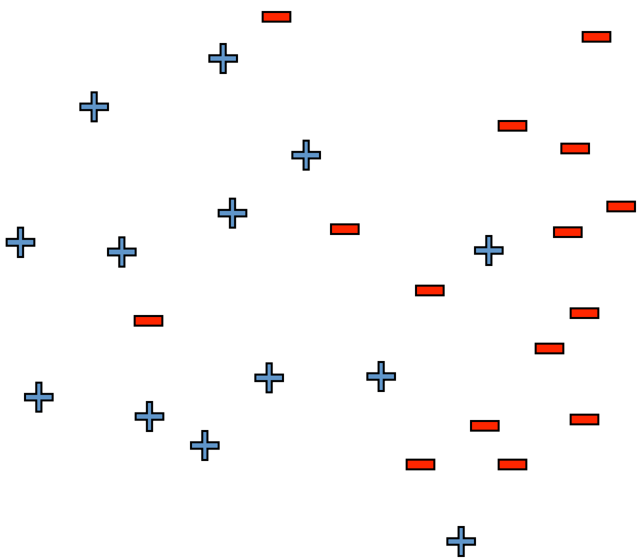
$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \ldots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \ldots \\ e^{x^{(1)}} \\ \ldots \end{pmatrix}$$

What about overfitting?

What if the data is still not linearly separable?

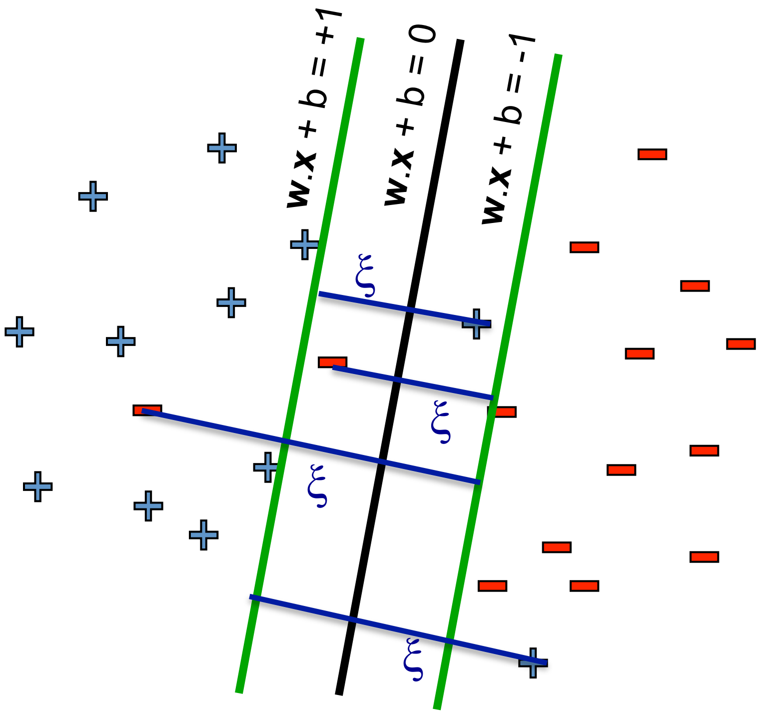
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} \ + \text{C \#(mistakes)}$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 \qquad , \forall j$$



- First Idea: Jointly minimize **w.w** and number of training mistakes
  - How to tradeoff two criteria?
  - Pick C on development / cross validation
- Tradeoff #(mistakes) and **w.w**
  - 0/1 loss
  - Slack penalty *C*
  - Not QP anymore
  - Also doesn't distinguish near misses and really bad mistakes

Slack variables – Hinge loss



$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\,\Sigma_j\,\xi_j$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j \quad , \forall j \quad \xi_j \geq 0$$

Slack Penalty $C > 0$:

- $C=\infty$ → have to separate the data!
- $C=0$ → ignore data entirely!
- Select on dev. set, etc.

For each data point:

- If margin ≥ 1, don't care
- If margin < 1, pay linear penalty

Side Note: Different Losses

Logistic regression:

$$\boxed{\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))}$$

Boosting :

$$\boxed{\frac{1}{m}\sum_{i} \exp(-y_i f(x_i)) = \prod_t Z_t}$$

SVM:

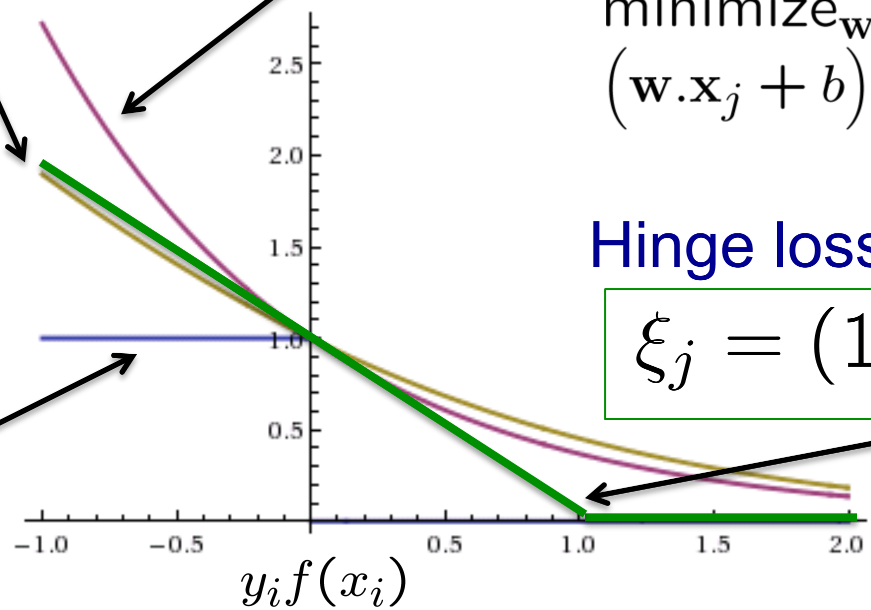
$$\text{minimize}_{\mathbf{w},b} \quad \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

Hinge loss:

$$\boxed{\xi_j = (1 - f(x_i)y_i)_+}$$

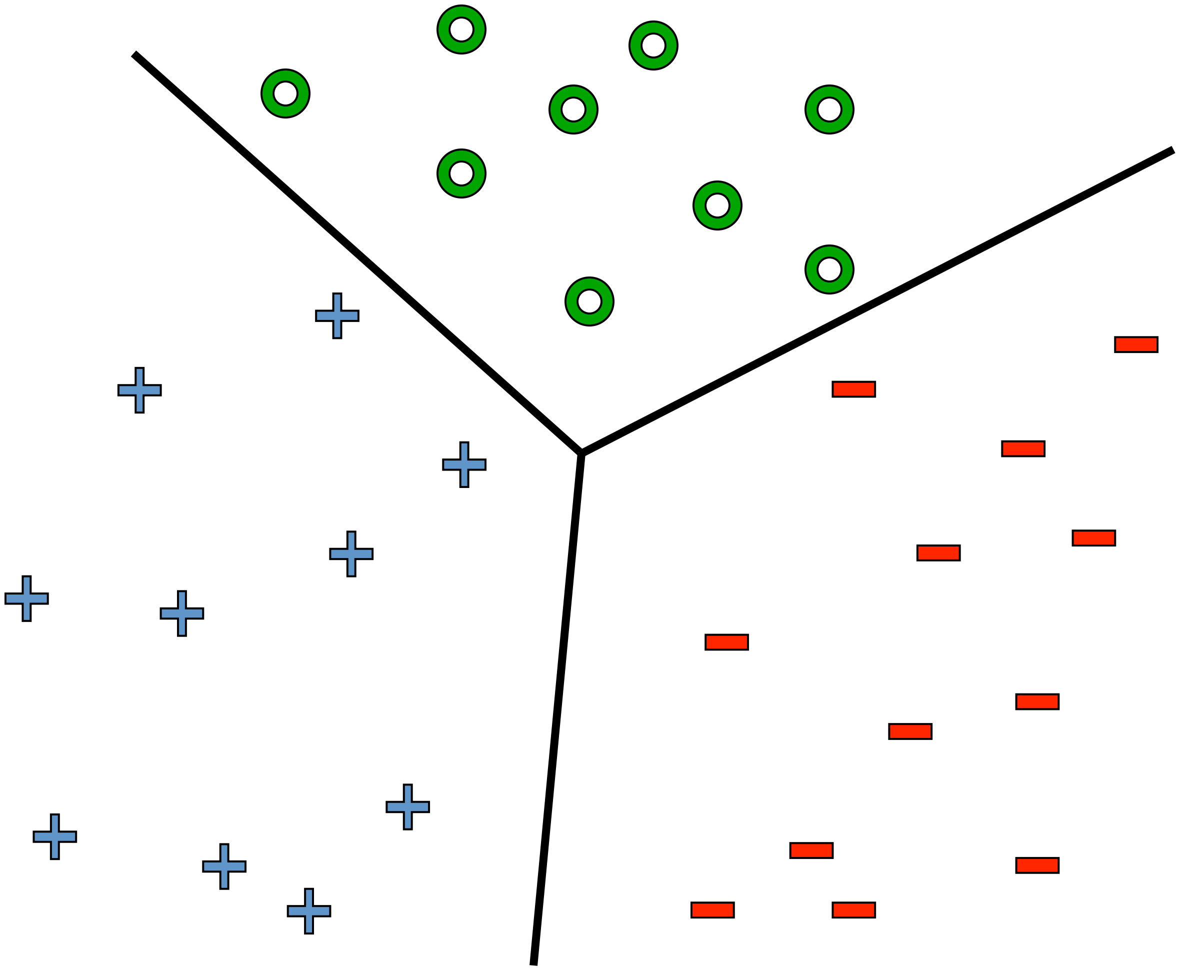
0-1 Loss:

$$\boxed{\delta(H(x_i) \neq y_i)}$$

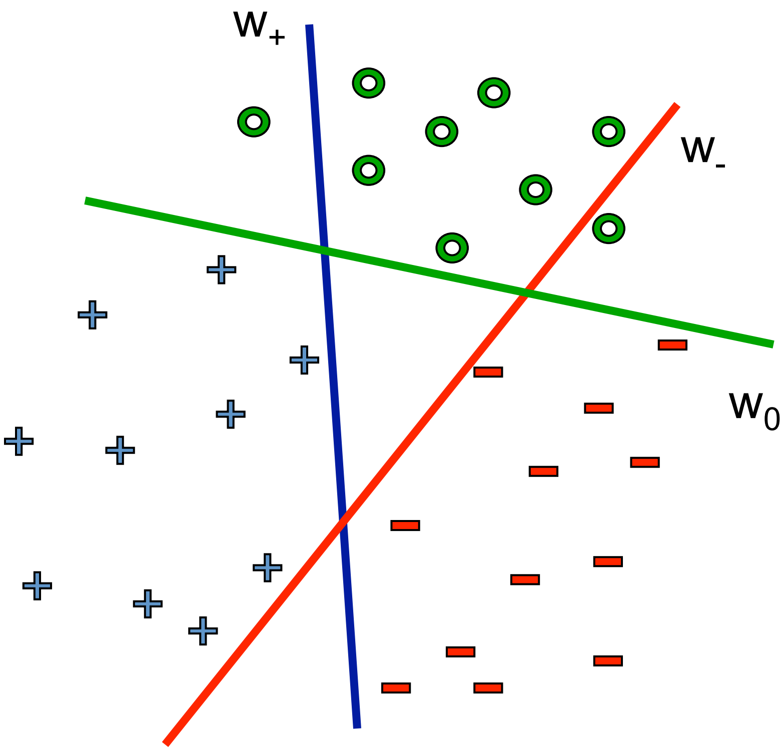


All approximations of 0/1 loss!

What about multiple classes?



One against All



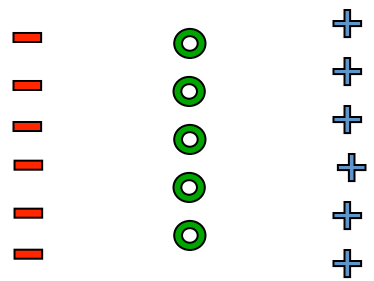
Learn 3 classifiers:

- $+$ vs $\{0,-\}$, weights w_+
- $-$ vs $\{0,+\}$, weights w_-
- 0 vs $\{+,-\}$, weights w_0

Output for x:

$$y = \text{argmax}_i \; w_i.x$$

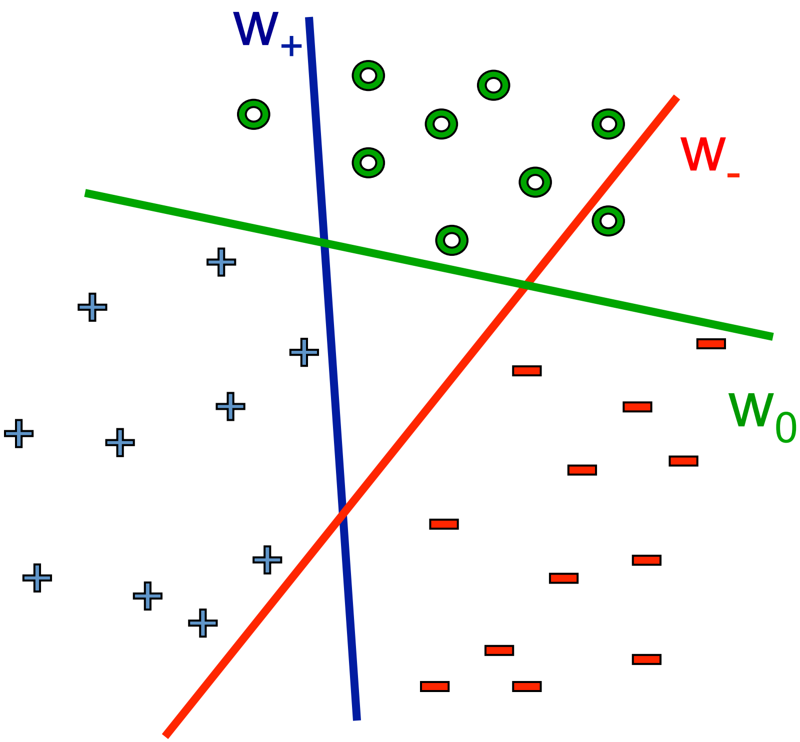
Any problems?
Could we learn this → dataset?



Learn 1 classifier: Multiclass SVM

Simultaneously learn 3 sets of weights:

- How do we guarantee the correct labels?
- Need new constraints



For j possible classes:

$$\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1, \ \forall y' \neq y_j, \ \forall j$$

Learn 1 classifier: Multiclass SVM

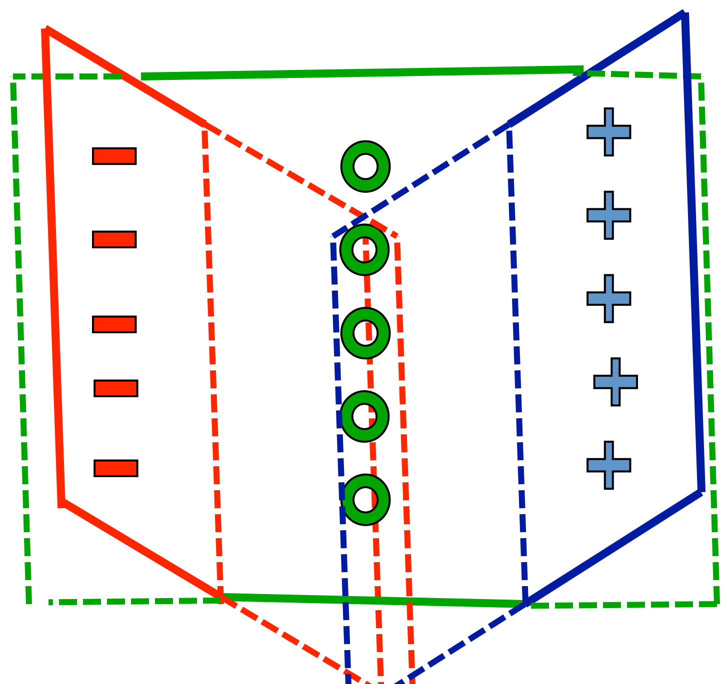
Introduce slack variables, as before:

$$\text{minimize}_{\mathbf{w},b} \quad \sum_y \mathbf{w}^{(y)}.\mathbf{w}^{(y)} + C\sum_j \xi_j$$

$$\mathbf{w}^{(y_j)}.\mathbf{x}_j + b^{(y_j)} \geq \mathbf{w}^{(y')}.\mathbf{x}_j + b^{(y')} + 1 - \xi_j, \ \forall y' \neq y_j, \ \forall j$$

$$\xi_j \geq 0, \ \forall j$$

Now, can we learn it? →



What you need to know

- Maximizing margin
- Derivation of SVM formulation
- Slack variables and hinge loss
- Relationship between SVMs and logistic regression
  - 0/1 loss
  - Hinge loss
  - Log loss
- Tackling multiple class
  - One against All
  - Multiclass SVMs

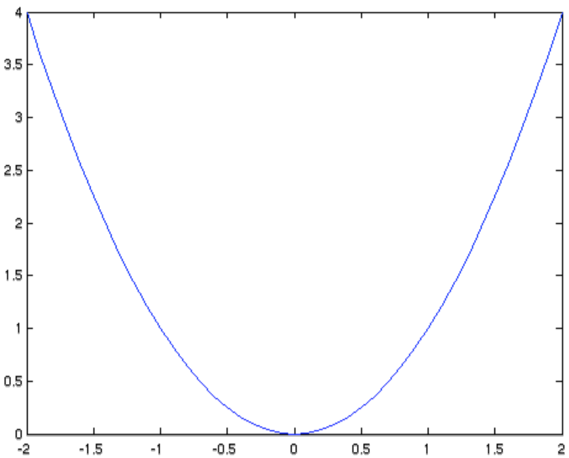
Whats Next!

- 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

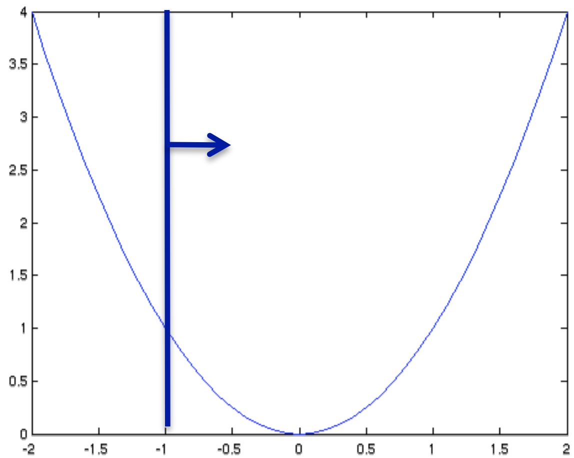
$$\min_x \; x^2 \qquad \text{s.t.} \quad x \geq b$$

No Constraint



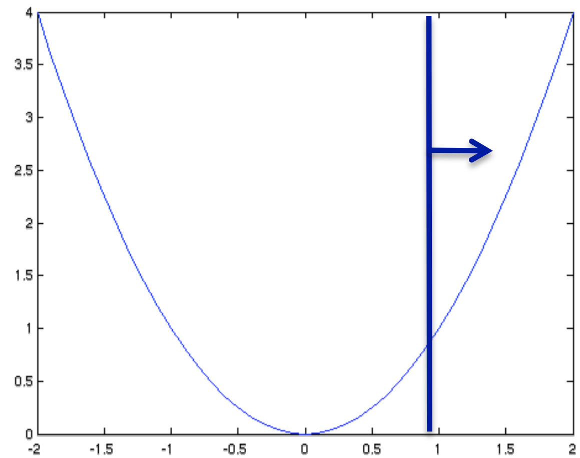
x*=0

x ≥ -1



x*=0

x ≥ 1

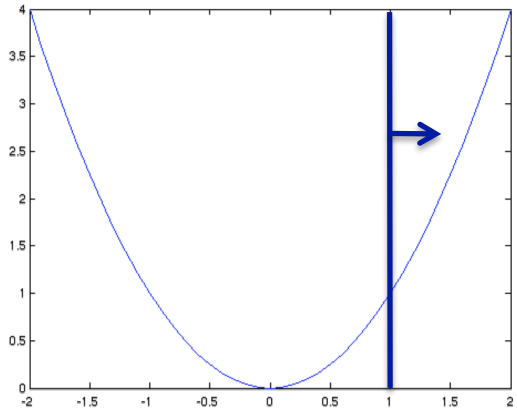


x*=1

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 does this work at all???

- min is fighting max!
- x<b → (x-b)<0 → $\max_\alpha -\alpha(x-b) = \infty$
  - min won't let that happen!!
- x>b, α>0→ (x-b)>0 → $\max_\alpha -\alpha(x-b) = 0$, α*=0
  - min is cool with 0, and L(x, α)=x^2 (original objective)
- x=b → α can be anything, and L(x, α)=x^2 (original objective)
- Since min is on the outside, can force max to behave and 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:

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

Dual SVM derivation (2) – the linearly separable case

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

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

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

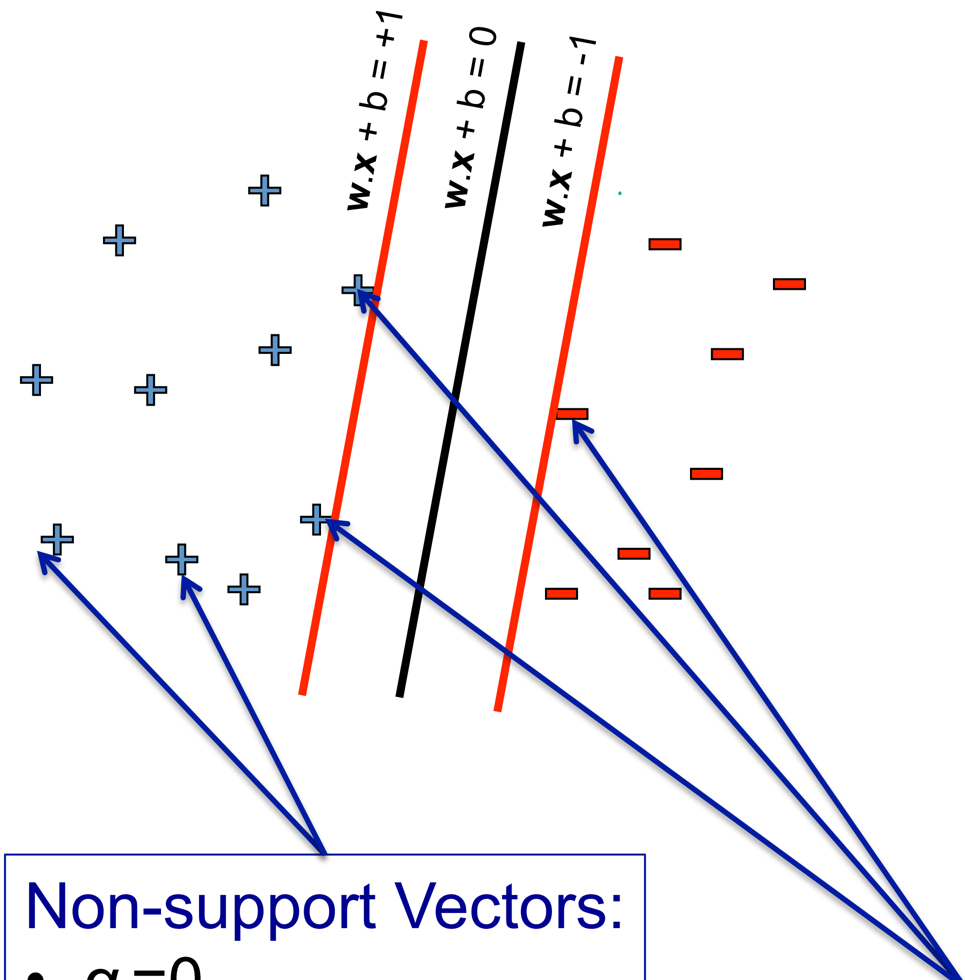
Also, α_k>0 implies constraint is tight → $b = y_k - \mathbf{w}.\mathbf{x}_k$

for any k where $\alpha_k > 0$

So, in dual formulation we solve for α directly!

- w,b are computed from α (if needed)

Dual SVM interpretation: Sparsity



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

Final solution tends to be sparse

- α_j=0 for most j
- don't need to store these points to compute w or make predictions

Non-support Vectors:

- α_j=0
- moving them will not change w

Support Vectors:

- $\alpha_j \geq 0$

Dual SVM formulation – linearly separable

Lagrangian:

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

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

Substituting (and some advanced math we are skipping) produces

$$\boxed{\begin{array}{l} \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \\ b = y_k - \mathbf{w}.\mathbf{x}_k \\ \text{for any } k \text{ where } \alpha_k > 0 \end{array}}$$

Dual SVM:

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

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Notes:

- max instead of min.
- One α for each training example

Sums over all training examples

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?

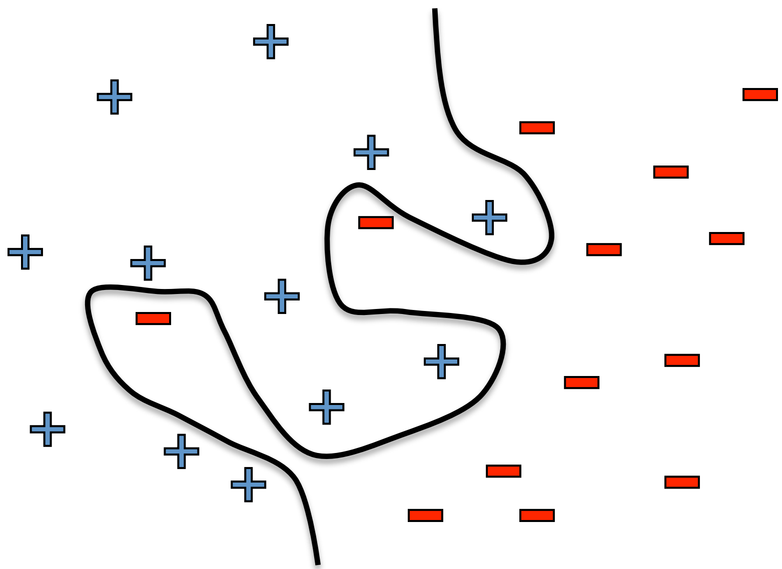
- Added upper bound of C on α_i!
- Intuitive explanation:
  - Without slack. $\alpha_i \rightarrow \infty$ when constraints are violated (points misclassified)
  - Upper bound of C limits the α_i, so misclassifications are allowed

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**"!!!
  - Another little detour…

Reminder: What if the data is not linearly separable?

Use features of features of features of features….

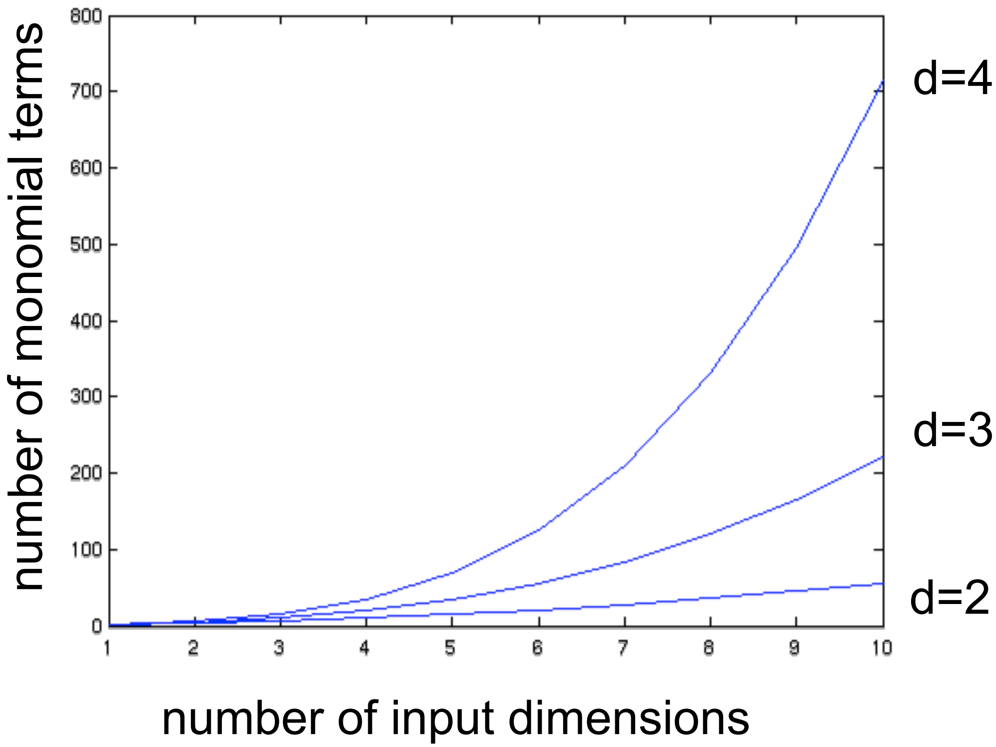


$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \dots \\ e^{x^{(1)}} \\ \dots \end{pmatrix}$$

Feature space can get really large really quickly!

Higher order polynomials

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



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

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

- Sigmoid

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

- And many others: very active area of research!

Overfitting?

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting
    - But everything overfits sometimes!!!
    - Can control by:
      - Setting C
      - Choosing a better Kernel
      - Varying parameters of the Kernel (width of Gaussian, etc.)

What about at classification time

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

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

$$\boxed{\begin{array}{l} \mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i) \\ b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k) \\ \text{for any } k \text{ where } C > \alpha_k > 0 \end{array}}$$

- Just need to store the support vectors and alphas

SVMs with kernels

- Choose a set of features and kernel function
- Solve dual problem to get support vectors and α_i
- At classification time: if we need to build $\Phi(\mathbf{x})$, we are in trouble!
  - instead compute:

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

$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

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

Classify as

$$sign\left(\mathbf{w} \cdot \Phi(\mathbf{x}) + b\right)$$

Only need to store support vectors and α_i!!!

Reminder: Kernel regression

Instance-based learning:

1. *A distance metric*

   Euclidian (and many more)

2. *How many nearby neighbors to look at?*

   All of them

3. *A weighting function*

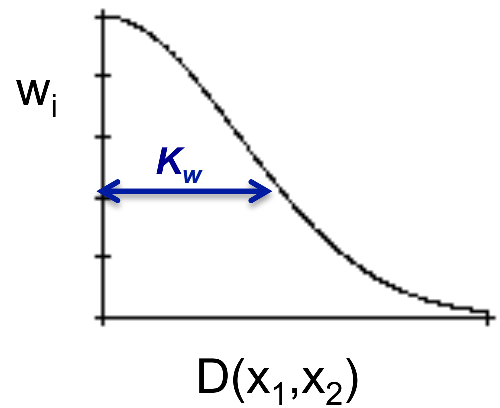
   $w_i = exp(-D(x_i, query)^2 / K_w^2)$

   Nearby points to the query are weighted strongly, far points weakly. The K_W parameter is the **Kernel Width**. Very important.

4. *How to fit with the local points?*

   Predict the weighted average of the outputs:

   $$\text{predict} = \Sigma w_i y_i / \Sigma w_i$$



SVMs v. Kernel Regression

SVMs

$$sign\left(\mathbf{w}\cdot\Phi(\mathbf{x})+b\right)$$

or

$$sign\left(\sum_i \alpha_i y_i K(\mathbf{x},\mathbf{x}_i)+b\right)$$

Kernel Regression

$$sign\left(\frac{\sum_i y_i K(\mathbf{x},\mathbf{x}_i)}{\sum_j K(\mathbf{x},\mathbf{x}_j)}\right)$$

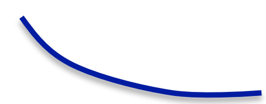
SVMs:

- Learn weights α_i (and bandwidth)
- Often sparse solution

KR:

- Fixed "weights", learn bandwidth
- Solution may not be sparse
- Much simpler to implement

What's the difference between SVMs and Logistic Regression?

	SVMs	**Logistic Regression**
Loss function	Hinge Loss 	Log Loss 
High dimensional features with kernels	**Yes!!!**	**Actually, yes!**

Kernels in logistic regression

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

- Define weights in terms of data points:

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

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

- Derive simple gradient descent rule on α_i,b
- Similar tricks for all linear models: Perceptron, etc

What’s the difference between SVMs and Logistic Regression? (Revisited)

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
Kernels	Yes!	Yes!
Solution sparse	Often yes!	Almost always no!
Semantics of learned model	Linear model from “Margin”	Probability Distribution

What you need to know

- Dual SVM formulation
  - How it's derived
- The kernel trick
- Derive polynomial kernel
- Common kernels
- Kernelized logistic regression
- SVMs vs kernel regression
- SVMs vs logistic regression