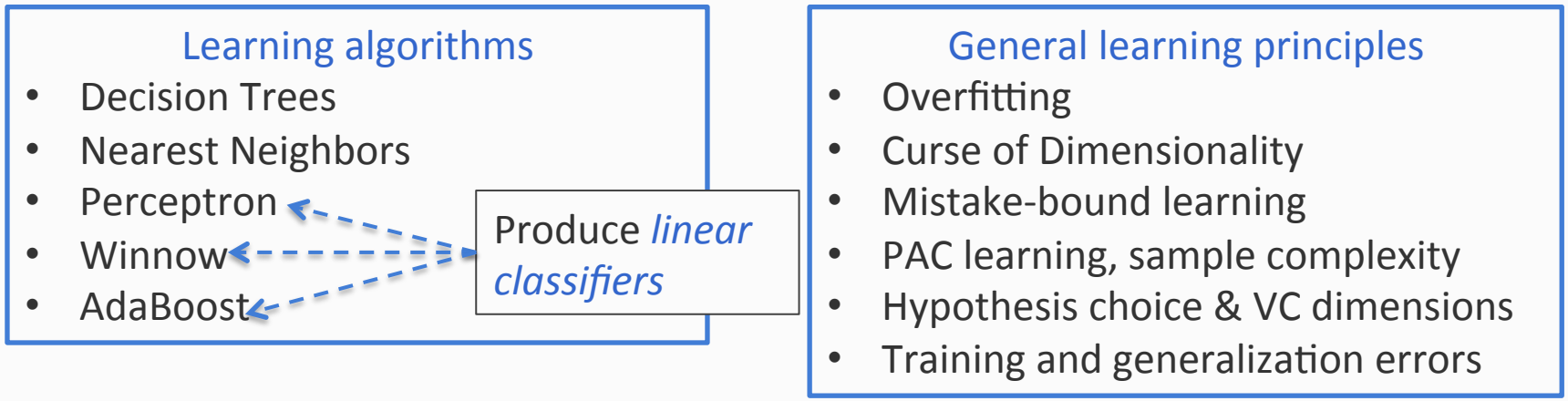
Support Vector Machines

Lecture 10

Machine Learning
Fall 2015

Where are we?

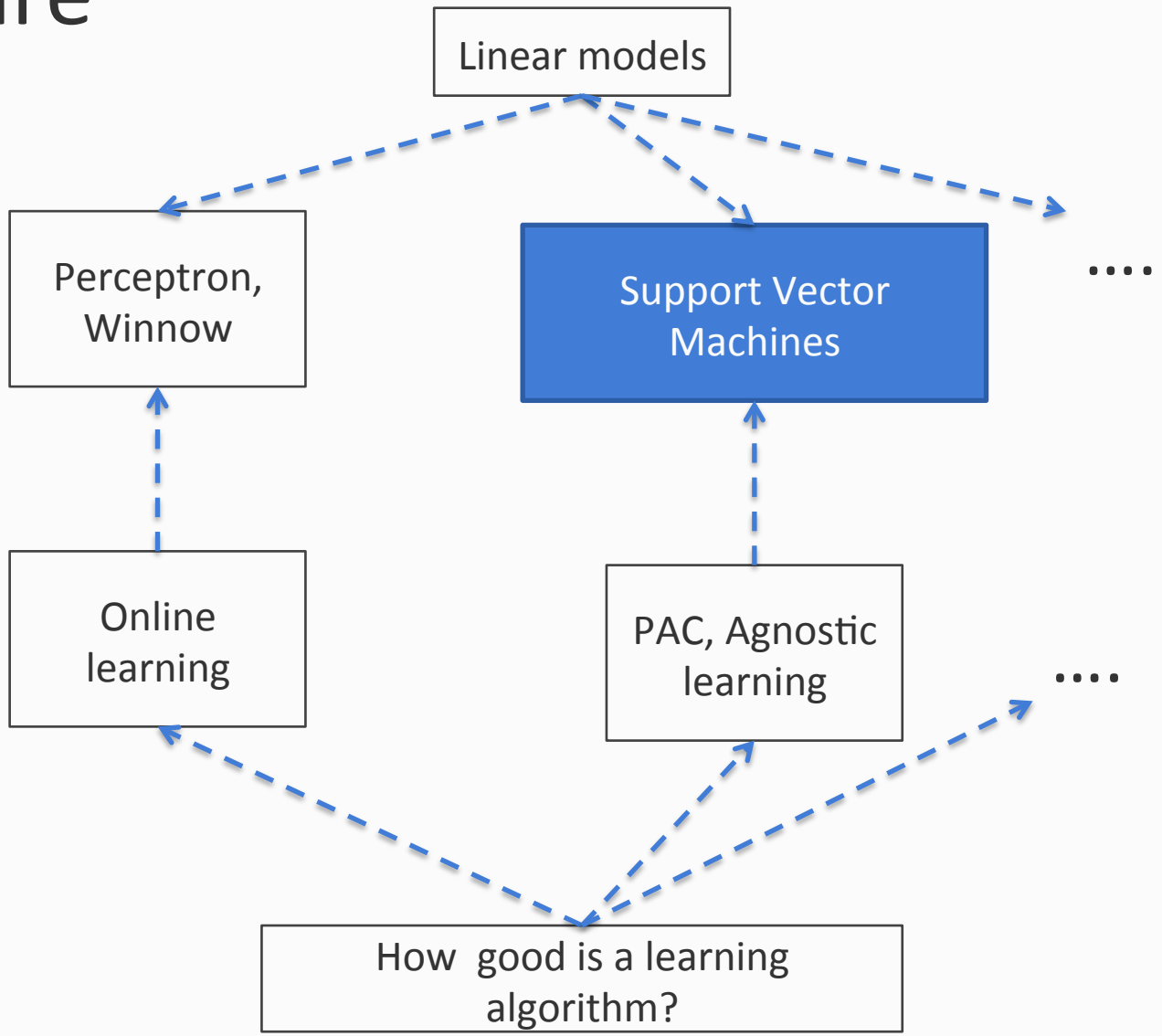


Coming up (next few lectures):

Learning theory $\rightarrow$ Training linear classifiers by minimizing *loss*

The Risk Minimization Principle

Big picture


Linear models
Perceptron, Winnow
Support Vector Machines
....
Online learning
PAC, Agnostic learning
....
How good is a learning algorithm?

This lecture: Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

This lecture: Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

VC dimensions and linear classifiers

What we know so far

1. If we have m examples, then with probability 1 - δ, the true error of a hypothesis h with training error $err_S(h)$ is bounded by

$$err_D(h) \leq err_S(h) + \sqrt{\frac{VC(H)\left(\ln \frac{2m}{VC(H)} + 1\right) + \ln \frac{4}{\delta}}{m}}$$

Generalization error — Training error — A function of VC dimension.

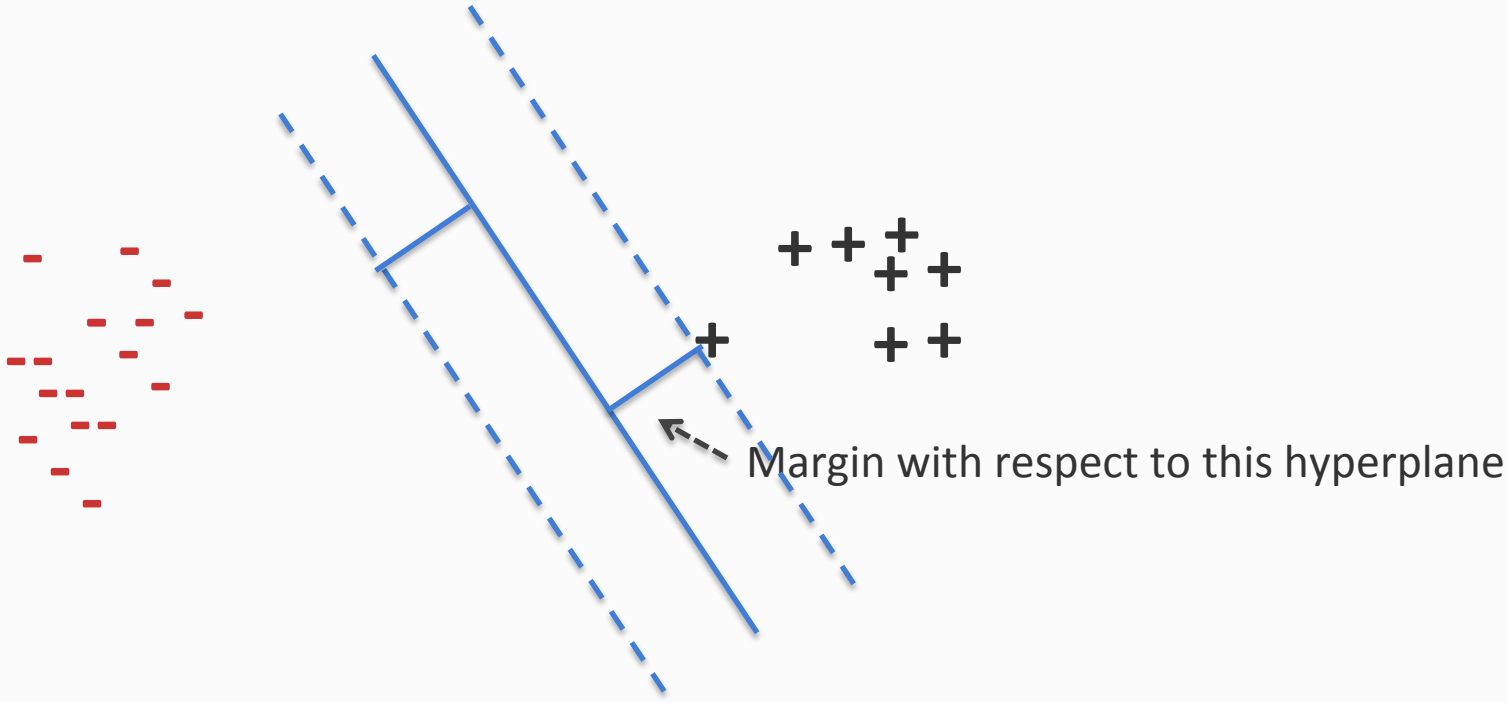
Low VC dimension gives tighter bound

2. VC dimension of a linear classifier in d dimensions = d + 1

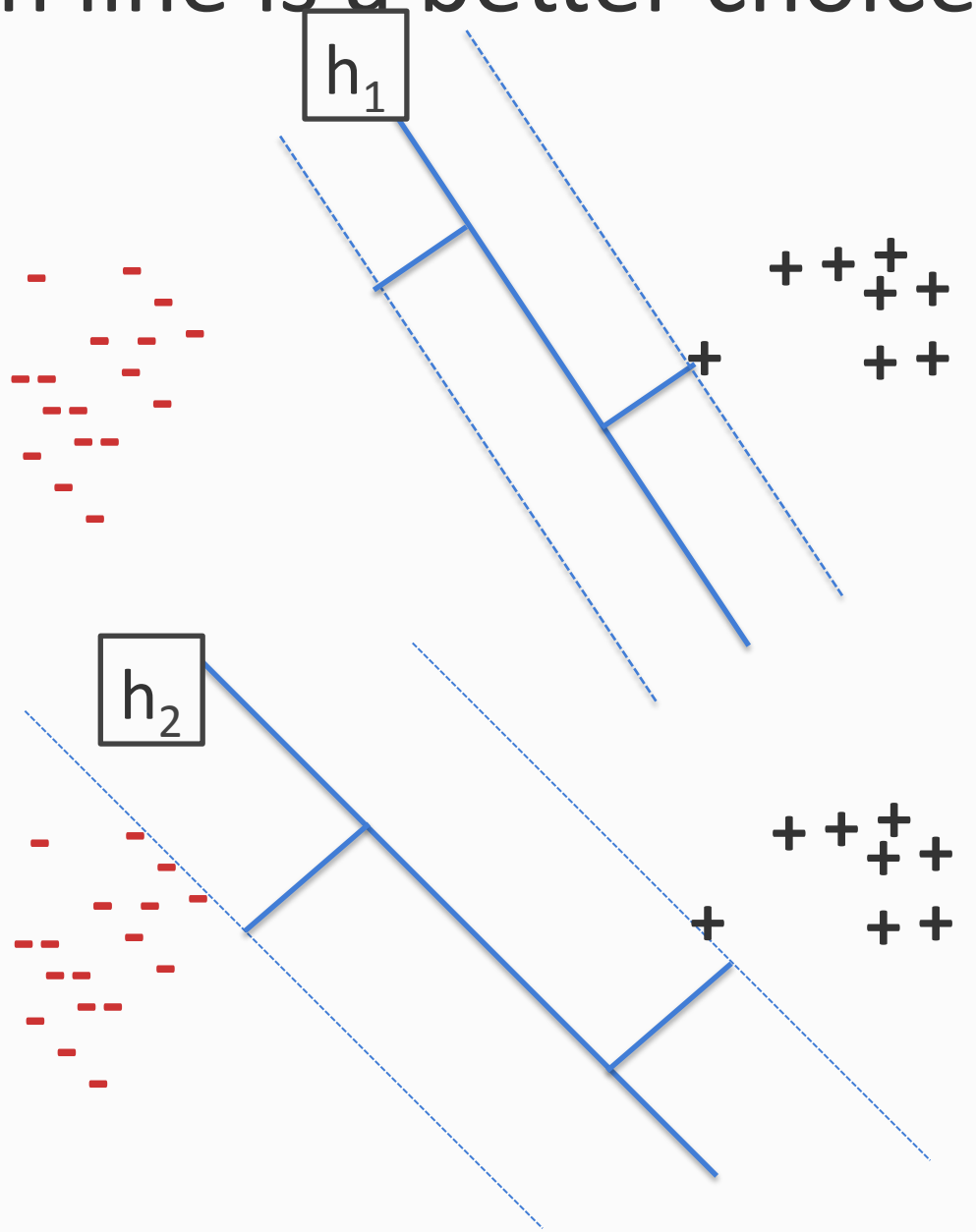
But are all linear classifiers the same?

Recall: Margin

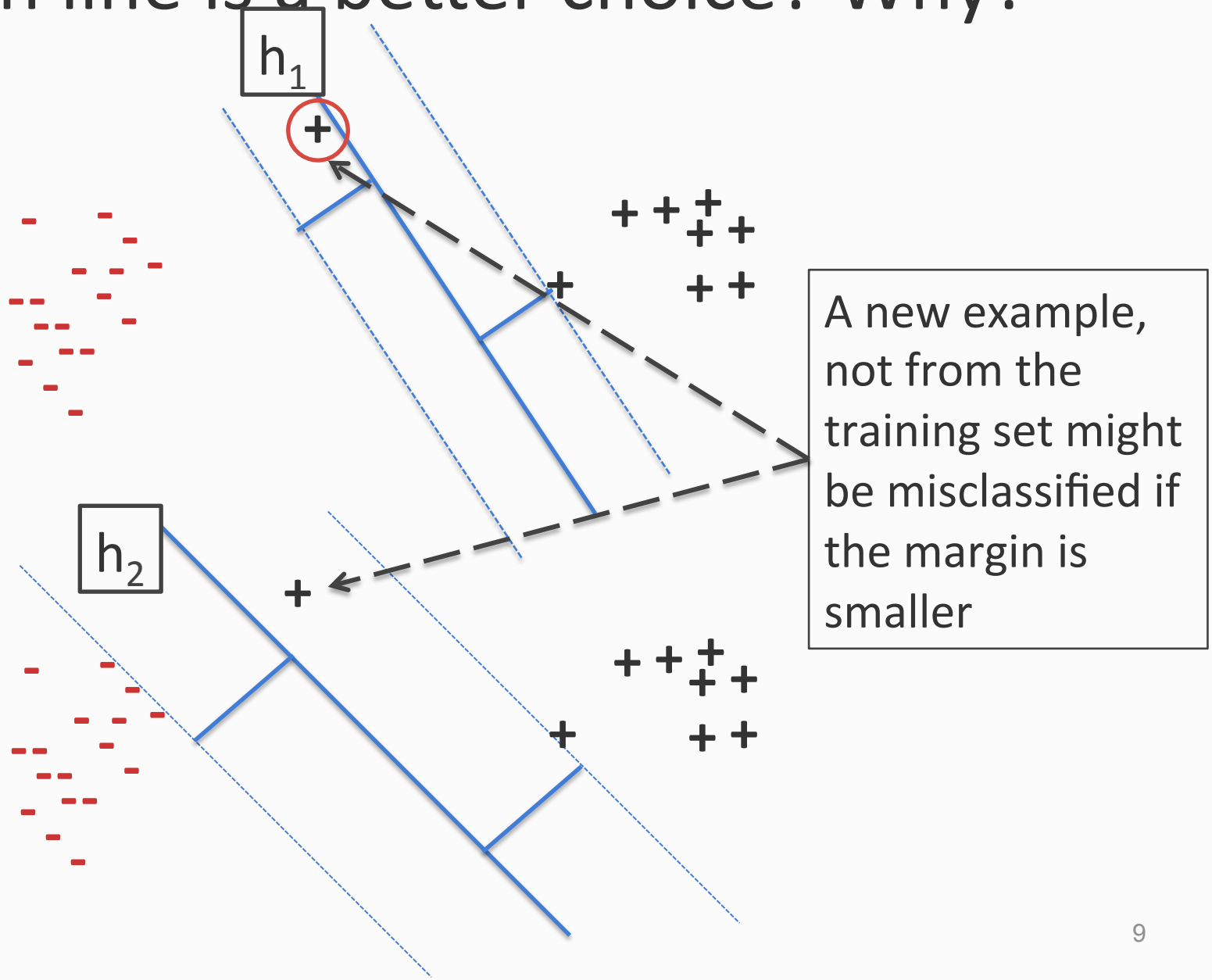
The margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it.



Which line is a better choice? Why?



Which line is a better choice? Why?



Data dependent VC dimension

- Intuitively, larger margins are better

- Suppose we only consider linear separators with margins γ_1 and γ_2
  - H_1 = linear separators that have a margin γ_1
  - H_2 = linear separators that have a margin γ_2
  - And $\gamma_1 > \gamma_2$

- The entire set of functions H_1 is "better"

Data dependent VC dimension

Theorem (Vapnik):

- Let H be the set of linear classifiers that separate the training set by a margin at least γ
- Then

$$VC(H) \leq \min\left(\frac{R^2}{\gamma^2}, d\right) + 1$$

- R is the radius of the smallest sphere containing the data

Larger margin $\Rightarrow$ Lower VC dimension

Lower VC dimension $\Rightarrow$ Better generalization bound

Learning strategy

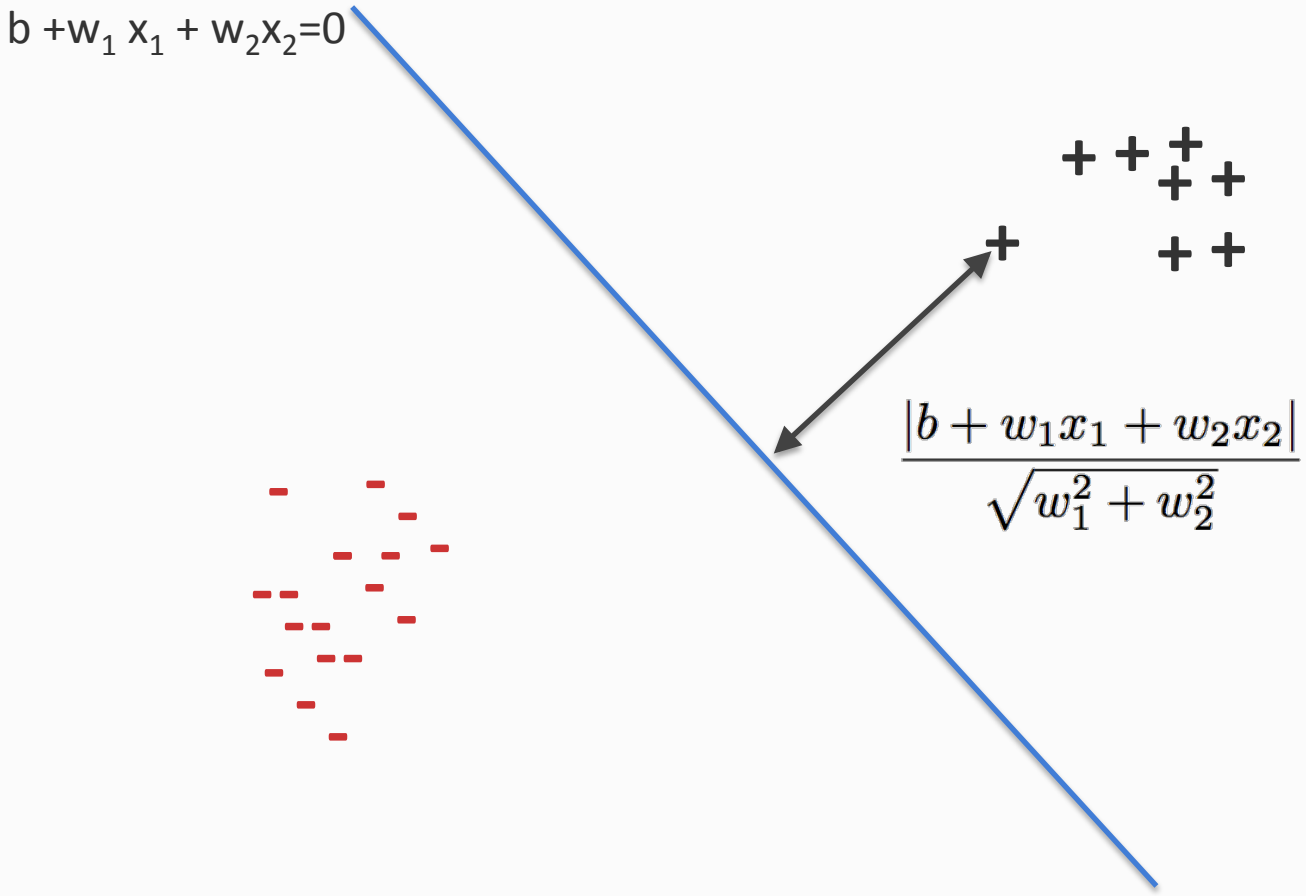
Find the linear separator that maximizes the margin

This lecture: Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

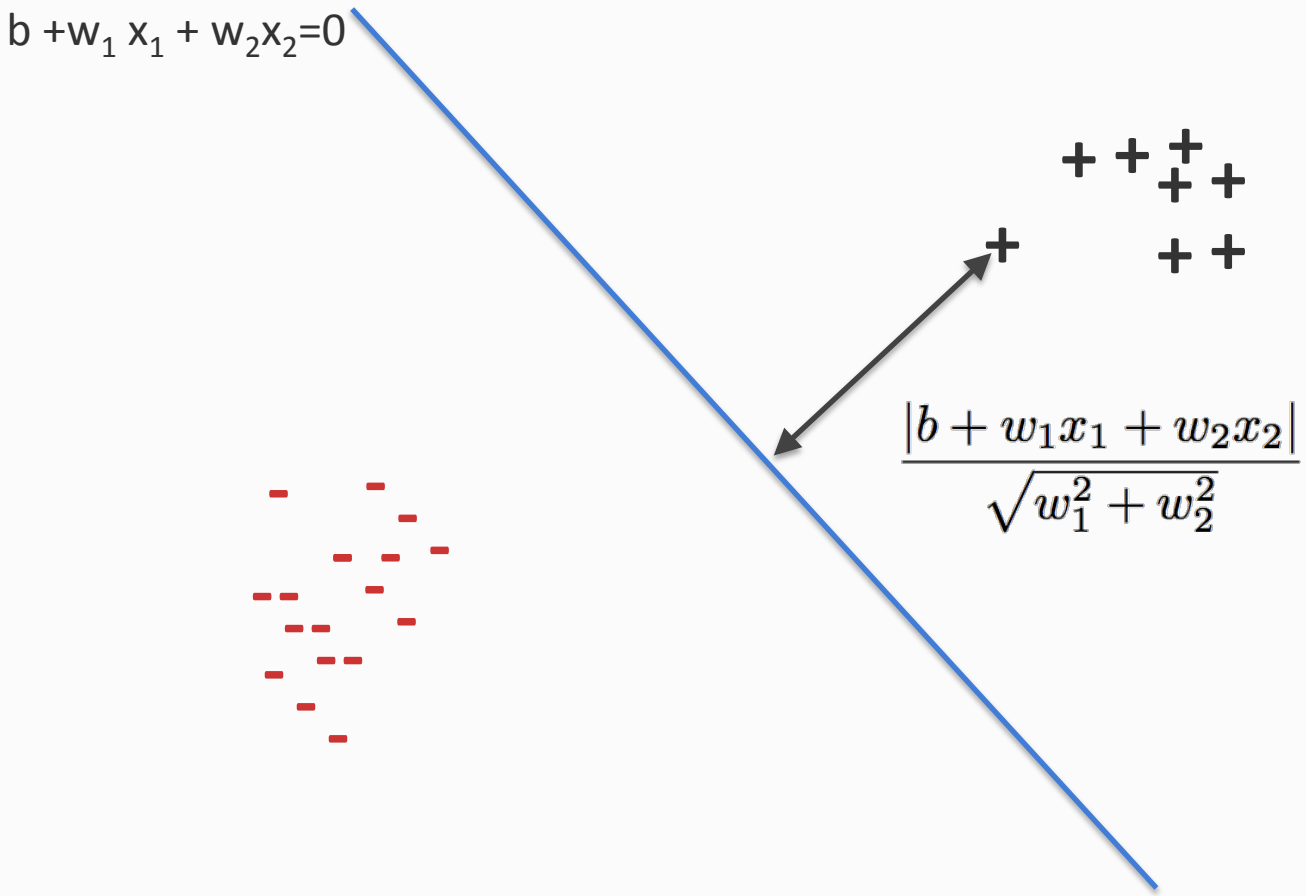
Recall: The geometry of a linear classifier

Prediction = sgn(b +w_1 x_1 + w_2x_2)



Recall: The geometry of a linear classifier

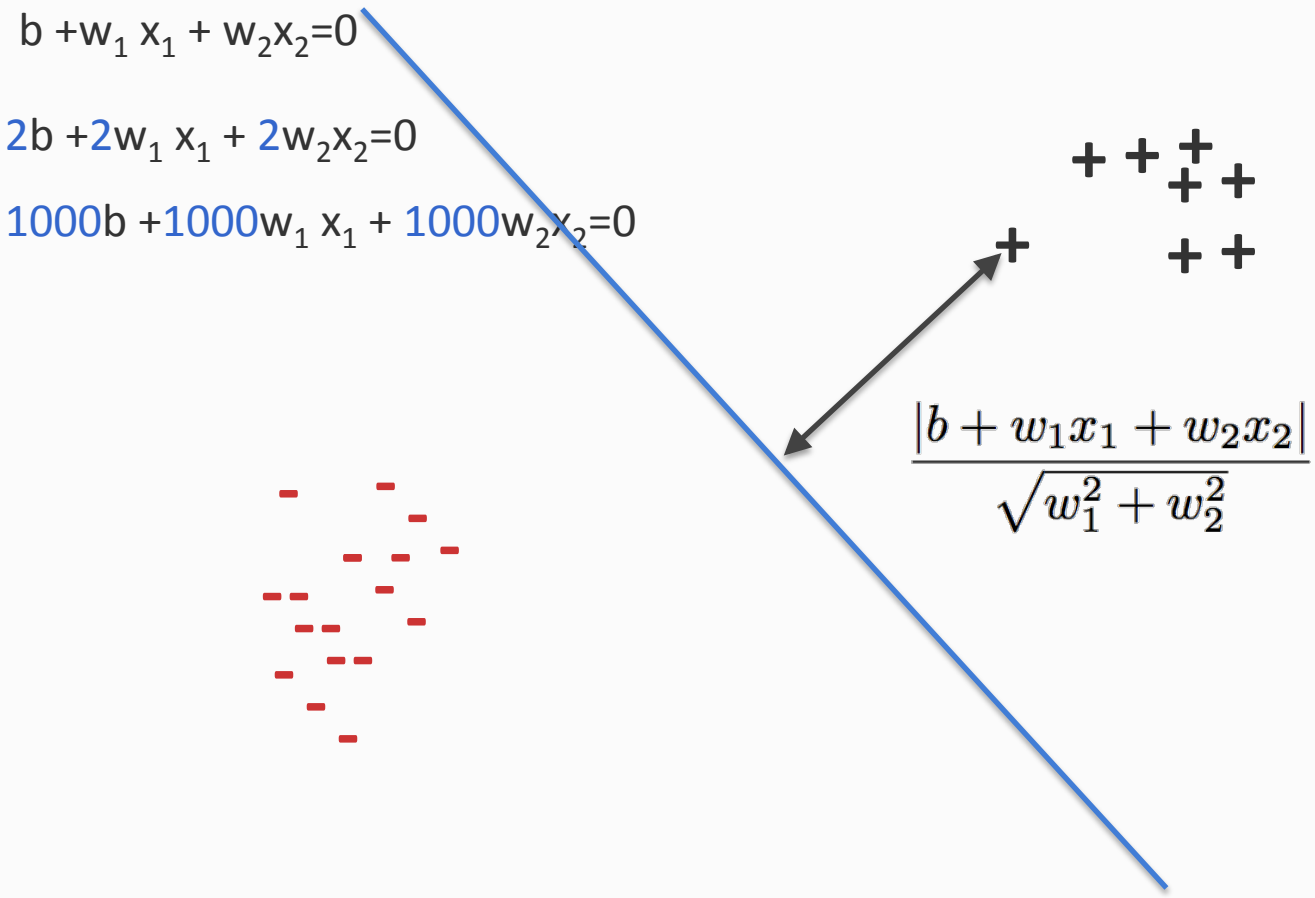
Prediction = *sgn*(b +w_1 x_1 + w_2x_2)



We only care about the sign, not the magnitude

Recall: The geometry of a linear classifier

Prediction = sgn(b +w_1 x_1 + w_2x_2)



We only care about the sign, not the magnitude

Maximizing margin

- Margin = distance of the closest point from the hyperplane

$$\gamma = \min_{\mathbf{x}_i, y_i} \frac{y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right)}{||\mathbf{w}||}$$

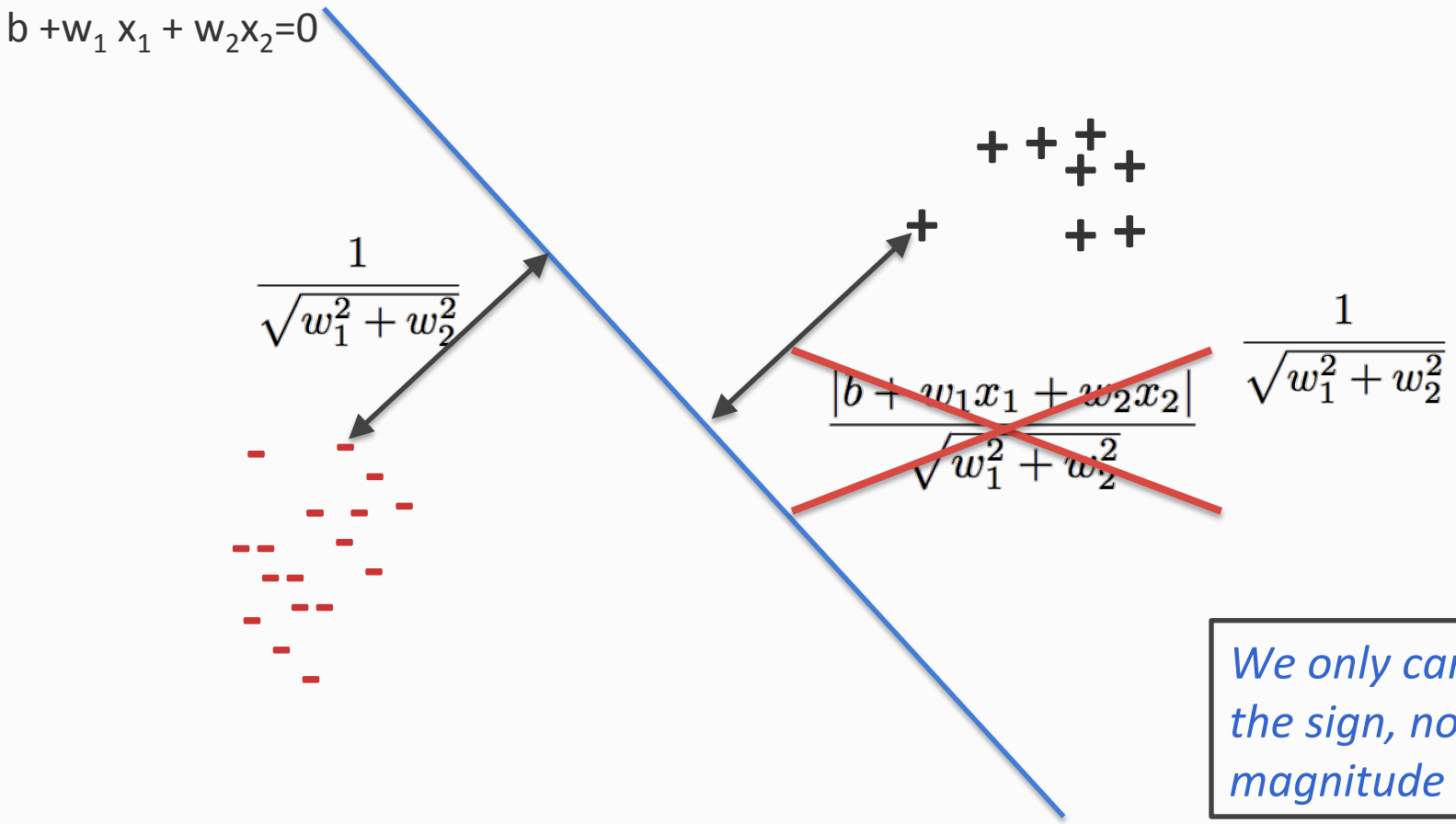
- We want $\max_{\mathbf{w}} \gamma$

Some people call this the *geometric margin*

The numerator alone is called the *functional margin*

Recall: The geometry of a linear classifier

Prediction = sgn(b +w_1 x_1 + w_2x_2)



We only care about the sign, not the magnitude

Maximizing margin

- Margin = distance of the closest point from the hyperplane

$$\gamma = \min_{\mathbf{x}_i, y_i} \frac{y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right)}{||\mathbf{w}||}$$

- We want $\max_{\mathbf{w}} \gamma$
- We only care about the sign of **w** and b in the end and not the magnitude
  - Set the absolute score (functional margin) of the closest point to be 1 and allow **w** to adjust itself

$\max_w \gamma$ is equivalent to $\min_w ||w||$ in this setting

Max-margin classifiers

$$\gamma = \min_{\mathbf{x}_i, y_i} \frac{y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right)}{||\mathbf{w}||}$$

- Learning a classifier:

  min ||**w**|| such that the closest example is at a distance 1/||**w**||

- Learning problem:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

Mimimizing gives us max 1/||**w**||

This condition is true for every example, specifically, for the example closest to the separator

For the closest point, $y_i \mathbf{w}^T \mathbf{x}_i = 1$
So, its distance from the separator is 1/||w||

Max-margin classifiers

$$\gamma = \min_{\mathbf{x}_i, y_i} \frac{y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right)}{||\mathbf{w}||}$$

- Learning a classifier:

  min ||**w**|| such that the closest example is at a distance 1/||**w**||

- Learning problem:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{s.t.} \forall i, \quad & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{aligned}$$

- This is called the "hard" Support Vector Machine

We will look at how to solve this optimization problem later

What if the data is not separable?

Hard SVM

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

Maximize margin

Every example has an functional margin of at least 1

What if the data is not separable?

Hard SVM

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

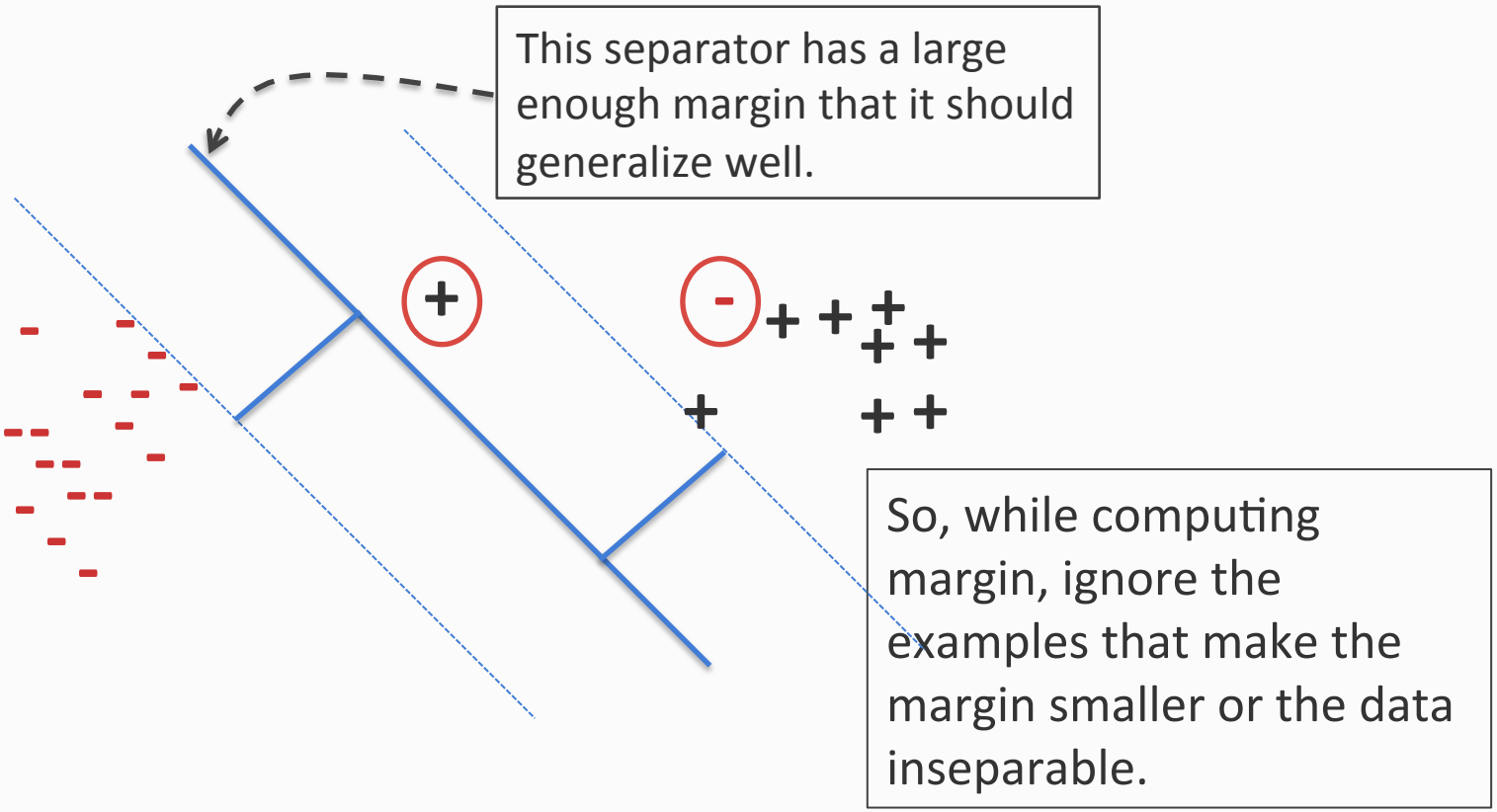
Maximize margin

Every example has an functional margin of at least 1

- This is a constrained optimization problem
- If the data is not separable, there is no **w** that will classify the data
- Infeasible problem, no solution!

Dealing with non-separable data

Key idea: Allow some examples to "break into the margin"



Soft SVM

- Hard SVM:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

Maximize margin

Every example has an functional margin of at least 1

- Introduce one *slack variable* ξ_i per example
  - And require $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$ and $\xi_i \geq 0$

Intuition: The slack variable allows examples to "break" into the margin

If the slack value is zero, then the example is either on or outside the margin

Soft SVM

- Hard SVM:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1$$

Maximize margin

Every example has an functional margin of at least 1

- Introduce one *slack variable* ξ_i per example
  - And require $y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$ and $\xi_i \geq 0$
- New optimization problem for learning

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$
$$\text{s.t.} \quad \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$$
$$\forall i, \quad \xi_i \geq 0.$$

Soft SVM

- Hard SVM:

$$\min_{\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \forall i, \quad y_i\mathbf{w}^T\mathbf{x}_i \geq 1$$

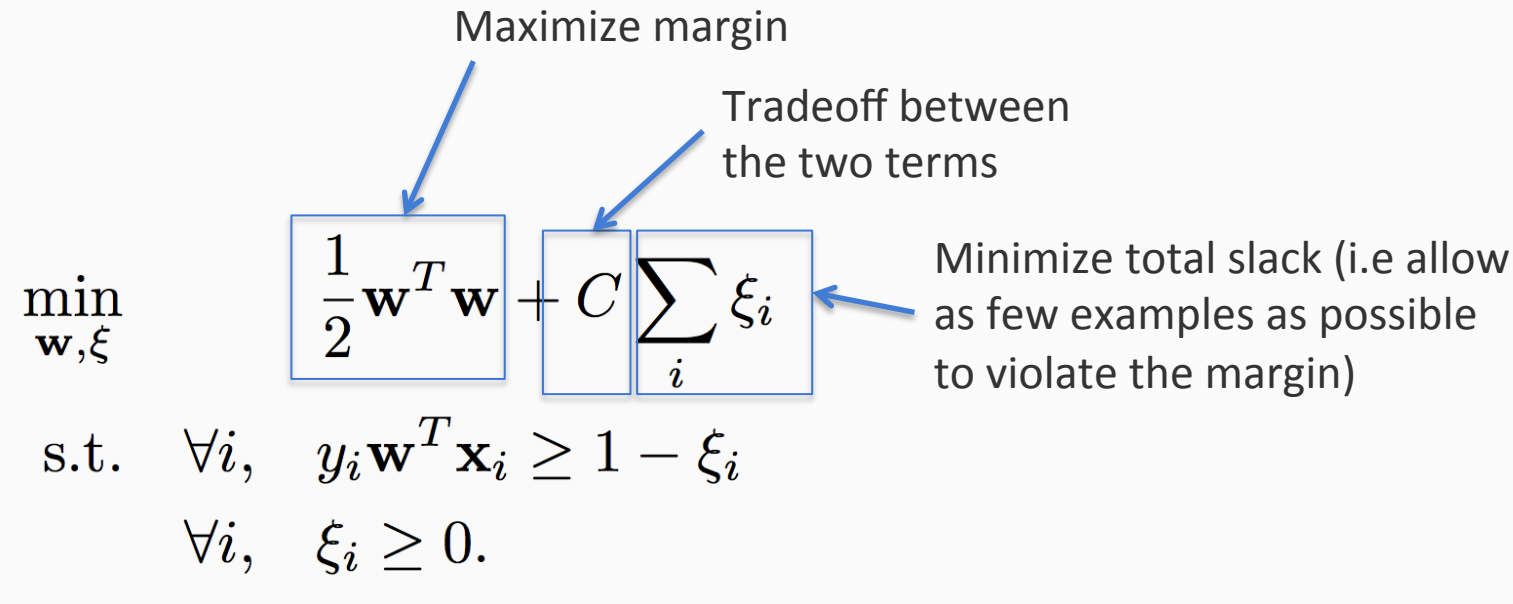
Maximize margin

Every example has an functional margin of at least 1

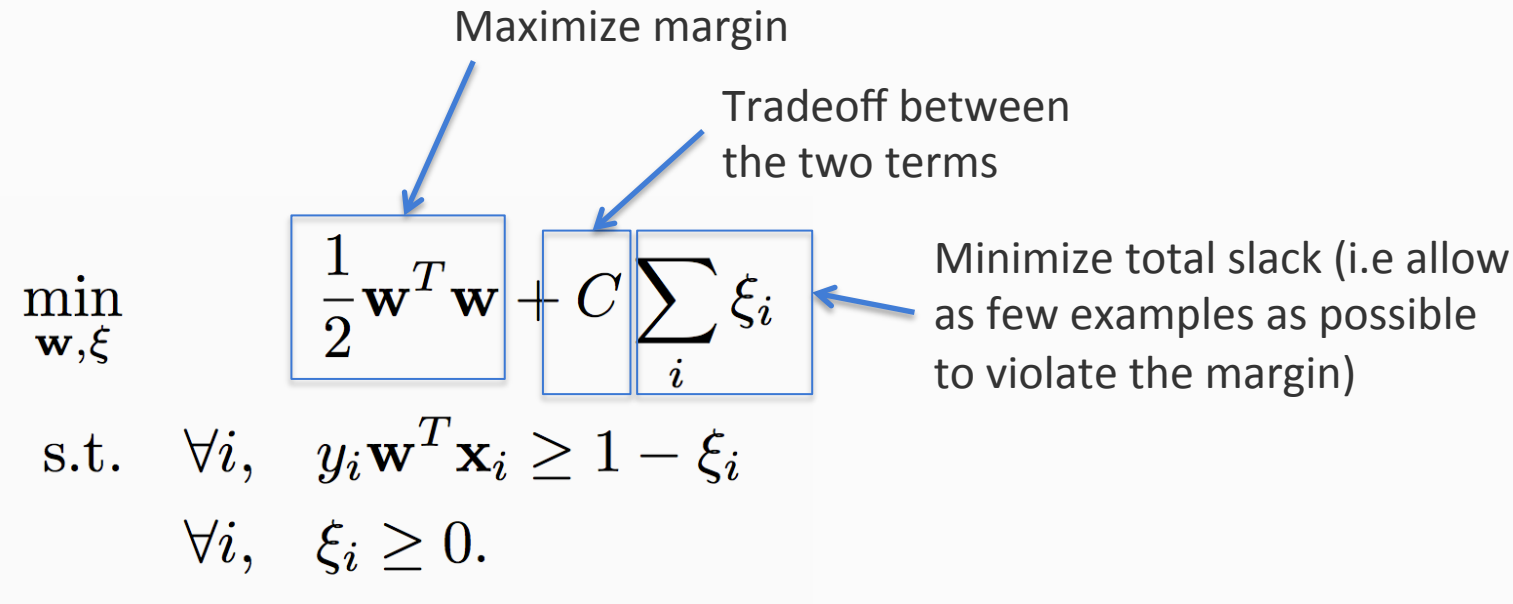
- Introduce one *slack variable* ξ_i per example
  - And require $y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$ and $\xi_i \geq 0$
- New optimization problem for learning

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$
$$\text{s.t.} \quad \forall i, \quad y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$$
$$\forall i, \quad \xi_i \geq 0.$$

Soft SVM

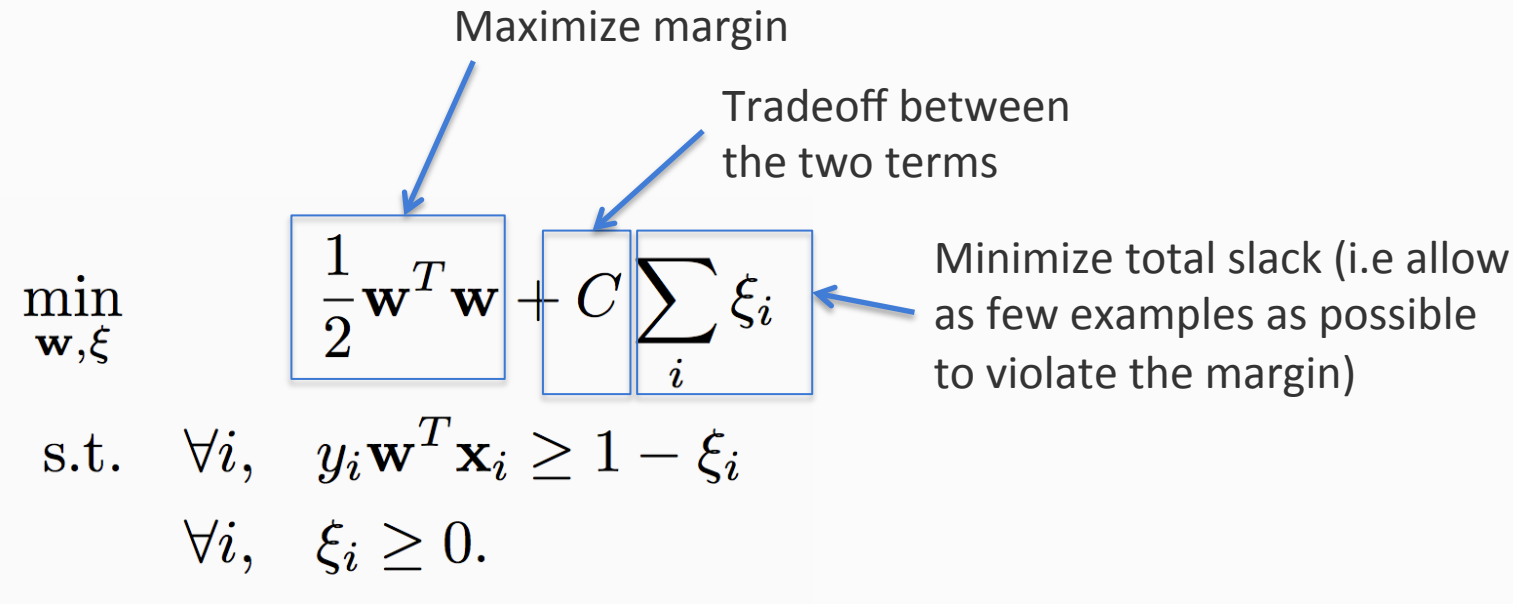

Maximize margin
Tradeoff between the two terms
Minimize total slack (i.e allow as few examples as possible to violate the margin)
$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$
s.t. $\forall i, \quad y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$
$\forall i, \quad \xi_i \geq 0.$

Soft SVM



Eliminate the slack variables to rewrite this

Soft SVM



$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

$$\text{s.t.} \quad \forall i, \quad y_i\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$$

$$\forall i, \quad \xi_i \geq 0.$$

Eliminate the slack variables to rewrite this as

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

This form is more interpretable

Maximizing margin and minimizing loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

Maximize margin

Penalty for the prediction

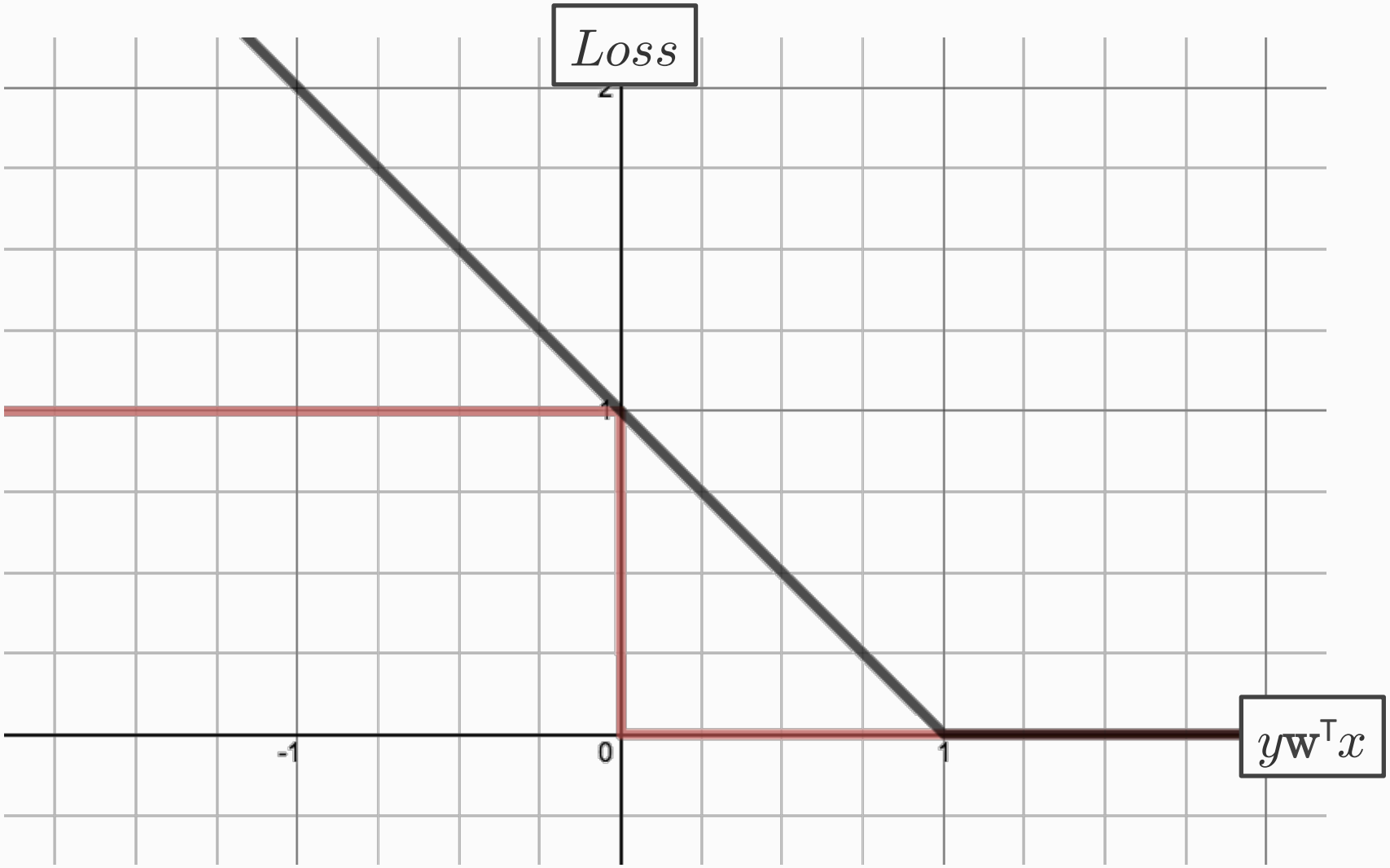
Three cases

- Example is correctly classified and is outside the margin: penalty = 0
- Example is incorrectly classified: penalty = $1 - y_i\ \mathbf{w}^T\mathbf{x}_i$
- Example is correctly classified but within the margin: penalty = $1 - y_i\ \mathbf{w}^T\mathbf{x}_i$

This is the hinge loss function

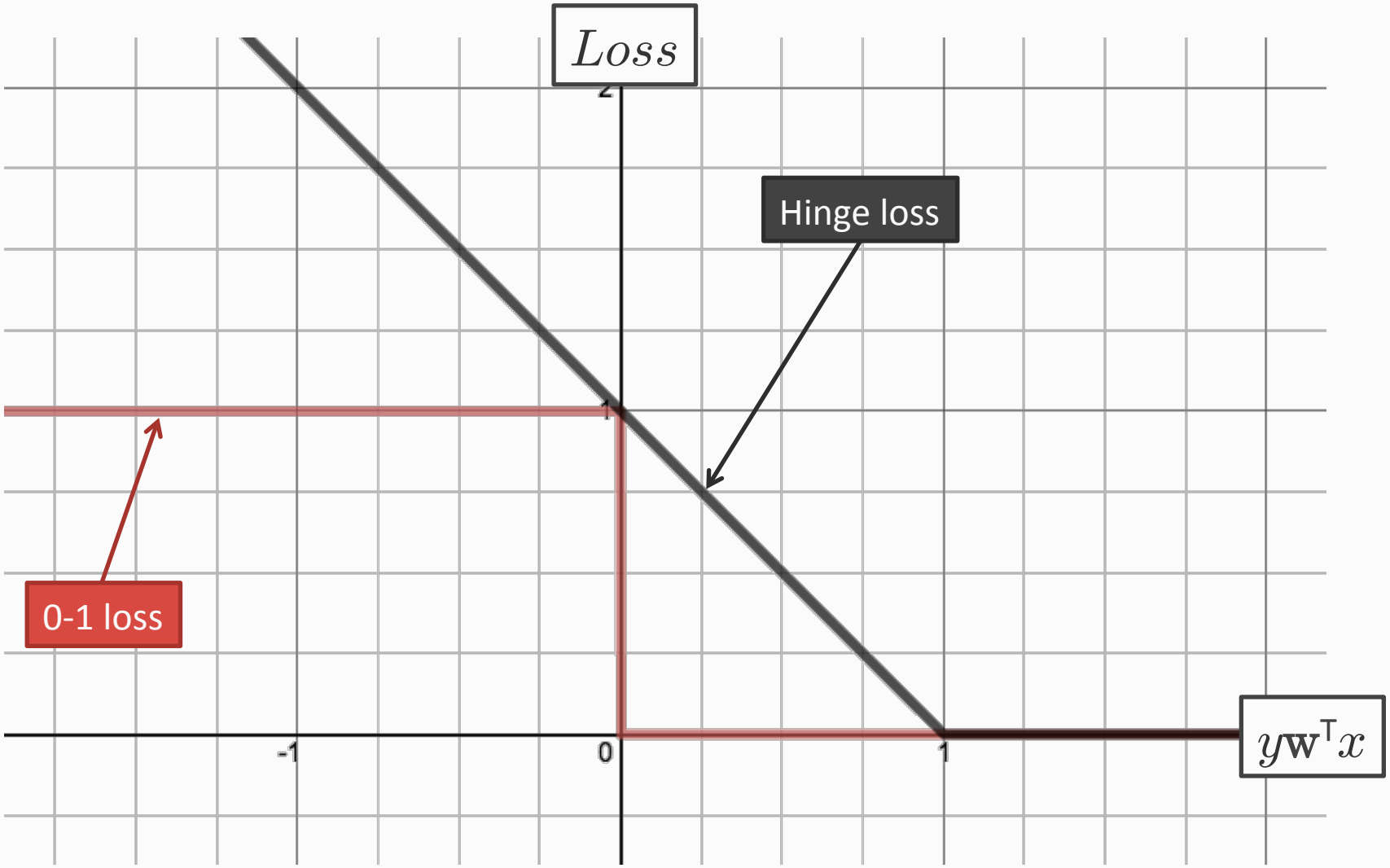
$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

The Hinge Loss



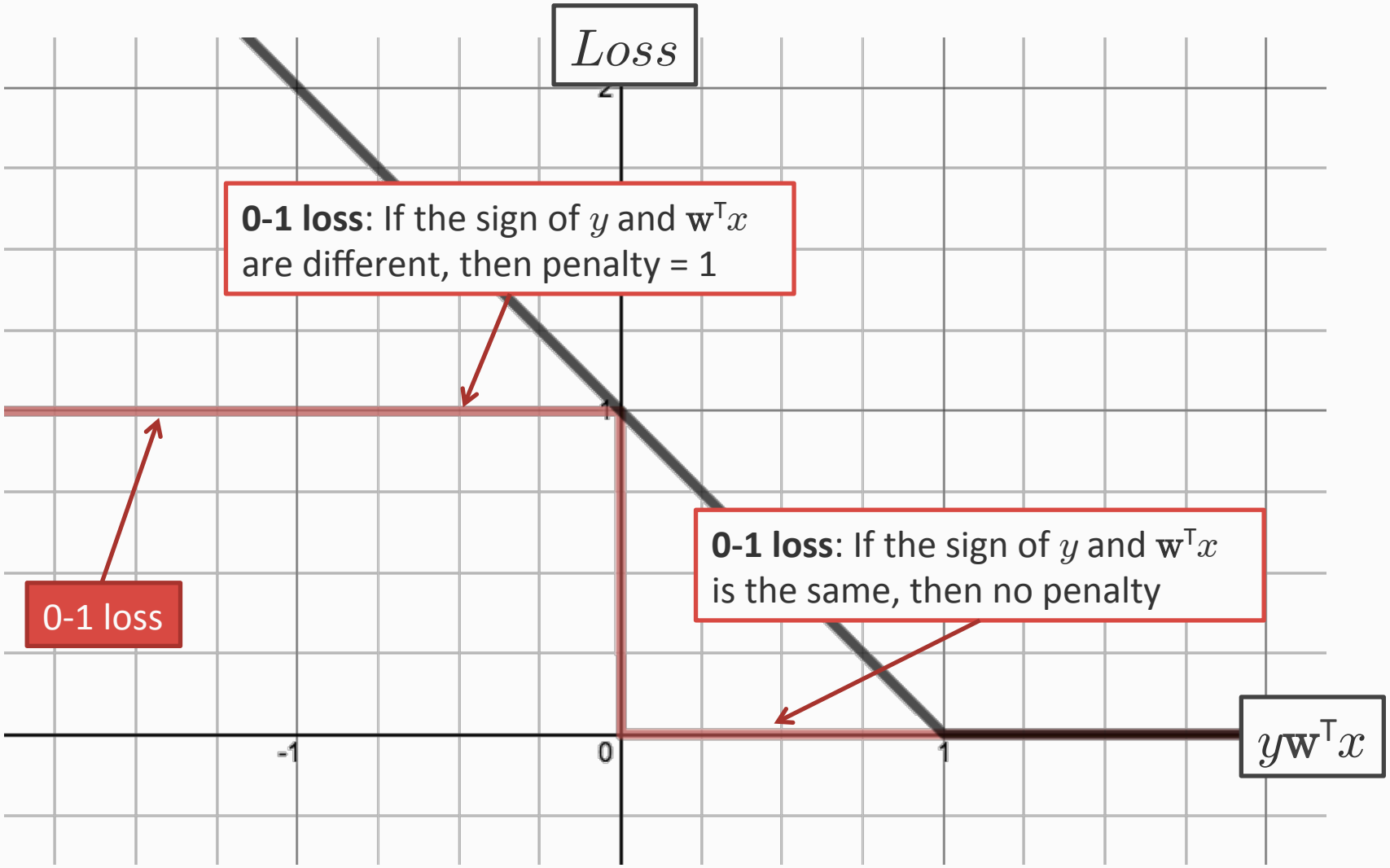
$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

The Hinge Loss



$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

The Hinge Loss

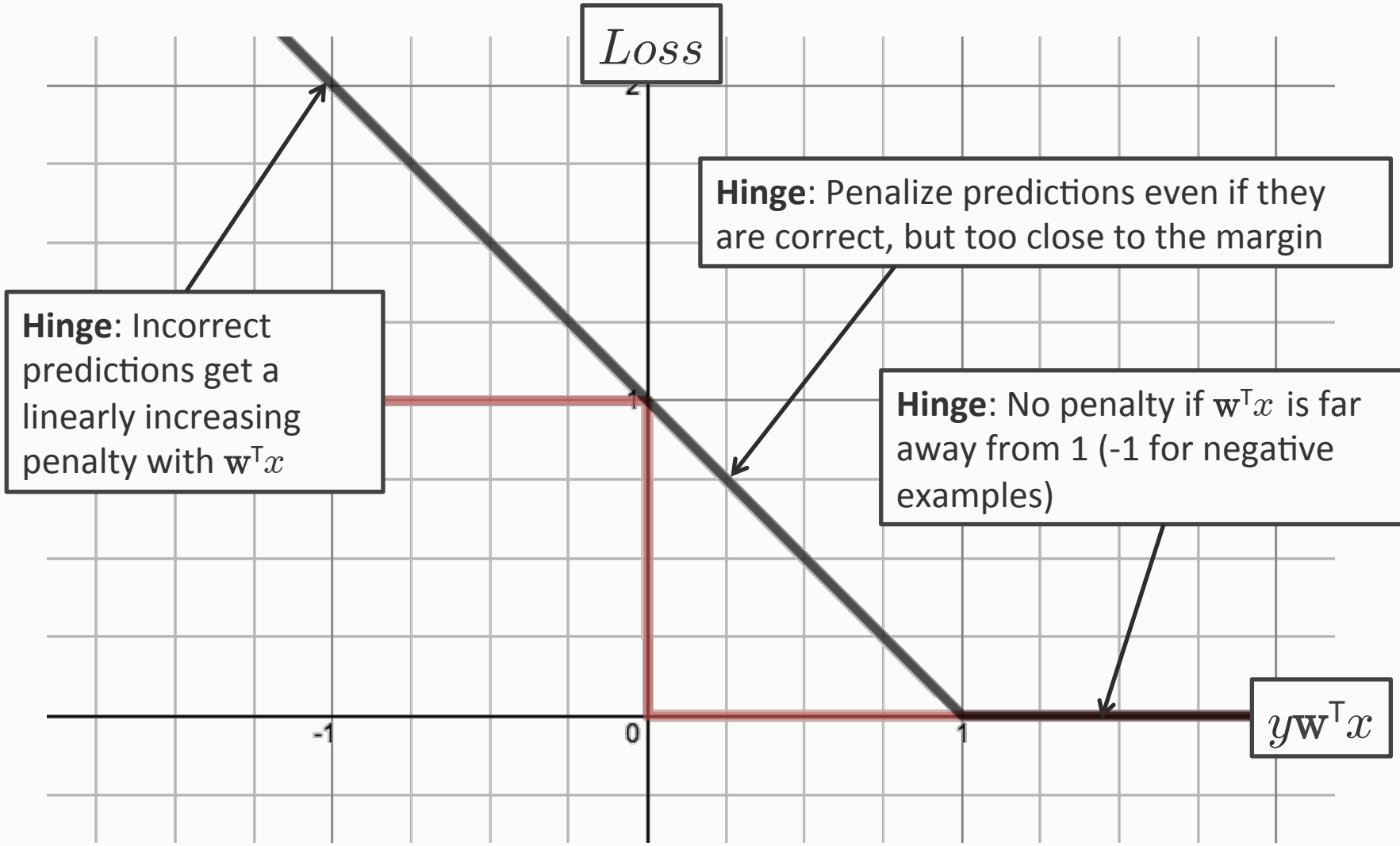


0-1 loss: If the sign of y and $\mathbf{w}^\mathsf{T}x$ are different, then penalty = 1

0-1 loss: If the sign of y and $\mathbf{w}^\mathsf{T}x$ is the same, then no penalty

$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

The Hinge Loss



Maximizing margin and minimizing loss

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

Maximize margin

Penalty for the prediction

Three cases

- Example is correctly classified and is outside the margin: penalty = 0
- Example is incorrectly classified: penalty = $1 - y_i\ \mathbf{w}^T\mathbf{x}_i$
- Example is correctly classified but within the margin: penalty = $1 - y_i\ \mathbf{w}^T\mathbf{x}_i$

General learning principle

Risk minimization

Define the notion of “loss” over the training data as a function of a hypothesis

Learning = find the hypothesis that has lowest loss on the training data

General learning principle

Regularized risk minimization

Define a regularization function that penalizes over-complex hypothesis.

Capacity control gives better generalization

Define the notion of "loss" over the training data as a function of a hypothesis

Learning =
find the hypothesis that has lowest
[Regularizer + loss on the training data]

SVM objective function

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

Regularization term:

- Maximize the margin
- Imposes a preference over the hypothesis space and pushes for better generalization
- Can be replaced with other regularization terms which impose other preferences

Empirical Loss:

- Hinge loss
- Penalizes weight vectors that make mistakes
- Can be replaced with other loss functions which impose other preferences

A hyper-parameter that controls the tradeoff between a large margin and a small hinge-loss

This lecture: Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

Solving the SVM optimization problem

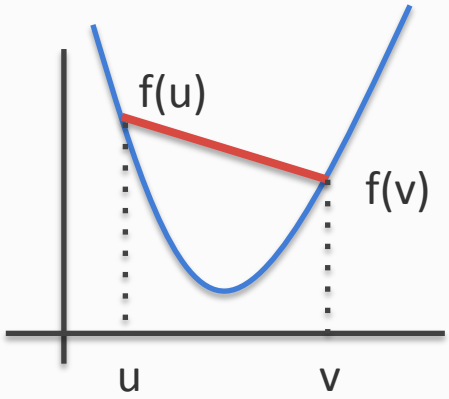
$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

This function is convex in $\mathbf{w}$

Detour: Convex functions

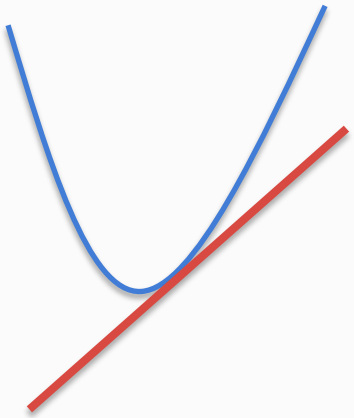
A function f is convex if for every u, v in the domain, and for every $\lambda \in [0,1]$ we have

$$f(\lambda u + (1-\lambda)v) \leq \lambda\, f(u) + (1-\lambda)\, f(v)$$



From geometric perspective

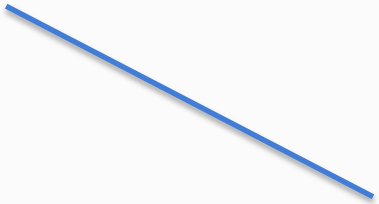
Every tangent plane lies below the function



Convex functions

$f(x) = -x$

Linear functions

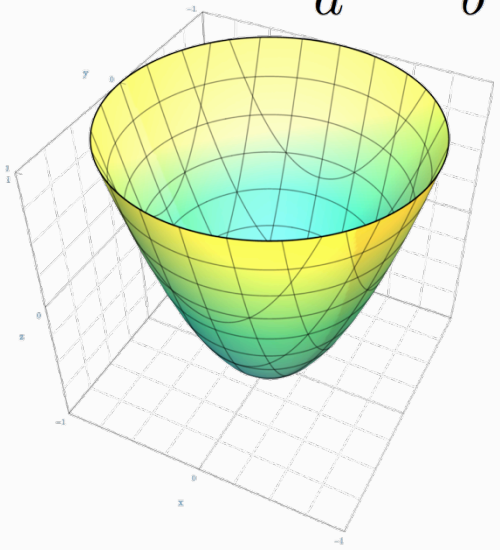


$f(x) = x^2$

$f(x) = \max(0, x)$

max is convex

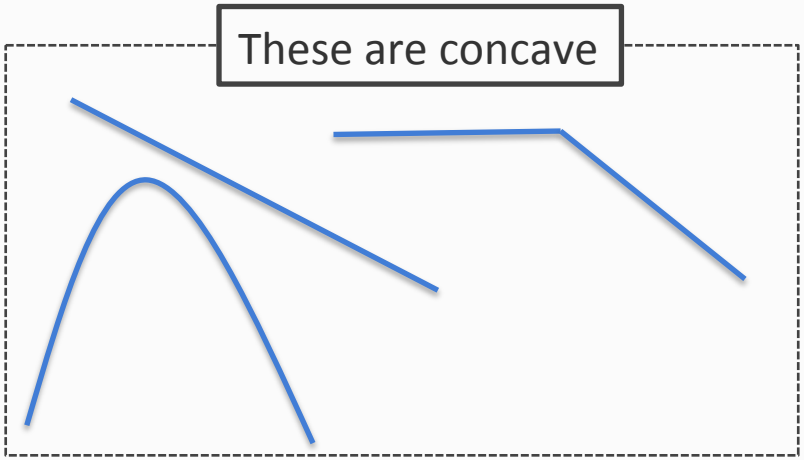
$$f(x_1, x_2) = \frac{x_1^2}{a^2} + \frac{x_2^2}{b^2}$$

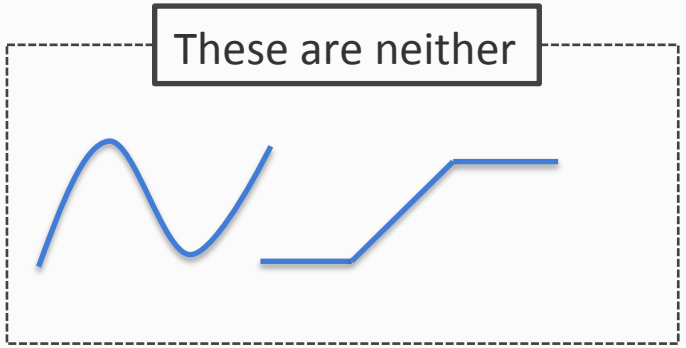


Some ways to show that a function is convex:

1. Using the definition of convexity
2. Showing that the second derivative is positive (for one dimensional functions)
3. Showing that the second derivative is positive semi-definite (for vector functions)

Not all functions are convex



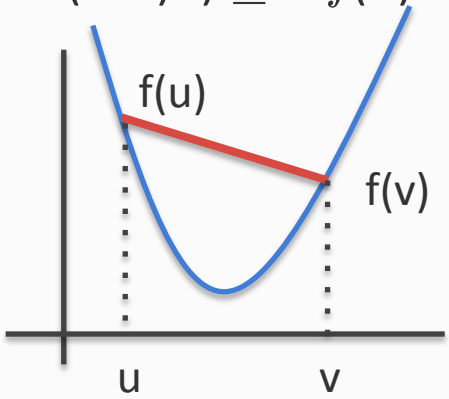


$f(\lambda u + (1-\lambda)v) \geq \lambda\, f(u) + (1-\lambda)\, f(v)$

Convex functions are convenient

A function f is convex if for every u, v in the domain, and for every $\lambda \in [0,1]$ we have

$$f(\lambda u + (1-\lambda)v) \leq \lambda\, f(u) + (1-\lambda)\, f(v)$$



In general: Necessary condition for x to be a minimum for the function f is ∇ f (x)= 0

For convex functions, this is both necessary *and* sufficient

Solving the SVM optimization problem

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i)$$

This function is convex in **w**

- This is a quadratic optimization problem because the objective is quadratic
- Older methods: Used techniques from Quadratic Programming
  - Very slow
- No constraints, can use *gradient descent*
  - Still very slow!

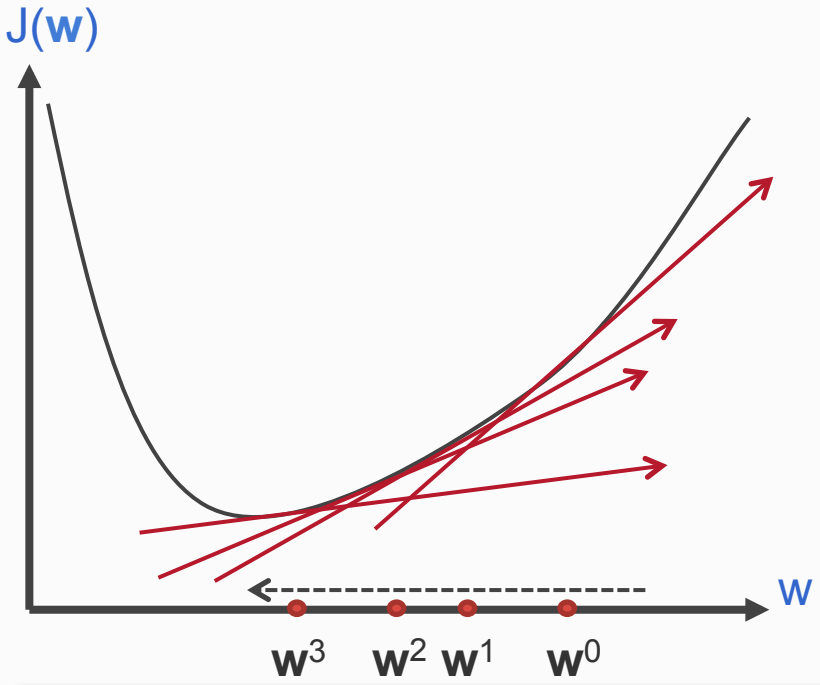
Gradient descent

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

General strategy for minimizing a function J(**w**)

- Start with an initial guess for **w**, say $\mathbf{w}^0$
- Iterate till convergence:
  - Compute the gradient of the gradient of J at $\mathbf{w}^t$
  - Update $\mathbf{w}^t$ to get $\mathbf{w}^{t+1}$ by taking a step in the opposite direction of the gradient



Intuition: The gradient is the direction of steepest increase in the function. To get to the minimum, go in the opposite direction

Gradient descent for SVM

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, ….
   1. Compute gradient of J($\mathbf{w}$) at $\mathbf{w}^t$. Call it ∇ J($\mathbf{w}^t$)
   2. Update w as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - r\nabla J(\mathbf{w}^t)$$

r: Called the learning rate .

Gradient descent for SVM

We are trying to minimize

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

1. Initialize $\mathbf{w}^0$
2. For t = 0, 1, 2, ….
   1. Compute gradient of J($\mathbf{w}$) at $\mathbf{w}^t$. Call it ∇ J($\mathbf{w}^t$)

Gradient of the SVM objective requires summing over the entire training set

Slow, does not really scale

r: Called the learning rate

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

Stochastic gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w}^0 = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Pick a random example $(\mathbf{x}_i, y_i)$ from the training set S
   2. Treat $(\mathbf{x}_i, y_i)$ as a full dataset and take the derivative of the SVM objective at the current $\mathbf{w}^{t-1}$ to be $\nabla J^t(\mathbf{w}^{t-1})$

$$J^t(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

   3. Update: $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \gamma_t \nabla J^t(\mathbf{w}^{t-1})$
3. Return final $\mathbf{w}$

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

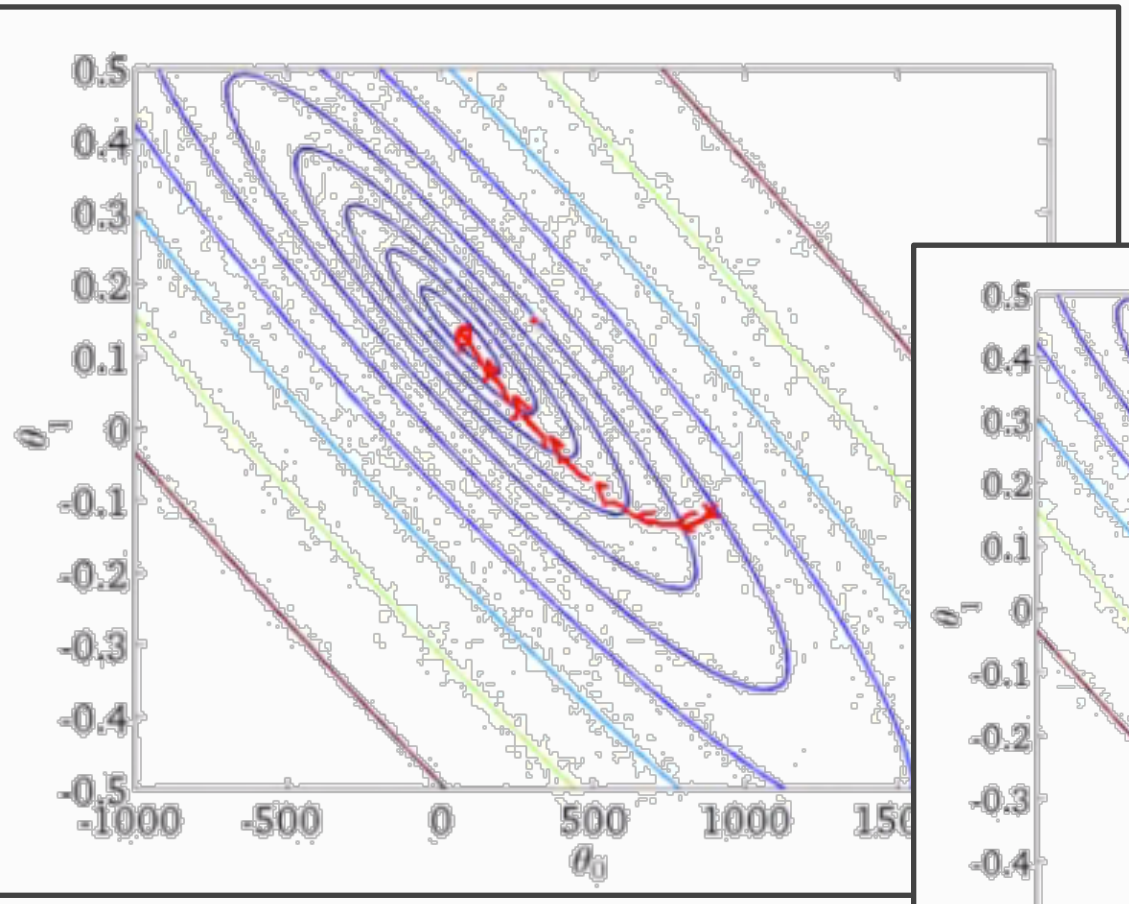
Stochastic gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w}^0 = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Pick a random example $(\mathbf{x}_i, y_i)$ from the training set S
   2. Treat $(\mathbf{x}_i, y_i)$ as a full dataset and take the derivative of the SVM objective at the current $\mathbf{w}^{t-1}$ to be $\nabla J^t(\mathbf{w}^{t-1})$
   3. Update: $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \gamma_t \nabla J^t(\mathbf{w}^{t-1})$
3. Return final $\mathbf{w}$

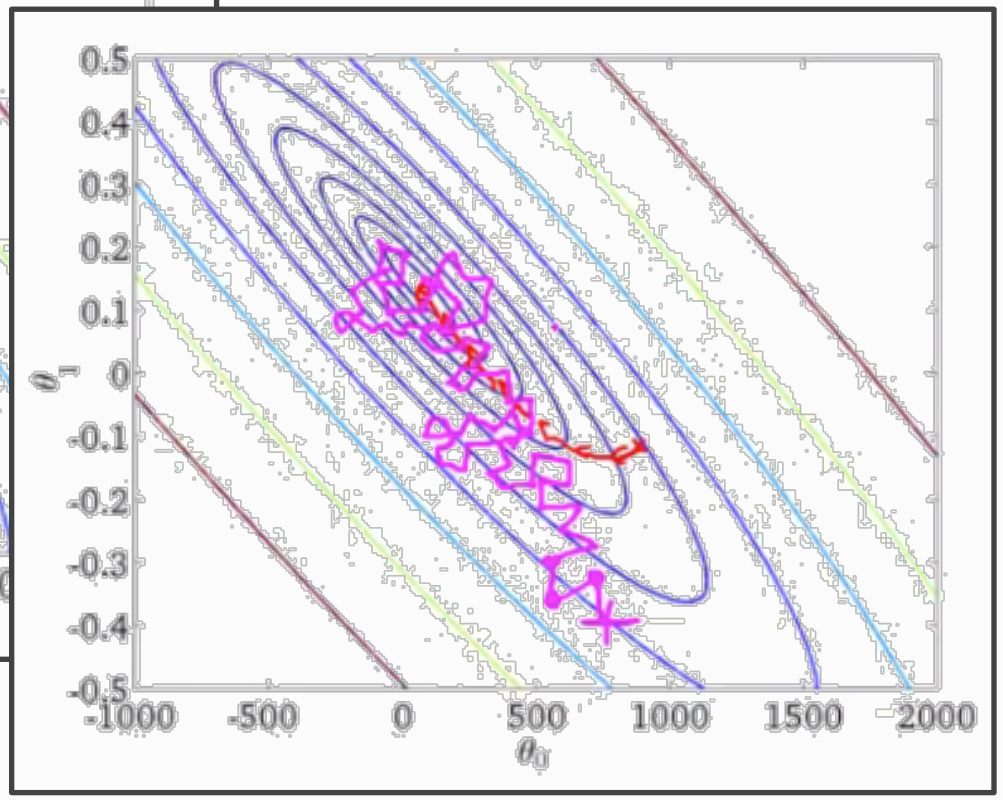
This algorithm is guaranteed to converge to the minimum of J if γ_t is small enough.
Why? The objective J(**w**) is a ***convex*** function

Gradient descent vs SGD



Gradient descent

Stochastic Gradient descent



[Images from Ng]

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

Stochastic gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w}^0 = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Pick a random example $(\mathbf{x}_i, y_i)$ from the training set S
   2. Treat $(\mathbf{x}_i, y_i)$ as a full dataset and take the *derivative of the SVM objective* at the current $\mathbf{w}^{t-1}$ to be $\nabla J^t(\mathbf{w}^{t-1})$
   3. Update: $\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} - \gamma_t \nabla J^t(\mathbf{w}^{t-1})$
3. Return final $\mathbf{w}$

What is the derivative of the hinge loss with respect to w?

(The hinge loss is not a differentiable function!)

Hinge loss is not differentiable!

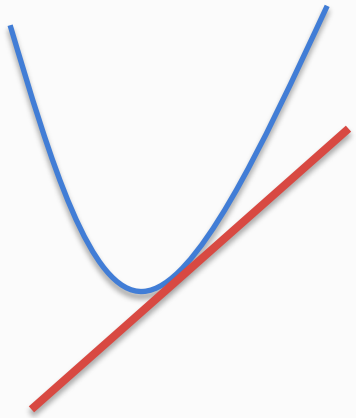
What is the derivative of the hinge loss with respect to w?

$$J^t(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

Another detour: Sub-gradients

Generalization of gradients to non-differentiable functions

(Remember that every tangent lies below the function for convex functions)



Informally, a sub-tangent at a point is any line lies below the function at the point.

A sub-gradient is the slope of that line

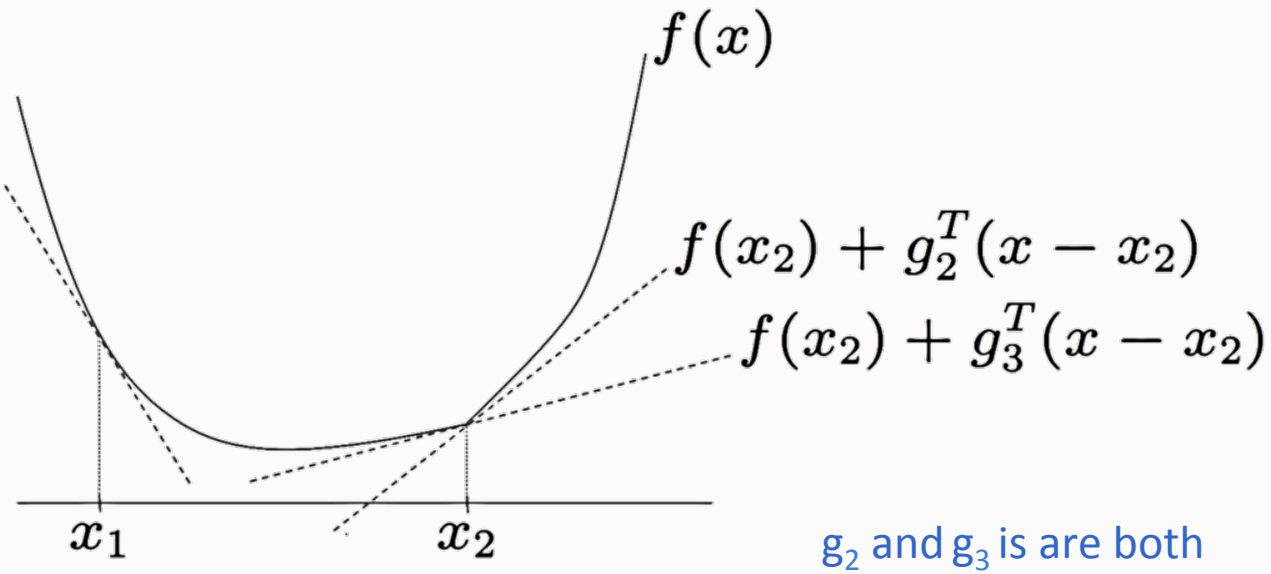
Sub-gradients

Formally, g is a subgradient to f at x if

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y$$

f is differentiable at x_1
Tangent at this point

$f(x_1) + g_1^T(x - x_1)$



g_1 is a gradient at x_1

g_2 and g_3 is are both subgradients at x_2

[Example from Boyd]

Sub-gradient of the SVM objective

$$J^t(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right)$$

General strategy: First solve the max and compute the gradient for each case

$$\nabla J^t = \begin{cases} \mathbf{w} & \text{if } \max\left(0, 1 - y_i\mathbf{w}^T\mathbf{x}_i\right) = 0 \\ \mathbf{w} - Cy_i\mathbf{x}_i & \text{otherwise} \end{cases}$$

Stochastic sub-gradient descent for SVM

$$\nabla J^t = \begin{cases} \mathbf{w} & \text{if } \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) = 0 \\ \mathbf{w} - C y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$

3. Return $\mathbf{w}$

Stochastic sub-gradient descent for SVM

$$\nabla J^t = \begin{cases} \mathbf{w} & \text{if } \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) = 0 \\ \mathbf{w} - C y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:

3. Return $\mathbf{w}$

Stochastic sub-gradient descent for SVM

$$\nabla J^t = \begin{cases} \mathbf{w} & \text{if } \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) = 0 \\ \mathbf{w} - C y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

Given a training set S = $\{(\mathbf{x}_i, \mathrm{y}_i)\}$, $\mathbf{x} \in \Re^n$, $\mathrm{y} \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}_i, \mathrm{y}_i) \in$ S:

      Update $\mathbf{w} \leftarrow \mathbf{w} - \gamma_t \nabla J^t$

3. Return $\mathbf{w}$

Stochastic sub-gradient descent for SVM

$$\nabla J^t = \begin{cases} \mathbf{w} & \text{if } \max\left(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\right) = 0 \\ \mathbf{w} - C y_i \mathbf{x}_i & \text{otherwise} \end{cases}$$

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
    1. For each training example $(\mathbf{x}_i, y_i) \in$ S:

       If $y_i \, \mathbf{w}^T \mathbf{x}_i \leq 1$,

       $\mathbf{w} \leftarrow (1 - \gamma_t) \, \mathbf{w} + \gamma_t \, C \, y_i \, \mathbf{x}_i$

       else

       $\mathbf{w} \leftarrow (1 - \gamma_t) \, \mathbf{w}$
3. Return $\mathbf{w}$

Stochastic sub-gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. For each training example $(\mathbf{x}_i, y_i) \in$ S:

      If $y_i \, \mathbf{w}^T \mathbf{x}_i \leq 1$,

      $\mathbf{w} \leftarrow (1 - \gamma_t) \, \mathbf{w} + \gamma_t \, C \, y_i \, \mathbf{x}_i$

      else

      $\mathbf{w} \leftarrow (1 - \gamma_t) \, \mathbf{w}$
3. Return $\mathbf{w}$

γ_t: learning rate, many tweaks possible

Important to shuffle examples at the start of each epoch

Stochastic sub-gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in$ S:

      If $y_i \mathbf{w}^T \mathbf{x}_i \leq 1$,

      $\mathbf{w} \leftarrow (1- \gamma_t)\ \mathbf{w} + \gamma_t\, C\, y_i\, \mathbf{x}_i$

      else

      $\mathbf{w} \leftarrow (1- \gamma_t)\ \mathbf{w}$
3. Return $\mathbf{w}$

γ_t: learning rate, many tweaks possible

Convergence and learning rates

With enough iterations, it will converge in expectation

Provided the step sizes are "*square summable, but not summable*"

- Step sizes γ_t are positive
- Sum of squares of step sizes over t = 1 to ∞ is not infinite
- Sum of step sizes over t = 1 to ∞ is infinity

- Example: $$\gamma_t = \frac{\gamma_0}{1 + \gamma_0 t / C}$$

Convergence and learning rates

- Number of iterations to get to accuracy within ϵ
- For strongly convex functions, N examples, d dimensional:
  - Gradient descent: O(Nd ln(1/ϵ))
  - Stochastic gradient descent: O(d/ϵ)
- More subtleties involved, but SGD is generally preferable when the data size is huge

Stochastic sub-gradient descent for SVM

Given a training set S = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x} \in \Re^n$, $y \in \{-1,1\}$

1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in$ S:

      If $y_i \, \mathbf{w}^T \mathbf{x}_i \leq 1$,

      $\mathbf{w} \leftarrow (1\text{-}\gamma_t) \, \mathbf{w} + \gamma_t \, C \, y_i \, \mathbf{x}_i$

      else

      $\mathbf{w} \leftarrow (1\text{-}\gamma_t) \, \mathbf{w}$
3. Return $\mathbf{w}$

Compare with the Perceptron update:
If $y \, \mathbf{w}^T \mathbf{x} \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + r \, y \, \mathbf{x}$

Perceptron vs. SVM

- Perceptron: Stochastic sub-gradient descent for a different loss
  - No regularization though

$$L_{Perceptron}(y, \mathbf{x}, \mathbf{w}) = \max(0, -y\mathbf{w}^T\mathbf{x})$$

- SVM optimizes the hinge loss
  - With regularization

$$L_{Hinge}(y, \mathbf{x}, \mathbf{w}) = \max(0, 1 - y\mathbf{w}^T\mathbf{x})$$

SVM summary from optimization perspective

- Minimize regularized hinge loss
- Solve using stochastic gradient descent
  - Very fast, run time does not depend on number of examples
  - Compare with Perceptron algorithm: Perceptron does not maximize margin width
    - Perceptron variants can force a margin
  - Convergence criterion is an issue; can be too aggressive in the beginning and get to a reasonably good solution fast; but convergence is slow for very accurate weight vector
- Other successful optimization algorithms exist
  - Eg: Dual coordinate descent, implemented in `liblinear`

Questions?

This lecture: Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

almost

And now for something completely different

So far we have seen

- Support vector machines

- Hinge loss and optimizing the regularized loss

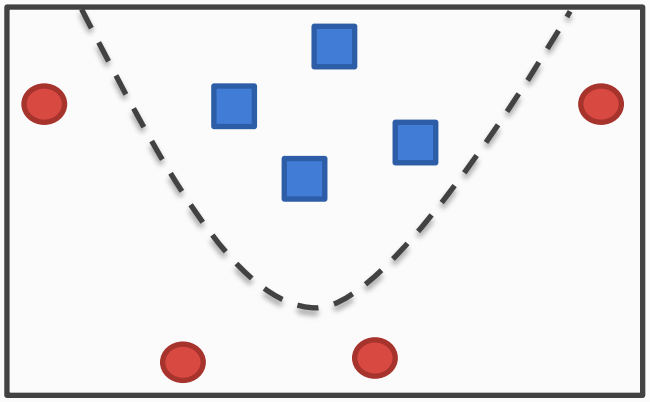
More broadly, different algorithms for learning linear classifiers

What about non-linear models?

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using $\mathbf{w}^T\phi(x_1, x_2) \geq b$

SVM: Primals and duals

The SVM objective

$$\min_{\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i$$

$$\text{s.t.} \quad \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i$$

$$\forall i, \quad \xi_i \geq 0.$$

This is called the *primal form* of the objective

This can be converted to its *dual form*, which will let us prove a very useful property

Another optimization problem

Has the property that max Dual = min Primal

Support vector machines

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ \text{s.t.} \quad & \forall i, \quad y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i \\ & \forall i, \quad \xi_i \geq 0. \end{aligned}$$

Let **w** be the minimizer of the SVM problem for some dataset with m examples: {($\mathbf{x}_i$, y_i)}

Then, for i = 1…m exist $\alpha_i \geq 0$ such that the optimum w can be written as

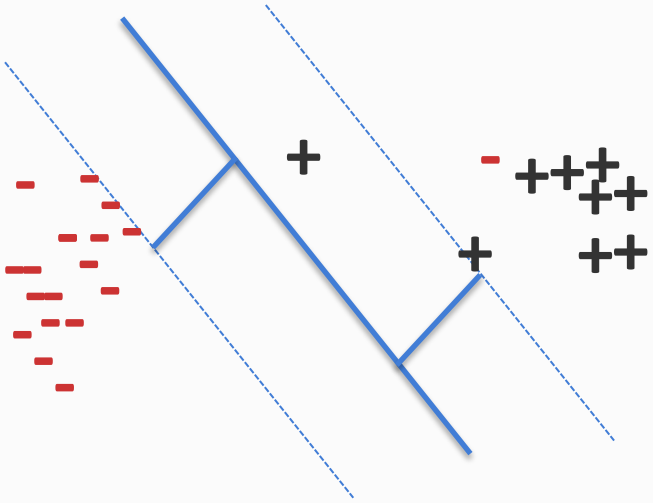
$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for i = 1…m exist $\alpha_i \geq 0$ such that the optimum w can be written as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$



Furthermore,

$$\alpha_i = 0 \qquad \Rightarrow y_i \mathbf{w}^T x_i \geq 1$$

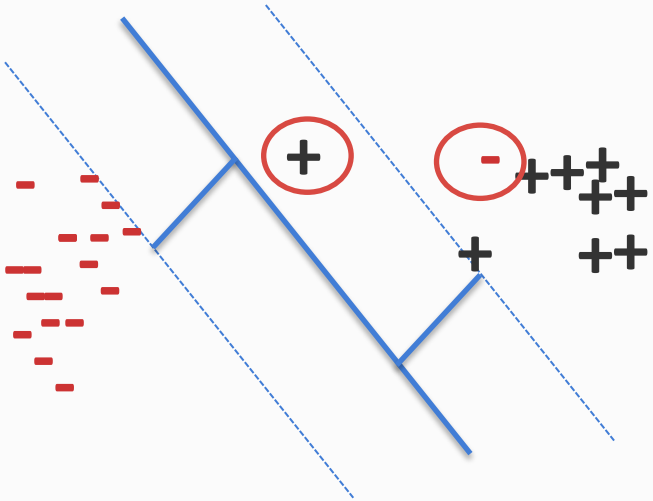
All points outside the margin

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for i = 1…m exist $\alpha_i \geq 0$ such that the optimum w can be written as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$



Furthermore,

$$\alpha_i = 0 \qquad \Rightarrow y_i \mathbf{w}^T x_i \geq 1$$

$$\alpha_i = C \qquad \Rightarrow y_i \mathbf{w}^T x_i \leq 1$$

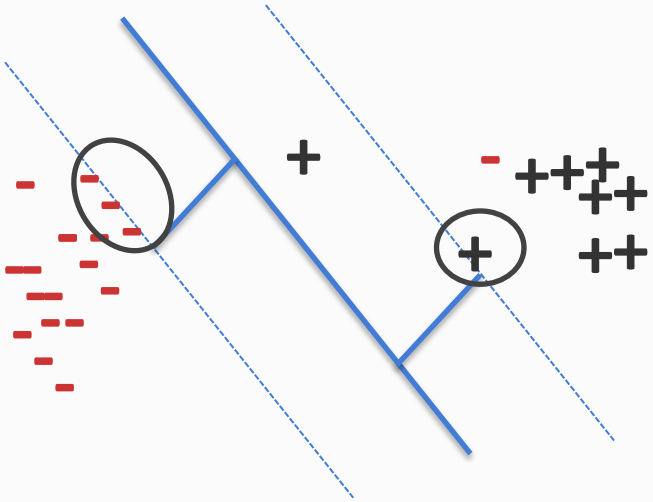
All points on the wrong side of the margin

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for i = 1…m exist $\alpha_i \geq 0$ such that the optimum w can be written as

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$



Furthermore,

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i \mathbf{w}^T x_i \geq 1 \\ \alpha_i = C &\Rightarrow y_i \mathbf{w}^T x_i \leq 1 \\ 0 \leq \alpha_i \leq C &\Rightarrow y_i \mathbf{w}^T x_i = 1 \end{aligned}$$

All points on the margin

Support vectors

The weight vector is completely defined by training examples whose α_is are not zero

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

These examples are called the *support vectors*

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- But we just showed that $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i^T\mathbf{x}$
  - That is we only need to compute dot products between training examples and the new example **x**

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- But we just showed that $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i^T\mathbf{x}$
  - That is we only need to compute dot products between training examples and the new example **x**
- This is true even if we map examples to a high dimensional space

$$\mathbf{w}^T\phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x})$$

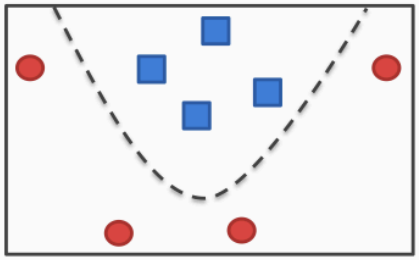
Predicting with linear classifiers

- Predi
- But w
  - Tha
    tra
- This i
  dime

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using $\mathbf{w}^T\phi(x_1, x_2) \geq b$

$$\mathbf{w}^T\phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$$

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

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

So prediction with this *high dimensional lifting map* is

$$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

Kernel based methods

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

$$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

What does this new formulation give us?

If we have to compute ϕ every time anyway, we gain nothing

If we can compute the value of K without explicitly writing the blown up representation, then we will have a computational advantage

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example- quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2]^T$$

and compute the dot product A = $\phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Theorem: A = B (Coefficients do not really matter)

Kernels: Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?
  - No! A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

- General condition: construct the Gram matrix $\{K(\mathbf{x}_i, \mathbf{z}_j)\}$; check that it's positive semi definite

Detour: Positive semi-definite matrices

A symmetric matrix M is positive semi-definite if it is

- For any vector non-zero **z**, we have $\mathbf{z}^T M \mathbf{z} \geq 0$

(A useful property characterizing many interesting mathematical objects)

The Kernel Matrix

- The Gram matrix of a set of n vectors $S = \{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is the $n \times n$ matrix $\mathbf{G}$ with $\mathbf{G}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
  - The kernel matrix is the Gram matrix of $\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$
  - (size depends on the # of examples, not dimensionality)

- Showing that a function K is a valid kernel
  - Direct approach: If you have the $\phi(\mathbf{x}_i)$, you have the Gram matrix (and it's easy to see that it will be positive semi-definite). *Why?*
  - Indirect: If you have the Kernel, write down the Kernel matrix K_{ij}, and show that it is a legitimate kernel, without an explicit construction of $\phi(\mathbf{x}_i)$

Mercer's condition

Let K(**x**, **z**) be a function that maps two n dimensional vectors to a real number

K is a valid kernel if for every finite set {x_1, x_2, $\cdots$ }, for any choice of real valued c_1, c_2, $\cdots$, we have

$$\sum_i \sum_j c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Polynomial kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$
- Polynomial kernel of degree d: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$
  - only dth-order interactions
- Polynomial kernel up to degree d: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$ ($c>0$)
  - all interactions of order d or lower

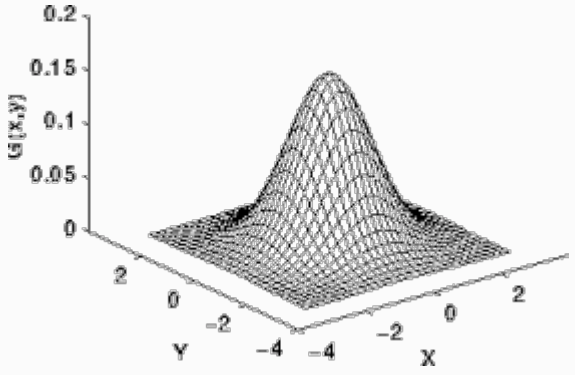
Gaussian Kernel

(aka radial basis function kernel)

$$K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{c}\right)$$

- $(x - z)^2$: squared Euclidean distance between **x** and **z**
- $c = \sigma^2$: a free parameter
- very small c: K ≈ identity matrix (every item is different)
- very large c: K ≈ unit matrix (all items are the same)

- k(**x**, **z**) ≈ 1 when **x**, **z** close
- k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



Exercises:

1. Prove that this is a kernel.
2. What is the "blown up" feature space for this kernel?

Constructing New Kernels

You can construct new kernels $k'(\mathbf{x}, \mathbf{x'})$ from existing ones:

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a constant c

  $ck(\mathbf{x}, \mathbf{x'})$

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a function f applied to $\mathbf{x}$ and $\mathbf{x'}$

  $f(\mathbf{x})k(\mathbf{x}, \mathbf{x'})f(\mathbf{x'})$

- Applying a polynomial (with non-negative coefficients) to $k(\mathbf{x}, \mathbf{x'})$

  $P(\ k(\mathbf{x}, \mathbf{x'})\ )$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq 0$

- Exponentiating $k(\mathbf{x}, \mathbf{x'})$

  $\exp(k(\mathbf{x}, \mathbf{x'}))$

Constructing New Kernels (2)

- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$ by:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$
- Also:
  - If $\phi(\mathbf{x}) \in R^m$ and $k_m(\mathbf{z}, \mathbf{z'})$ a valid kernel in R^m,
    $k(\mathbf{x}, \mathbf{x}') = k_m(\phi(\mathbf{x}), \phi(\mathbf{x}'))$ is also a valid kernel
  - If $\mathbf{A}$ is a symmetric positive semi-definite matrix,
    $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}\mathbf{A}\mathbf{x}'$ is also a valid kernel

Kernel Trick: An example

Consider the space of all 3^n monomials (allowing both positive and negative literals). Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have monomials of size 0,1,2,3

Proof: let m=same(**x**, **z**); construct "surviving" monomials by

1. choosing to include one of these k literals with the right polarity in the monomial, or
2. choosing to not include it at all.

Monomials with literals outside this set disappear.

Exercise

- Take $X=\{x_1, x_2, x_3, x_4\}$
- $\phi(\mathbf{x})$ = The space of all 3^n monomials; $|\phi(\mathbf{x})| = 81$
- Consider $\mathbf{x}=(1100)$, $\mathbf{z}=(1101)$
- Write $\phi(\mathbf{x})$, $\phi(\mathbf{z})$, the representation of $\mathbf{x}$, $\mathbf{z}$ in the ϕ space
- Compute $\phi(\mathbf{x})^T\phi(\mathbf{z})$
- Show that

$$K(x,z) = \phi(\mathbf{x})^T\phi(\mathbf{z}) = \sum_i \phi_i(z)\,\phi_i(x) = 2^{\text{same}(\mathbf{x},\mathbf{z})} = 8$$

- Try to develop another kernel, e.g., where the space of all conjunctions of size 3 (exactly)

Summary: Kernel trick

- To make the final prediction, we are computing dot products
- The kernel trick is a computational trick to compute dot products in higher dimensional spaces
- This is applicable not just to SVMs. The same idea can be extended to Perceptron too: the Kernel Perceptron
- Important: All the bounds we have seen (eg: Perceptron bound, etc) depend on the underlying dimensionality
  - By moving to a higher dimensional space, we are incurring a penalty on sample complexity