Perceptron: Geometric Margins

Kernels Methods

Support Vector Machines (SVMs)

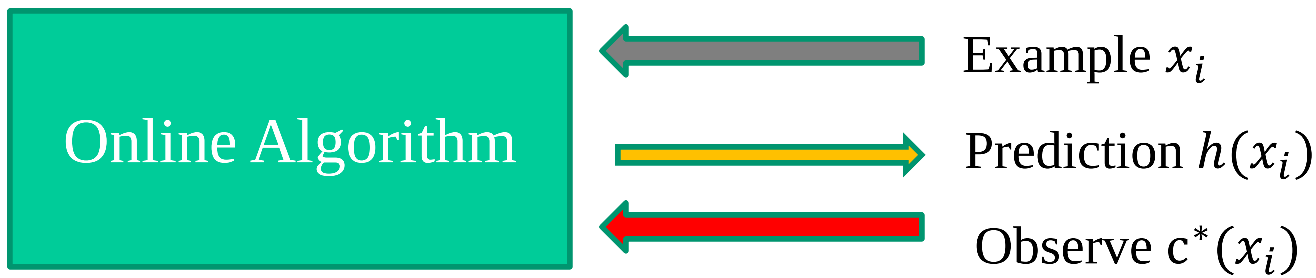
Perceptron: Geometric Margins

The Online Learning Model: Perceptron Algorithm

- Examples arrive sequentially.
- We need to make a prediction.
  Afterwards observe the outcome.

For i=1, 2, …, :

Phase i:

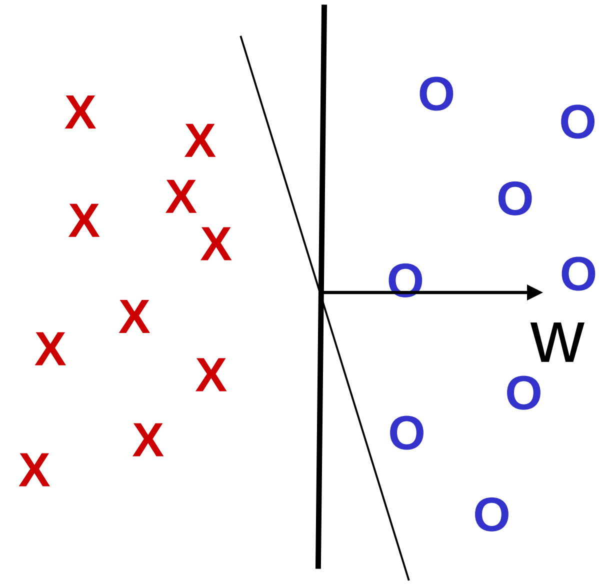


Mistake bound model

- Analysis wise, make no distributional assumptions.
- **Goal**: Minimize the number of mistakes.

Linear Separators

- Instance space $X = R^d$
- Hypothesis class of linear decision surfaces in R^d.



- $h(x) = w \cdot x + w_0$, if $h(x) \geq 0$, then label x as $+$, otherwise label it as $-$

Claim: WLOG $w_0 = 0$.

Proof: Can simulate a non-zero threshold with a dummy input feature x_0 that is always set up to 1.

- $x = (x_1, \ldots, x_d) \rightarrow \tilde{x} = (x_1, \ldots, x_d, 1)$
- $w \cdot x + w_0 \geq 0$ iff $(w_1, \ldots, w_d, w_0) \cdot \tilde{x} \geq 0$

where $w = (w_1, \ldots, w_d)$

Linear Separators: Perceptron Algorithm

- Set t=1, start with the all zero vector w_1.
- Given example x, predict positive iff $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, then update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, then update $w_{t+1} \leftarrow w_t - x$

Note: w_t is weighted sum of incorrectly classified examples

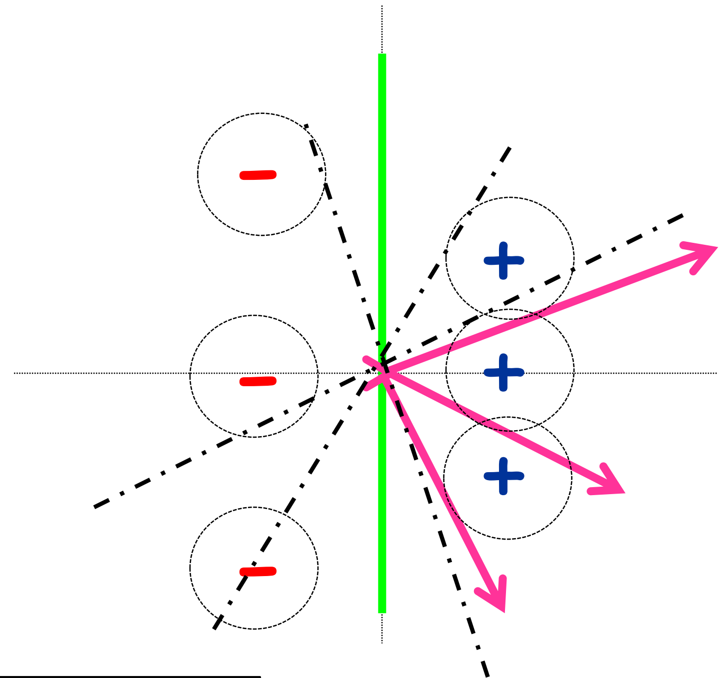
$$w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$$

$$w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \cdots + a_{i_k} x_{i_k} \cdot x$$

Perceptron Algorithm: Example

Example:

$(-1,2)\ -$ ✗
$(1,0)\ +$ ✓
$(1,1)\ +$ ✗
$(-1,0)\ -$ ✓
$(-1,-2)\ -$ ✗
$(1,-1)\ +$ ✓



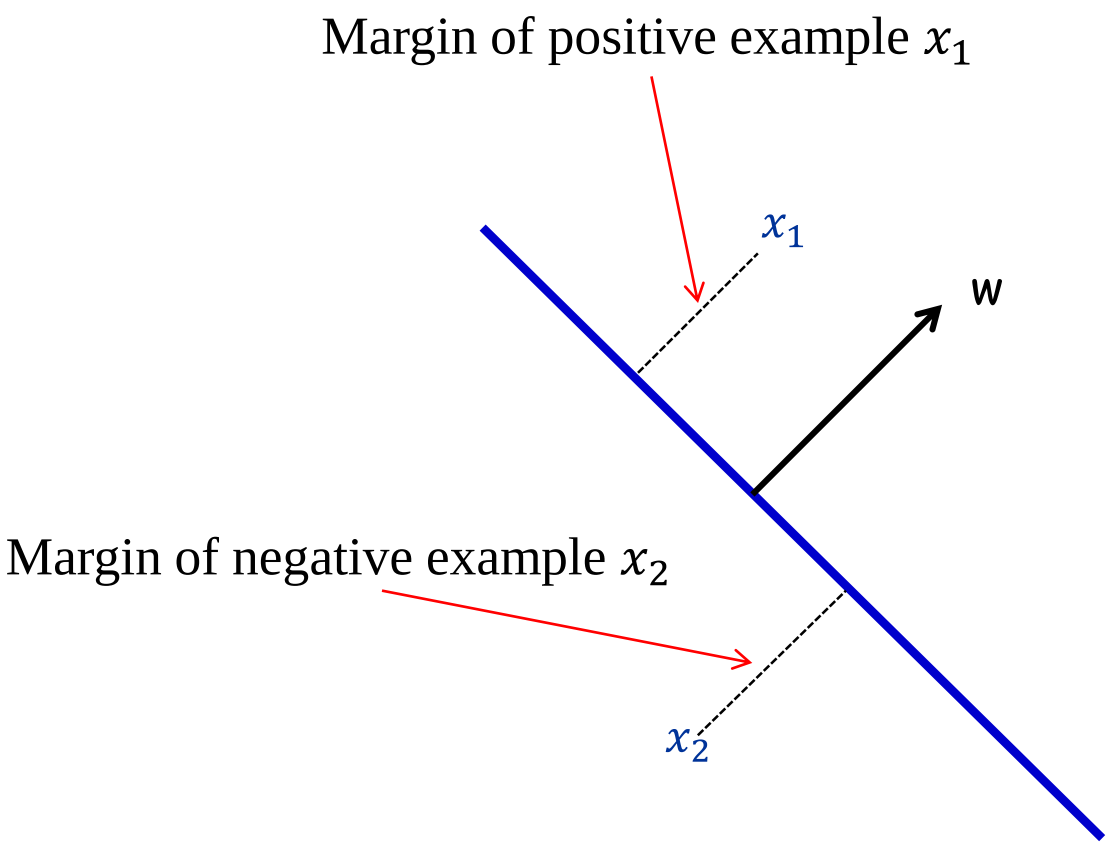
$w_1 = (0,0)$
$w_2 = w_1 - (-1,2) = (1,-2)$
$w_3 = w_2 + (1,1) = (2,-1)$
$w_4 = w_3 - (-1,-2) = (3,1)$

Algorithm:

- Set t=1, start with all-zeroes weight vector w_1.
- Given example x, predict positive iff $w_t \cdot x \geq 0$.
- On a mistake, update as follows:
    - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
    - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

Geometric Margin

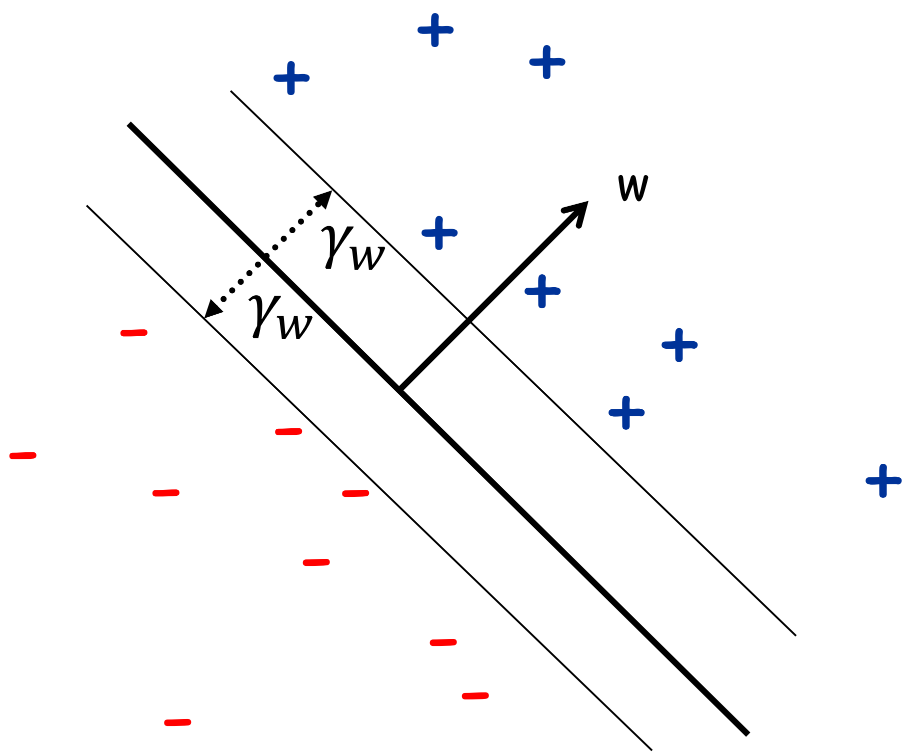
Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)



Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

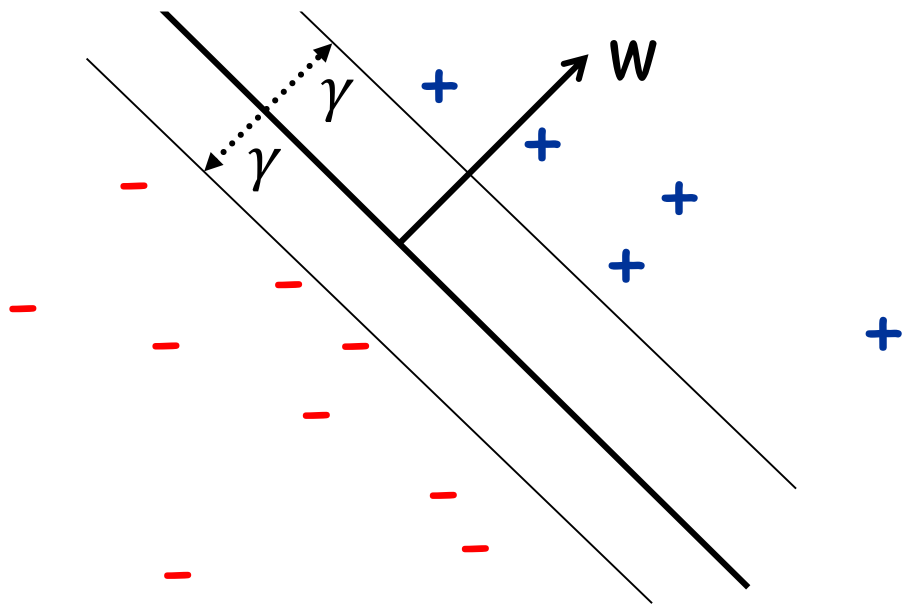


Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

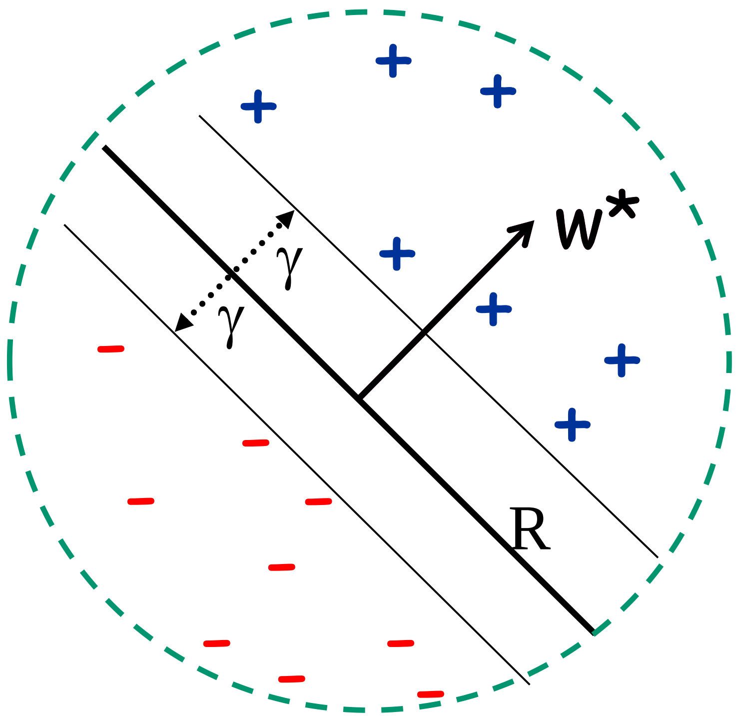
Definition: The margin γ of a set of examples S is the maximum γ_w over all linear separators w.



Perceptron: Mistake Bound

Theorem: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algorithm is invariant to scaling.)



Perceptron Algorithm: Analysis

Theorem: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

Update rule:

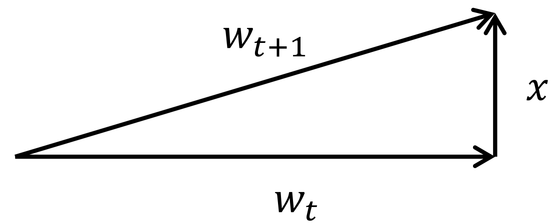
- Mistake on positive: $w_{t+1} \leftarrow w_t + x$
- Mistake on negative: $w_{t+1} \leftarrow w_t - x$

Proof:

Idea: analyze $w_t \cdot w^*$ and $\|w_t\|$, where w^* is the max-margin sep, $\|w^*\| = 1$.

Claim 1: $w_{t+1} \cdot w^* \geq w_t \cdot w^* + \gamma$. (because $l(x)x \cdot w^* \geq \gamma$)

Claim 2: $\|w_{t+1}\|^2 \leq \|w_t\|^2 + R^2$. (by Pythagorean Theorem)

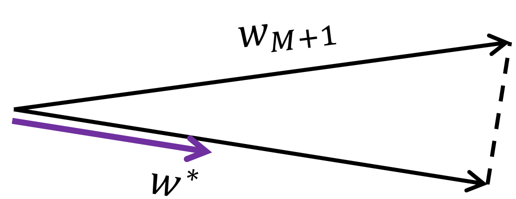


After M mistakes:

$w_{M+1} \cdot w^* \geq \gamma M$ (by Claim 1)

$\|w_{M+1}\| \leq R\sqrt{M}$ (by Claim 2)

$w_{M+1} \cdot w^* \leq \|w_{M+1}\|$ (since w^* is unit length)



So, $\gamma M \leq R\sqrt{M}$, so $M \leq \left(\frac{R}{\gamma}\right)^2$.

Perceptron Extensions

- Can use it to find a consistent separator (by cycling through the data) with a given set S linearly separable by margin γ (by cycling through the data).
- One can convert the mistake bound guarantee into a distributional guarantee too (for the case where the x_is come from a fixed distribution).
- Can be adapted to the case where there is no perfect separator as long as the so called hinge loss (*i.e., the total distance needed to move the points to classify them correctly large margin*) is small.
- Can be kernelized to handle non-linear decision boundaries!

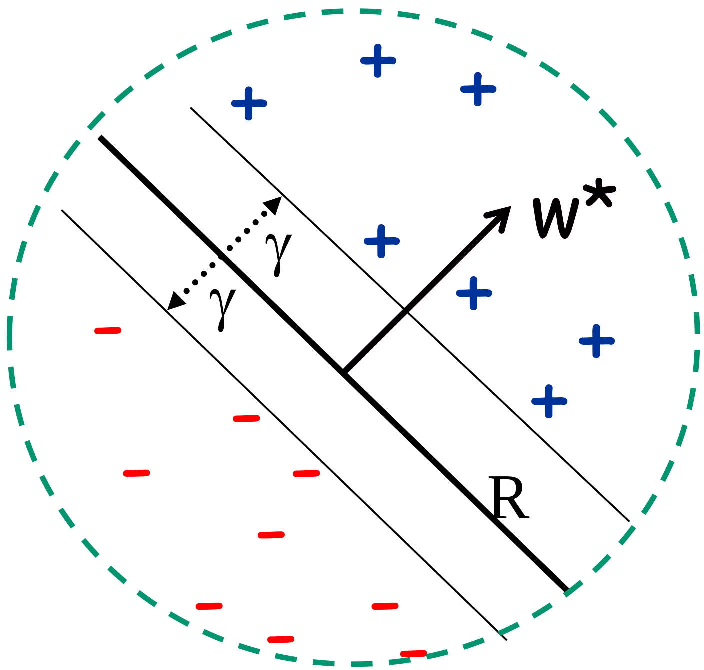
Perceptron Discussion

- Simple online algorithm for learning linear separators with a nice guarantee that depends only on the geometric (aka L_2, L_2) margin.
- It can be kernelized to handle non-linear decision boundaries
- Simple, but very useful in applications like Branch prediction; it also has interesting extensions to structured prediction.

Perceptron: Mistake Bound

Theorem: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



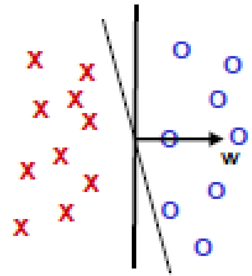
Margin: the amount of wiggle-room available for a solution.

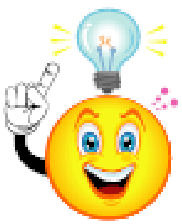
Implies that large margin classifiers have smaller complexity!

Complexity of Large Margin Linear Sep.

- Know that in $\mathbb{R}^n$ we can shatter $n+1$ points with linear separators, but not $n+2$ points (VC-dim of linear sep is $n+1$).

What if we require that the points be linearly separated by margin γ?





Can have at most $\left(\frac{R}{\gamma}\right)^2$ points inside ball of radius R that can be shattered at margin γ (meaning that every labeling is achievable by a separator of margin γ).

- So, large margin classifiers have smaller complexity!
  - Nice implications for usual distributional learning setting.
  - Less classifiers to worry about that will look good over the sample, but bad over all….
- Less prone to overfitting!!!!

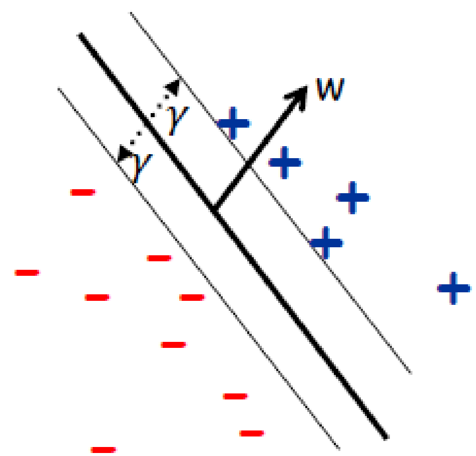
Margin Important Theme in ML

Both sample complexity and algorithmic implications.

Sample/Mistake Bound complexity:

- If large margin, # mistakes Peceptron makes is small (independent on the dim of the space)!

- If large margin γ and if alg. produces a large margin classifier, then amount of data needed depends only on R/γ [Bartlett & Shawe-Taylor '99].
  - Suggests searching for a large margin classifier…



Algorithmic Implications:

- Perceptron, Kernels, SVMs…

What if Dataset is Not Linearly Separable

Problem: data not linearly separable in the most natural feature representation.

Example:  vs  No good linear separator in pixel representation.

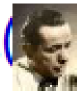
Solutions:

- "Learn a more complex class of functions"
  - (e.g., decision trees, neural networks, boosting).
- "Use a Kernel" (a neat solution that attracted a lot of attention)
- "Use a Deep Network"
- "Combine Kernels and Deep Networks"

Kernels Methods

Overview of Kernel Methods

What is a Kernel?

A kernel K is a legal def of dot-product: i.e. there exists an implicit mapping Φ s.t. K( , ) =Φ()$\cdot$ Φ()

E.g., $K(x,y) = (x \cdot y + 1)^d$

ϕ: (n-dimensional space) $\rightarrow$ n^d-dimensional space

Why Kernels matter?

- Many algorithms interact with data only via dot-products.
- So, if replace $x \cdot z$ with $K(x, z)$ they act implicitly as if data was in the higher-dimensional Φ-space.
- If data is linearly separable by large margin in the Φ-space, then good sample complexity.

  [Or other regularity properties for controlling the capacity.]

Kernels

Definition

$K(\cdot,\cdot)$ is a kernel if it can be viewed as a legal definition of inner product:

- $\exists\ \phi: X \to R^N$ s.t. $K(x, z) = \phi(x) \cdot \phi(z)$
  - Range of ϕ is called the Φ-space.
  - N can be very large.

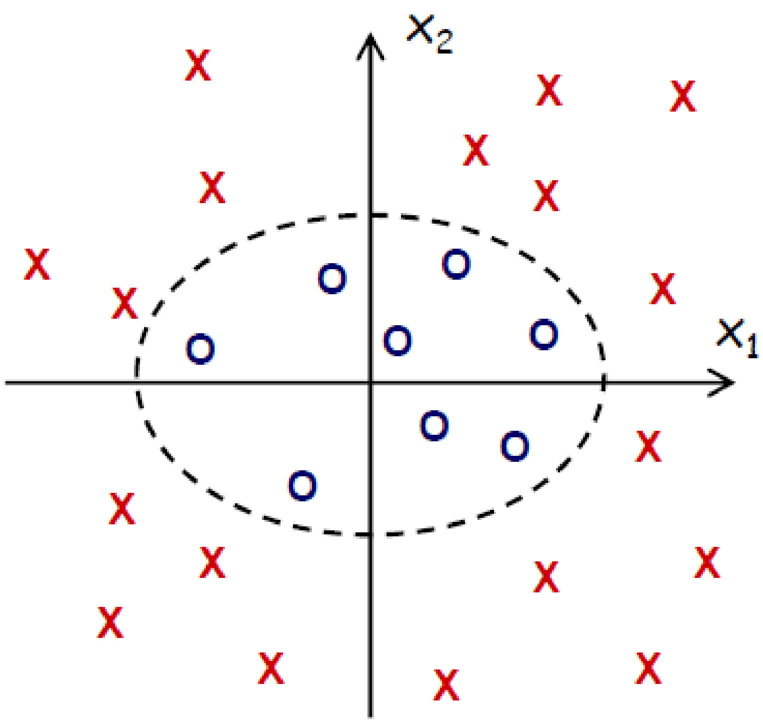
- But think of ϕ as implicit, not explicit!!!!

Kernels: Example

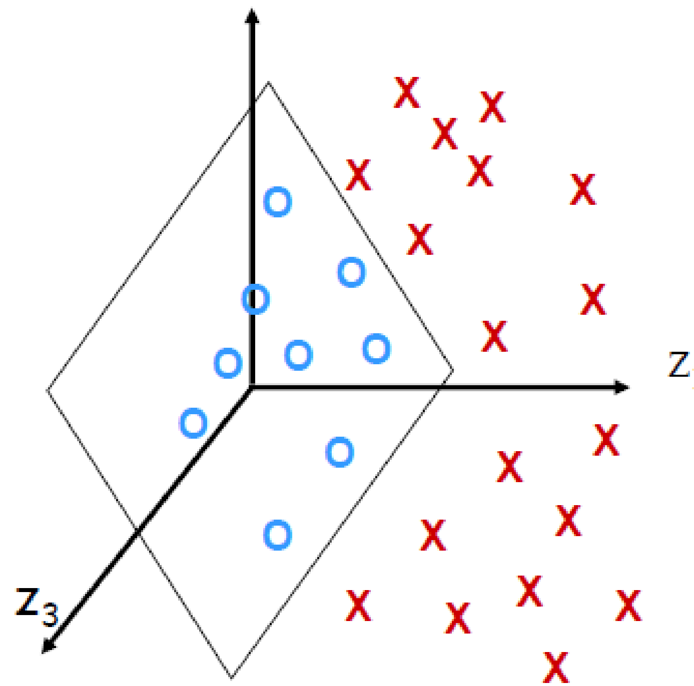
For n=2, d=2, the kernel $\mathrm{K}(\mathrm{x}, \mathrm{z}) = (\mathrm{x} \cdot \mathrm{z})^{\mathrm{d}}$ corresponds to

$$(x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$

Original space



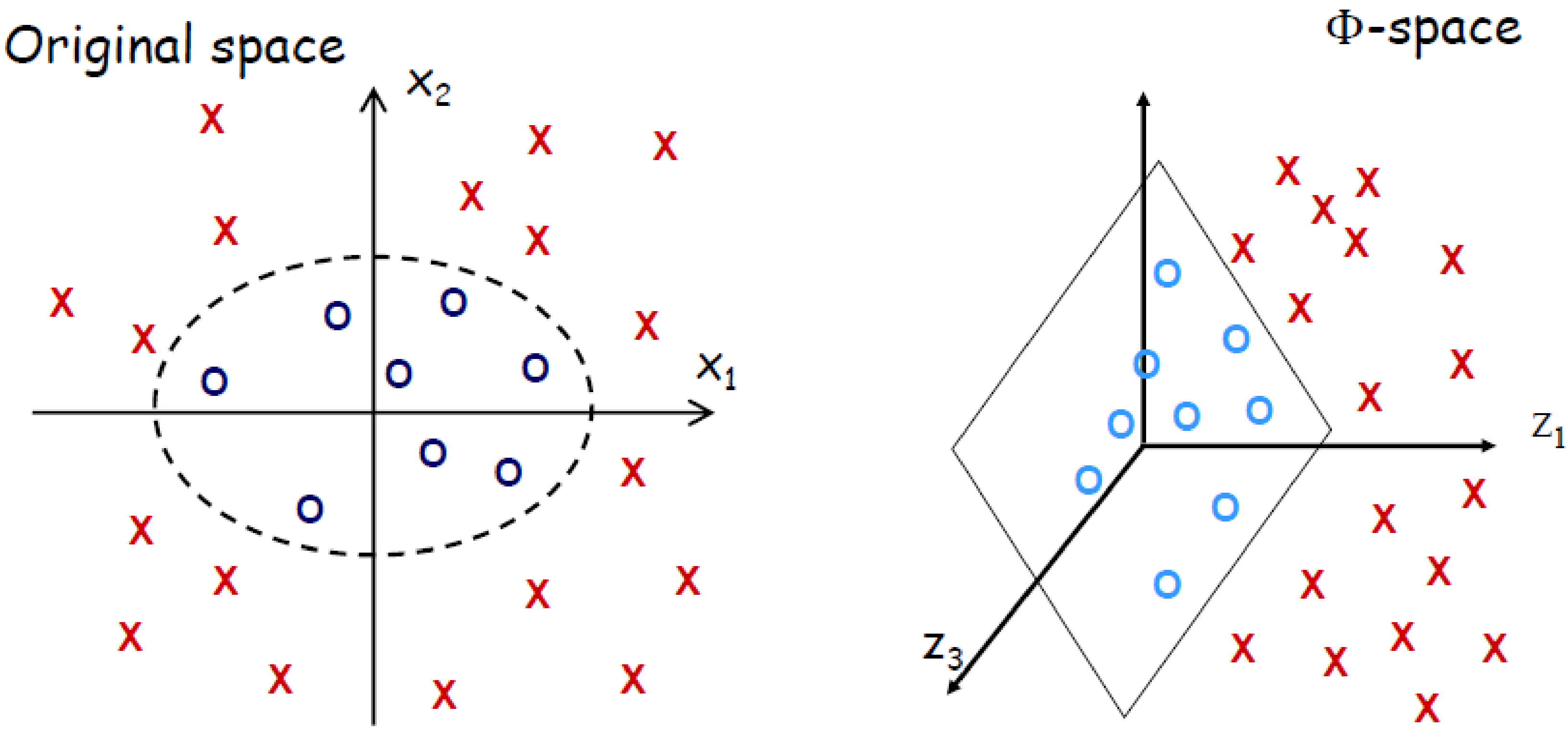
Φ-space



Kernels: Example

$$\phi: R^2 \rightarrow R^3,\ (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\begin{aligned}\phi(x) \cdot \phi(z) &= \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \\ &= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)\end{aligned}$$



Kernels: Example

Note: feature space might not be unique.

$$\phi: R^2 \to R^3,\ (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$\phi(x) \cdot \phi(z) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2)$$

$$= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)$$

$$\phi: R^2 \to R^4,\ (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1)$$

$$\phi(x) \cdot \phi(z) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \cdot (z_1^2, z_2^2, z_1z_2, z_2z_1)$$

$$= (x \cdot z)^2 = K(x, z)$$

Avoid explicitly expanding the features

Feature space can grow really large and really quickly….

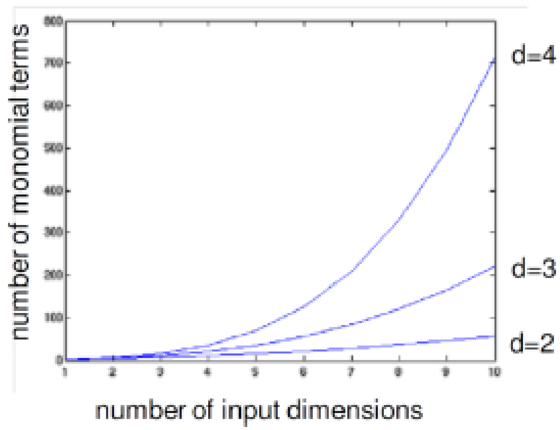
Crucial to think of ϕ as implicit, not explicit!!!!

Polynomial kernel degreee d, $k(x,z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$

- $x_1^d, x_1 x_2 \dots x_d, x_1^2 x_2 \dots x_{d-1}$
- Total number of such feature is

$$\binom{d+n-1}{d} = \frac{(d+n-1)!}{d!\,(n-1)!}$$

- $d = 6, n = 100$, there are 1.6 billion terms



$O(n)$ computation!

$$k(x,z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$$

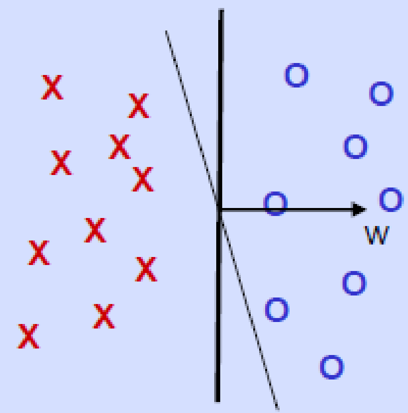
Kernelizing a learning algorithm

- **If all computations involving instances are in terms of inner products then:**
  - Conceptually, work in a very high diml space and the alg's performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each $x \cdot z$ with a $K(x, z)$.

- **Examples of kernalizable algos:**
  - classification: Perceptron, SVM.
  - regression: linear, ridge regression.
  - clustering: k-means.

Kernelizing the Perceptron Algorithm

- Set t=1, start with the all zero vector w_1.
- Given example x, predict + iff $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, $w_{t+1} \leftarrow w_t - x$



Easy to kernelize since w_t is weighted sum of incorrectly classified examples $w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$

Replace $w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \cdots + a_{i_k} x_{i_k} \cdot x$ with

$$a_{i_1} K(x_{i_1}, x) + \cdots + a_{i_k} K(x_{i_k}, x)$$

Note: need to store all the mistakes so far.

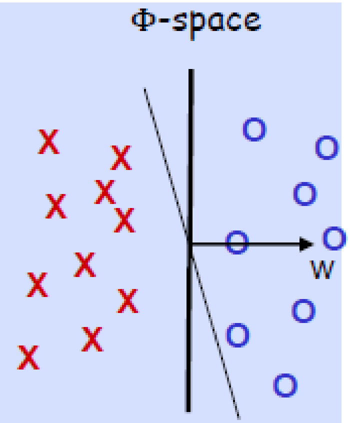
Kernelizing the Perceptron Algorithm

- Given x, predict + iff

$$\phi(x_{i_{t-1}}) \cdot \phi(x)$$

$$a_{i_1} K(x_{i_1}, x) + \cdots + a_{i_{t-1}} K(x_{i_{t-1}}, x) \geq 0$$

- On the t th mistake, update as follows:
  - Mistake on positive, set $a_{i_t} \leftarrow 1$; store x_{i_t}
  - Mistake on negative, $a_{i_t} \leftarrow -1$; store x_{i_t}



Perceptron $w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$

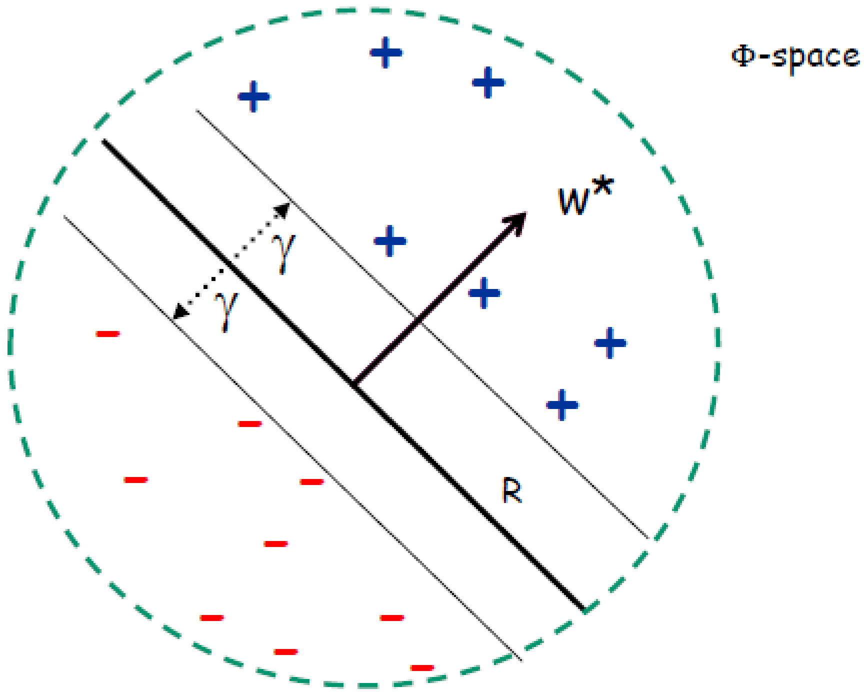
$$w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \cdots + a_{i_k} x_{i_k} \cdot x \quad \rightarrow \quad a_{i_1} K(x_{i_1}, x) + \cdots + a_{i_k} K(x_{i_k}, x)$$

Exact same behavior/prediction rule as if mapped data in the ϕ-space and ran Perceptron there!

Do this implicitly, so computational savings!!!!!

Generalize Well if Good Margin

- If data is linearly separable by margin in the ϕ-space, then small mistake bound.
- If margin γ in ϕ-space, then Perceptron makes $\left(\frac{R}{\gamma}\right)^2$ mistakes.



Kernels: More Examples

- Linear: $K(x, z) = x \cdot z$
- Polynomial: $K(x, z) = (x \cdot z)^d$ or $K(x, z) = (1 + x \cdot z)^d$
- Gaussian: $K(x, z) = \exp\left[-\frac{||x-z||^2}{2\,\sigma^2}\right]$
- Laplace Kernel: $K(x, z) = \exp\left[-\frac{||x-z||}{2\,\sigma^2}\right]$

Support Vector Machines (SVMs)

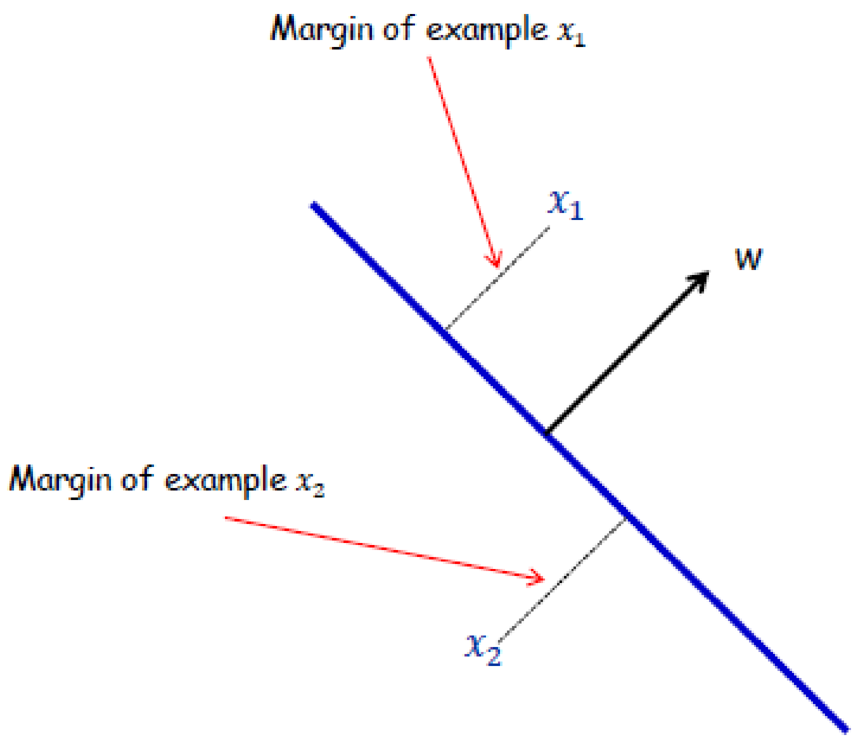
Support Vector Machines

- Support Vector Machines (SVMs)
  - One of the most theoretically well motivated and practically most effective classification algorithms in machine learning.
  - Directly motivated by Margins and Kernels!

Geometric Margin

WLOG homogeneous linear separators $[w_0 = 0]$.

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$.



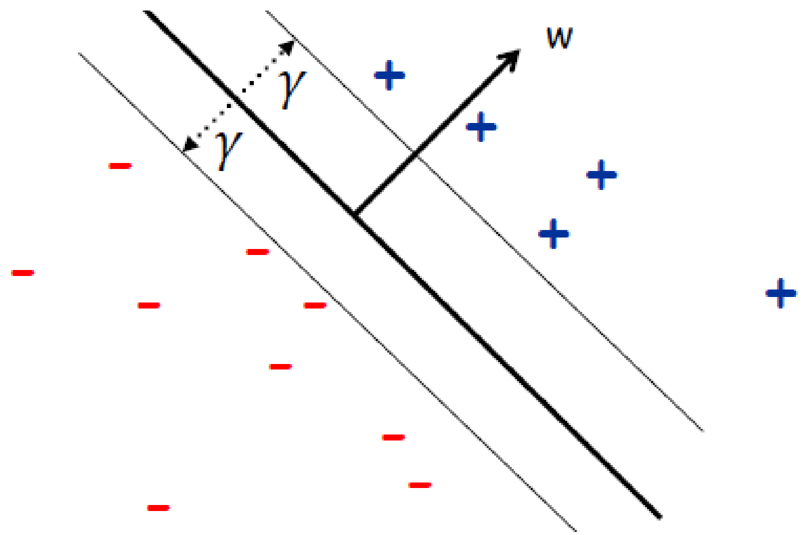
If $||w|| = 1$, margin of x w.r.t. w is $|x \cdot w|$.

Geometric Margin

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$.

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

Definition: The margin γ of a set of examples S is the maximum γ_w over all linear separators w.

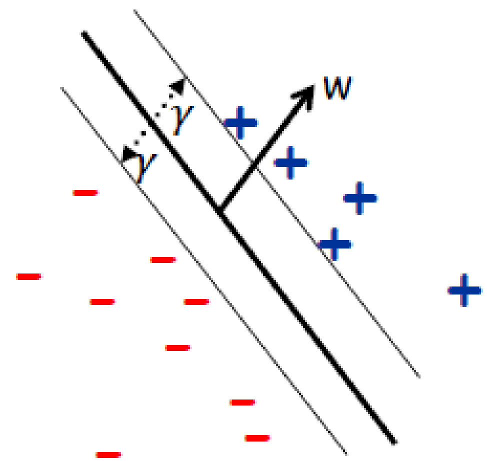


Margin Important Theme in ML

Both sample complexity and algorithmic implications.

Sample/Mistake Bound complexity:

- If large margin, # mistakes Peceptron makes is small (independent on the dim of the space)!
- If large margin γ and if alg. produces a large margin classifier, then amount of data needed depends only on R/γ [Bartlett & Shawe-Taylor '99].



Algorithmic Implications

Suggests searching for a large margin classifier… SVMs

Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

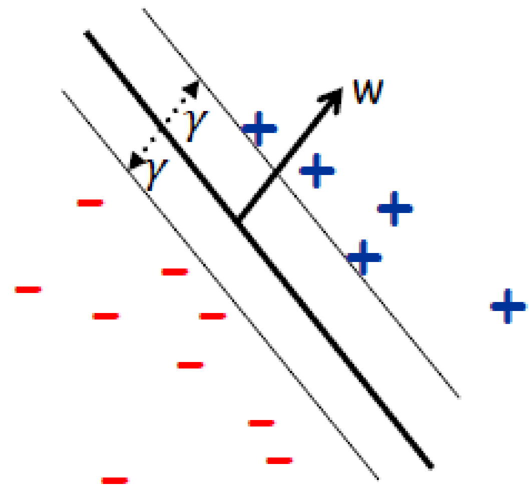
First, assume we know a lower bound on the margin γ

<u>Input</u>: γ, $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

<u>Find</u>: some w where:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$

<u>Output</u>: w, a separator of margin γ over S

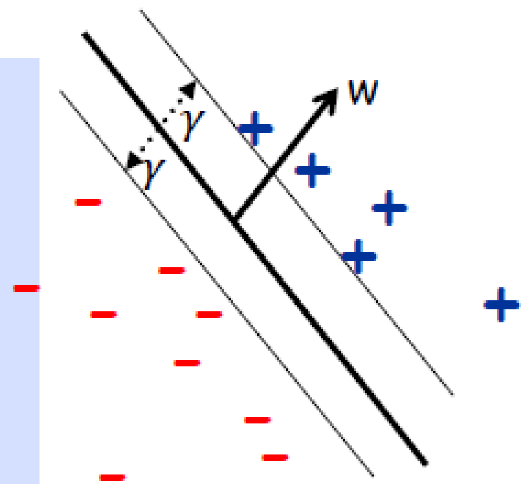


Realizable case, where the data is linearly separable by margin γ

Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

E.g., search for the best possible γ



Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

Find: some w and maximum γ where:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$

Output: maximum margin separator over S

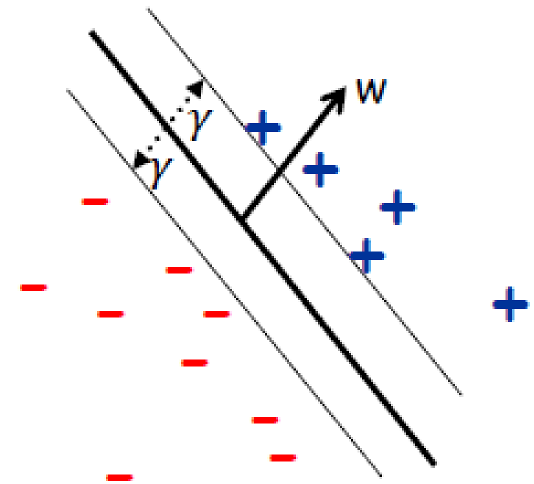
Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

Maximize γ under the constraint:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$



Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

Input: S={(x_1, y_1), …,(x_m, y_m)};

Maximize γ under the constraint:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$

objective function

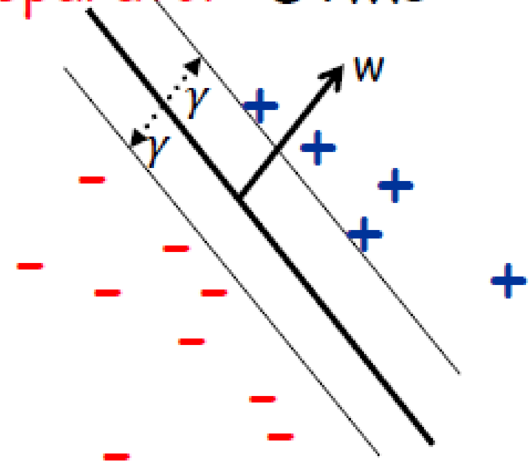
constraints

This is a **constrained optimization** problem.

- Famous example of constrained optimization: **linear programming**, where objective fn is linear, constraints are linear (in)equalities

Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs



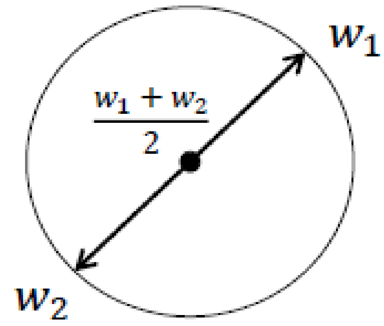
<u>Input</u>: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

Maximize γ under the constraint:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$

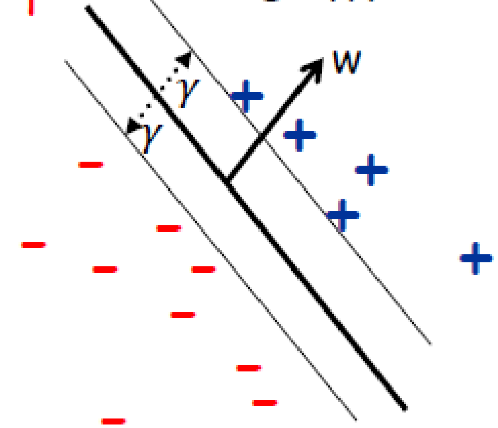
This constraint is non-linear.

In fact, it's even non-convex



Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

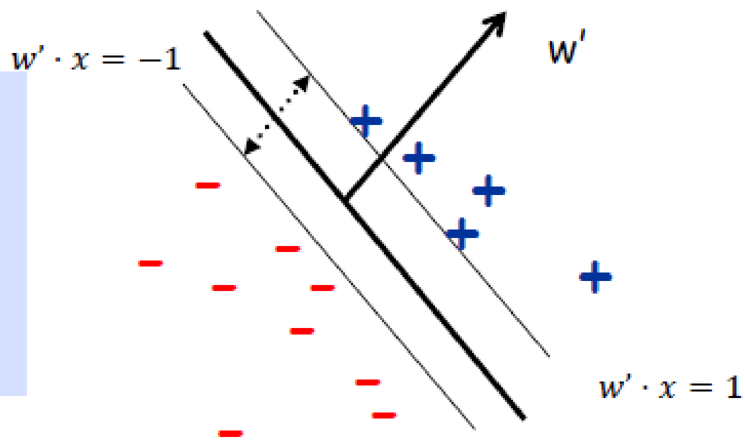


Input: S={(x_1, y_1), …,(x_m, y_m)};

Maximize γ under the constraint:

- $||w||^2 = 1$
- For all i, $y_i w \cdot x_i \geq \gamma$

$w' = w/\gamma$, then max γ is equiv. to minimizing $||w'||^2$ (since $||w'||^2 = 1/\gamma^2$).
So, dividing both sides by γ and writing in terms of w′ we get:



Input: S={(x_1, y_1), …,(x_m, y_m)};

Minimize $||w'||^2$ under the constraint:

- For all i, $y_i w' \cdot x_i \geq 1$

Support Vector Machines (SVMs)

Directly optimize for the maximum margin separator: SVMs

Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

$\mathrm{argmin}_w \, ||w||^2$ s.t.:

- For all i, $y_i w \cdot x_i \geq 1$

This is a **constrained optimization** problem.

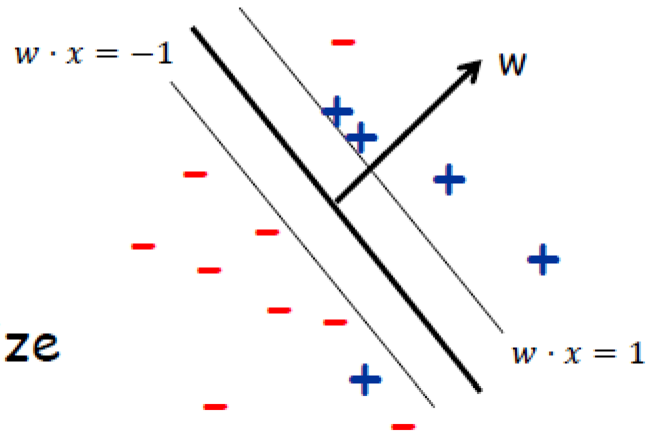
- The objective is convex (quadratic)
- All constraints are linear
- Can solve efficiently (in poly time) using standard **quadratic programing** (QP) software

Support Vector Machines (SVMs)

Question: what if data isn't perfectly linearly separable?

Issue 1: now have two objectives

- maximize margin
- minimize # of misclassifications.



Ans 1: Let's optimize their sum: minimize

$$||w||^2 + C(\text{\# misclassifications})$$

where C is some tradeoff constant.

Issue 2: This is computationally hard (NP-hard). 

[even if didn't care about margin and minimized # mistakes]

NP-hard [Guruswami-Raghavendra'06]

Support Vector Machines (SVMs)

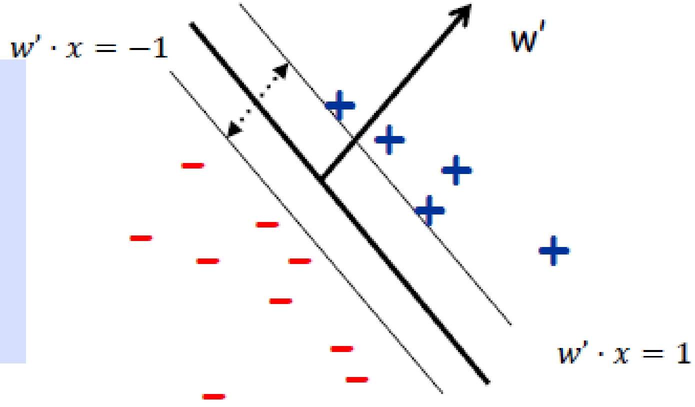
Question: what if data isn't perfectly linearly separable?

Replace "# mistakes" with upper bound called "hinge loss"

Input: S={(x_1, y_1), …,(x_m, y_m)};

Minimize $||w'||^2$ under the constraint:

- For all i, $y_i w' \cdot x_i \geq 1$



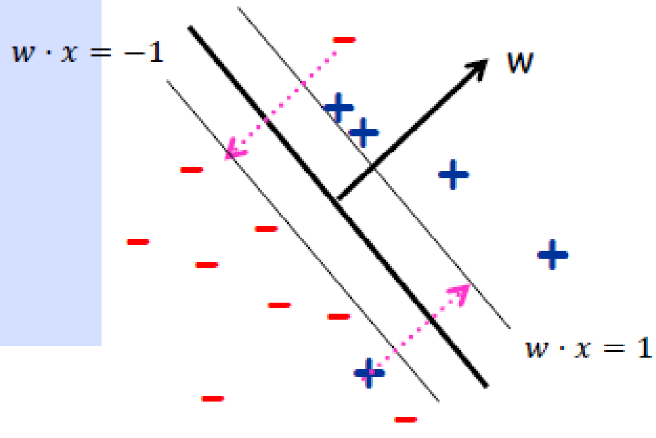
Input: S={(x_1, y_1), …,(x_m, y_m)};

Find $\text{argmin}_{w,\xi_1,\ldots,\xi_m} ||w||^2 + C \sum_i \xi_i$ s.t.:

- For all i, $y_i w \cdot x_i \geq 1 - \xi_i$

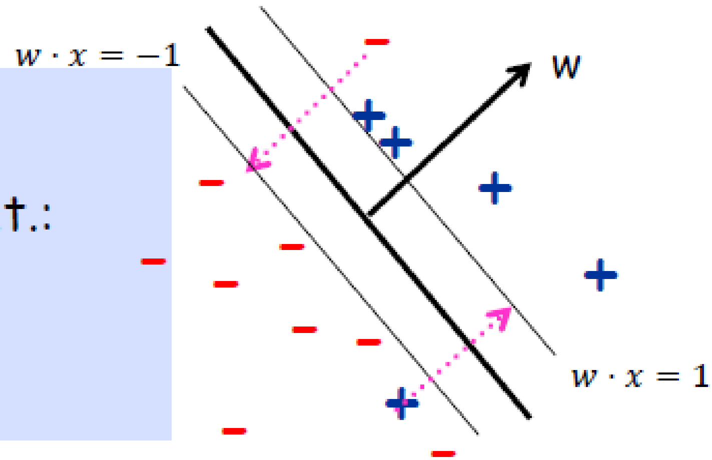
  $\xi_i \geq 0$

ξ_i are "slack variables"



Support Vector Machines (SVMs)

Question: what if data isn't perfectly linearly separable?
Replace "# mistakes" with upper bound called "hinge loss"



Input: S={(x_1, y_1), …,(x_m, y_m)};

Find $\text{argmin}_{w,\xi_1,\ldots,\xi_m} ||w||^2 + C\sum_i \xi_i$ s.t.:

- For all i, $y_i w \cdot x_i \geq 1 - \xi_i$
  $\xi_i \geq 0$

Total amount have to move the points to get them on the correct side of the lines $w \cdot x = +1/-1$, where the distance between the lines $w \cdot x = 0$ and $w \cdot x = 1$ counts as "1 unit".

Support Vector Machines (SVMs)

What if the data is far from being linearly separable?

Example:  vs 

No good linear separator in pixel representation.

SVM philosophy: "Use a Kernel"

Support Vector Machines (SVMs)

Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

Find $\operatorname{argmin}_{w,\xi_1,\ldots,\xi_m} ||w||^2 + C\sum_i \xi_i$ s.t.:

- For all i, $y_i w \cdot x_i \geq 1 - \xi_i$

  $\xi_i \geq 0$

Primal form

Which is equivalent to:

Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$;

Find $\operatorname{argmin}_{\alpha} \frac{1}{2}\sum_i \sum_j y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_i \alpha_i$ s.t.:

- For all i, $0 \leq \alpha_i \leq C_i$

  $$\sum_i y_i \alpha_i = 0$$

Lagrangian Dual

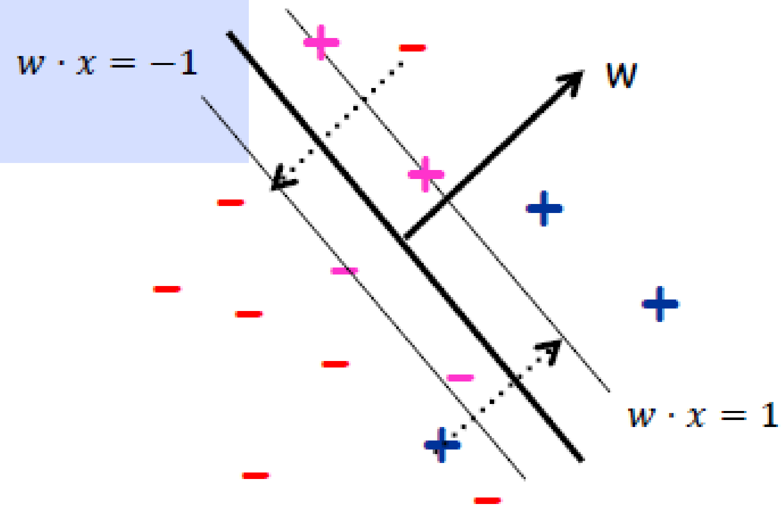
SVMs (Lagrangian Dual)

<u>Input</u>: $S=\{(x_1,y_1), \ldots,(x_m,y_m)\}$;

<u>Find</u> $\text{argmin}_\alpha \frac{1}{2}\sum_i \sum_j y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_i \alpha_i$ s.t.:

- For all i, $0 \leq \alpha_i \leq C_i$

$$\sum_i y_i \alpha_i = 0$$



- Final classifier is: $w = \sum_i \alpha_i y_i x_i$
- The points x_i for which $\alpha_i \neq 0$ are called the "support vectors"

Kernelizing the Dual SVMs

Input: S={(x_1, y_1), …,(x_m, y_m)};

Find $\operatorname{argmin}_\alpha \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_i \alpha_i$ s.t.:

- For all i, $0 \leq \alpha_i \leq C_i$

$$\sum_i y_i \alpha_i = 0$$

Replace $x_i \cdot x_j$ with $K(x_i, x_j)$.

- Final classifier is: $w = \sum_i \alpha_i y_i x_i$
- The points x_i for which $\alpha_i \neq 0$ are called the "support vectors"
- With a kernel, classify x using $\sum_i \alpha_i y_i K(x, x_i)$

SVM - Support Vector Machines

- A new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., "decision boundary")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)
- SVMs are currently among the best performers for a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types beyond feature vectors (e.g. graphs, sequences, relational data) by designing kernel functions for such data.