CS578 Statistical Machine Learning Lecture 1

Jean Honorio
Purdue University

(based on slides by Tommi Jaakkola, MIT CSAIL)

First things first…

- Website: http://www.cs.purdue.edu/homes/jhonorio/20spring-cs57800.html
  - TAs
  - Textbooks
  - Assignments
  - Grading
  - Late policy
  - Schedule

- Office hours: Doodle on Friday, we will decide on Monday
- Piazza will be made available soon

- **Answer prerequisite survey sent on January 6**
- **Solve Homework 0, due on January 16**

Course topics

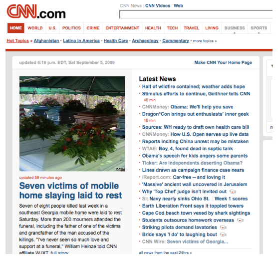
- Supervised learning
  - linear and non-linear classifiers, kernels
  - rating, ranking, collaborative filtering
  - model selection, complexity, generalization
  - conditional Random fields, structured prediction

- Unsupervised learning, modeling
  - mixture models, topic models
  - Hidden Markov Models
  - Bayesian networks
  - Markov Random Fields, factor graphs

Machine learning

- Learning from examples

+1



-1

+1

-1

training set

Machine learning

- Learning from examples

+1

-1

+1

-1

training set

The training set of labeled examples specifies the learning task only implicitly

Machine learning

- Learning from examples

+1

-1

+1

-1

training set

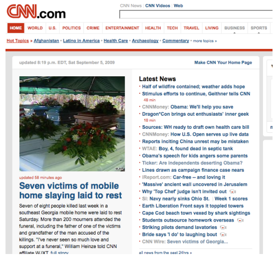
?

Our goal is to accurately label new websites that were not part of the training set

Machine learning

- Learning from examples

+1



-1

+1

-1

training set

predicted label

new website

$$y = f\Bigg($$  $$\Bigg)$$

classifier = mapping from websites to labels

“Examples”

- We will have to first represent the examples (websites) in a manner that can be easily mapped to labels

news article

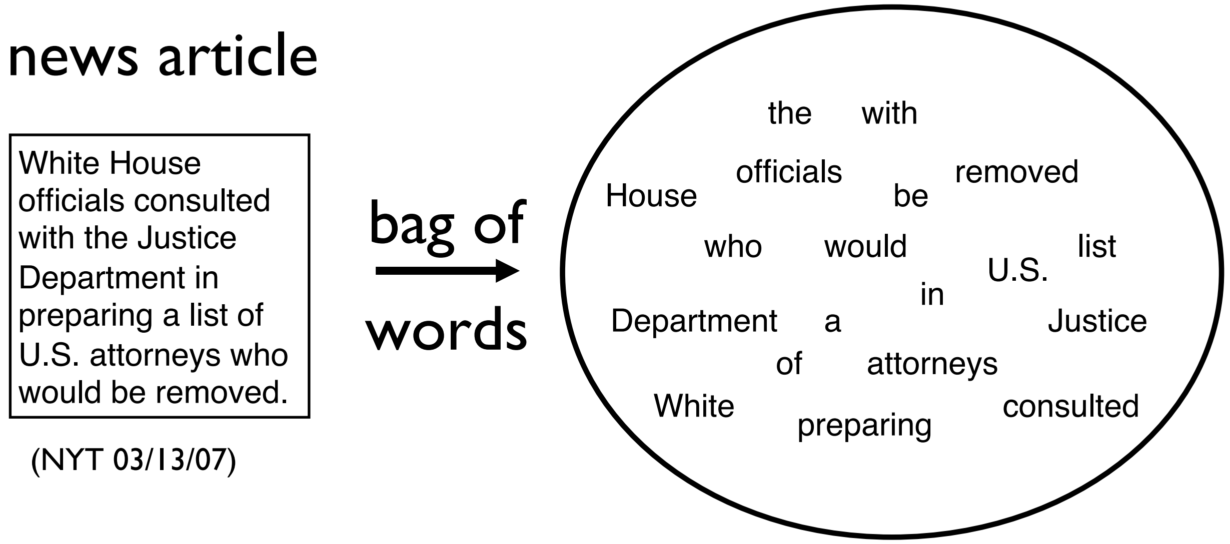
White House officials consulted with the Justice Department in preparing a list of U.S. attorneys who would be removed.

(NYT 03/13/07)

“Examples”

- We will have to first represent the examples (websites) in a manner that can be easily mapped to labels

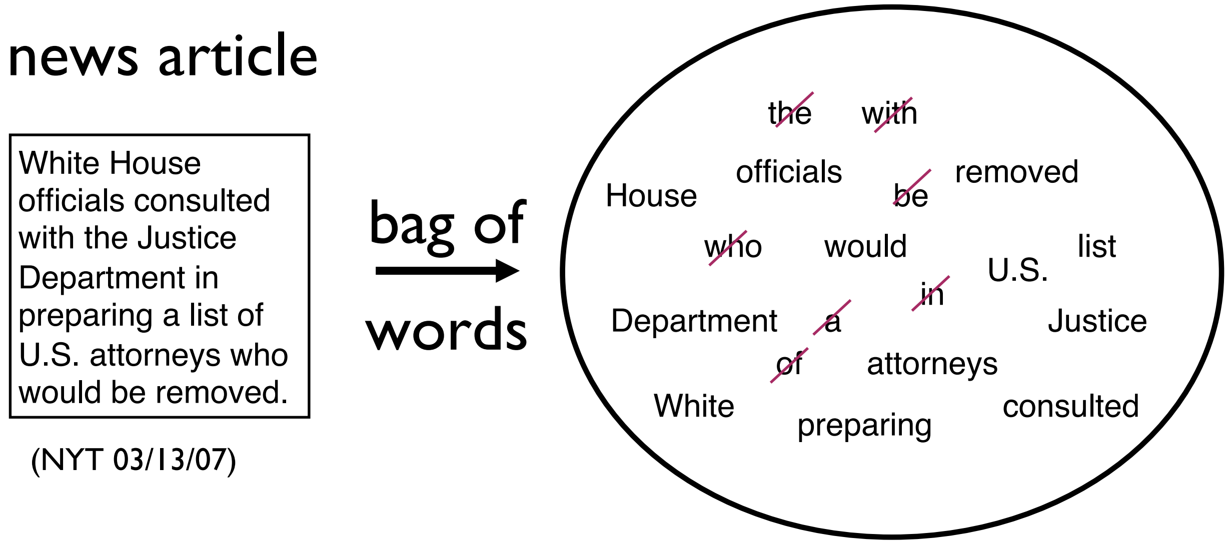
news article



(NYT 03/13/07)

"Examples"

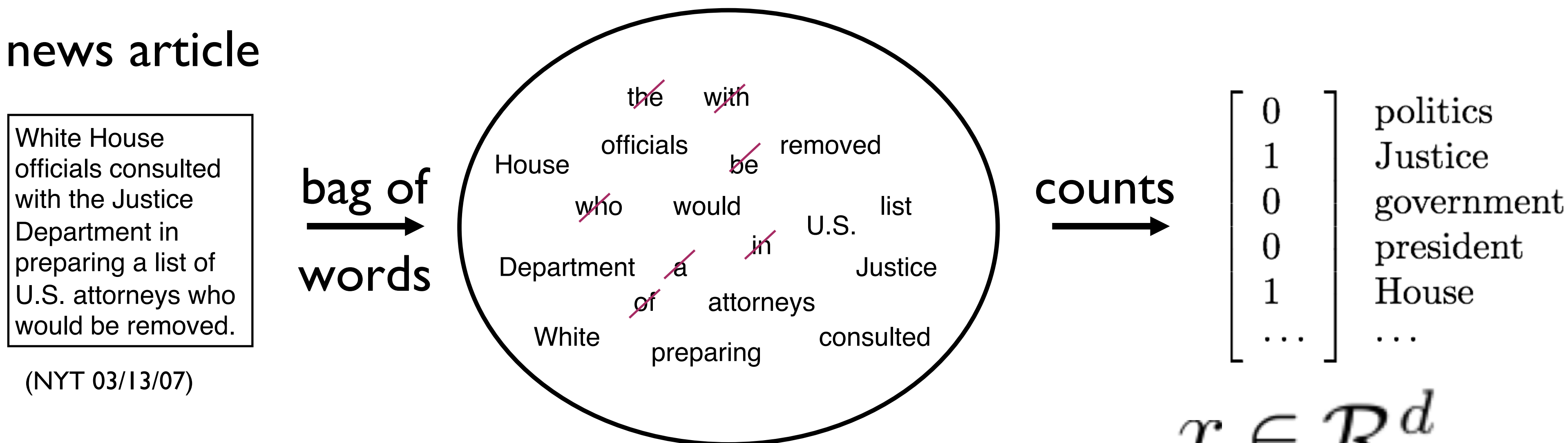
- We will have to first represent the examples (websites) in a manner that can be easily mapped to labels



“Examples”

- We will have to first represent the examples (websites) in a manner that can be easily mapped to labels

news article



$$\underline{x} \in \mathcal{R}^d$$

a vector whose coordinates (features) specify how many times (or whether) particular words appeared in the article

$$y \in \{-1, 1\}$$

the label of the website

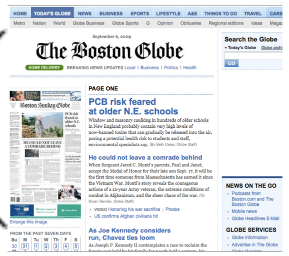
Classifier

- Intuitively:

predicted label

new website

$$y = f\left(\quad \right)$$



- Programming point of view:

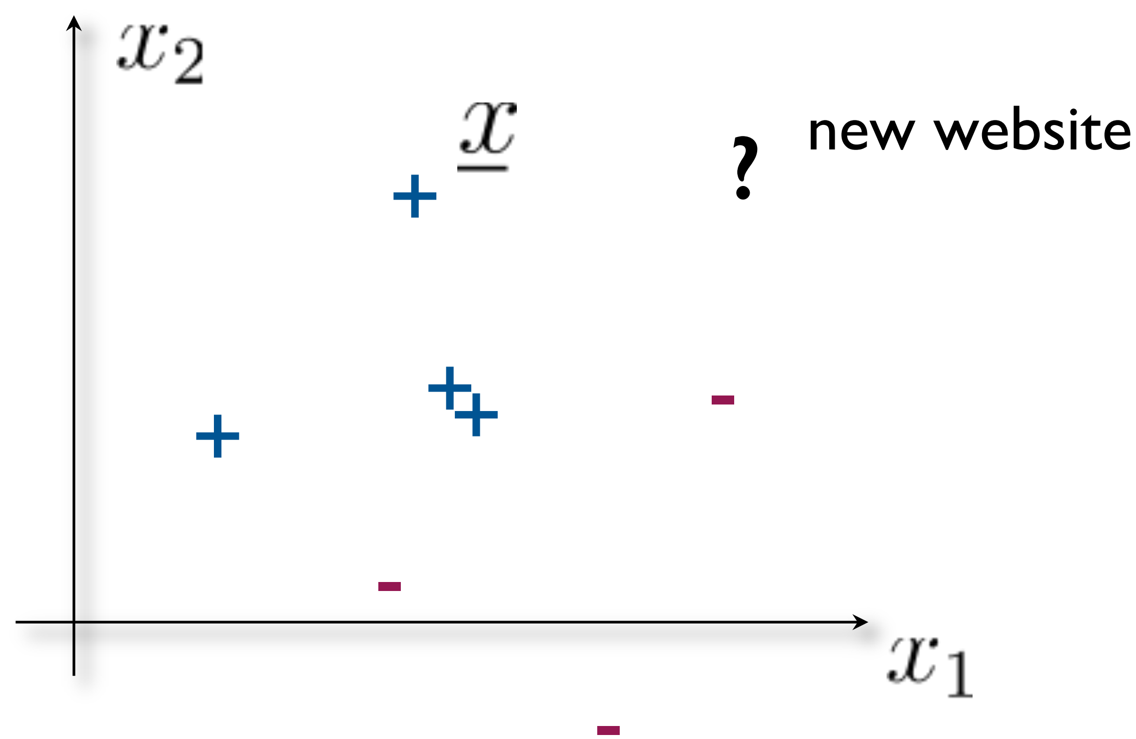
  f is a piece of code that receives a vector $\underline{x} \in \mathcal{R}^d$ as input, and outputs a label $y \in \{-1, 1\}$

- Mathematical notation:

$$f : \mathcal{R}^d \rightarrow \{-1, 1\}$$

The learning task

- The training set is now a set of labeled points



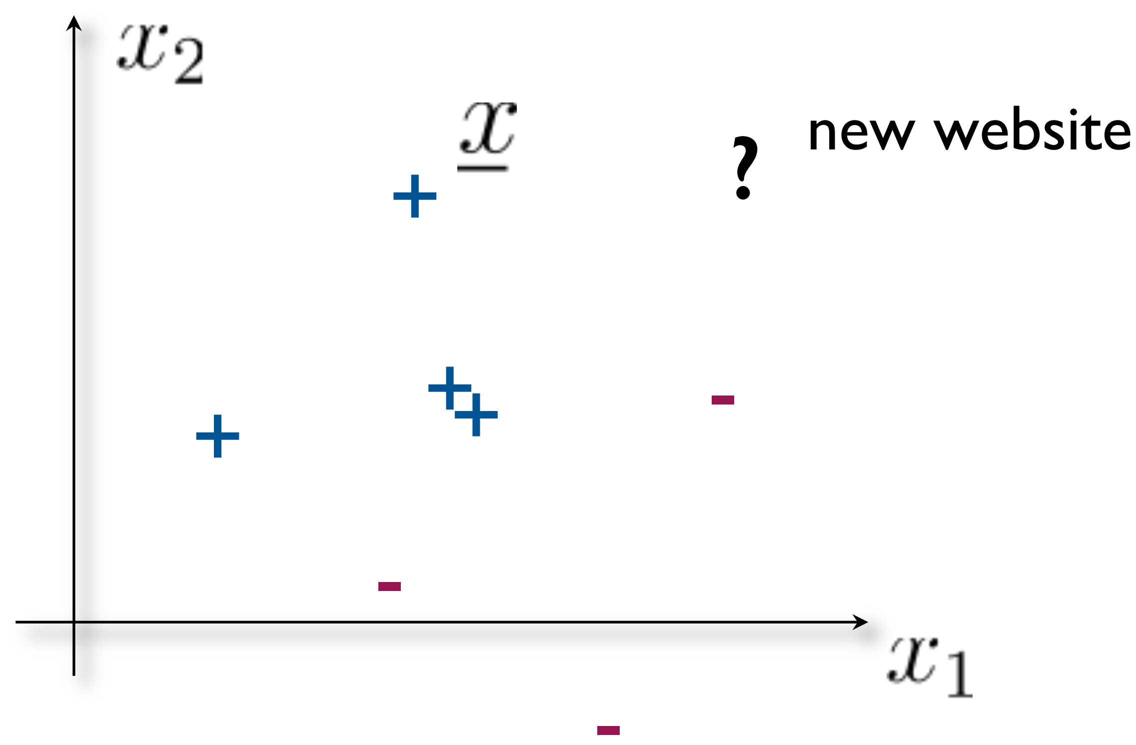
- We seek for a “good” classifier $f : \mathcal{R}^d \longrightarrow \{-1, 1\}$
based on the training set $D = \{\underline{x}_1, y_1, \underline{x}_2, y_2, \ldots, \underline{x}_n, y_n\}$
so that $f(\underline{x})$ correctly labels any new websites $\underline{x}$

$\{-1, 1\}$

$\mathcal{R}^d$

The learning task

- The training set is now a set of labeled points



Part 1:
Model selection
what type of classifiers should we consider?

- We seek for a "good" classifier $f : \mathcal{R}^d \rightarrow \{-1, 1\}$

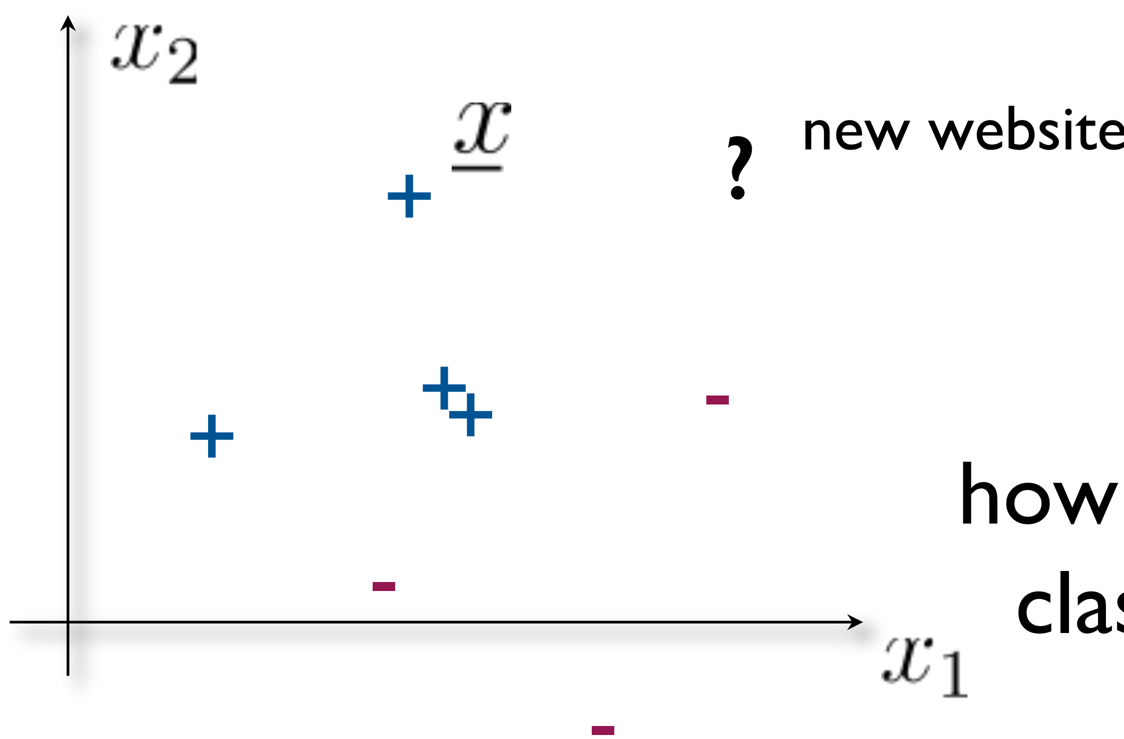
based on the training set $D = \{\underline{x}_1, y_1, \underline{x}_2, y_2, \ldots, \underline{x}_n, y_n\}$

so that $f(\underline{x})$ correctly labels any new websites $\underline{x}$

The learning task

- The training set is now a set of labeled points

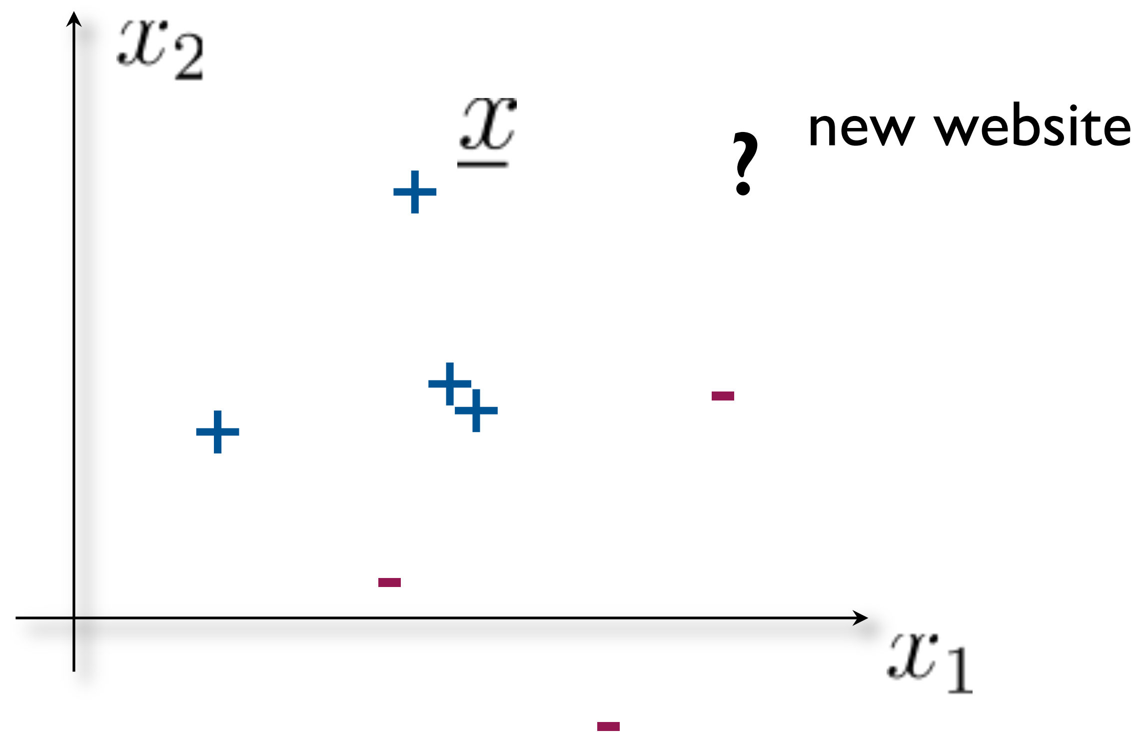
Part 1:
Model selection
what type of classifiers should we consider?



Part 2:
Estimation
how to select the best classifier in the set?

- We seek for a "good" classifier $f : \mathcal{R}^d \rightarrow \{-1, 1\}$ based on the training set $D = \{\underline{x}_1, y_1, \underline{x}_2, y_2, \ldots, \underline{x}_n, y_n\}$ so that $f(\underline{x})$ correctly labels any new websites $\underline{x}$

Part 1: allowing all classifiers?

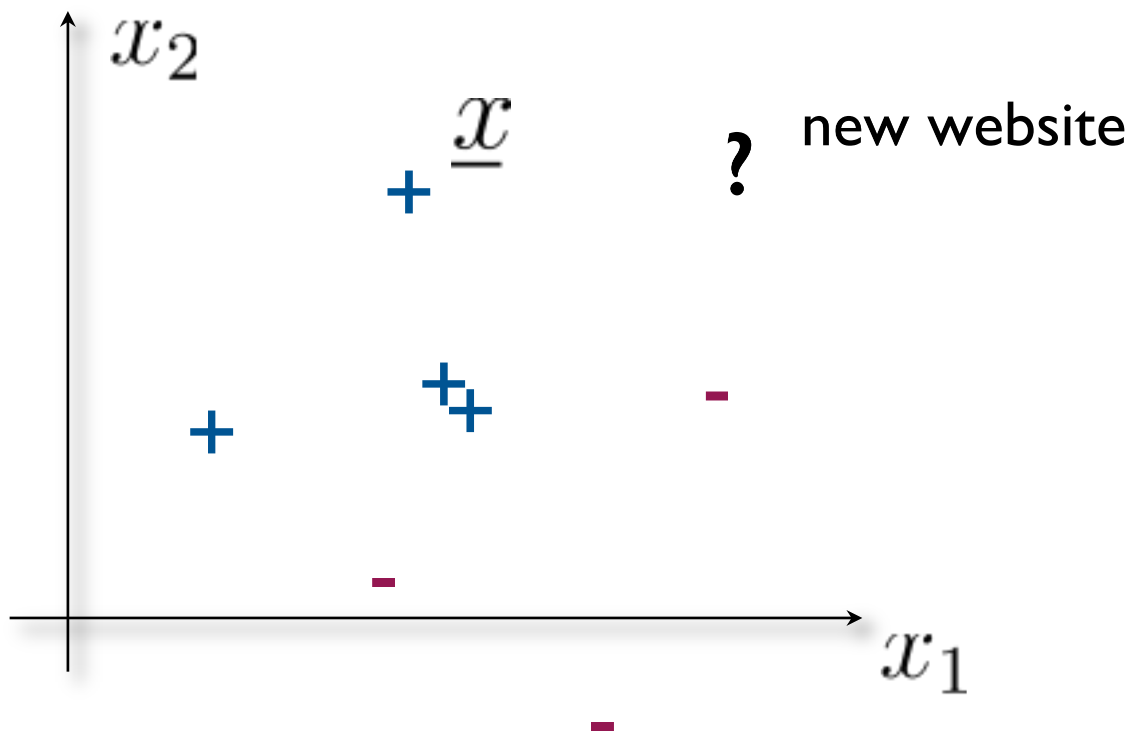


- We can easily construct a "silly classifier" that perfectly classifiers any distinct set of training points

$$f(\underline{x}) = \begin{cases} y_i, & \text{if } \underline{x} = \underline{x}_i \text{ for some } i \\ -1, & \text{otherwise} \end{cases}$$

- But it doesn't "generalize" (it doesn't classify new points very well)

Part 1: allowing few classifiers?



- We could instead consider very few alternatives such as

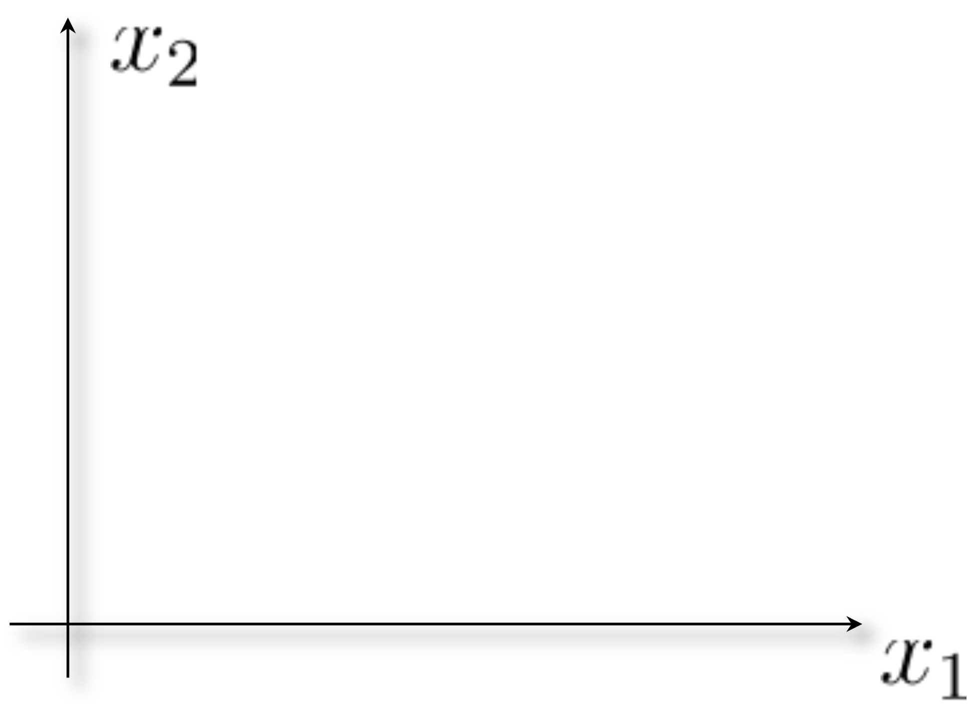
$$\begin{aligned} f(\underline{x}) &= 1, \quad \text{for all} \quad \underline{x} \in \mathcal{R}^d \quad \text{or} \\ f(\underline{x}) &= -1, \quad \text{for all} \quad \underline{x} \in \mathcal{R}^d \end{aligned}$$

- But neither one classifies even training points very well

Part 1: linear classifiers

- A linear classifier (through origin) with parameters $\underline{\theta} \in \mathcal{R}^d$ divides the space into positive and negative halves

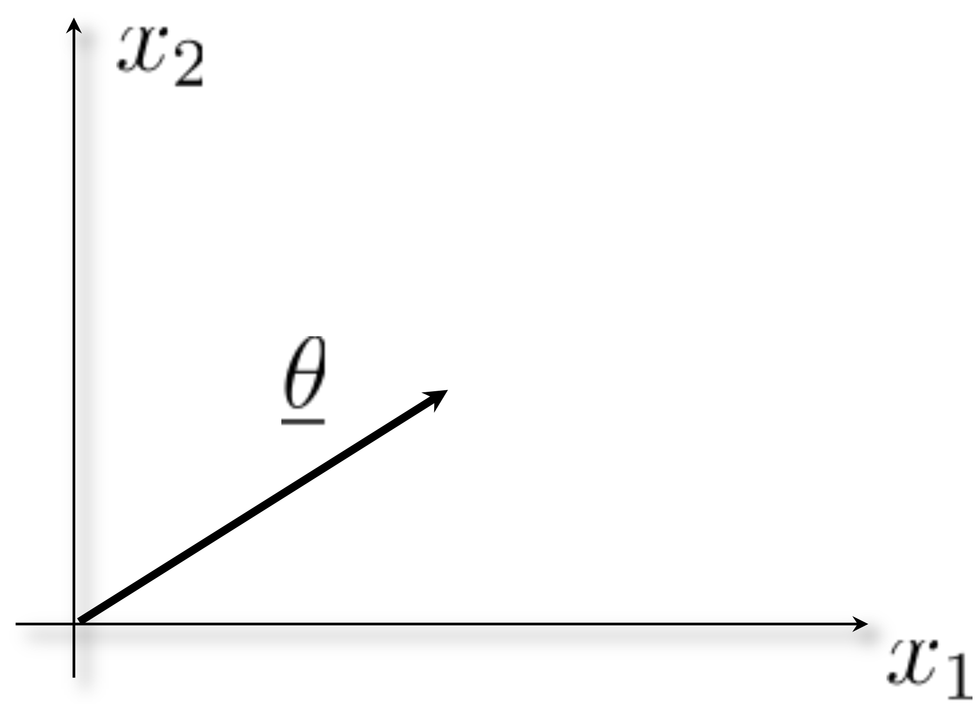
$$
\begin{aligned}
f(\underline{x};\underline{\theta}) &= \operatorname{sign}\big(\,\underline{\theta}\cdot\underline{x}\,\big) = \operatorname{sign}\big(\,\theta_1 x_1 + \ldots + \theta_d x_d\,\big) \\
&= \begin{cases} +1, & \text{if } \underline{\theta}\cdot\underline{x} > 0 \\ -1, & \text{if } \underline{\theta}\cdot\underline{x} \le 0 \end{cases}
\end{aligned}
$$



Part 1: linear classifiers

- A linear classifier (through origin) with parameters $\underline{\theta} \in \mathcal{R}^d$ divides the space into positive and negative halves

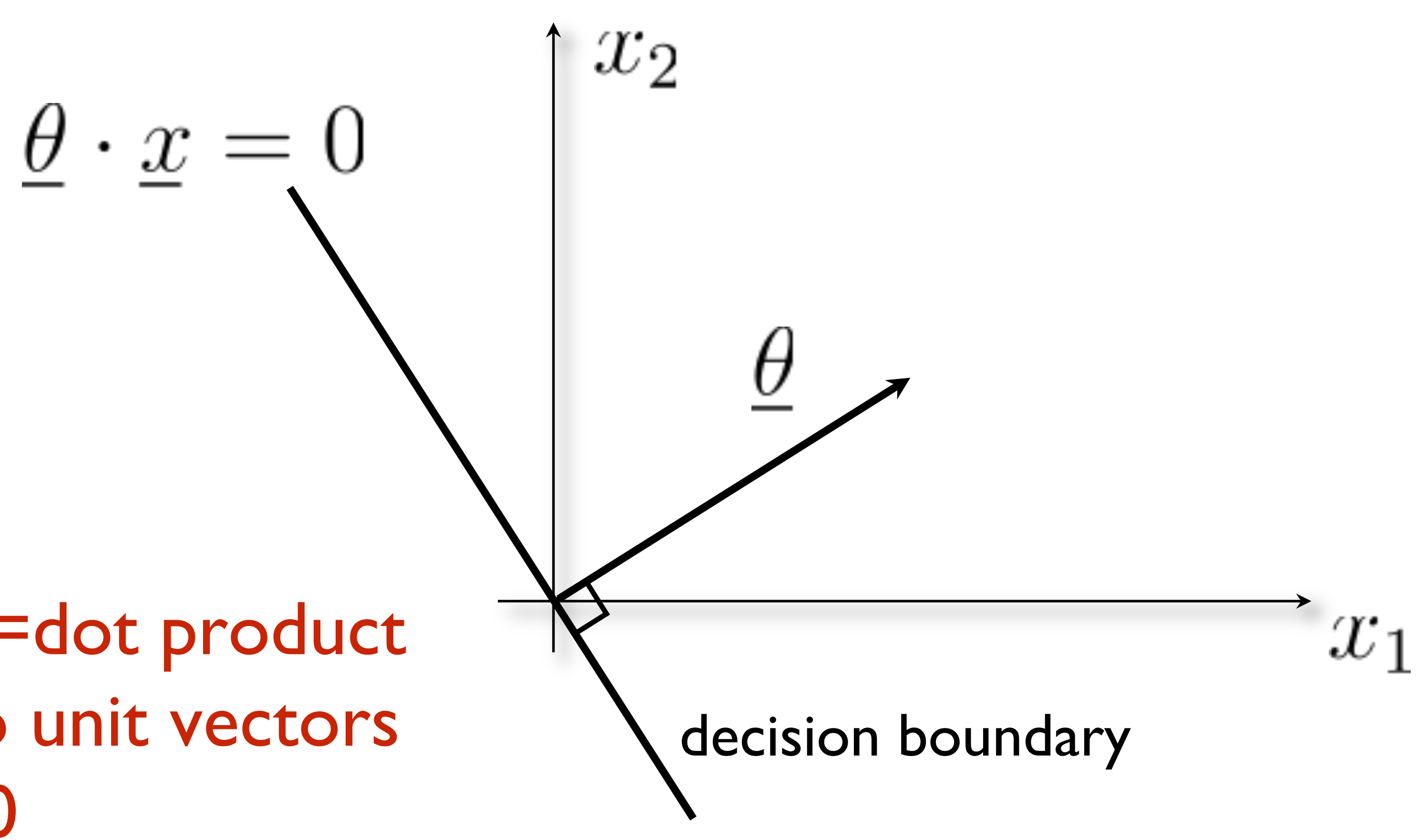
$$
\begin{aligned}
f(\underline{x};\underline{\theta}) &= \text{sign}\big(\,\underline{\theta}\cdot\underline{x}\,\big) = \text{sign}\big(\,\theta_1 x_1 + \ldots + \theta_d x_d\,\big) \\
&= \begin{cases} +1, & \text{if } \underline{\theta}\cdot\underline{x} > 0 \\ -1, & \text{if } \underline{\theta}\cdot\underline{x} \leq 0 \end{cases}
\end{aligned}
$$



Part 1: linear classifiers

- A linear classifier (through origin) with parameters $\underline{\theta} \in \mathcal{R}^d$ divides the space into positive and negative halves

$$
\begin{aligned}
f(\underline{x};\underline{\theta}) &= \operatorname{sign}\big(\,\underline{\theta}\cdot\underline{x}\,\big) = \operatorname{sign}\big(\,\theta_1 x_1 + \ldots + \theta_d x_d\,\big) \\
&= \begin{cases} +1, & \text{if } \underline{\theta}\cdot\underline{x} > 0 \\ -1, & \text{if } \underline{\theta}\cdot\underline{x} \leq 0 \end{cases}
\end{aligned}
$$

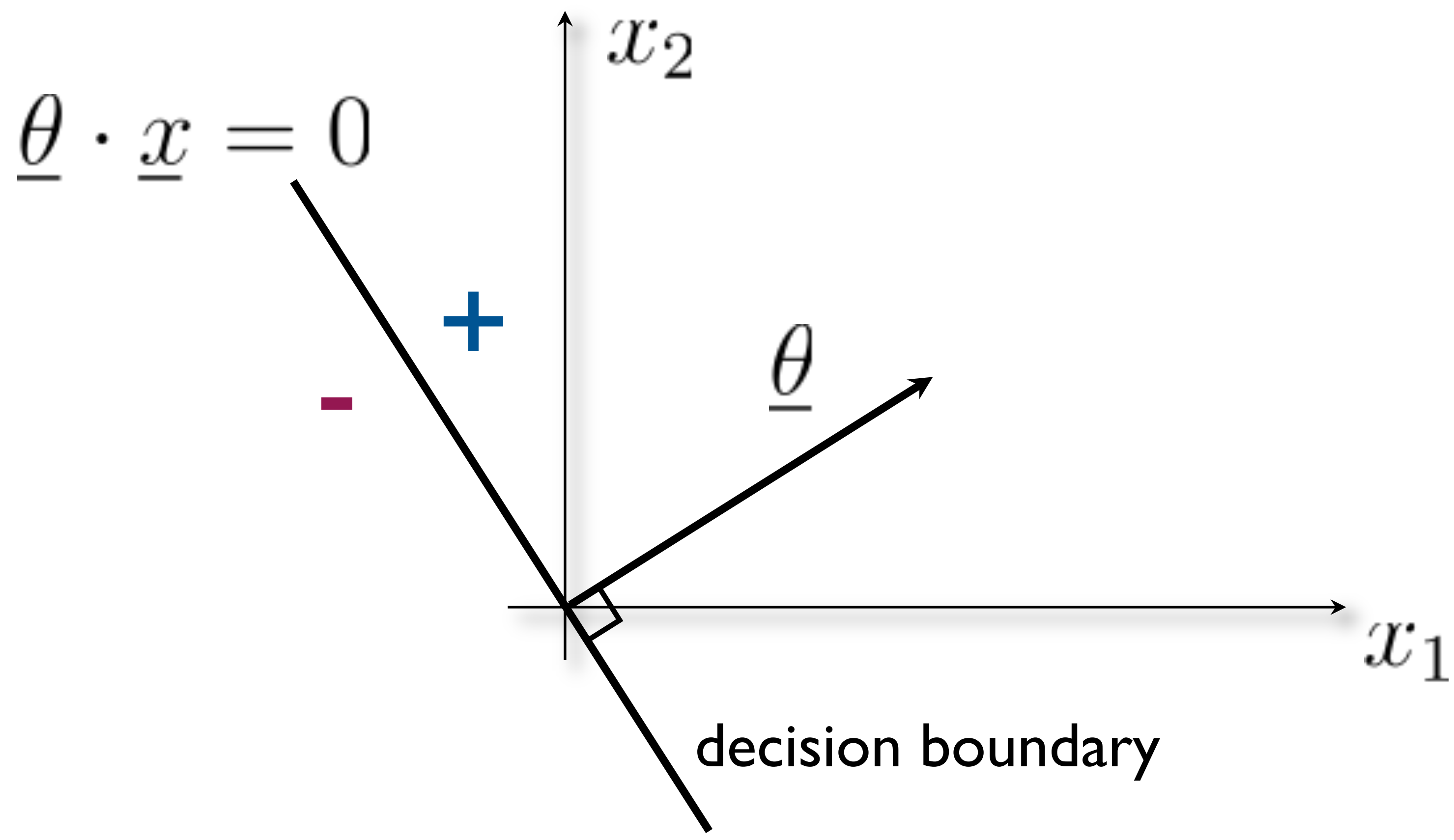


Recall:
cos(angle)=dot product of two unit vectors
cos(90°)=0

Part 1: linear classifiers

- A linear classifier (through origin) with parameters $\underline{\theta} \in \mathcal{R}^d$ divides the space into positive and negative halves

$$\begin{aligned} f(\underline{x};\underline{\theta}) &= \operatorname{sign}\big(\,\underline{\theta}\cdot\underline{x}\,\big) = \operatorname{sign}\big(\,\theta_1 x_1 + \ldots + \theta_d x_d\,\big) \\ &= \begin{cases} +1, & \text{if } \underline{\theta}\cdot\underline{x} > 0 \\ -1, & \text{if } \underline{\theta}\cdot\underline{x} \leq 0 \end{cases} \end{aligned}$$



Part 2: estimation

- We can use the training error as a surrogate criterion for finding the best linear classifier (through origin)

$$\hat{R}_n(\underline{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{Loss}\big(\, y_i, f(\underline{x}_i; \underline{\theta})\,\big)$$

$$\text{where } \ \mathrm{Loss}\big(\, y, y'\,\big) = \begin{cases} 1, & \text{if } y \neq y' \\ 0, & \text{o.w.} \end{cases}$$

- Other choices are possible (and often preferable)

Perceptron algorithm

- We can use the training error as a surrogate criterion for finding the best linear classifier (through origin)

$$\hat{R}_n(\underline{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \text{Loss}\big(\, y_i, f(\underline{x}_i; \underline{\theta})\,\big)$$

$$\text{where} \;\; \text{Loss}\big(\, y, y' \,\big) = \begin{cases} 1, & \text{if } y \neq y' \\ 0, & \text{o.w.} \end{cases}$$

- Since $f(\underline{x}; \underline{\theta}) = \text{sign}\big(\, \underline{\theta} \cdot \underline{x} \,\big)$

$$\text{Loss}(y_t, f(\underline{x}_t; \underline{\theta})) = \begin{cases} 1, & y_t \neq \text{sign}(\underline{\theta} \cdot \underline{x}_t) \\ 0, & \text{o.w.} \end{cases}$$

Mistake

- Checking for the condition $y_t \neq \text{sign}(\underline{\theta} \cdot \underline{x}_t)$ is equivalent to checking for the condition $y_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$

Perceptron algorithm

- The perceptron algorithm considers each training point in turn, adjusting the parameters to correct any mistakes

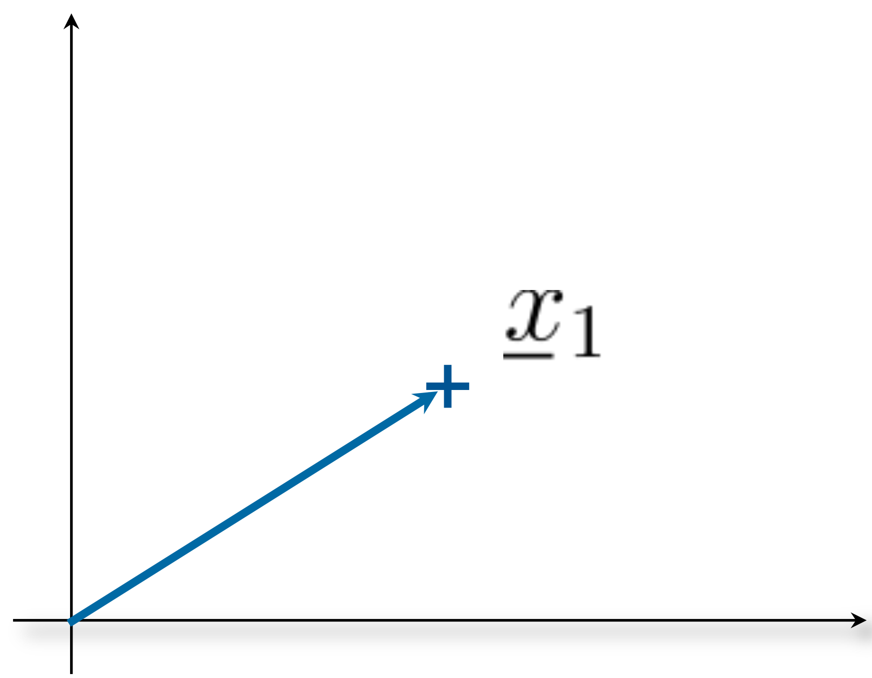
$$
\begin{aligned}
&\text{Initialize: } \underline{\theta} = 0 \\
&\text{Repeat until convergence:} \\
&\qquad \text{for } t = 1, \ldots, n \\
&\qquad\qquad \text{if } y_t(\underline{\theta} \cdot \underline{x}_t) \leq 0 \text{ (mistake)} \\
&\qquad\qquad\qquad \underline{\theta} \leftarrow \underline{\theta} + y_t \underline{x}_t
\end{aligned}
$$

- The algorithm will converge (no mistakes) if the training points are *linearly separable*, otherwise it won't converge

Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

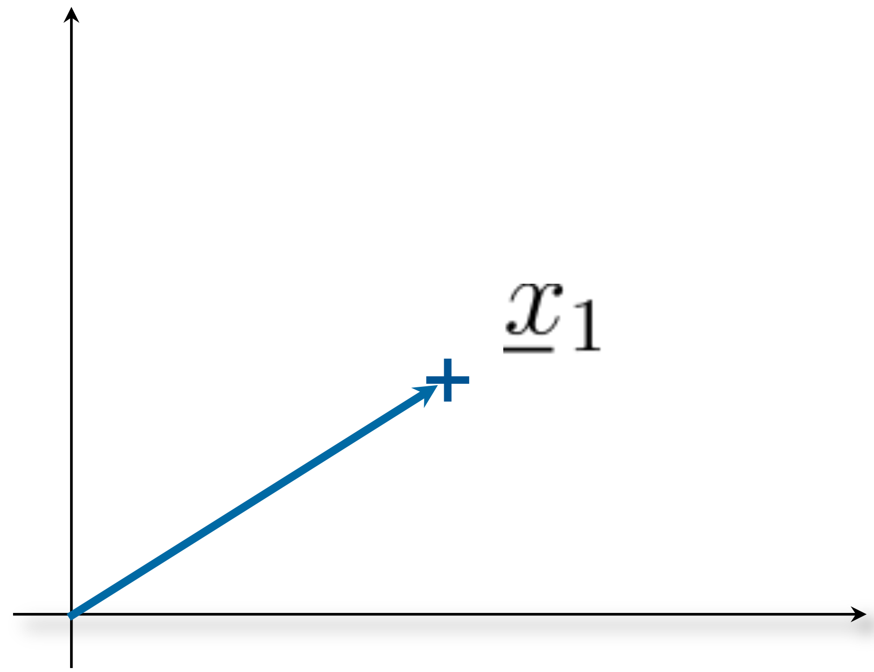


Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

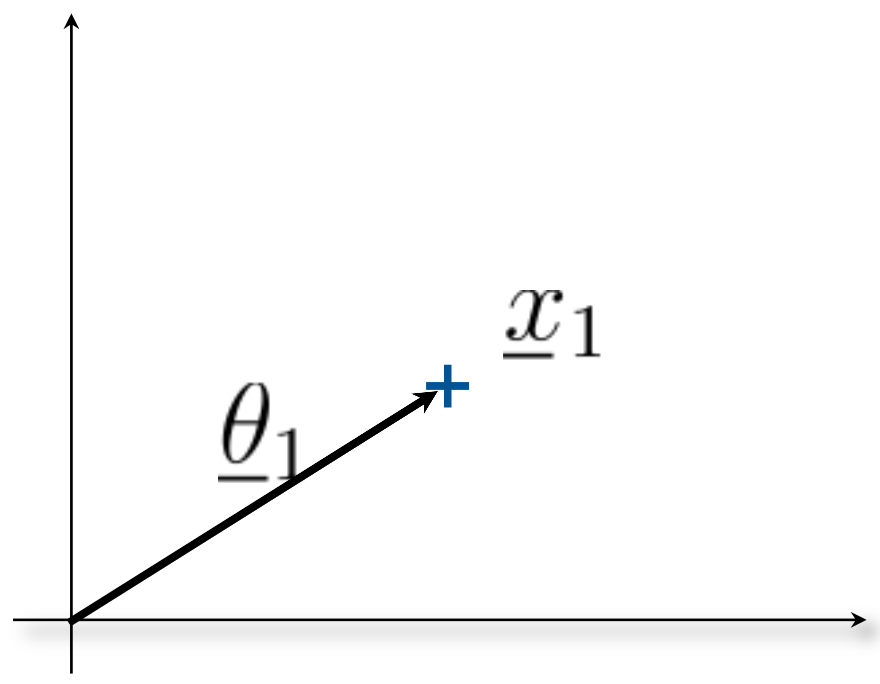


Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

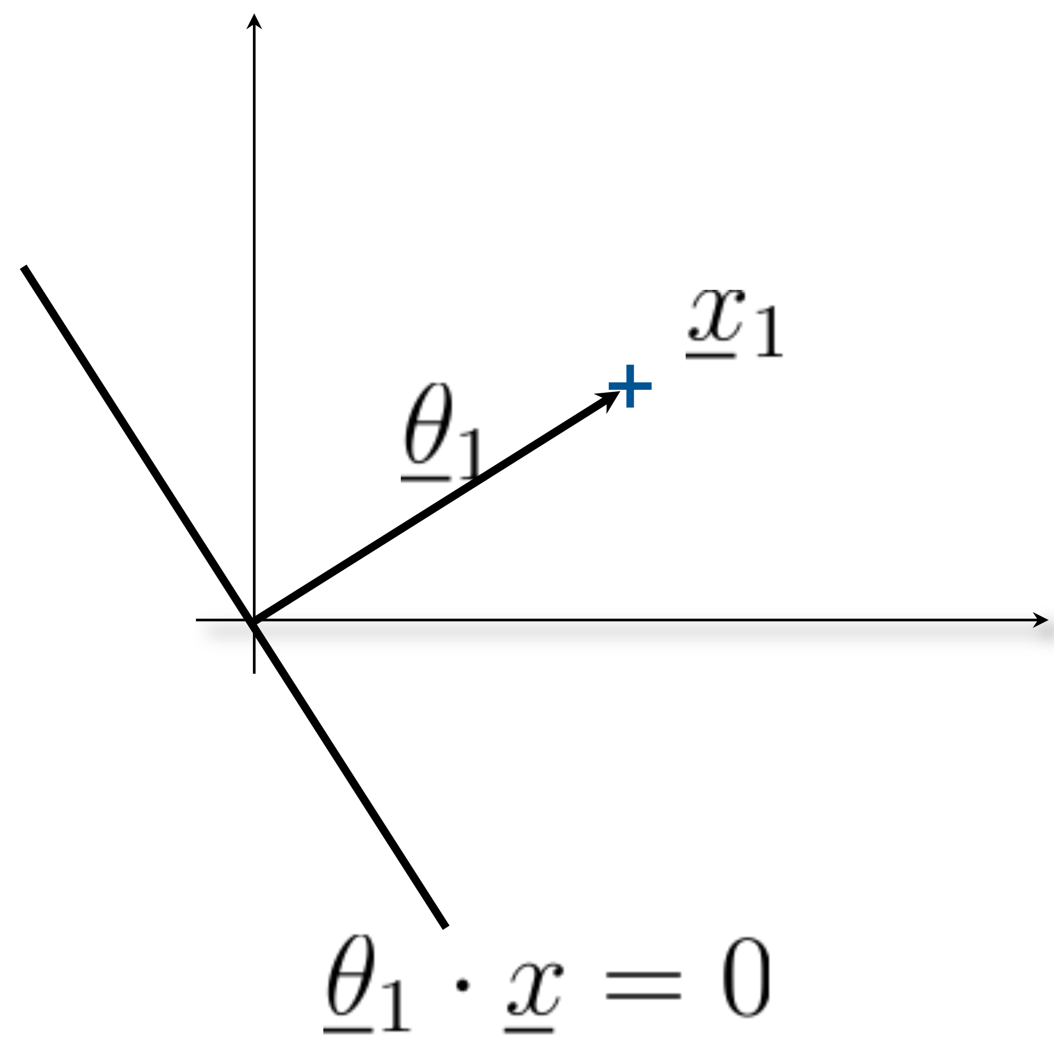
$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$



Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$
$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

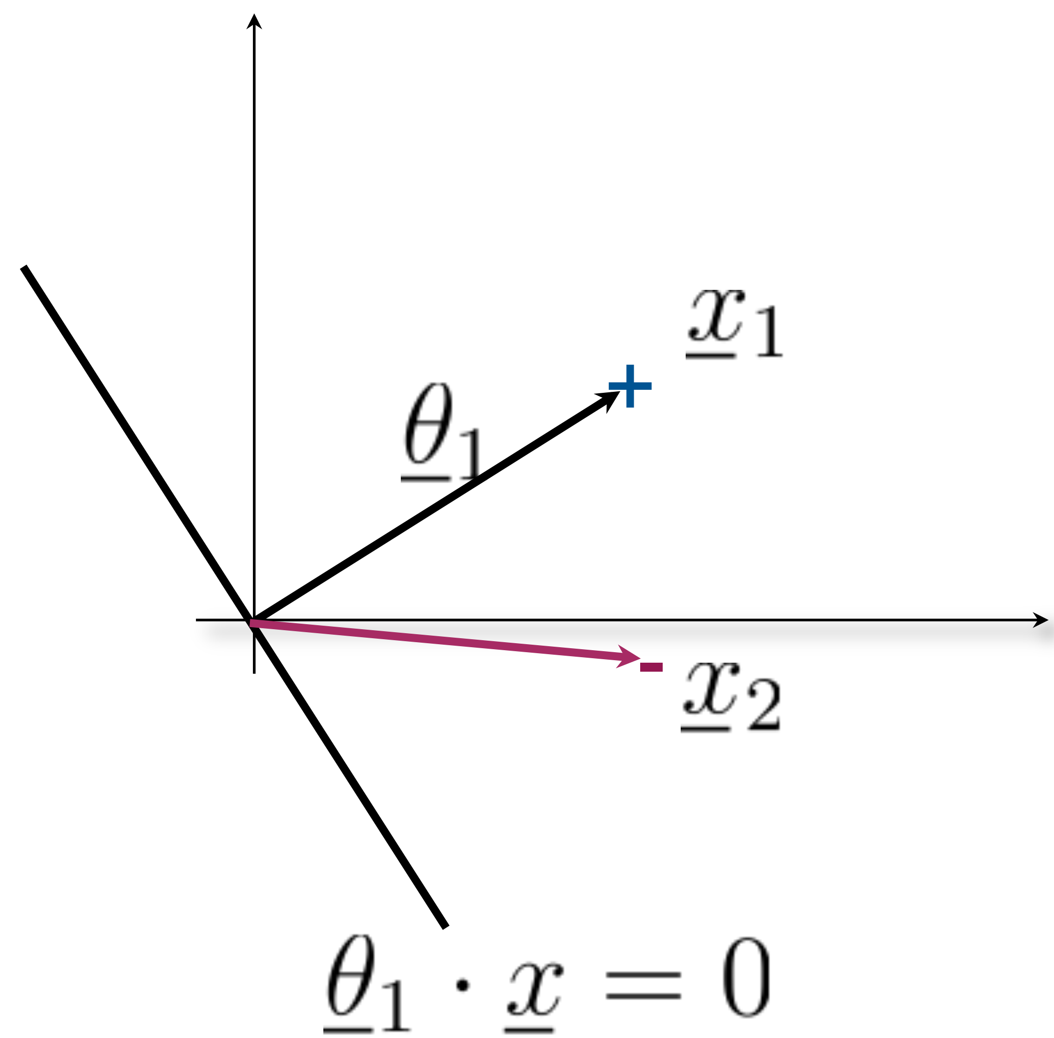


Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$



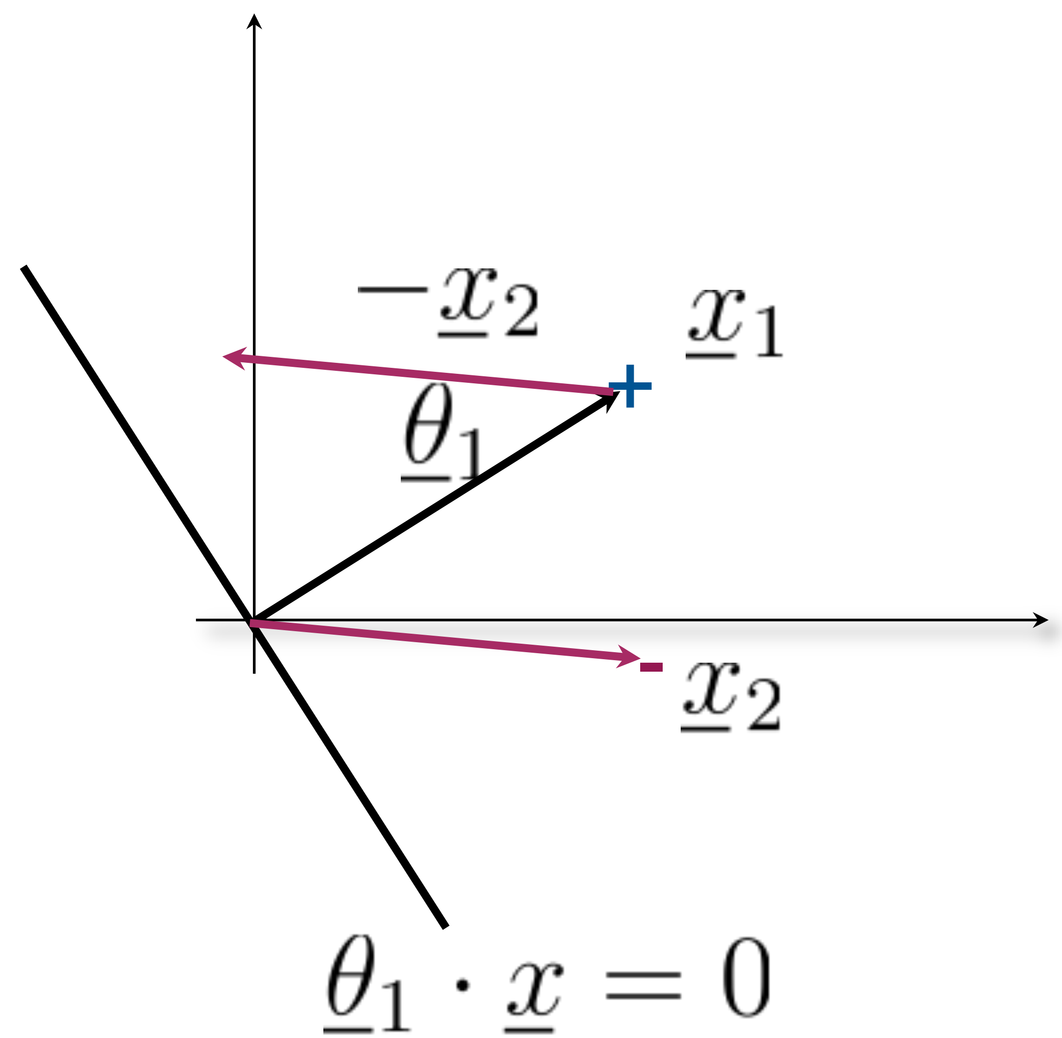
Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



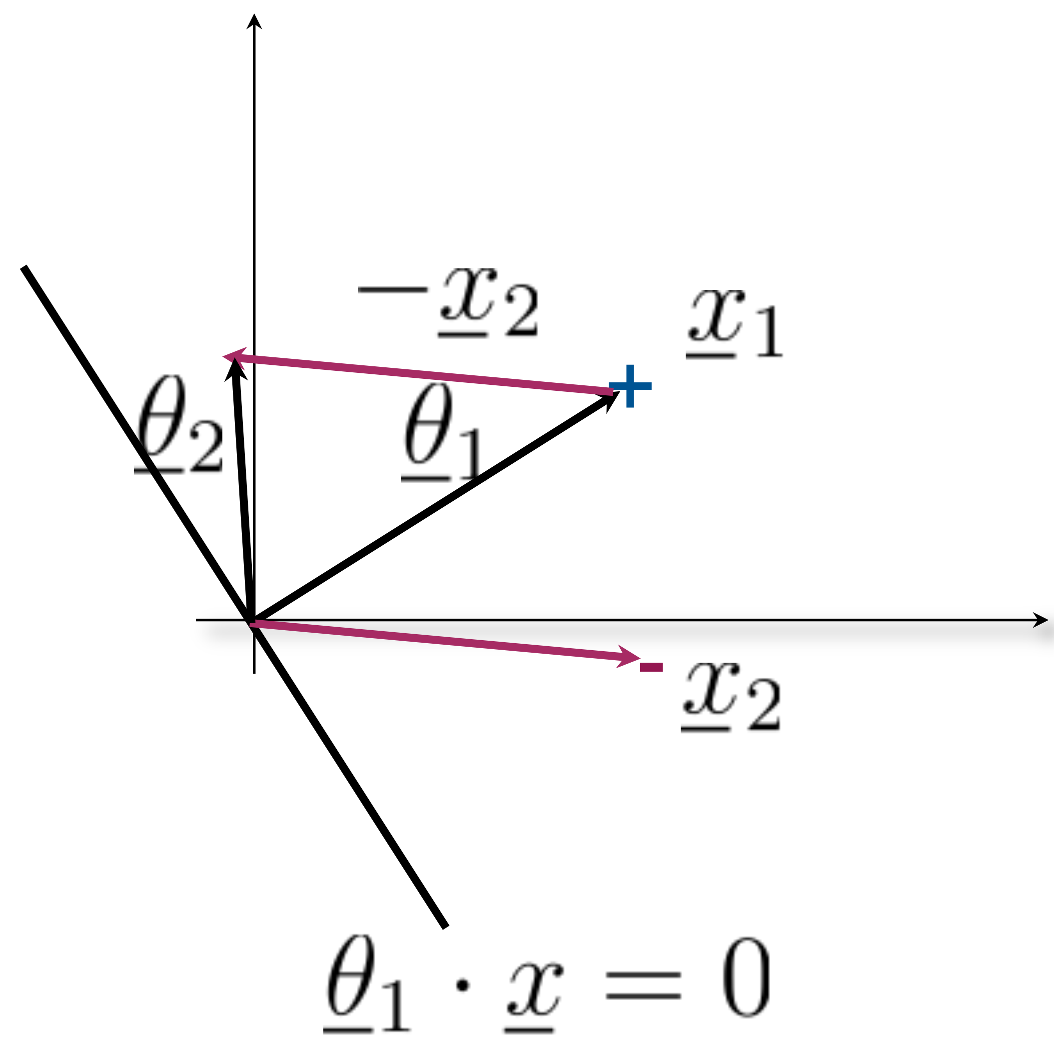
Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



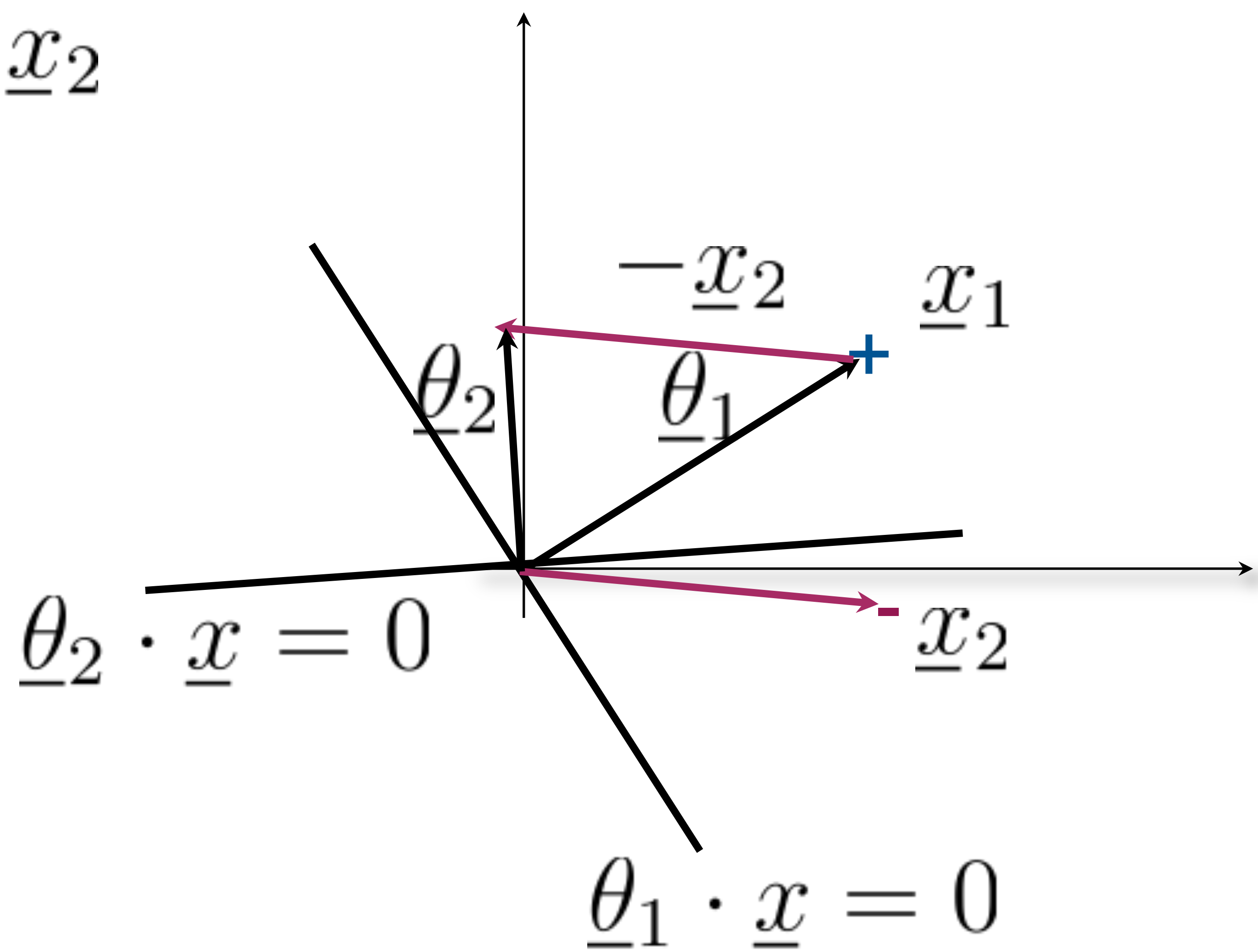
Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



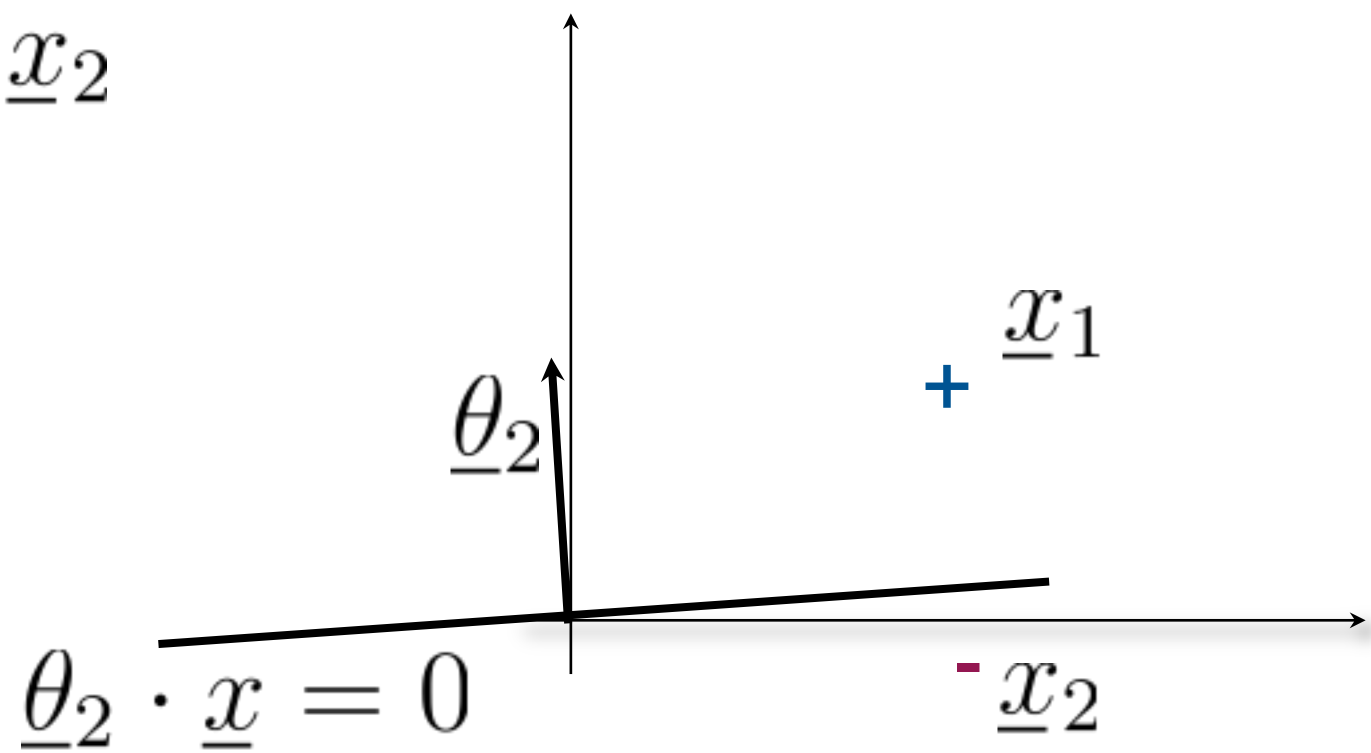
Perceptron algorithm

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

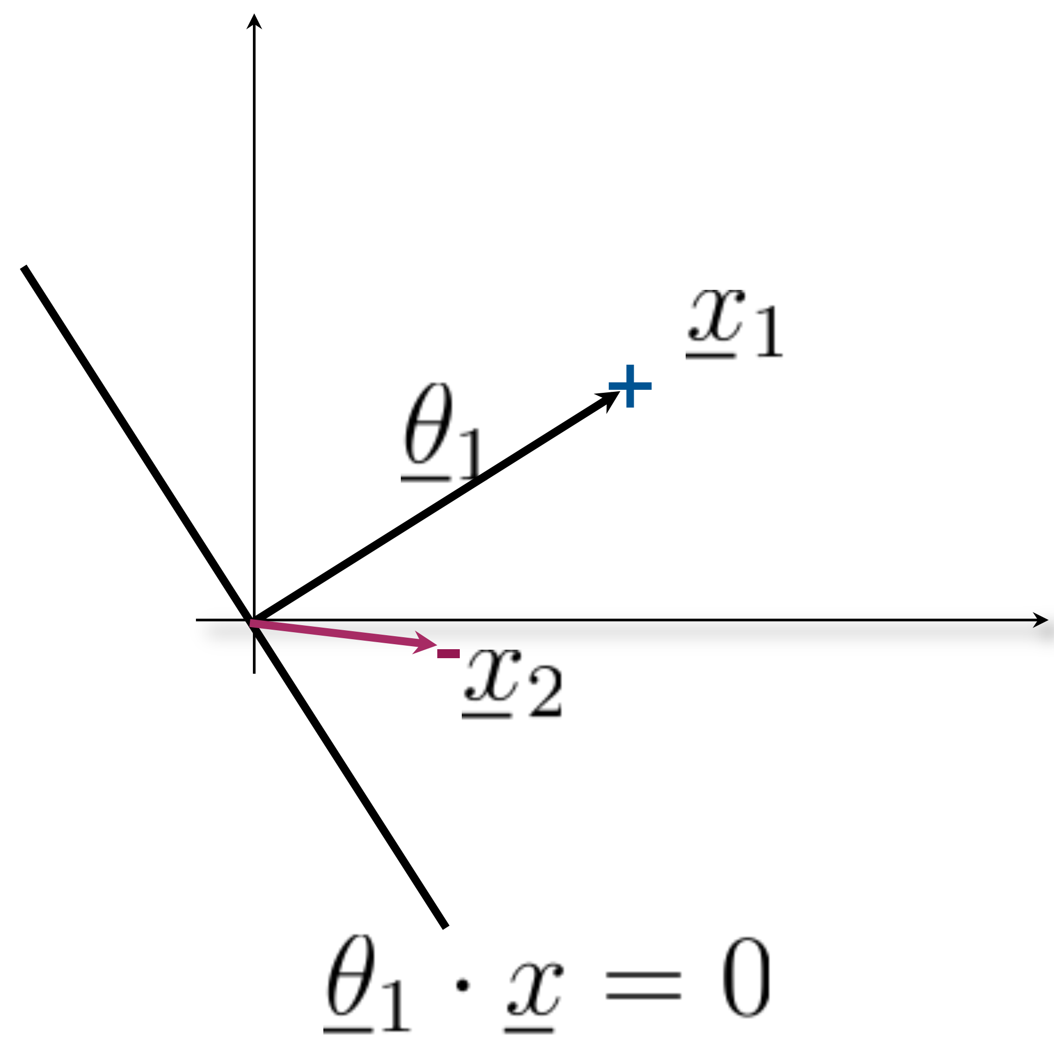
$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



Perceptron algorithm (take 2)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$
$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$



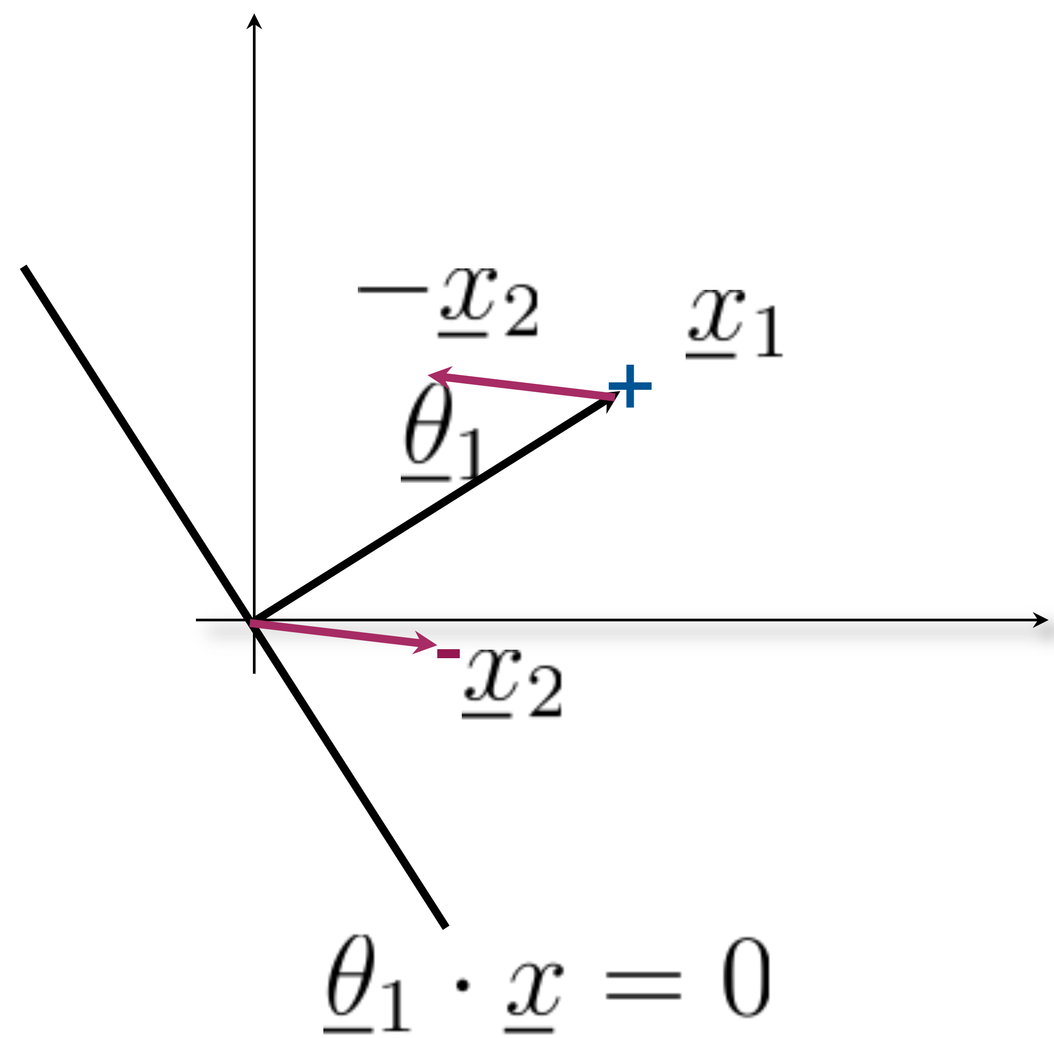
Perceptron algorithm (take 2)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



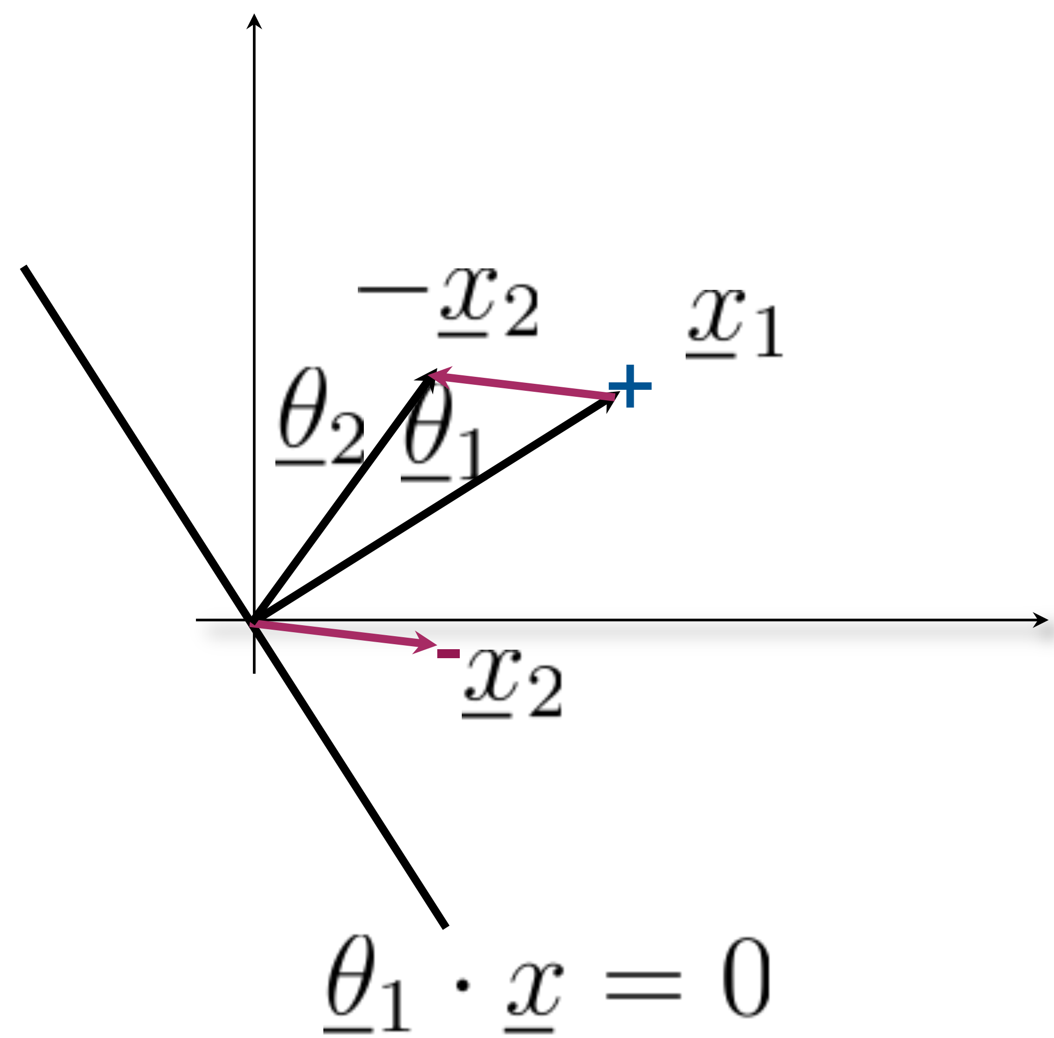
Perceptron algorithm (take 2)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



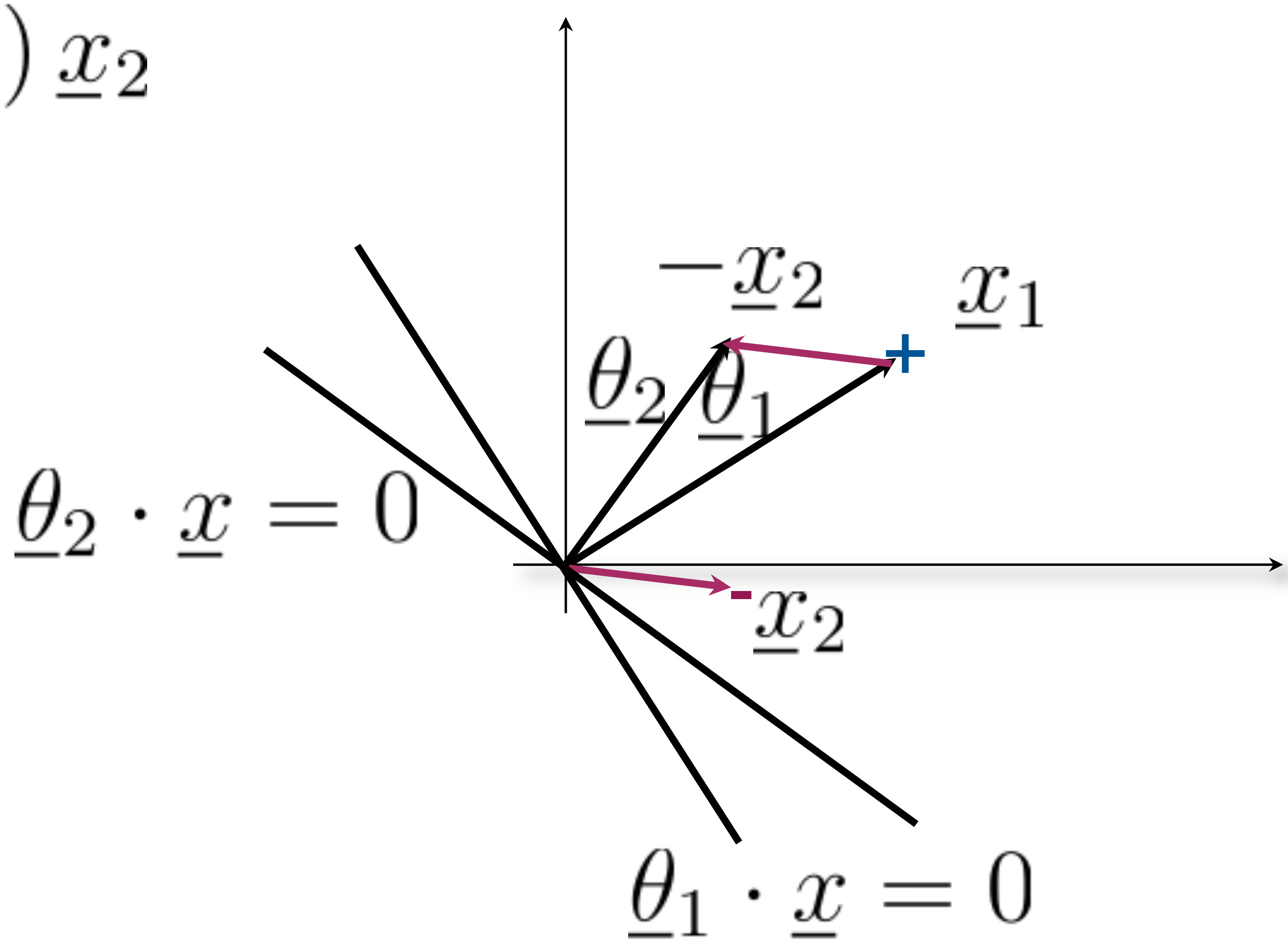
Perceptron algorithm (take 2)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



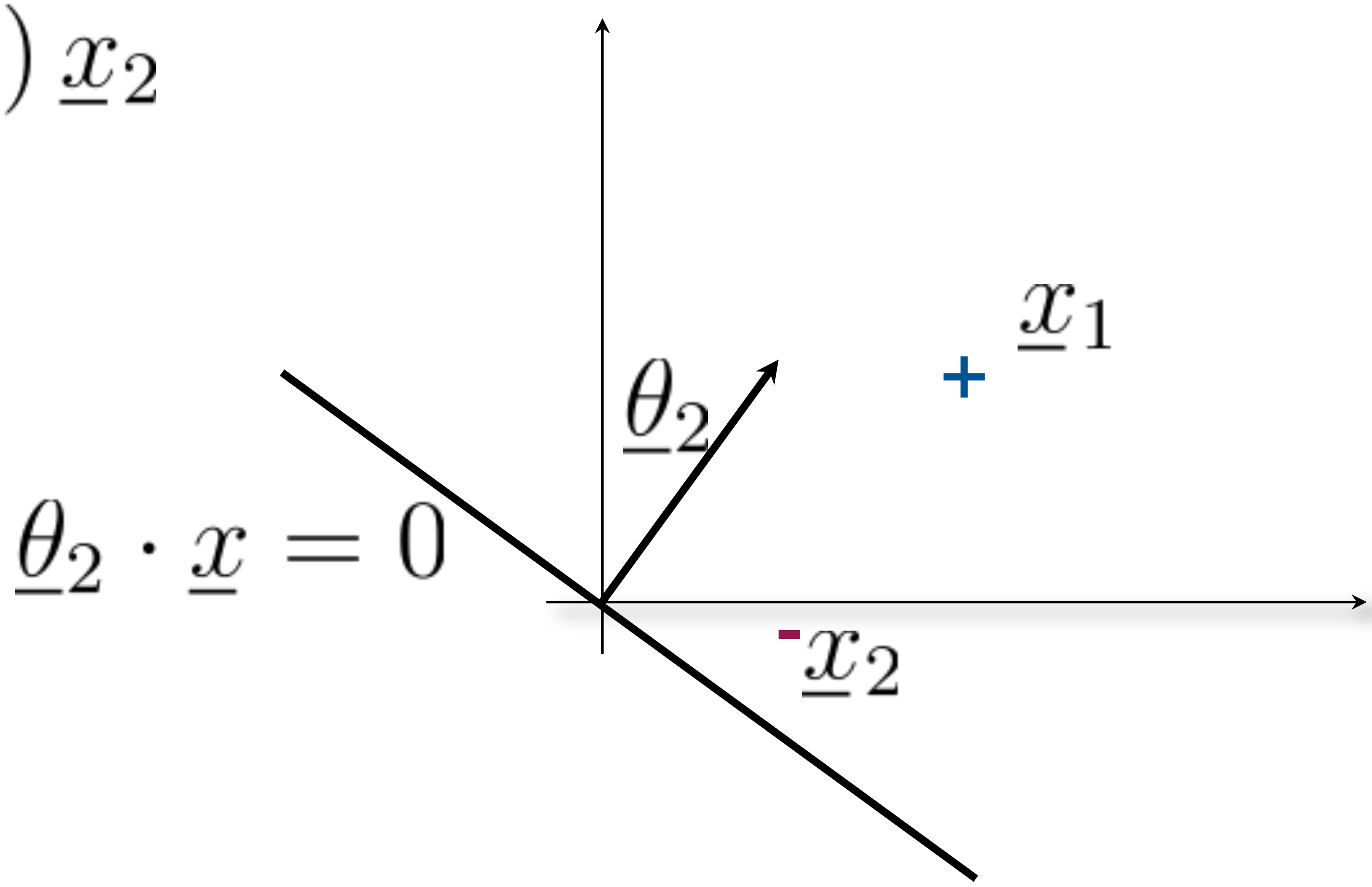
Perceptron algorithm (take 2)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

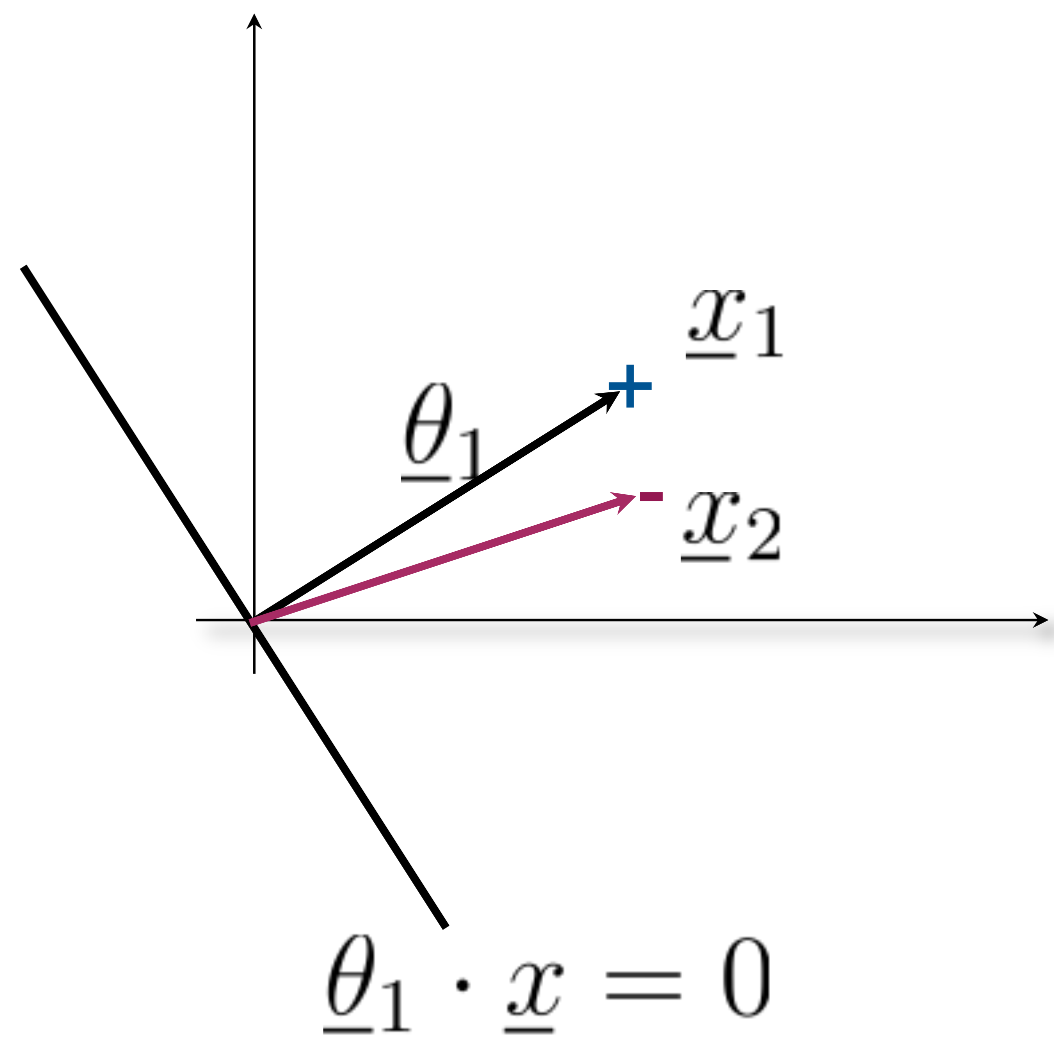
$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



Perceptron algorithm (take 3)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$
$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$



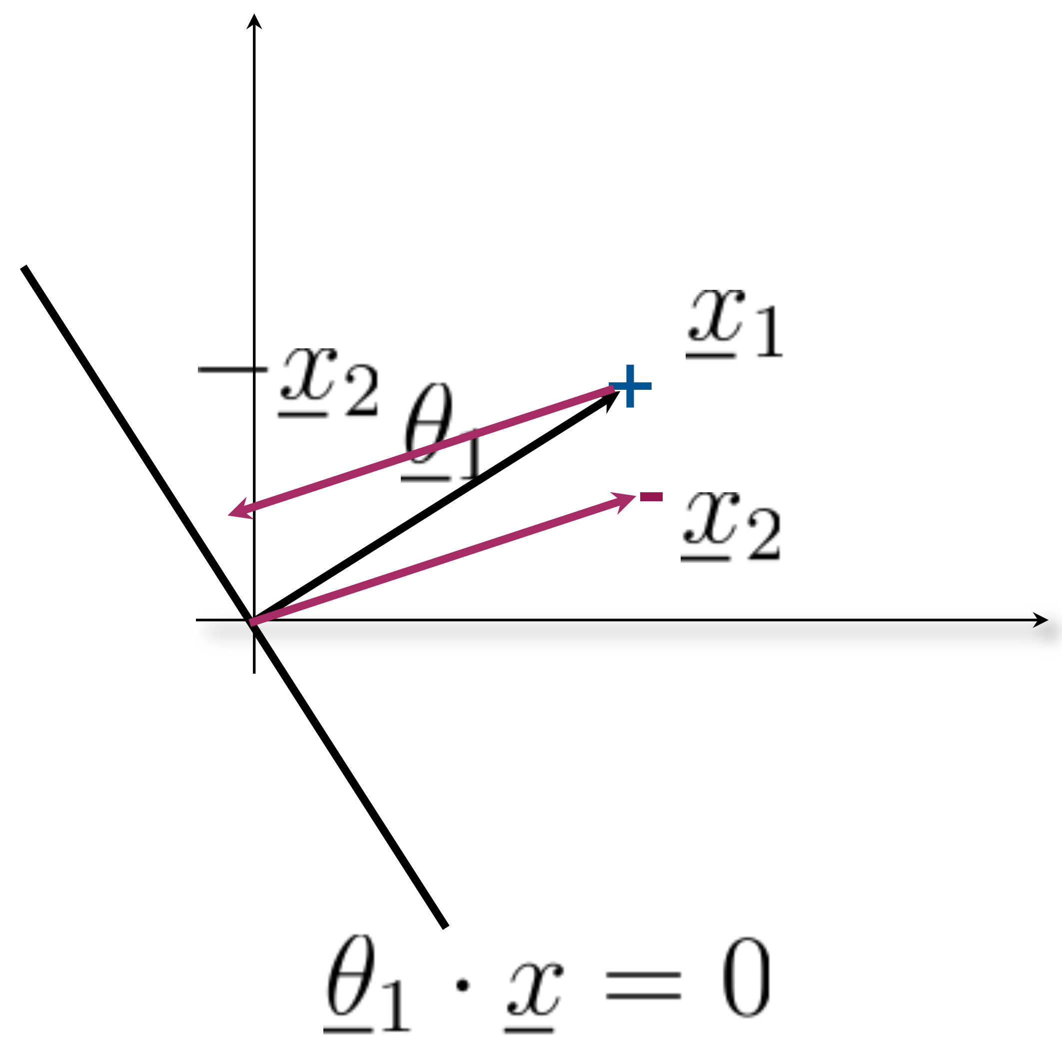
Perceptron algorithm (take 3)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



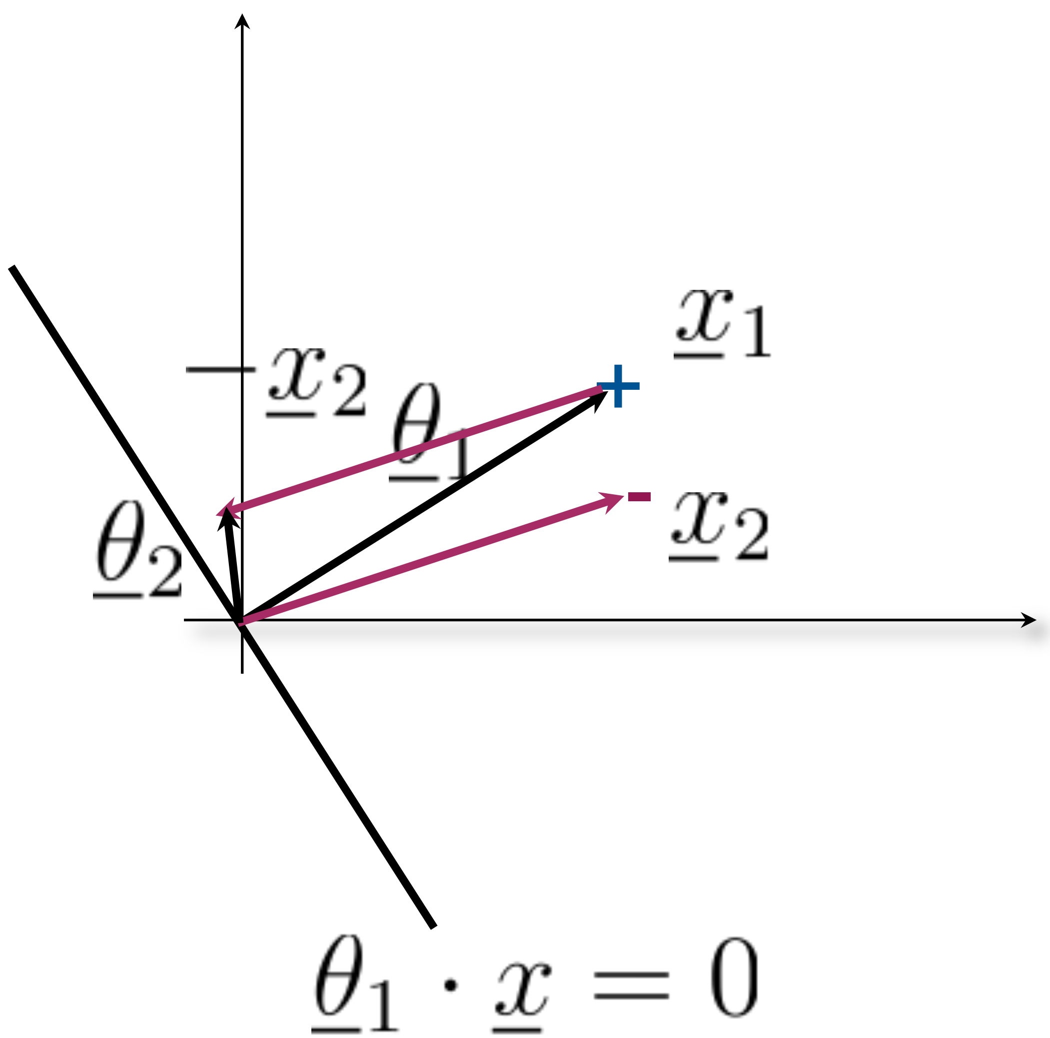
Perceptron algorithm (take 3)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



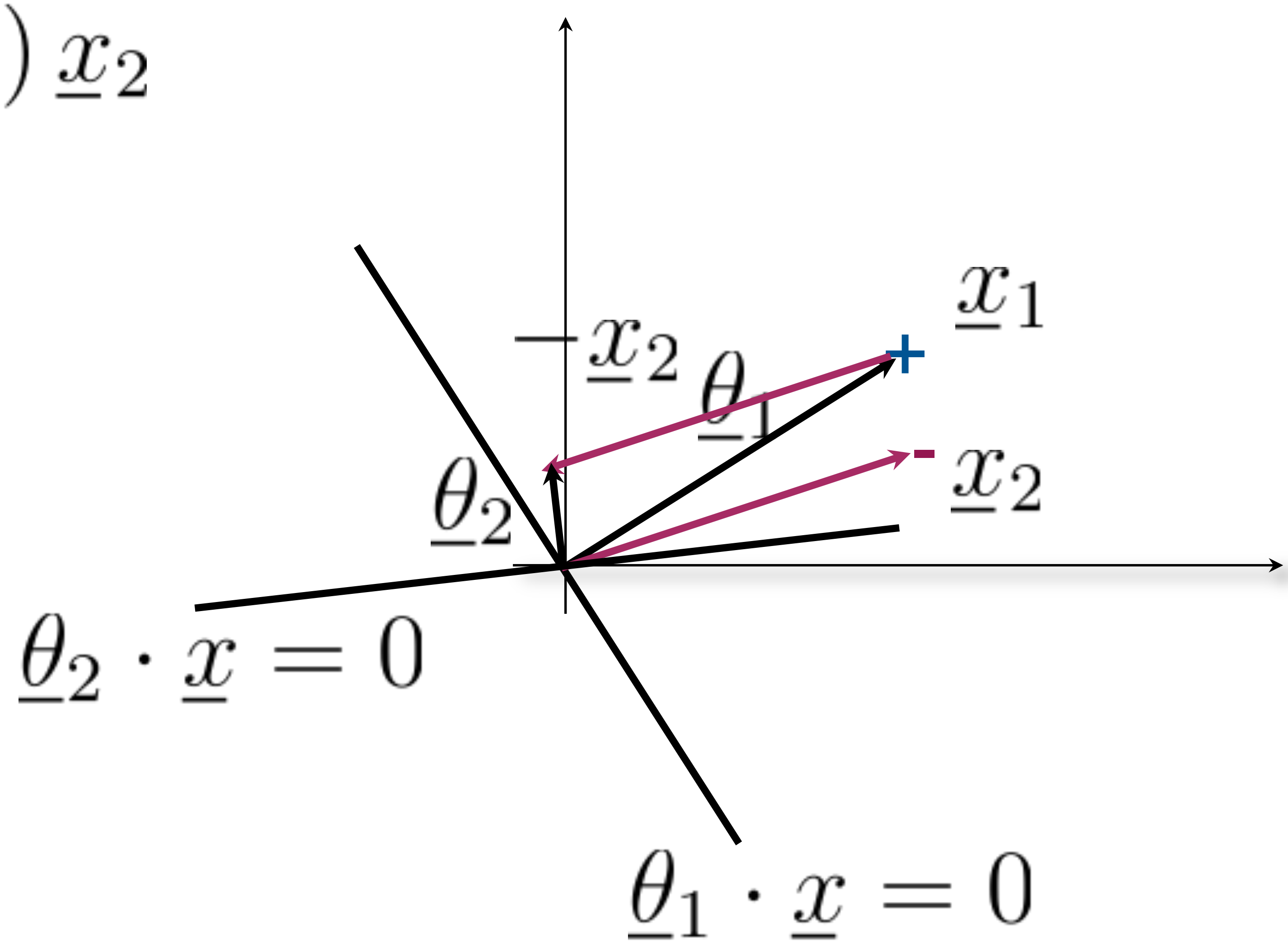
Perceptron algorithm (take 3)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



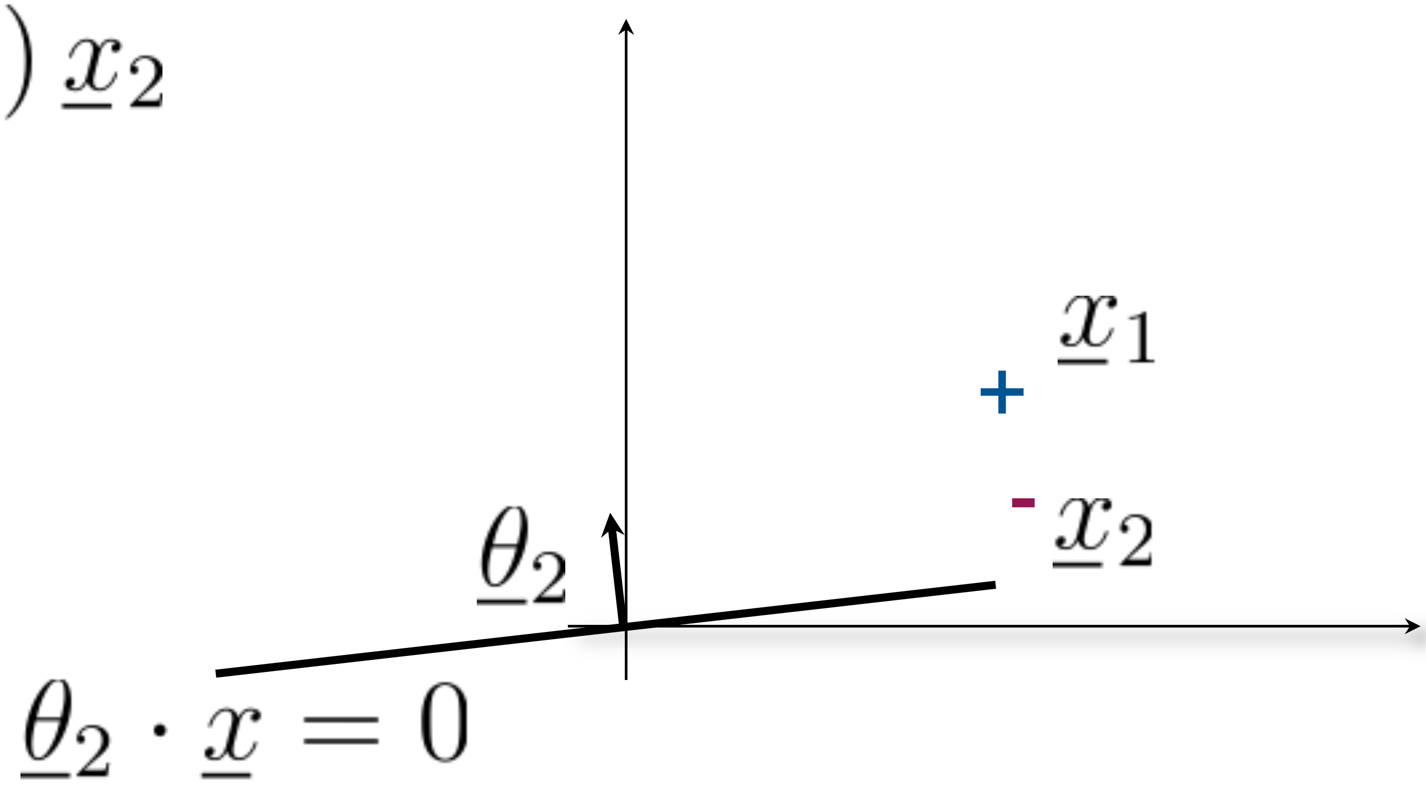
Perceptron algorithm (take 3)

- Iterative updates based on mistakes

$$\underline{\theta}_0 = 0$$

$$\underline{\theta}_1 = \underline{\theta}_0 + 1\,\underline{x}_1$$

$$\underline{\theta}_2 = \underline{\theta}_1 + (-1)\,\underline{x}_2$$



Perceptron algorithm: motivation

- If we make a mistake on the t^th training point, then

$$y_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

Perceptron algorithm: motivation

- If we make a mistake on the t^th training point, then

$$\mathrm{y}_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

$$y_t(\underline{\theta}' \cdot \underline{x}_t) \quad = \quad y_t([\underline{\theta} + y_t \underline{x}_t] \cdot \underline{x}_t)$$

Perceptron algorithm: motivation

- If we make a mistake on the t^th training point, then

$$\mathrm{y}_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

$$\begin{aligned} y_t(\underline{\theta}' \cdot \underline{x}_t) &= y_t([\underline{\theta} + y_t \underline{x}_t] \cdot \underline{x}_t) \\ &= y_t(\underline{\theta} \cdot \underline{x}_t + y_t \underline{x}_t \cdot \underline{x}_t) \end{aligned}$$

Perceptron algorithm: motivation

- If we make a mistake on the t[th] training point, then

$$\mathrm{y}_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

$$\begin{aligned} y_t(\underline{\theta}' \cdot \underline{x}_t) &= y_t([\underline{\theta} + y_t \underline{x}_t] \cdot \underline{x}_t) \\ &= y_t(\underline{\theta} \cdot \underline{x}_t + y_t \underline{x}_t \cdot \underline{x}_t) \\ &= y_t(\underline{\theta} \cdot \underline{x}_t) + y_t^2 \underline{x}_t \cdot \underline{x}_t \end{aligned}$$

Perceptron algorithm: motivation

- If we make a mistake on the t^th^ training point, then

$$\mathrm{y}_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

$$\begin{aligned} y_t(\underline{\theta}' \cdot \underline{x}_t) &= y_t([\underline{\theta} + y_t \underline{x}_t] \cdot \underline{x}_t) \\ &= y_t(\underline{\theta} \cdot \underline{x}_t + y_t \underline{x}_t \cdot \underline{x}_t) \\ &= y_t(\underline{\theta} \cdot \underline{x}_t) + y_t^2 \underline{x}_t \cdot \underline{x}_t \\ &= y_t(\underline{\theta} \cdot \underline{x}_t) + \|\underline{x}_t\|^2 \end{aligned}$$

Perceptron algorithm: motivation

- If we make a mistake on the t^{th} training point, then

$$\mathrm{y}_t(\underline{\theta} \cdot \underline{x}_t) \leq 0$$

- After the update, we have

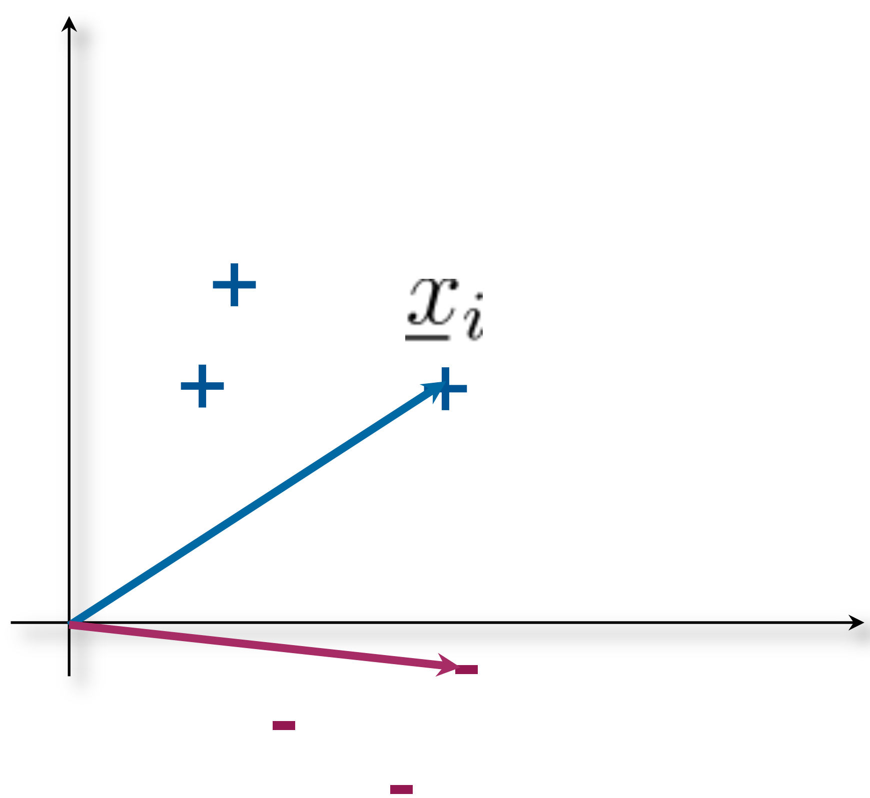
$$\underline{\theta}' = \underline{\theta} + y_t \underline{x}_t$$

$$\begin{aligned}
y_t(\underline{\theta}' \cdot \underline{x}_t) &= y_t([\underline{\theta} + y_t \underline{x}_t] \cdot \underline{x}_t) \\
&= y_t(\underline{\theta} \cdot \underline{x}_t + y_t \underline{x}_t \cdot \underline{x}_t) \\
&= y_t(\underline{\theta} \cdot \underline{x}_t) + y_t^2 \underline{x}_t \cdot \underline{x}_t \\
&= y_t(\underline{\theta} \cdot \underline{x}_t) + \|\underline{x}_t\|^2
\end{aligned}$$

- So that $y_t(\underline{\theta}' \cdot \underline{x}_t)$ increases based on the update

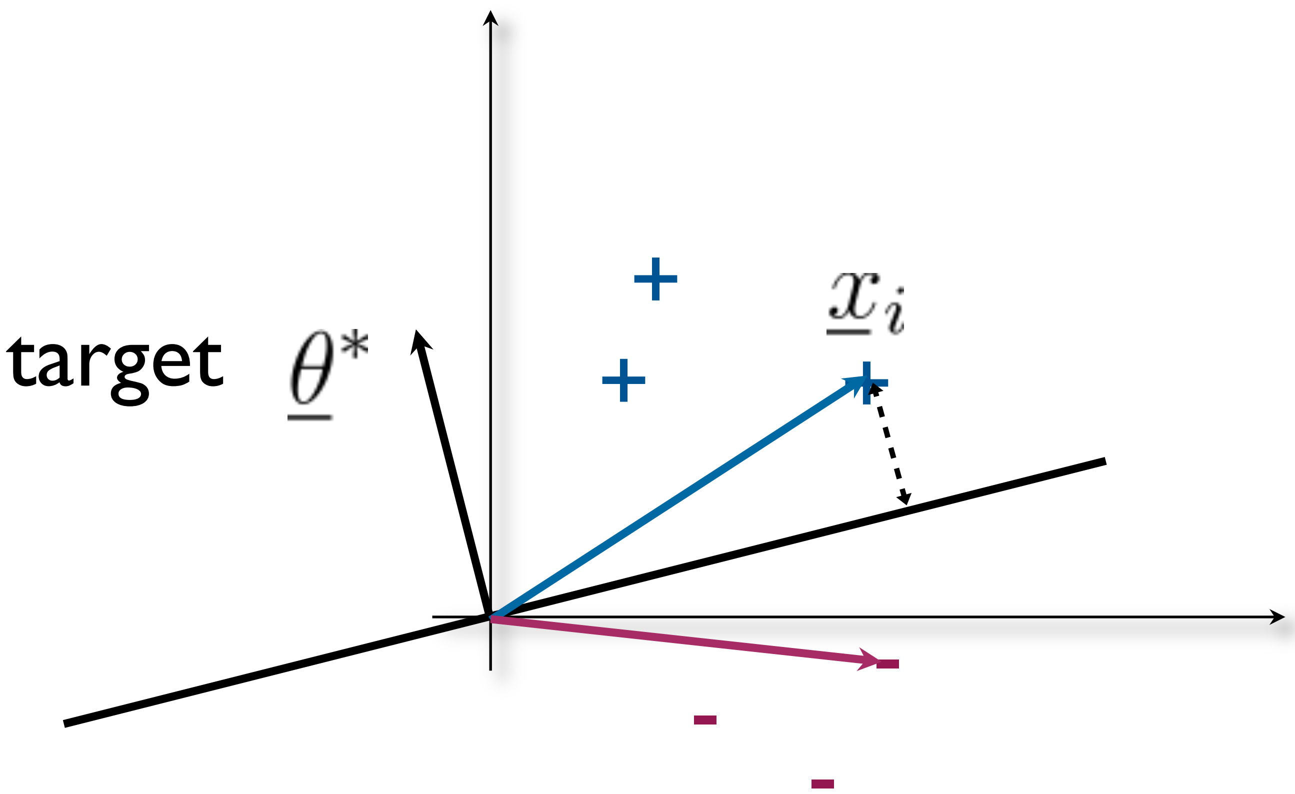
“Margin”

- We can get a handle on convergence by assuming that there exists a target classifier with good properties
- One such property is geometric margin, i.e., how close the separating boundary is to the points



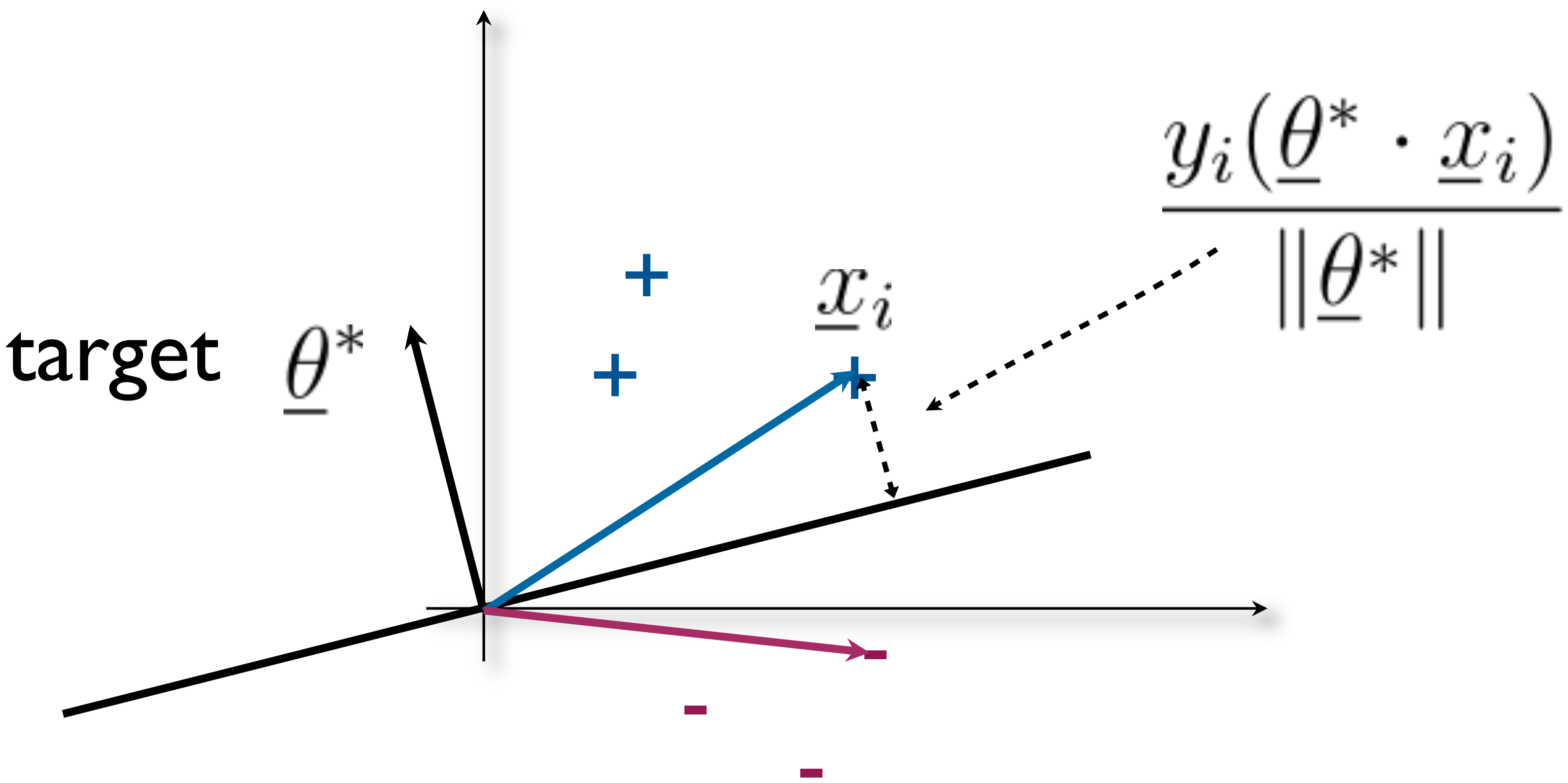
“Margin”

- We can get a handle on convergence by assuming that there exists a target classifier with good properties
- One such property is geometric margin, i.e., how close the separating boundary is to the points



“Margin”

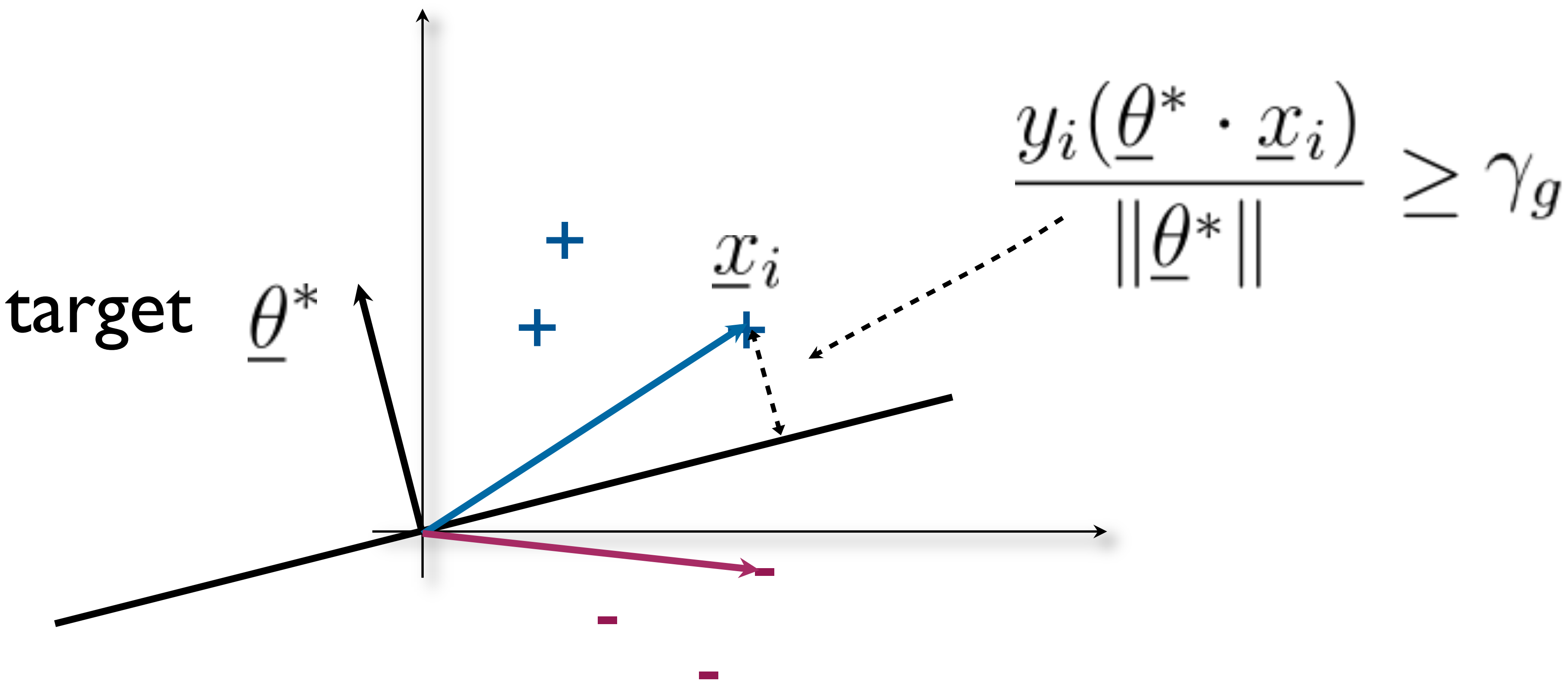
- We can get a handle on convergence by assuming that there exists a target classifier with good properties
- One such property is geometric margin, i.e., how close the separating boundary is to the points



Recall:
cos(angle)=dot product of two unit vectors
dotted distance = radius * cos(angle)

“Margin”

- We can get a handle on convergence by assuming that there exists a target classifier with good properties
- One such property is geometric margin, i.e., how close the separating boundary is to the points



Perceptron convergence theorem

- If there exists $\underline{\theta}^*$ such that

$$\frac{y_i(\underline{\theta}^* \cdot \underline{x}_i)}{\|\underline{\theta}^*\|} \geq \gamma_g\,,\; i = 1, \ldots, n$$

and $\|\underline{x}_i\| \leq R$ then the perceptron algorithm makes at most

$$\frac{R^2}{\gamma_g^2}$$

mistakes (on the training set).

- Note that the result does NOT depend on d or n