# Support Vector Machines

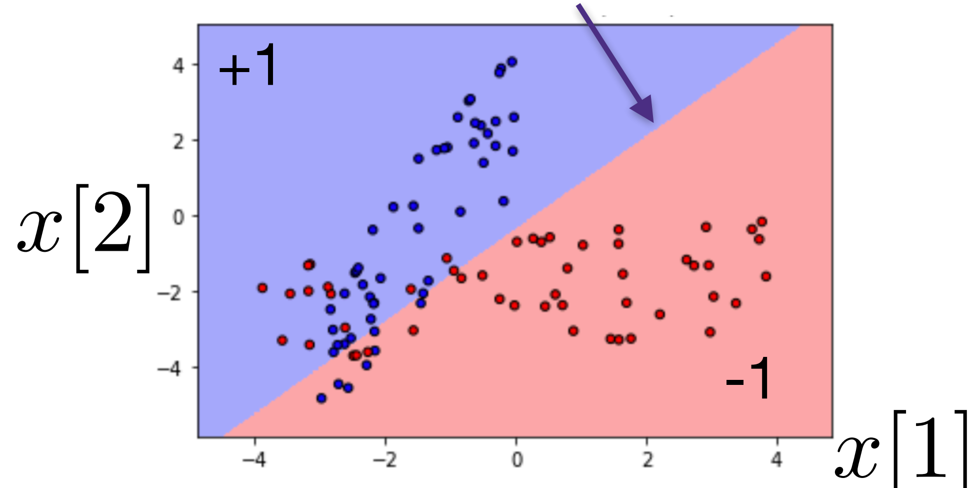
# Logistic regression for binary classification

- Data $\mathscr{D} = \{(x_i \in \mathbb{R}^d, y_i \in \{-1, +1\})\}_{i=1}^n$
- Model: $\hat{y} = x^T w + b$
- Loss function: logistic loss $\ell(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$
- Optimization: solve for

$$(\widehat{b}, \widehat{w}) = \arg\min_{b,w} \sum_{i=1}^{n} \log(1 + e^{-y_i(b + x_i^T w)})$$

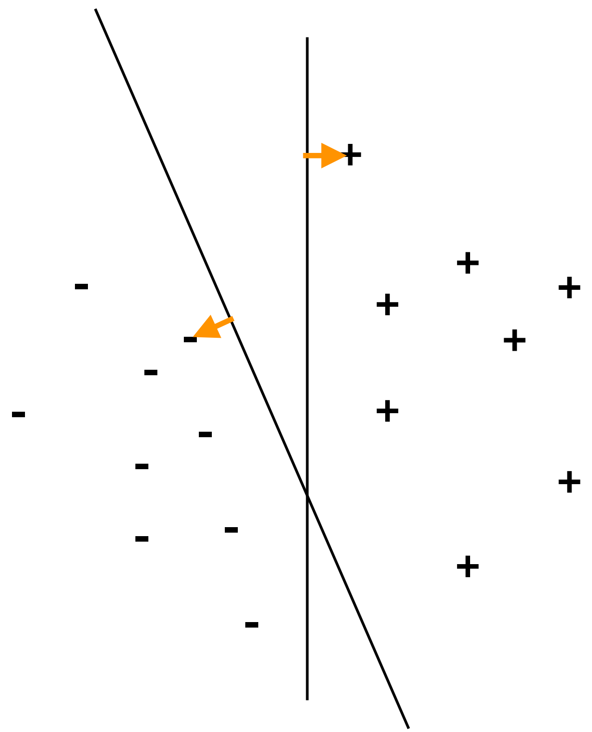
- As this is a **smooth convex** optimization, it can be solved efficiently using gradient descent
- Prediction: $\text{sign}(b + x^T w)$

decision boundary at $w^T x + b = 0$

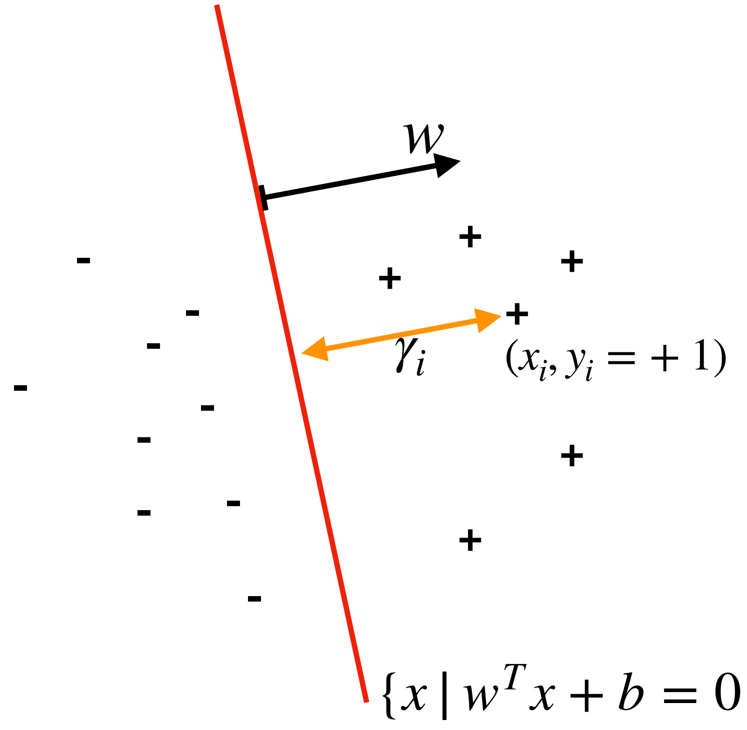


# How do we choose the best linear classifier?

- Informally, **margin** of a set of examples to a decision boundary is the distance to the closest point to the decision boundary
- For linearly separable datasets, **maximum margin** classifier is a natural choice
- Large margin implies that the decision boundary can change without losing accuracy, so the learned model is more robust against new data points

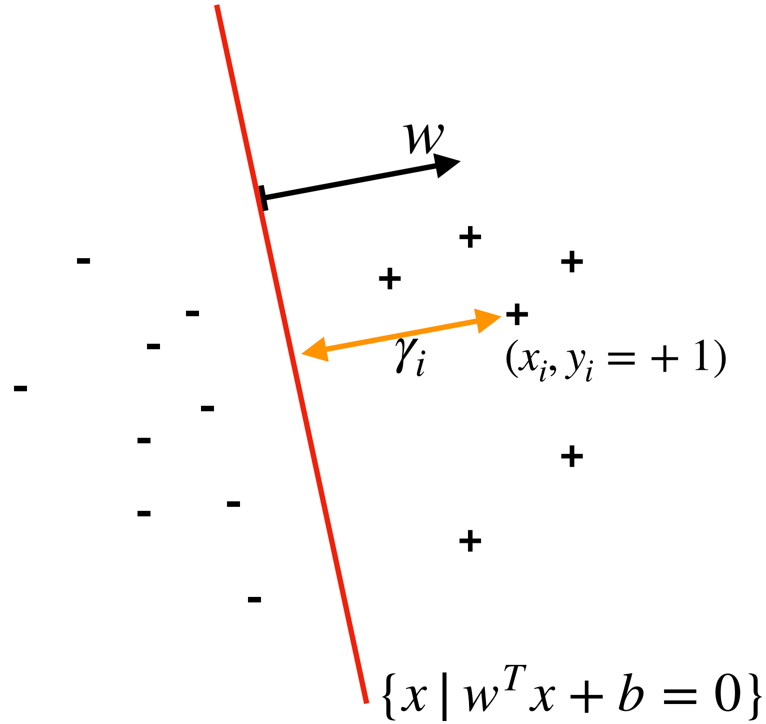


# Geometric margin



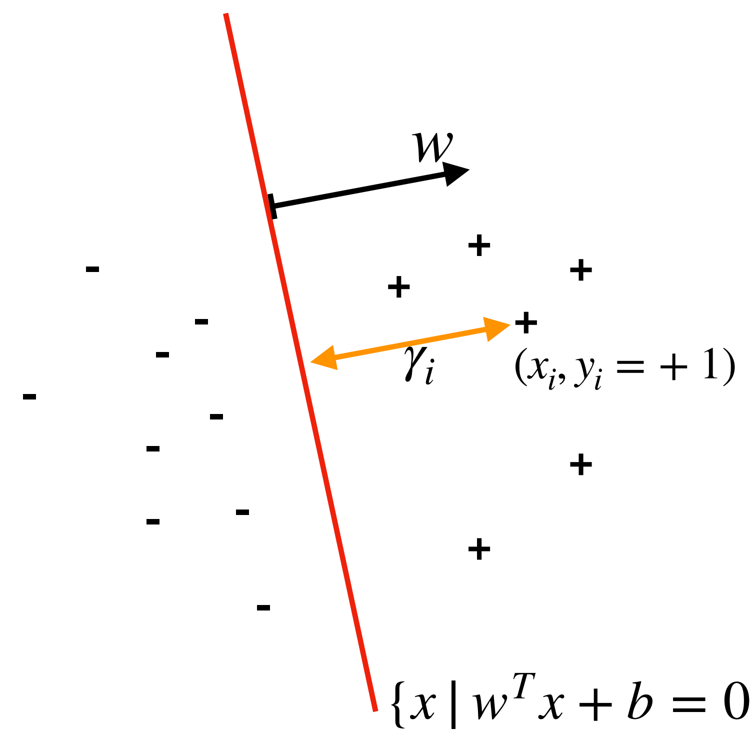
# Geometric margin

- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^n$, with $y_i \in \{-1, +1\}$



# Geometric margin

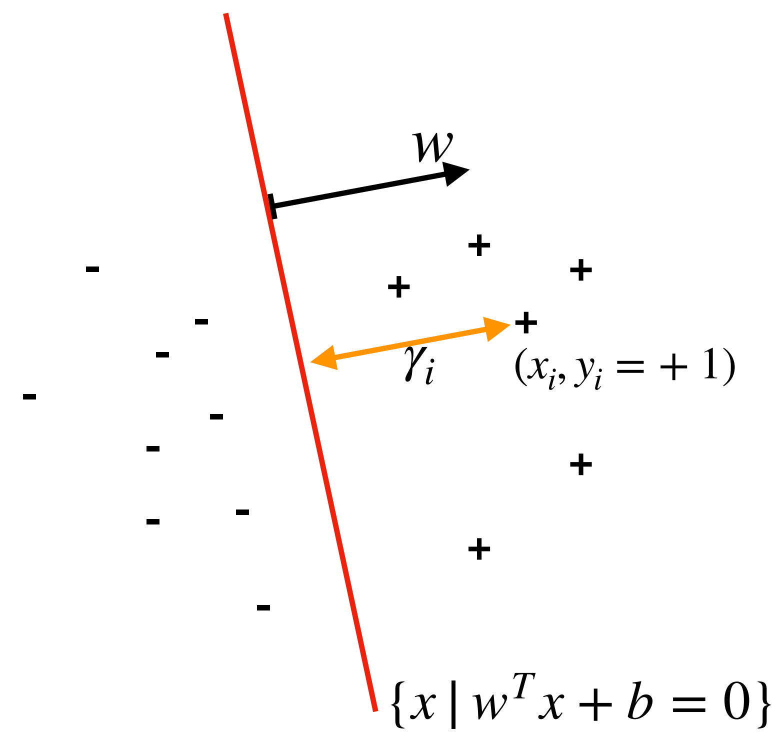
- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^n$, with $y_i \in \{-1, +1\}$
- and a linear classifier $(w, b) \in \mathbb{R}^d \times \mathbb{R}$



# Geometric margin

- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^n$, with $y_i \in \{-1, +1\}$
- and a linear classifier $(w, b) \in \mathbb{R}^d \times \mathbb{R}$
- such that the decision boundary is
  a separating hyperplane $\{x \mid \underbrace{b + w_1 x[1] + w_2 x[2] + \cdots + w_d x[d]}_{w^T x + b} = 0\}$,

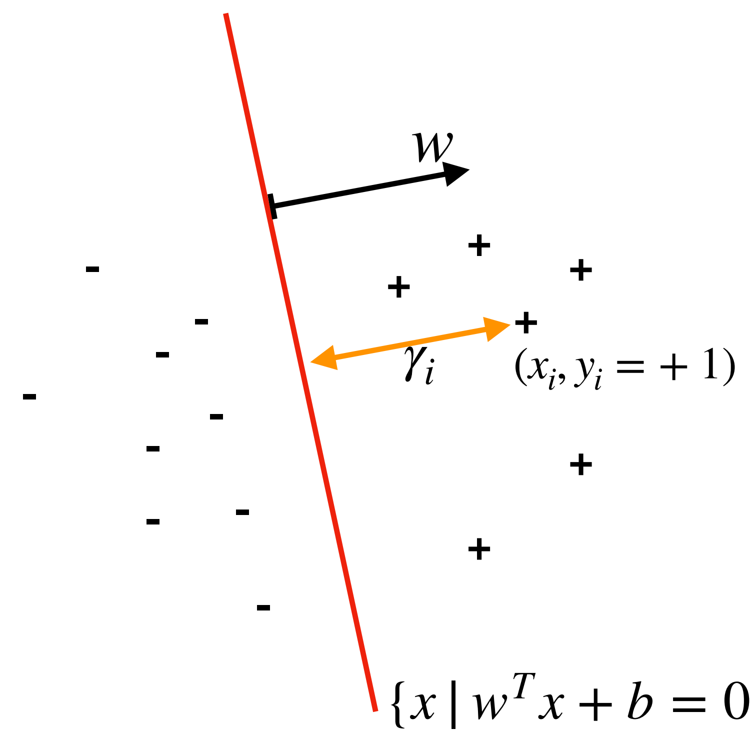
  which is the hyperplane orthogonal to $w$ with a shift of $b$



# Geometric margin

- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^n$, with $y_i \in \{-1, +1\}$
- and a linear classifier $(w, b) \in \mathbb{R}^d \times \mathbb{R}$
- such that the decision boundary is
  a separating hyperplane $\{x \mid \underbrace{b + w_1 x[1] + w_2 x[2] + \cdots + w_d x[d]}_{w^T x + b} = 0\}$,
  which is the hyperplane orthogonal to $w$ with a shift of $b$
- we define **margin** of $(b, w)$
  with respect to a training example $(x_i, y_i)$ as
  the distance from the point $(x_i, y_i)$ to the
  decision boundary, which is

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



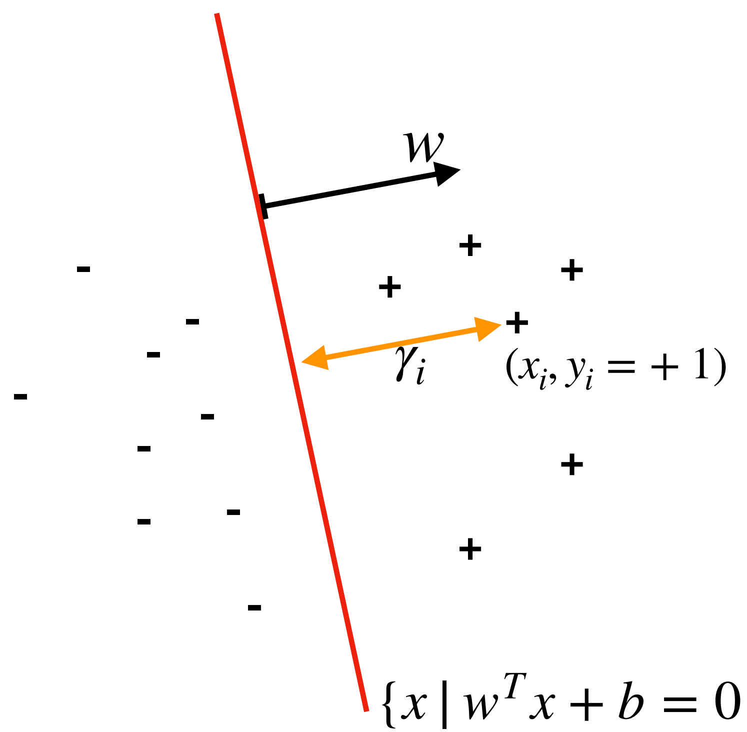
# Geometric margin

- Given a set of training examples $\{(x_i, y_i)\}_{i=1}^{n}$, with $y_i \in \{-1, +1\}$
- and a linear classifier $(w, b) \in \mathbb{R}^d \times \mathbb{R}$
- such that the decision boundary is
  a separating hyperplane $\{x \mid \underbrace{b + w_1 x[1] + w_2 x[2] + \cdots + w_d x[d]}_{w^T x + b} = 0\}$,

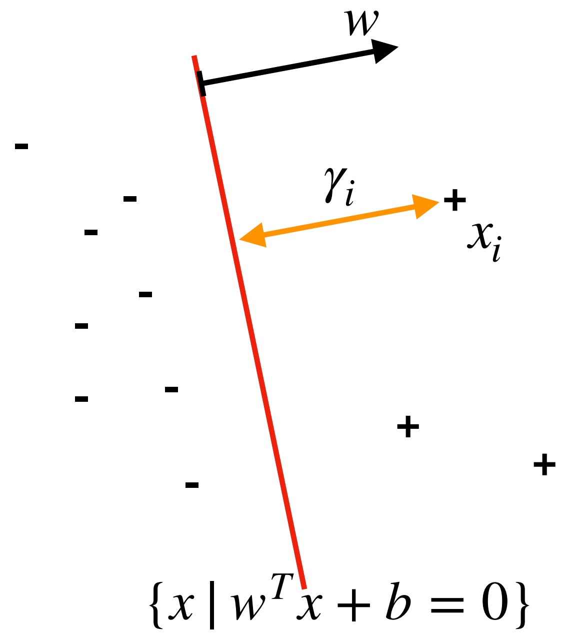
  which is the hyperplane orthogonal to $w$ with a shift of $b$
- we define **margin** of $(b, w)$
  with respect to a training example $(x_i, y_i)$ as
  the distance from the point $(x_i, y_i)$ to the
  decision boundary, which is

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$

(The proof is on the next slide)

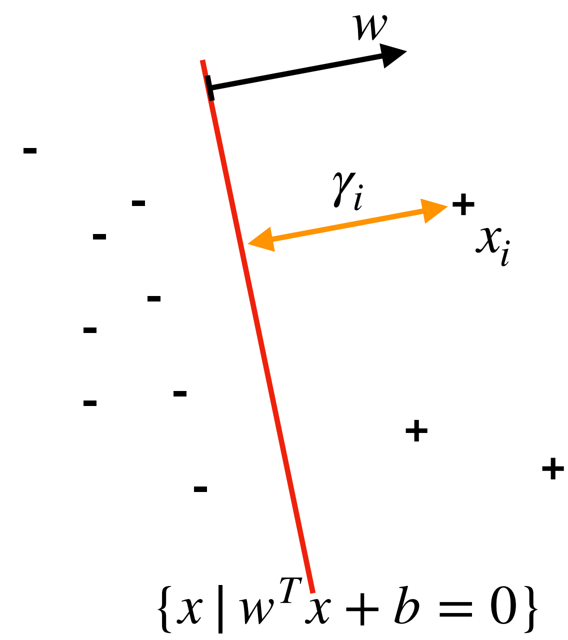


# Geometric margin



# Geometric margin

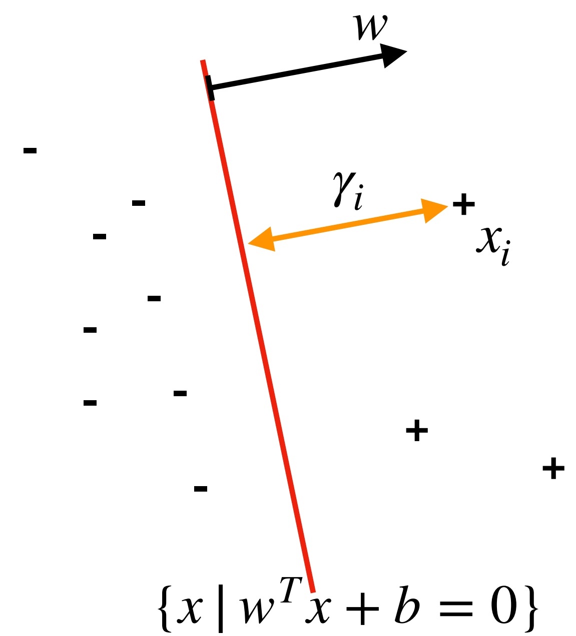
- The distance $\gamma_i$ from a hyperplane $\{x \,|\, w^T x + b = 0\}$ to a point $x_i$ can be computed geometrically as follows:



# Geometric margin

- The distance $\gamma_i$ from a hyperplane $\{x \mid w^T x + b = 0\}$ to a point $x_i$ can be computed geometrically as follows:
- We know that if you move from $x_i$ in the negative direction of $w$ by length $\gamma_i$, you arrive at the line, which can be written as

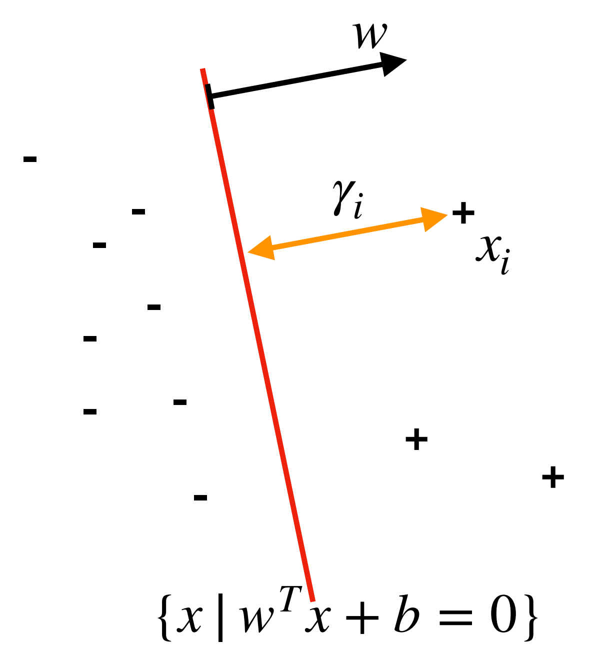
$$\left( x_i - \frac{w}{\|w\|_2} \gamma_i \right) \text{ is in } \{x \mid w^T x + b = 0\}$$



# Geometric margin

- The distance $\gamma_i$ from a hyperplane $\{x \,|\, w^T x + b = 0\}$ to a point $x_i$ can be computed geometrically as follows:
- We know that if you move from $x_i$ in the negative direction of $w$ by length $\gamma_i$, you arrive at the line, which can be written as

$$\left( x_i - \frac{w}{\|w\|_2} \gamma_i \right) \text{ is in } \{x \,|\, w^T x + b = 0\}$$




- So we can plug the point in the formula:

$$w^T \left( x_i - \frac{w}{\|w\|_2} \gamma_i \right) + b \;=\; 0$$

which is

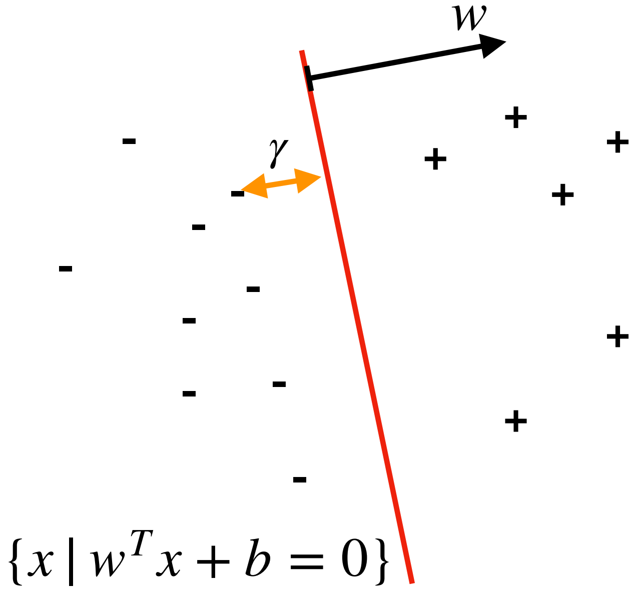
$$w^T x_i - \frac{\|w\|_2^2}{\|w\|_2} \gamma_i + b \;=\; 0$$

and hence

$$\gamma_i \;=\; \frac{w^T x_i + b}{\|w\|_2},$$

We multiply the formula by $y_i$ so that for negative samples we use the opposite direction of $-w$ instead of $w$

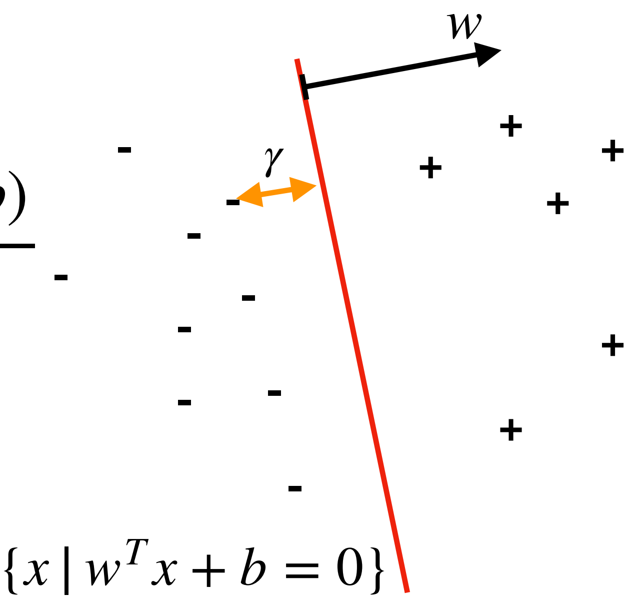
# Maximum margin classifiers



# Maximum margin classifiers

- The **margin** with respect to a set is defined as

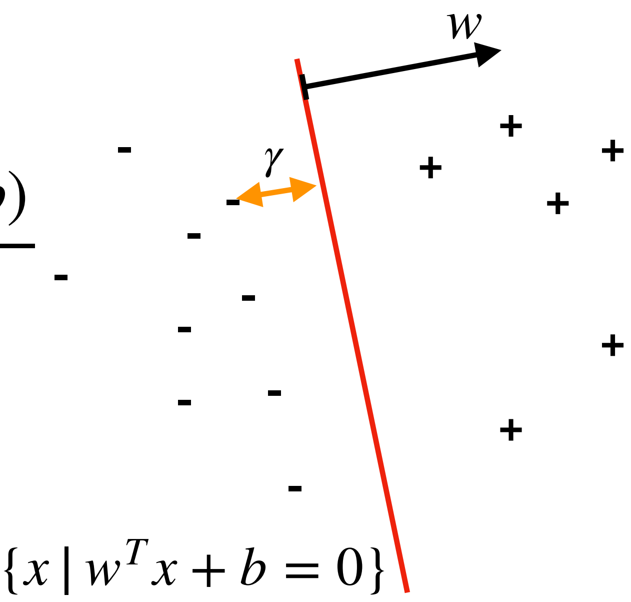
$$\gamma = \min_{i \in \{1,\ldots,n\}} \gamma_i = \min_i y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



# Maximum margin classifiers

- The **margin** with respect to a set is defined as

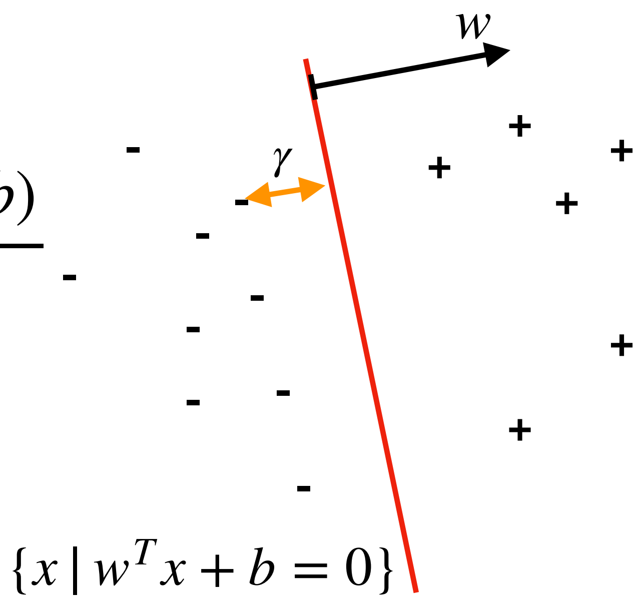
$$\gamma = \min_{i \in \{1,\ldots,n\}} \gamma_i = \min_i y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



# Maximum margin classifiers

- The **margin** with respect to a set is defined as

$$\gamma = \min_{i \in \{1,\ldots,n\}} \gamma_i = \min_i y_i \frac{(w^T x_i + b)}{\|w\|_2}$$

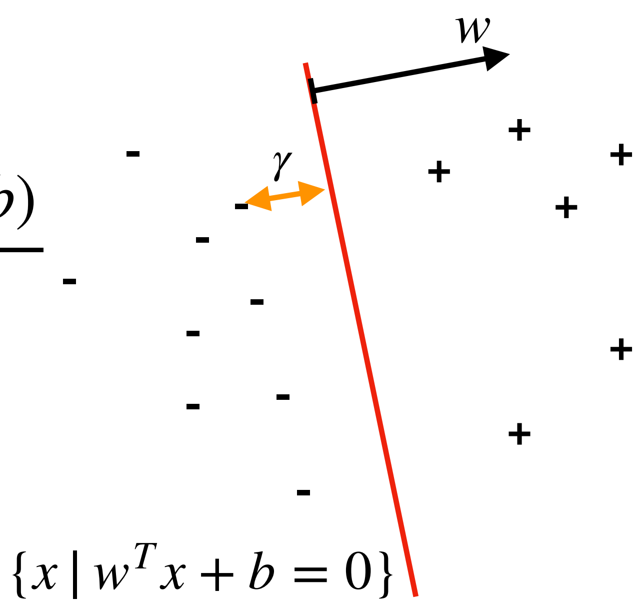



- Among all linear classifiers, we would like to find one that has the **maximum margin**

# Maximum margin classifiers

- The **margin** with respect to a set is defined as

$$\gamma = \min_{i \in \{1,\ldots,n\}} \gamma_i = \min_i y_i \frac{(w^T x_i + b)}{\|w\|_2}$$

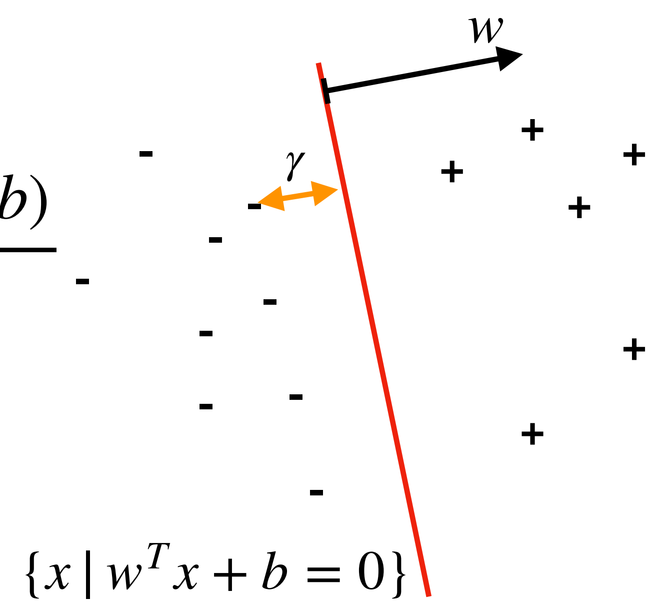



- Among all linear classifiers, we would like to find one that has the **maximum margin**

# Maximum margin classifiers

- The **margin** with respect to a set is defined as

$$\gamma = \min_{i \in \{1,\ldots,n\}} \gamma_i = \min_i y_i \frac{(w^T x_i + b)}{\|w\|_2}$$




- Among all linear classifiers, we would like to find one that has the **maximum margin**

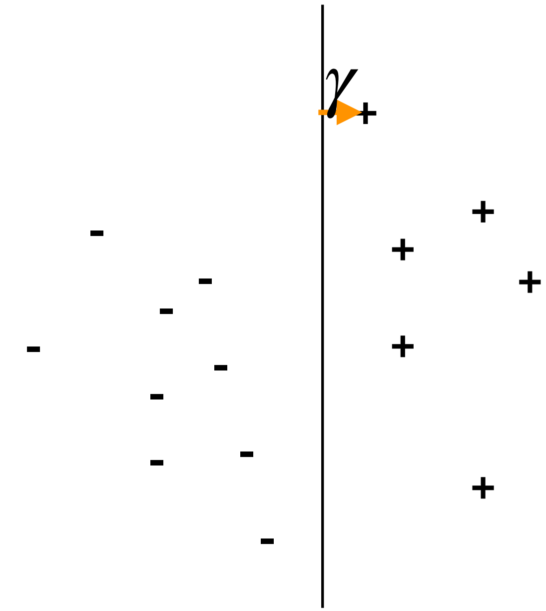
- We will derive an algorithm that finds the maximum margin classifier, by transforming a difficult to solve optimization into an efficient one

# Maximum margin classifier

(we transform the optimization into an efficient one)

**(maximize the margin)**

**(s.t. $\gamma$ is a lower bound on the margin)**



# Maximum margin classifier

**(we transform the optimization into an efficient one)**

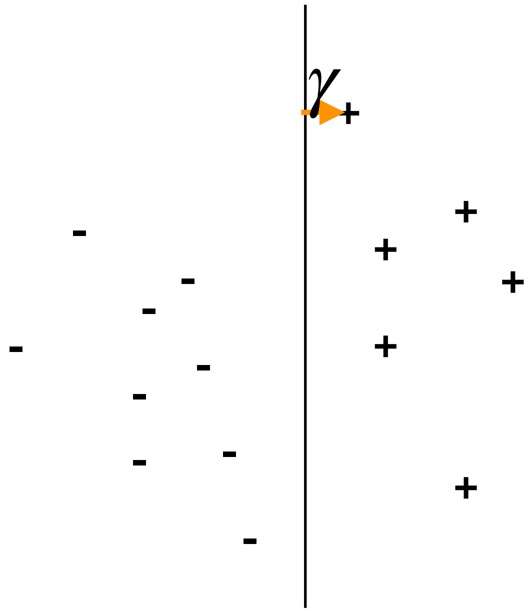
- We propose the following optimization problem:

$$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1, \ldots, n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**



# Maximum margin classifier

**(we transform the optimization into an efficient one)**

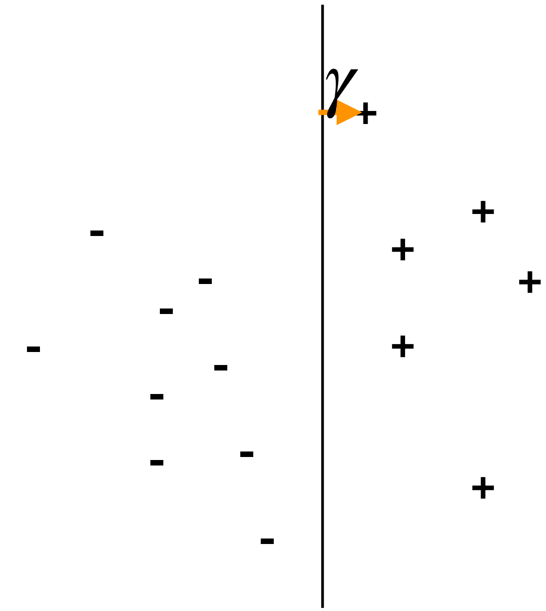
- We propose the following optimization problem:

$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**



# Maximum margin classifier

**(we transform the optimization into an efficient one)**

- We propose the following optimization problem:

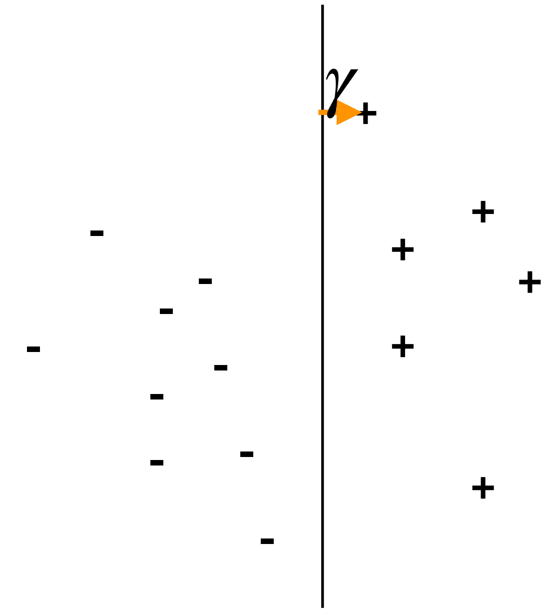
$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**

- If we fix $(w, b)$, the optimal solution of the optimization is the margin



# Maximum margin classifier

**(we transform the optimization into an efficient one)**

- We propose the following optimization problem:

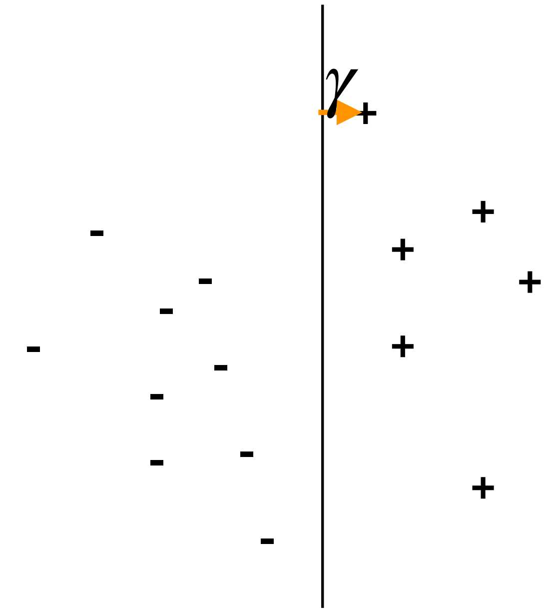
$$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1, \ldots, n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**

- If we fix $(w, b)$, the optimal solution of the optimization is the margin
- Together with $(w, b)$, this finds the classifier with the maximum margin



# Maximum margin classifier

(we transform the optimization into an efficient one)

- We propose the following optimization problem:

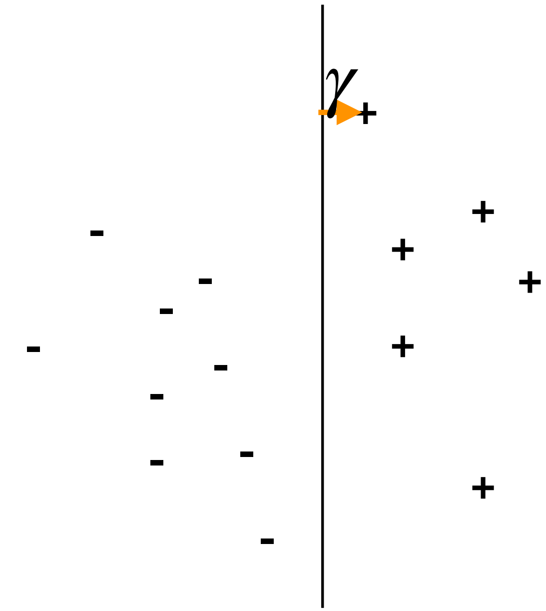
$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**

- If we fix $(w, b)$, the optimal solution of the optimization is the margin
- Together with $(w, b)$, this finds the classifier with the maximum margin
- Note that this problem is **scale invariant** in $(w, b)$, i.e. changing a $(w, b)$ to $(2w, 2b)$ does not change either the feasibility or the objective value, hence the following reparametrization is valid



# Maximum margin classifier

(we transform the optimization into an efficient one)

- We propose the following optimization problem:

$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

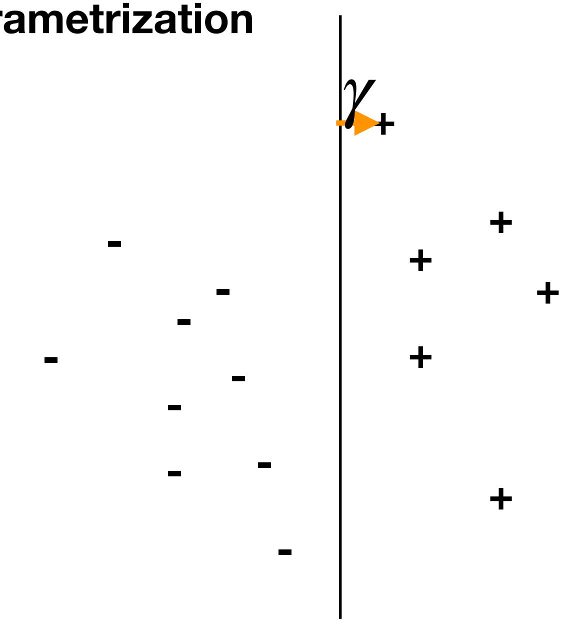
**(s.t. $\gamma$ is a lower bound on the margin)**

- If we fix $(w, b)$, the optimal solution of the optimization is the margin
- Together with $(w, b)$, this finds the classifier with the maximum margin
- Note that this problem is **scale invariant** in $(w, b)$, i.e. changing a $(w, b)$ to $(2w, 2b)$ does not change either the feasibility or the objective value, hence the following reparametrization is valid
- The above optimization looks difficult, so we transform it using **reparametrization**

$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\|w\|_2 = \frac{1}{\gamma}$$



# Maximum margin classifier

(we transform the optimization into an efficient one)

- We propose the following optimization problem:

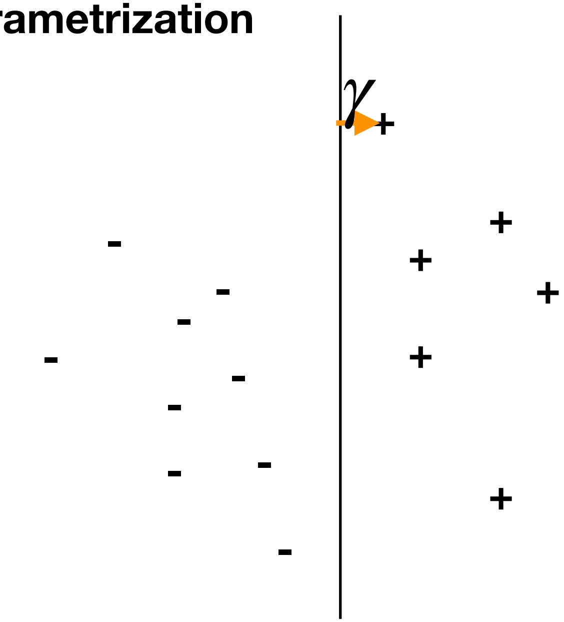
$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

**(maximize the margin)**

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

**(s.t. $\gamma$ is a lower bound on the margin)**

- If we fix $(w, b)$, the optimal solution of the optimization is the margin
- Together with $(w, b)$, this finds the classifier with the maximum margin
- Note that this problem is **scale invariant** in $(w, b)$, i.e. changing a $(w, b)$ to $(2w, 2b)$ does not change either the feasibility or the objective value, hence the following reparametrization is valid
- The above optimization looks difficult, so we transform it using **reparametrization**




$$\text{maximize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \gamma\in\mathbb{R}} \quad \gamma$$

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\|w\|_2 = \frac{1}{\gamma}$$

- Because of scale invariance, the optimal solution does not change, as the solutions to the original problem did not depend on $\|w\|_2$, and only depends on the direction of $w$

**(maximize the margin)**

**(now** $\frac{1}{\|w\|_2}$ **plays the role of a lower bound on the margin)**

- $$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$$

$$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \text{ for all } i \in \{1, \ldots, n\}$$

$$\|w\|_2 = \frac{1}{\gamma}$$

**(maximize the margin)**

**(now $\frac{1}{\|w\|_2}$ plays the role of a lower bound on the margin)**

- $\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$

  subject to $\dfrac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma$ for all $i \in \{1, \ldots, n\}$

  $$\|w\|_2 = \frac{1}{\gamma}$$

- The above optimization still looks difficult, but can be transformed into

  $\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \dfrac{1}{\|w\|_2}$ **(maximize the margin)**

  subject to $\dfrac{y_i(w^T x_i + b)}{\|w\|_2} \geq \dfrac{1}{\|w\|_2}$ for all $i \in \{1, \ldots, n\}$ **(now $\dfrac{1}{\|w\|_2}$ plays the role of a lower bound on the margin)**

  which simplifies to

  $\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$

  subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1, \ldots, n\}$

- $$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$$

  $$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \text{ for all } i \in \{1, \ldots, n\}$$

  $$\|w\|_2 = \frac{1}{\gamma}$$

- The above optimization still looks difficult, but can be transformed into

  $$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{\|w\|_2}$$

  **(maximize the margin)**

  $$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \frac{1}{\|w\|_2} \quad \text{for all } i \in \{1, \ldots, n\}$$

  **(now $\frac{1}{\|w\|_2}$ plays the role of a lower bound on the margin)**

  which simplifies to

  $$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

  $$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 \text{ for all } i \in \{1, \ldots, n\}$$

- This is a **quadratic program with linear constraints**, which can be easily solved

- $$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \gamma \in \mathbb{R}} \quad \gamma$$

  $$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \ \text{ for all } i \in \{1, \ldots, n\}$$

  $$\|w\|_2 = \frac{1}{\gamma}$$

- The above optimization still looks difficult, but can be transformed into

  $$\text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{\|w\|_2}$$

  **(maximize the margin)**

  $$\text{subject to} \quad \frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \frac{1}{\|w\|_2} \quad \text{for all } i \in \{1, \ldots, n\}$$

  **(now $\frac{1}{\|w\|_2}$ plays the role of a lower bound on the margin)**

  which simplifies to

  $$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

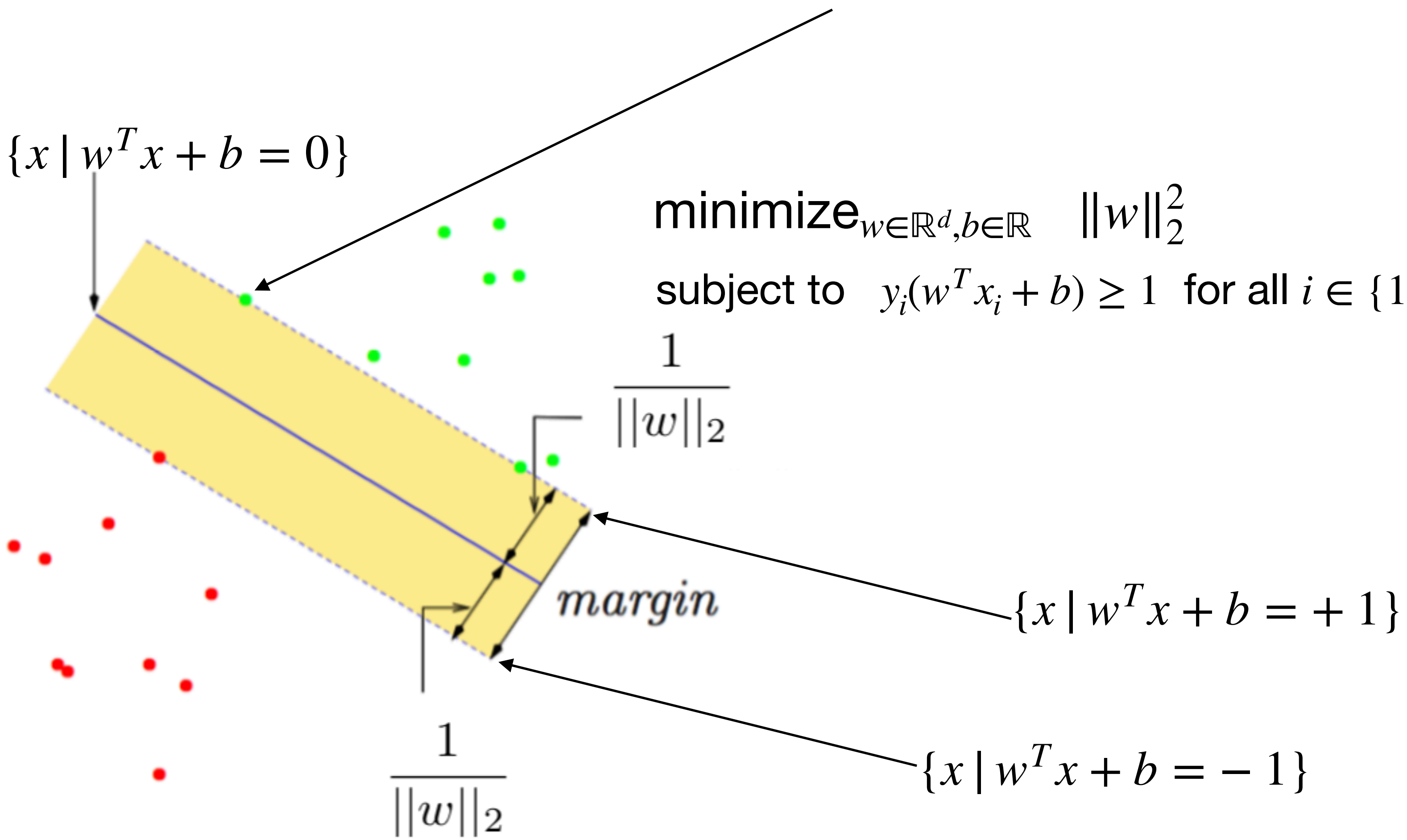
  $$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 \ \text{ for all } i \in \{1, \ldots, n\}$$

- This is a **quadratic program with linear constraints**, which can be easily solved
- Once the optimal solution is found, the margin of that classifier $(w, b)$ is $\frac{1}{\|w\|_2}$

# What if the data is not separable?

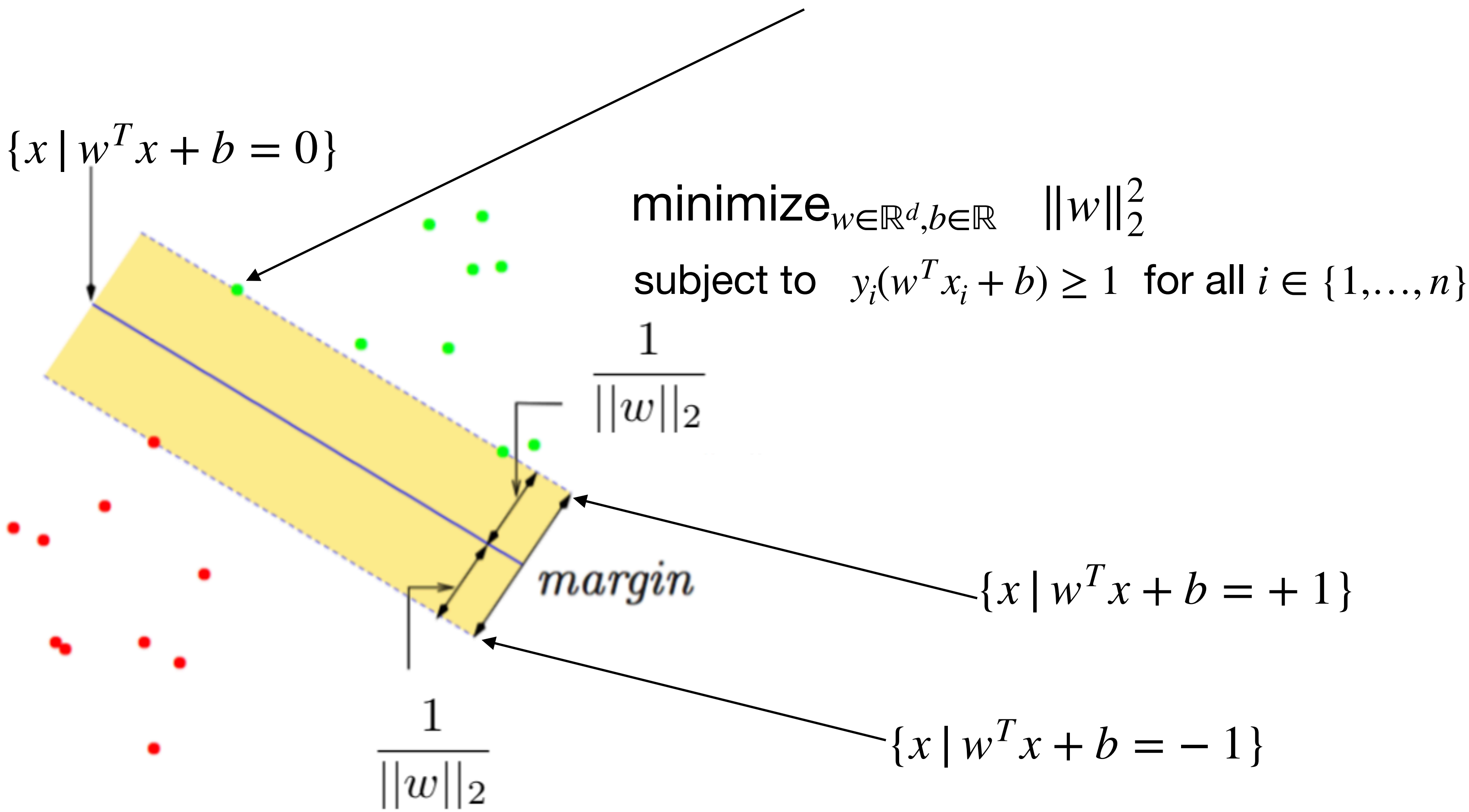
$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 \quad \text{for all } i \in \{1, \ldots, n\}$$



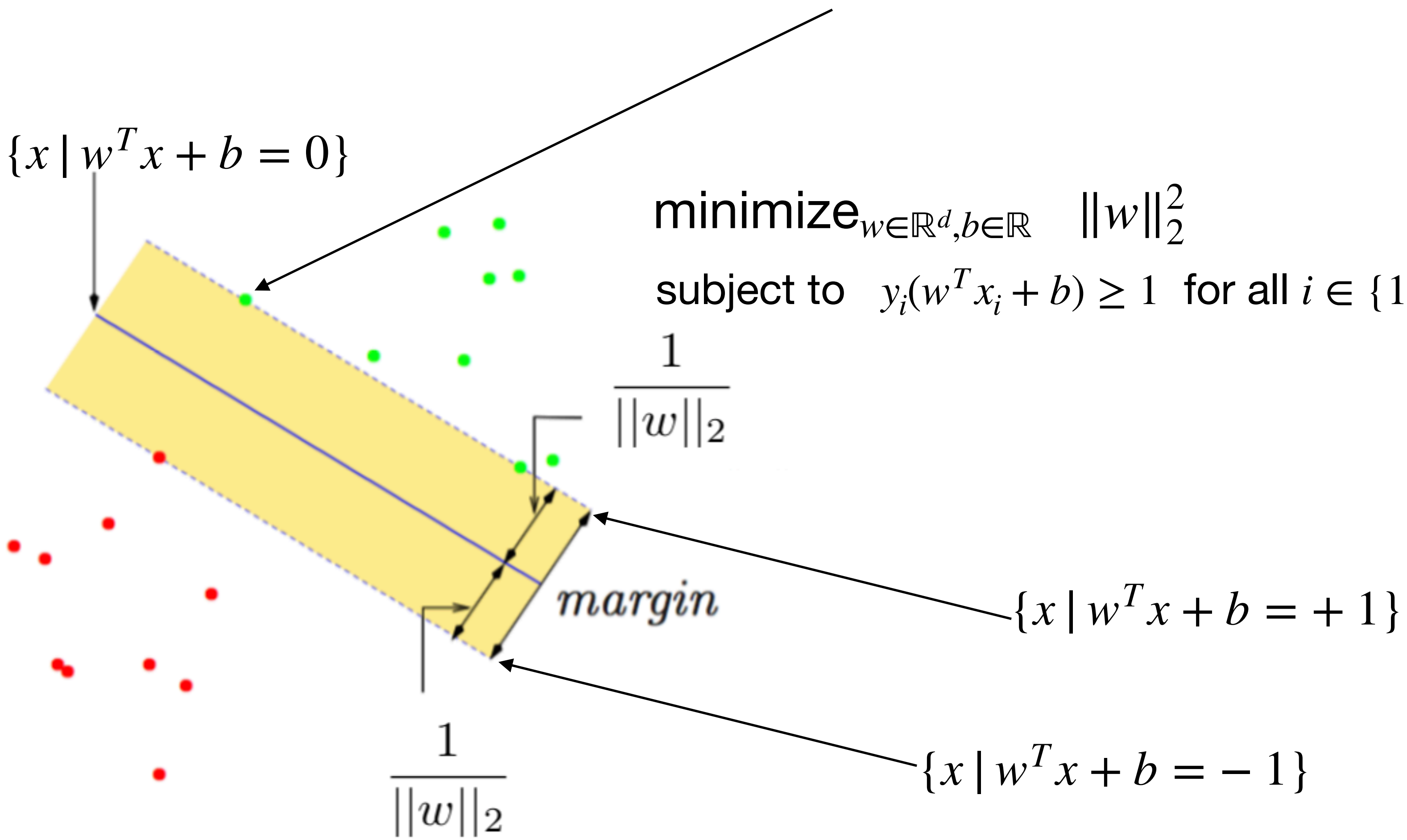
# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin



# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution




$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

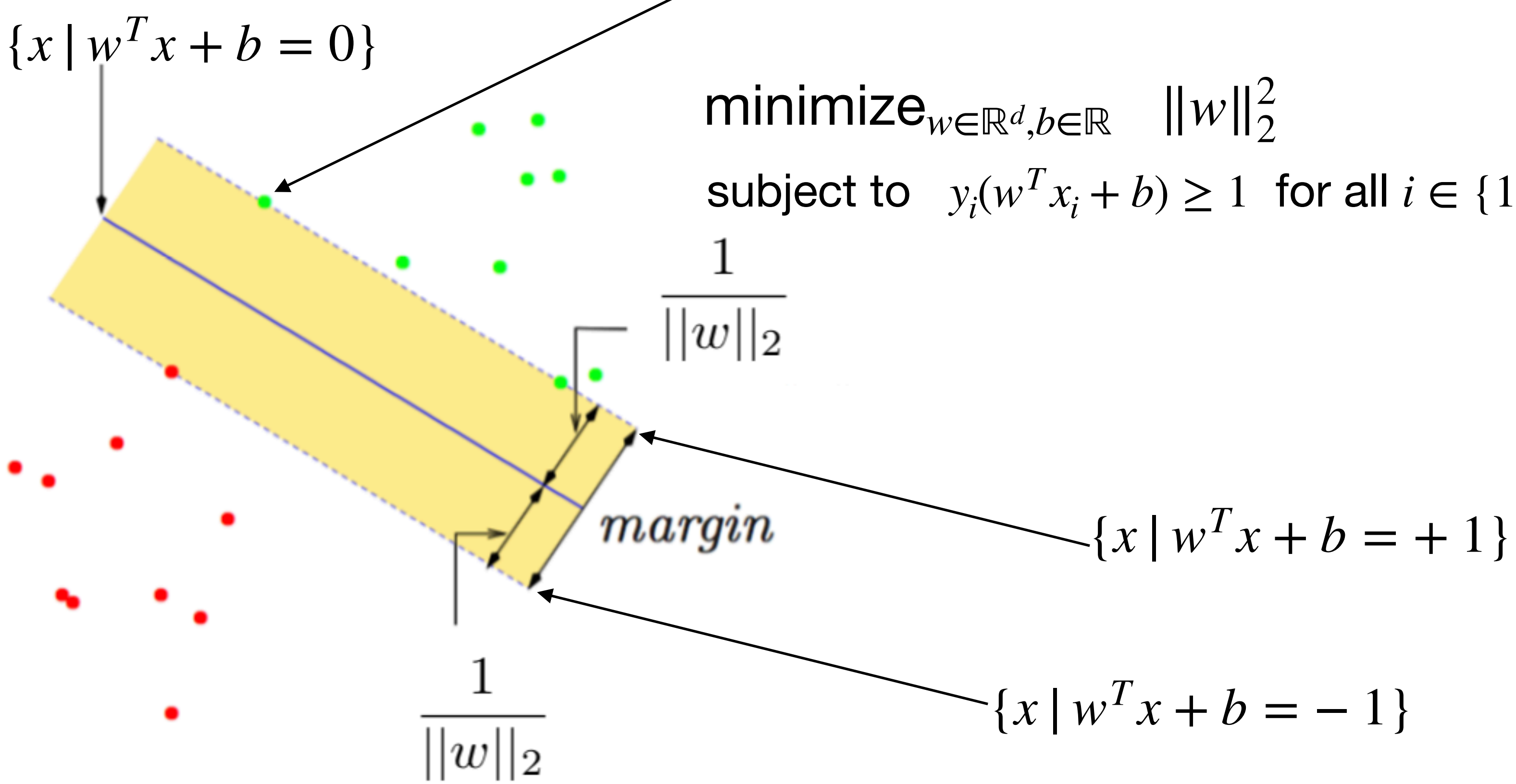
subject to $\; y_i(w^T x_i + b) \geq 1 \;$ for all $i \in \{1,\ldots,n\}$

# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \|w\|_2^2$$

subject to $\; y_i(w^T x_i + b) \geq 1 \;$ for all $i \in \{1,\ldots,n\}$

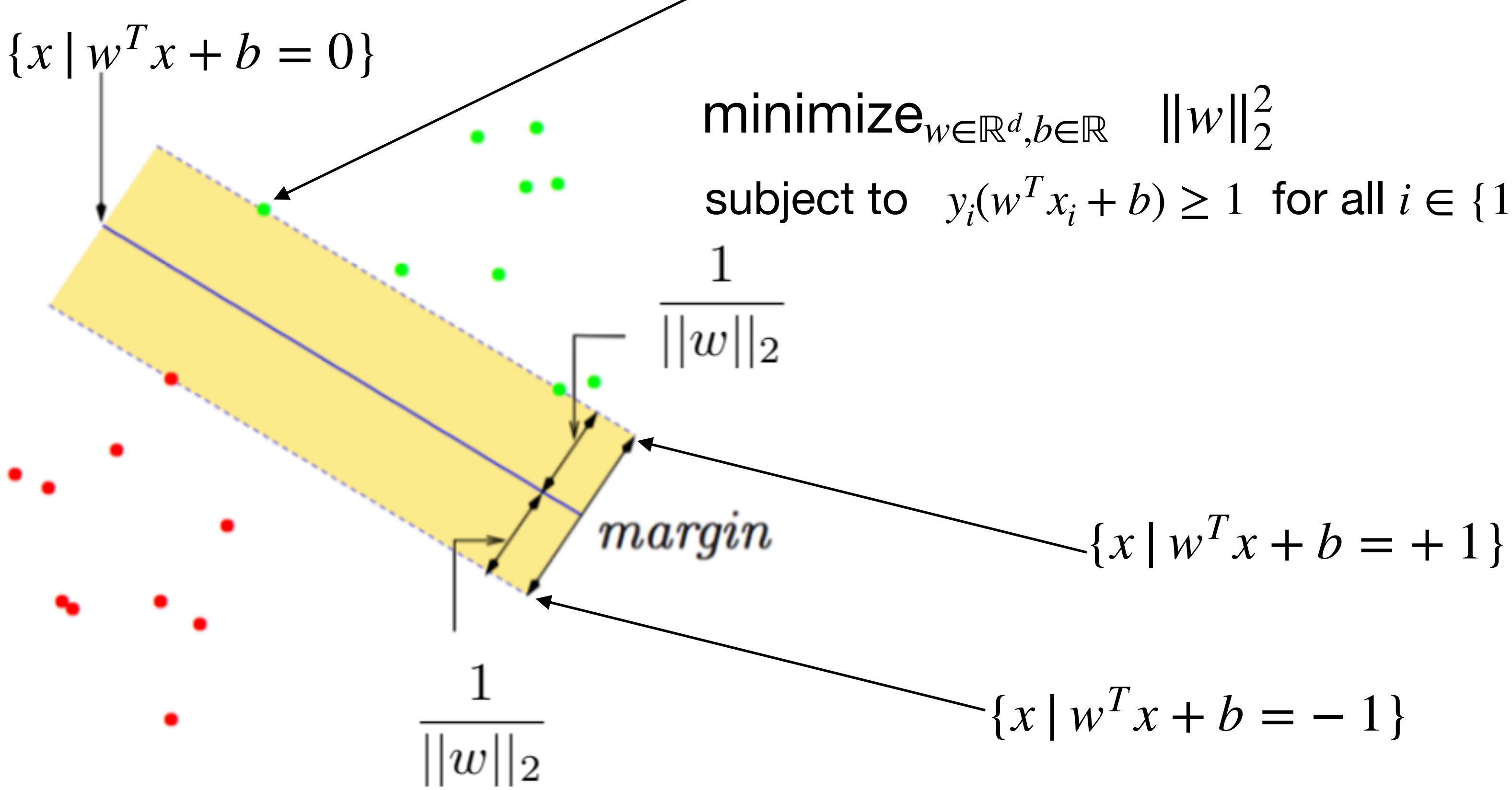


# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1, \ldots, n\}$

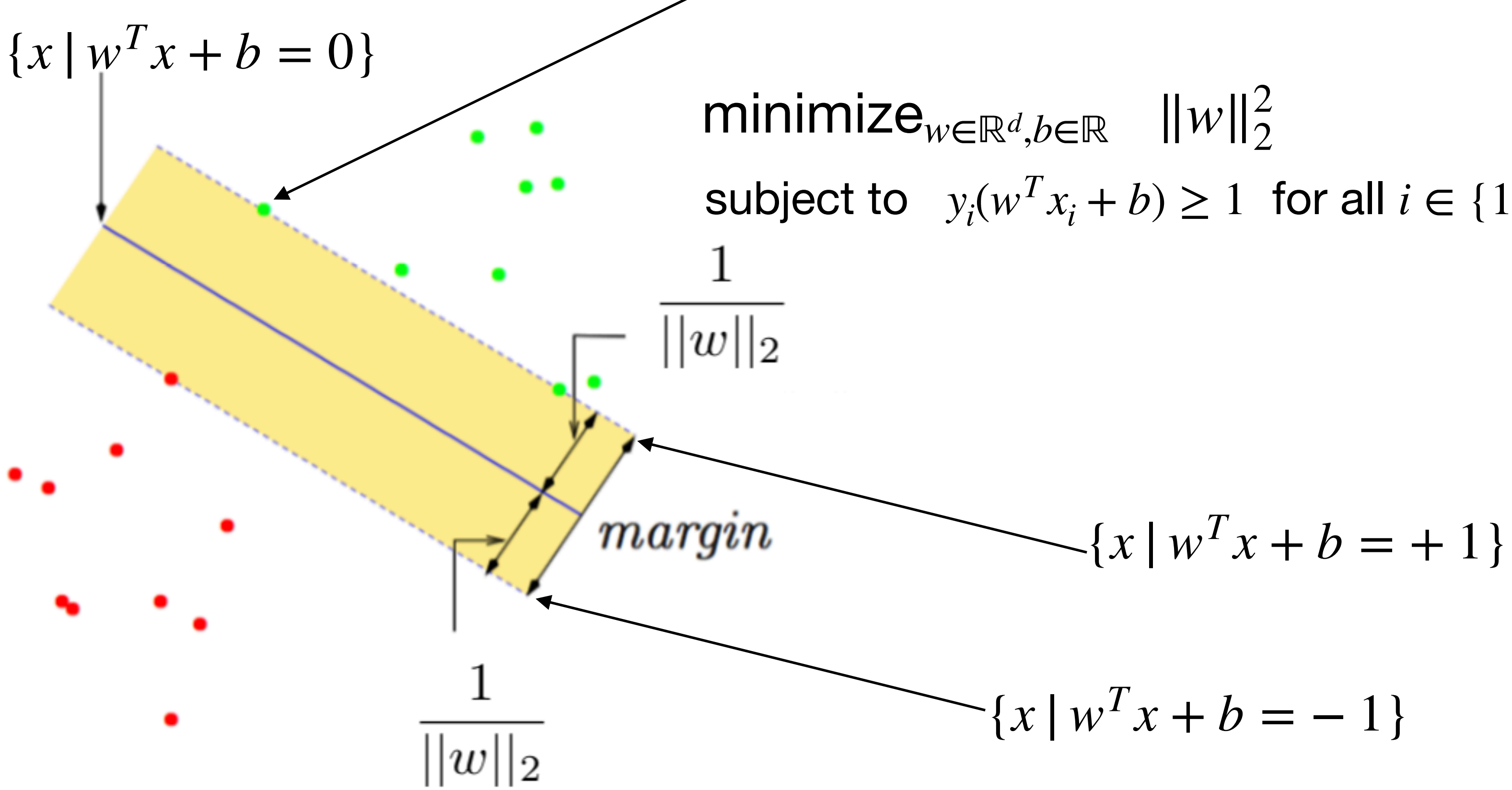


# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1, \ldots, n\}$

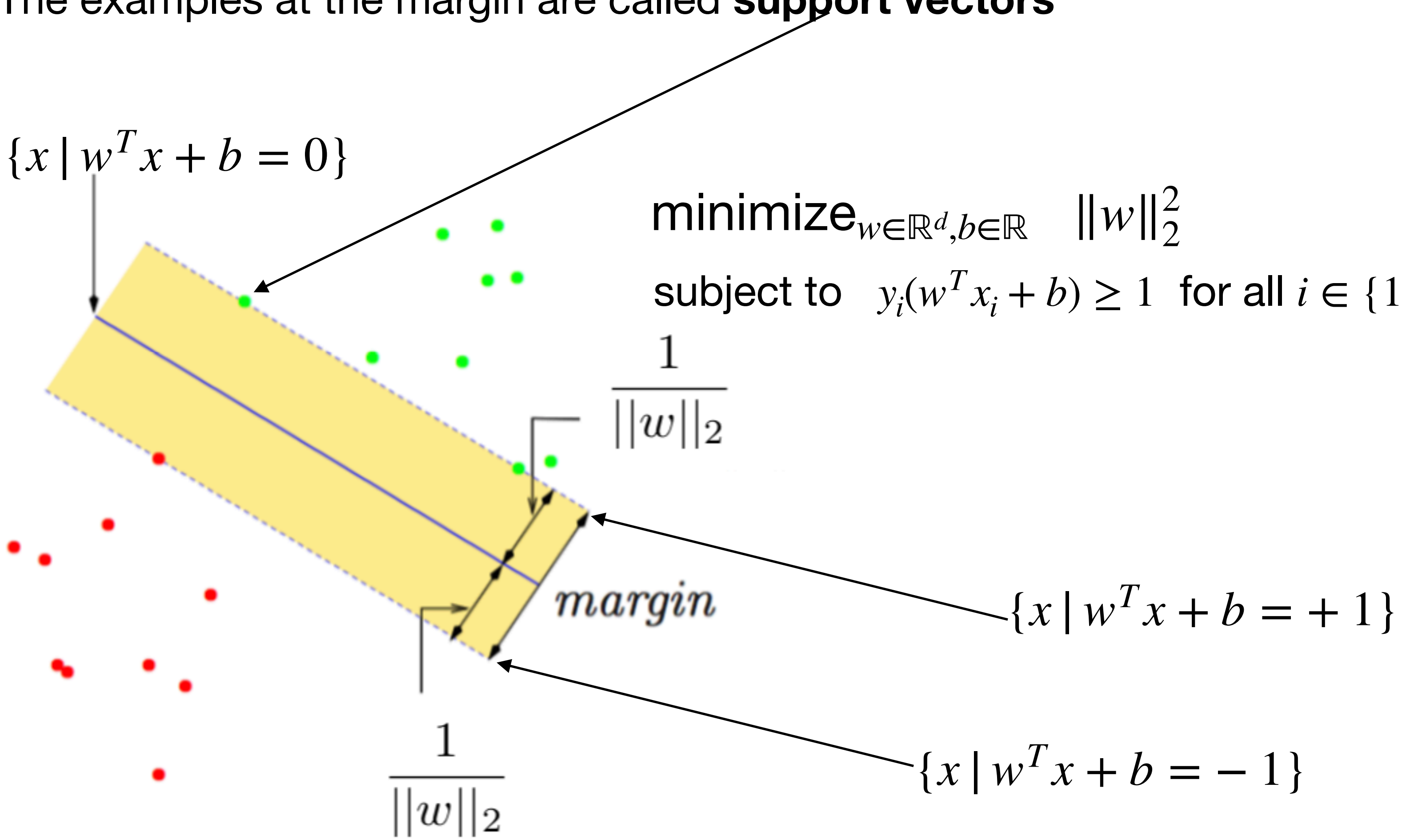


# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

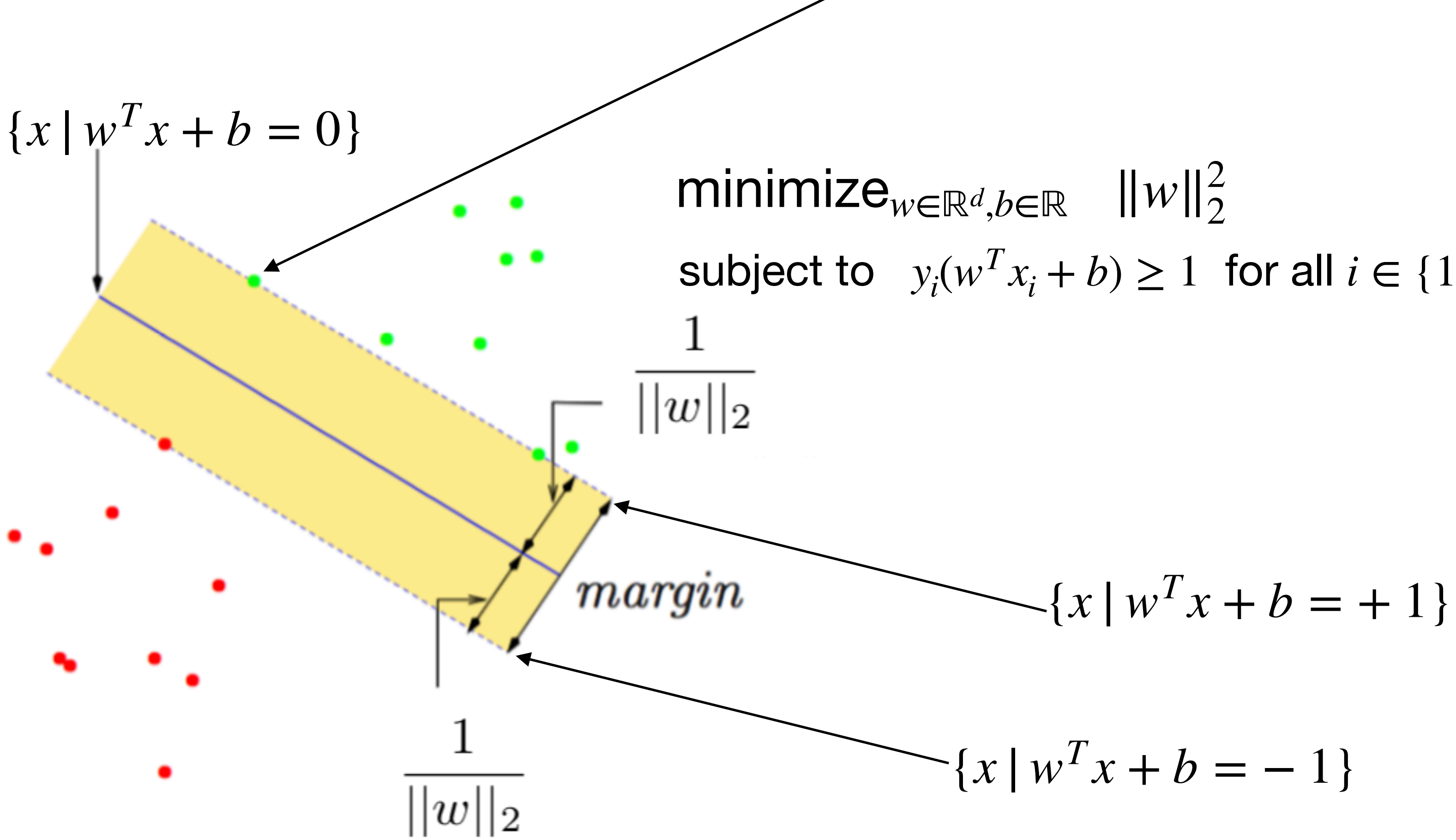
$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

subject to $\quad y_i(w^T x_i + b) \geq 1$ for all $i \in \{1,\ldots,n\}$



# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

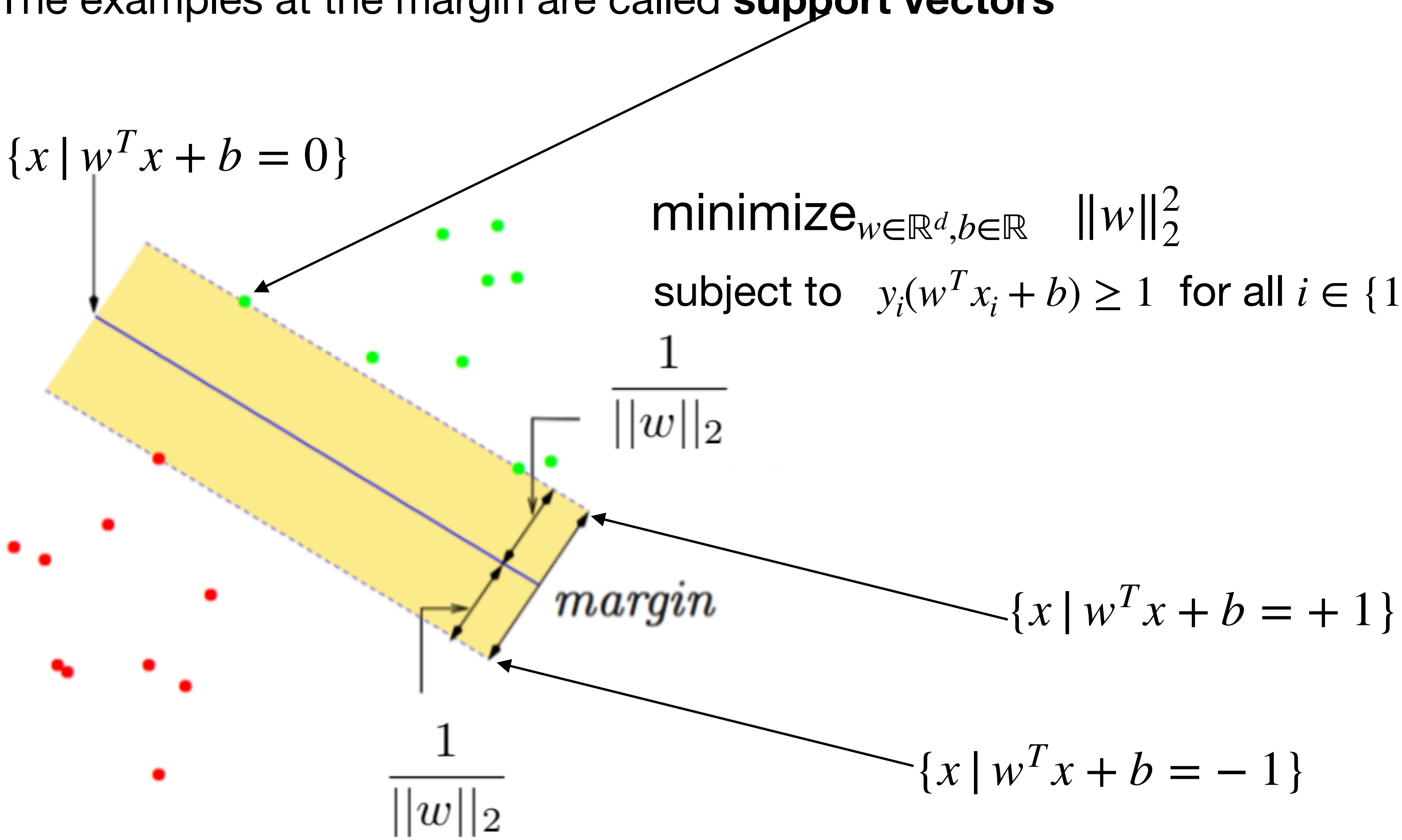



$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

subject to $\quad y_i(w^T x_i + b) \geq 1 \quad$ for all $i \in \{1,\ldots,n\}$

# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**




$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

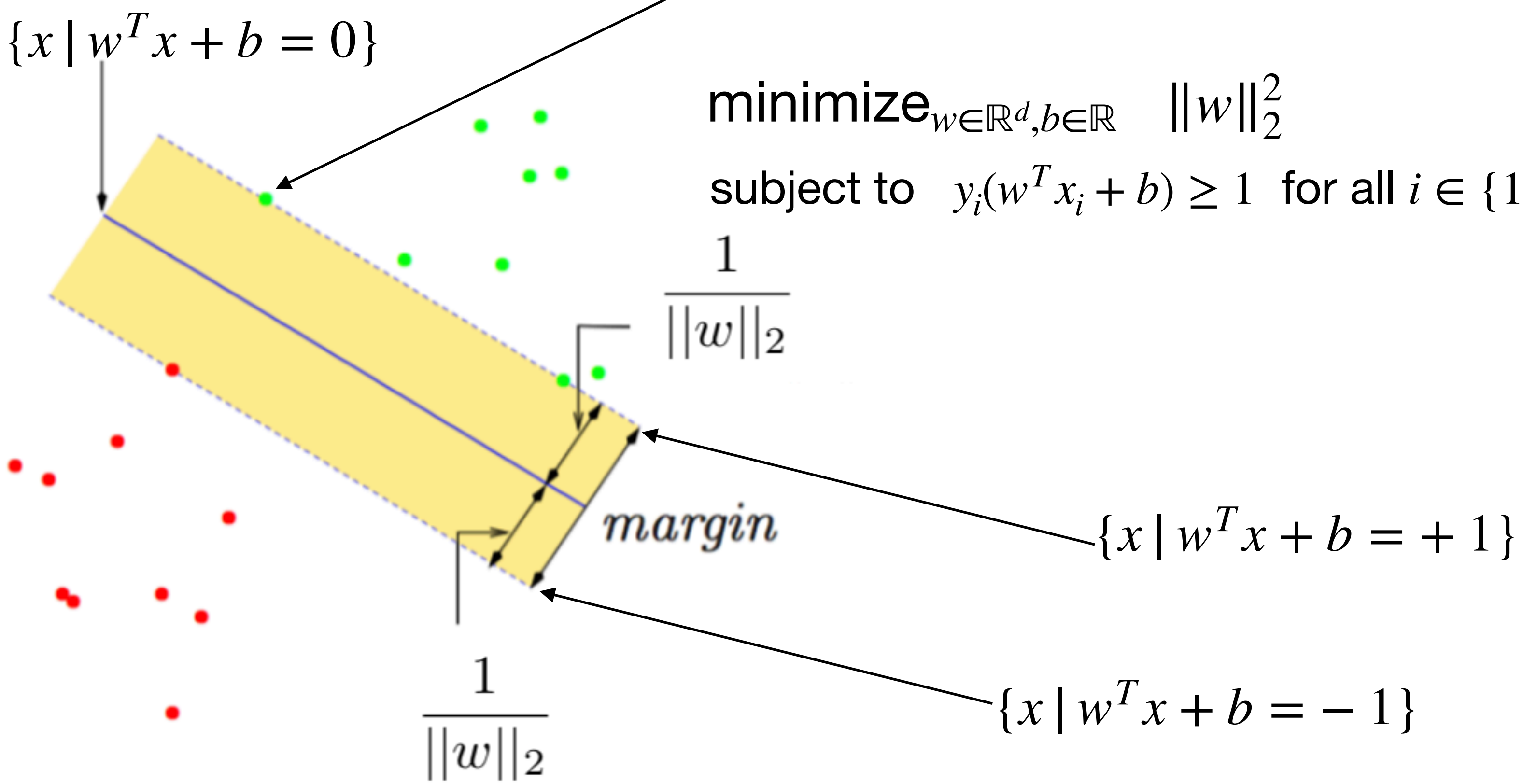
subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1,\ldots,n\}$

# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

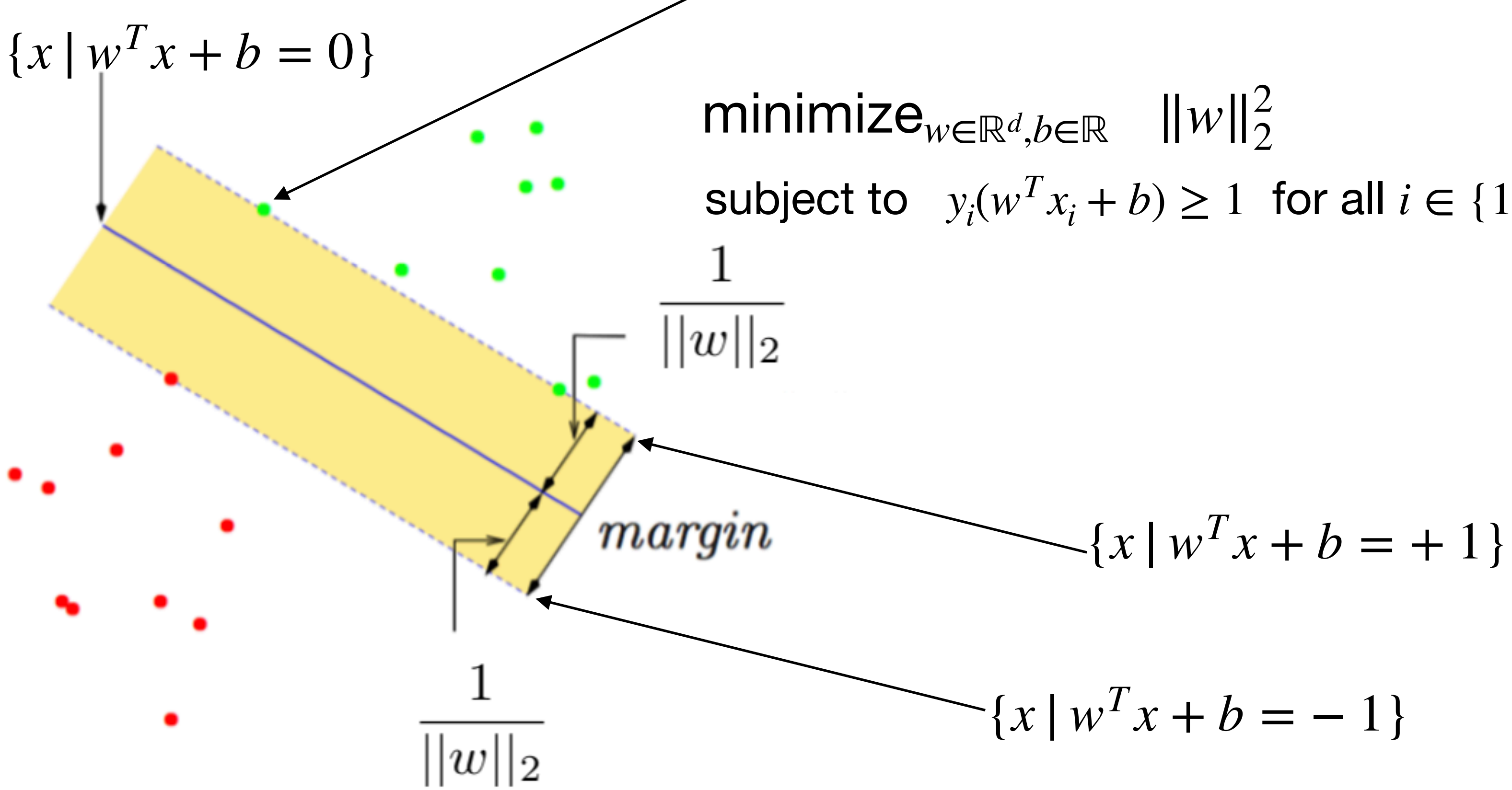
$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

subject to $\; y_i(w^T x_i + b) \geq 1 \;$ for all $i \in \{1, \ldots, n\}$



# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**




$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \|w\|_2^2$$

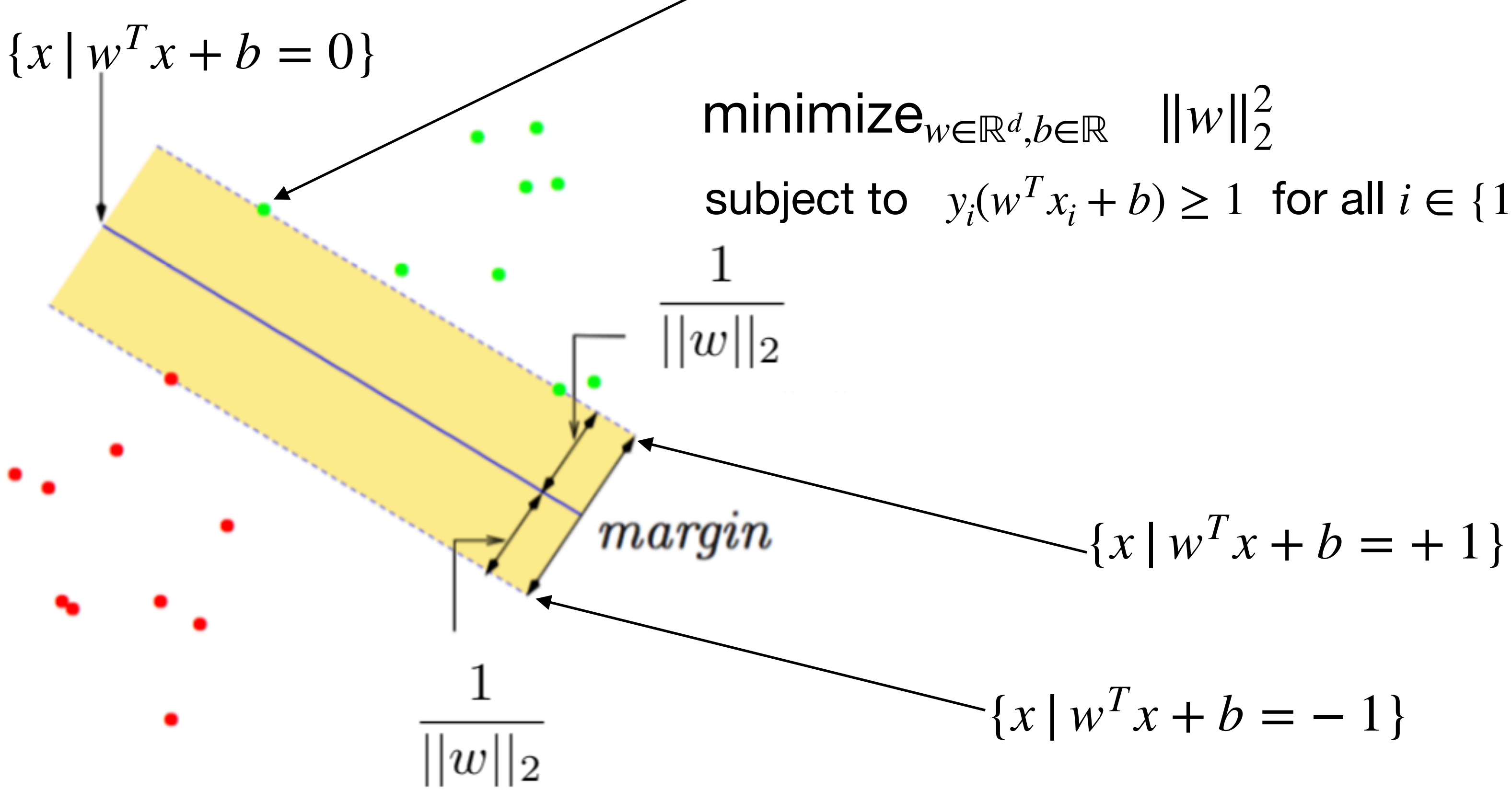
subject to $\; y_i(w^T x_i + b) \geq 1 \;$ for all $i \in \{1,\ldots,n\}$

# What if the data is not separable?

- We cheated a little in the sense that the reparametrization of $\|w\|_2 = \dfrac{1}{\gamma}$ is possible only if the the margins are positive, i.e. the data is linearly separable with a positive margin
- Otherwise, there is no feasible solution
- The examples at the margin are called **support vectors**

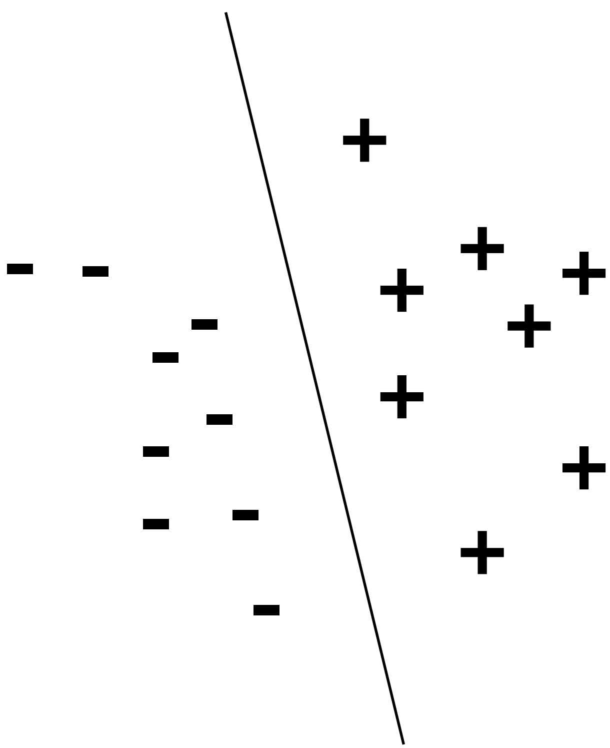
minimize$_{w \in \mathbb{R}^d, b \in \mathbb{R}}$ $\|w\|_2^2$

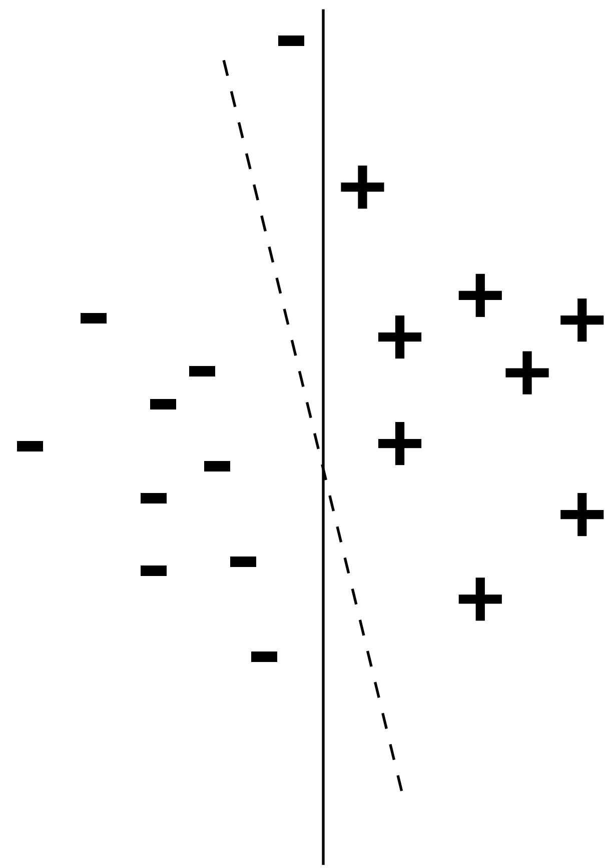
subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1, \ldots, n\}$



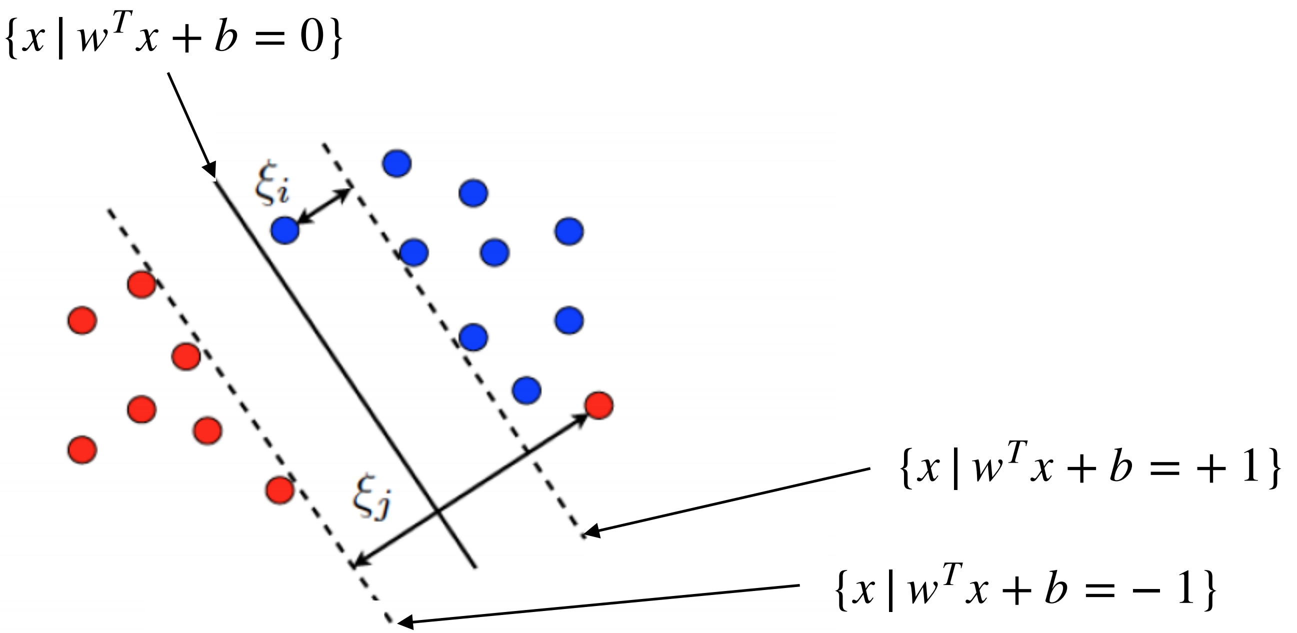
# Two issues

- it does not generalize to non-separable datasets
- max-margin formulation we proposed is sensitive to outliers

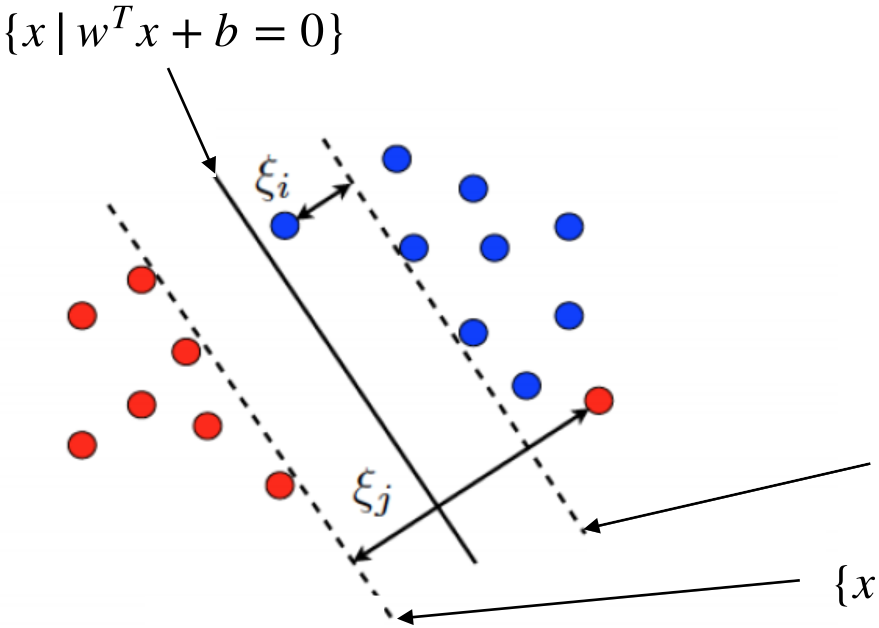




# What if the data is not separable?



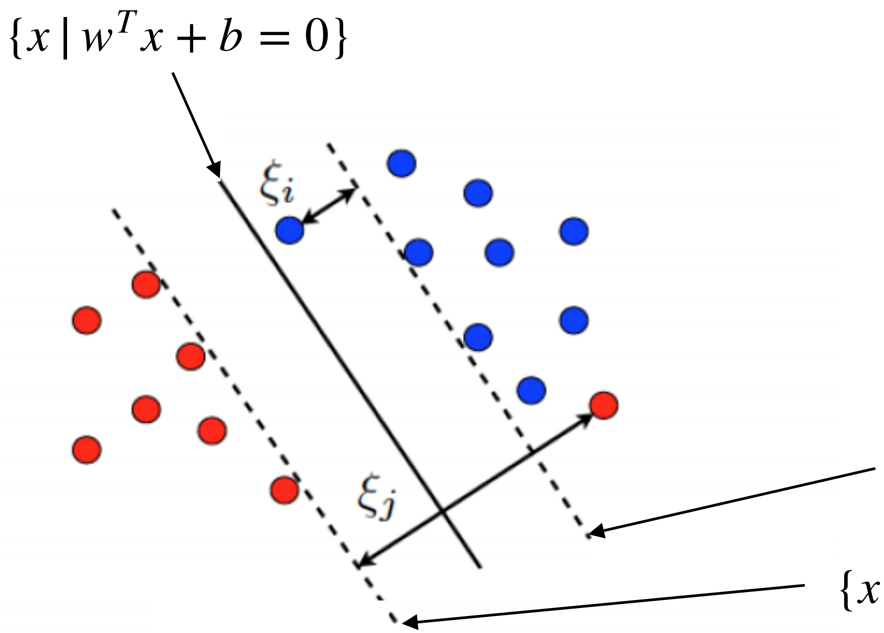
# What if the data is not separable?




- We introduce **slack** so that some points can violate the margin condition

$$y_i(w^T x_i + b) \;\geq\; 1 - \xi_i$$

# What if the data is not separable?



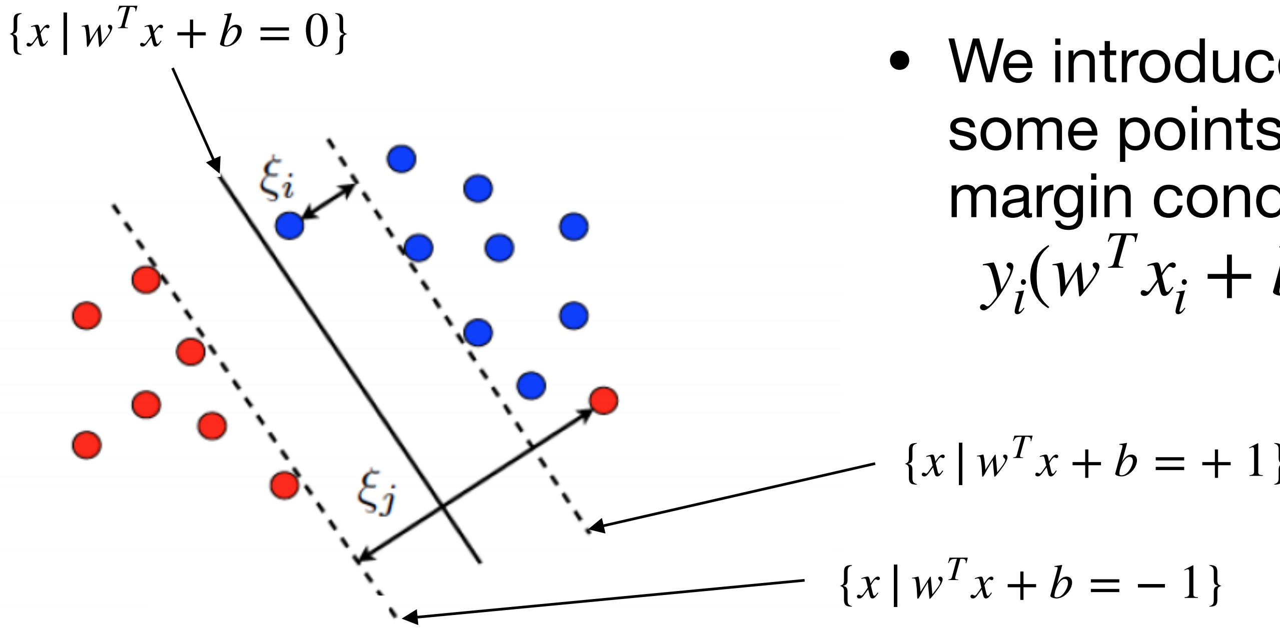

- We introduce **slack** so that some points can violate the margin condition
  $$y_i(w^T x_i + b) \;\geq\; 1 - \xi_i$$

- This gives a new optimization problem with some positive constant $c \in \mathbb{R}$

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

# What if the data is not separable?




- We introduce **slack** so that some points can violate the margin condition

$$y_i(w^T x_i + b) \;\geq\; 1 - \xi_i$$

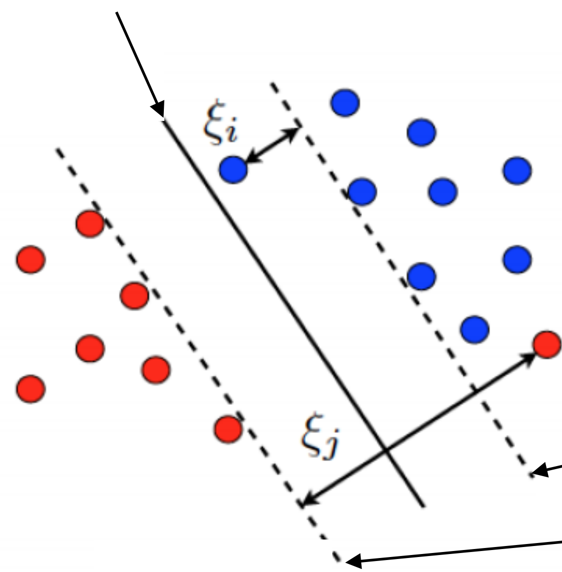
- This gives a new optimization problem with some positive constant $c \in \mathbb{R}$

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

$$\xi_i \;\geq 0 \quad \text{for all } i \in \{1, \ldots, n\}$$

# What if the data is not separable?




- We introduce **slack** so that some points can violate the margin condition
$$y_i(w^T x_i + b) \;\geq\; 1 - \xi_i$$

- This gives a new optimization problem with some positive constant $c \in \mathbb{R}$

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

the (re-scaled) margin (for each sample) is allowed to be less than one,
but you pay $c\xi_i$ in the cost, and $c$ balances the two goals:
maximizing the margin for most examples vs. having small number of violations

# Support Vector Machine

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \qquad \text{for all } i \in \{1, \ldots, n\}$$

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1, \ldots, n\}$$

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1, \ldots, n\}$$

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^Tx_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^Tx_i + b) \geq 1$, or

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1, \ldots, n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1, \ldots, n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1, \ldots, n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

- So one can write

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

- So one can write
    - $\xi_i = \max\{0, 1 - y_i(w^T x_i + b)\}$, which gives

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

- So one can write
  - $\xi_i = \max\{0, 1 - y_i(w^T x_i + b)\}$, which gives

# Support Vector Machine

- For the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, $\xi_i$'s satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

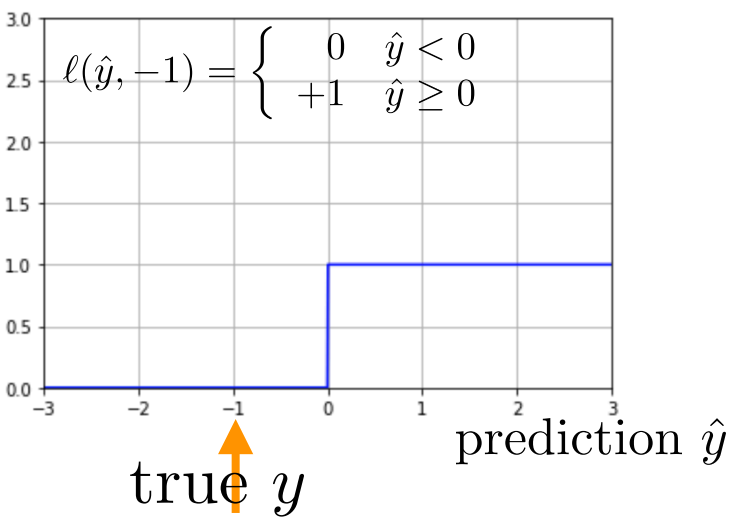
- So one can write
    - $\xi_i = \max\{0, 1 - y_i(w^T x_i + b)\}$, which gives

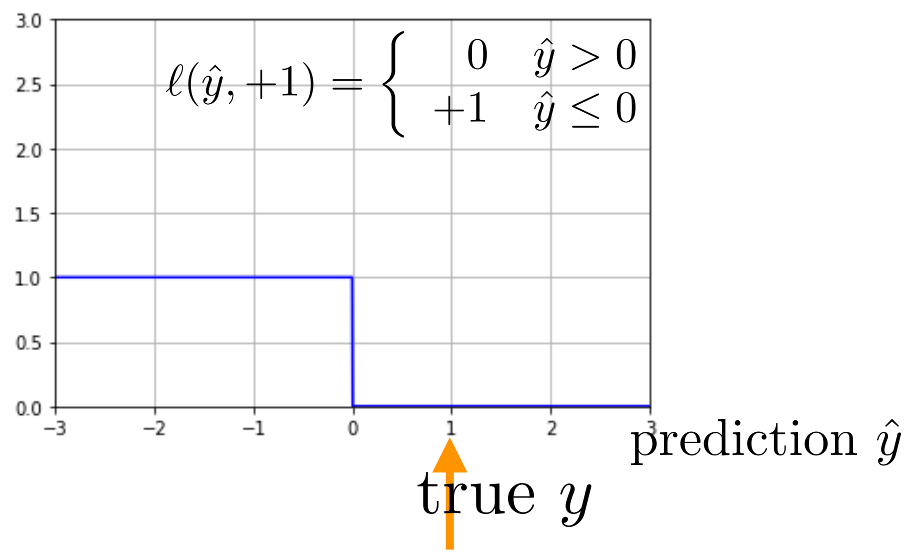
$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n}\max\{0, 1 - y_i(w^T x_i + b)\}$$

# Recall: we were looking for a loss function

- We want a loss function that
  - approximates (captures the flavor of) the 0-1 loss
  - can be easily optimized (e.g. convex and/or non-zero derivatives)
- More formally, we want a **loss function**
  - with $\ell(\hat{y}, -1)$ small when $\hat{y} < 0$ and larger when $\hat{y} > 0$
  - with $\ell(\hat{y}, 1)$ small when $\hat{y} > 0$ and larger when $\hat{y} < 0$
  - which has other nice characteristics, e.g., differentiable or convex
- We now have a new loss function from the SVM optimization problem:

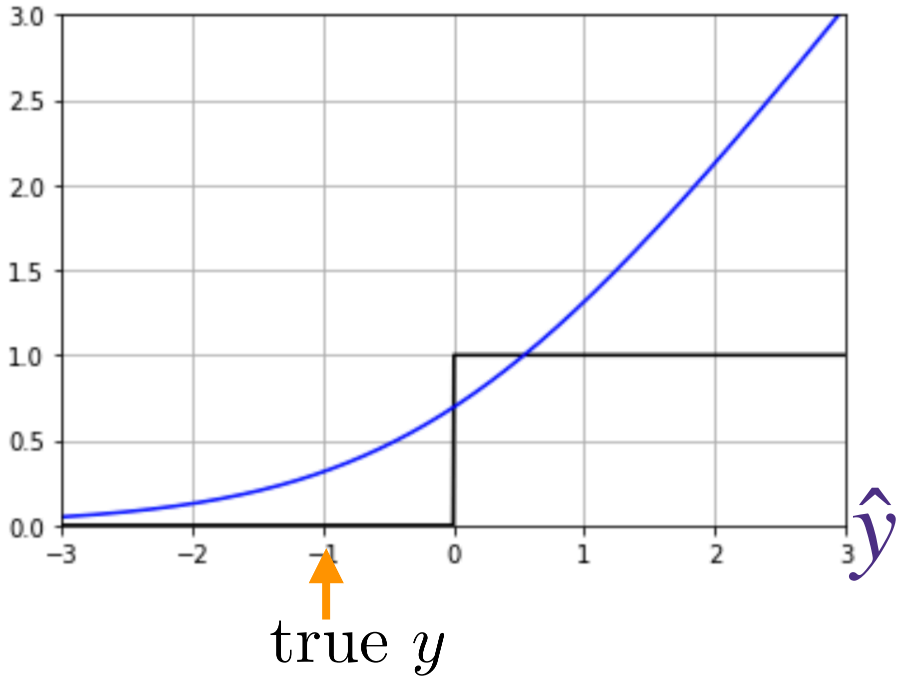
$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

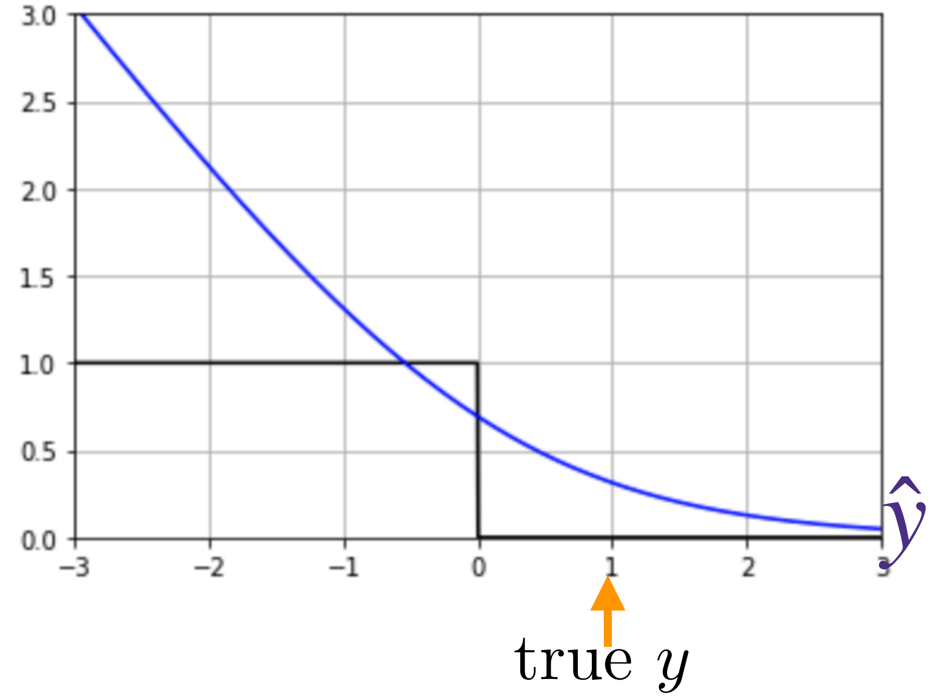


# Logistic loss $\ell(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$

$$\ell(\hat{y}, -1) = \log(1 + e^{\hat{y}}) \qquad \ell(\hat{y}, +1) = \log(1 + e^{-\hat{y}})$$




- Differentiable and convex in $\hat{y}$
- Approximation of 0-1 loss
- Most popular choice of a loss function for classification problems

# Sub-gradient descent for SVM

# Sub-gradient descent for SVM

- SVM is the solution of

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

# Sub-gradient descent for SVM

- SVM is the solution of

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

- As it is non-differentiable, we solve it using **sub-gradient descent**

# Sub-gradient descent for SVM

- SVM is the solution of

$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

- As it is non-differentiable, we solve it using **sub-gradient descent**
- which is exactly the same as gradient descent, except when we are at a non-differentiable point, we take one of the sub-gradients instead of the gradient (recall sub-gradient is a set)

# Sub-gradient descent for SVM

- SVM is the solution of

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

- As it is non-differentiable, we solve it using **sub-gradient descent**
- which is exactly the same as gradient descent, except when we are at a non-differentiable point, we take one of the sub-gradients instead of the gradient (recall sub-gradient is a set)
- this means that we can take (a generic form derived from previous page)

$$\partial_w \ell(w^T x_i + b, y_i) \;=\; \mathbf{I}\{y_i(w^T x_i + b) \le 1\}(-y_i x_i)$$

and apply

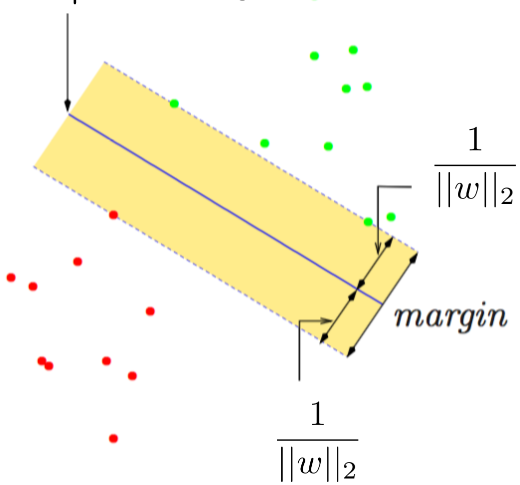
$$w^{(t+1)} \;\leftarrow\; w^{(t)} - \eta \left( \sum_{i=1}^{n} \mathbf{I}\{y_i((w^{(t)})^T x_i + b^{(t)}) \le 1\}(-y_i x_i) \;+\; \frac{2}{c} w^{(t)} \right)$$

$$b^{(t+1)} \;\leftarrow\; b^{(t)} - \eta \sum_{i=1}^{n} \mathbf{I}\{y_i((w^{(t)})^T x_i + b^{(t)}) \le 1)\}(-y_i)$$

# Kernels

# What if the data is not linearly separable?

- $x^T w + b = 0$




Some points do not satisfy margin constraint:

$$\min_{w,b} ||w||_2^2$$

$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

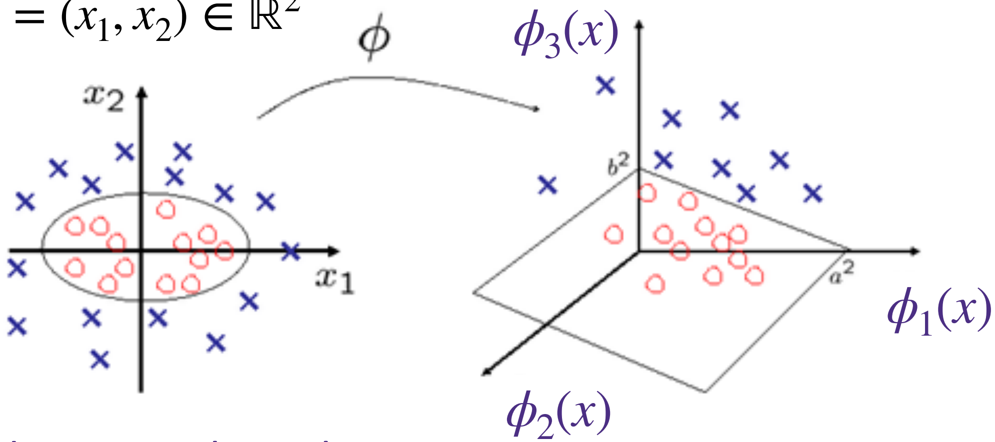
Two options:

1. Introduce slack to this optimization problem (Support Vector Machine)
2. **Lift to higher dimensional space (Kernels)**

# What if the data is not linearly separable?

- Use features, for example,

$x = (x_1, x_2) \in \mathbb{R}^2$

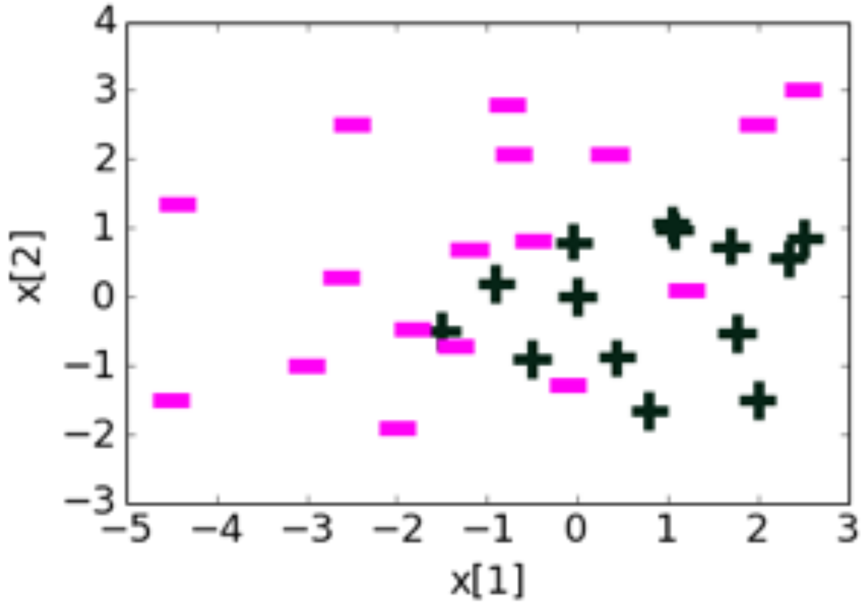



This data is not linearly separable

Can you suggest some features $\phi_1(x_1, x_2), \phi_2(x_1, x_2), \phi_3(x_1, x_2)$ such that this data is linearly separable in this 3-dimensional space?

- Generally, in high dimensional feature space, it is easier to linearly separate different classes
- However, it is hard to know which feature map will work for given data
- So the rule of thumb is to use high-dimensional features and hope that the algorithm will automatically pick the right set of features

# Example: adding more polynomial features



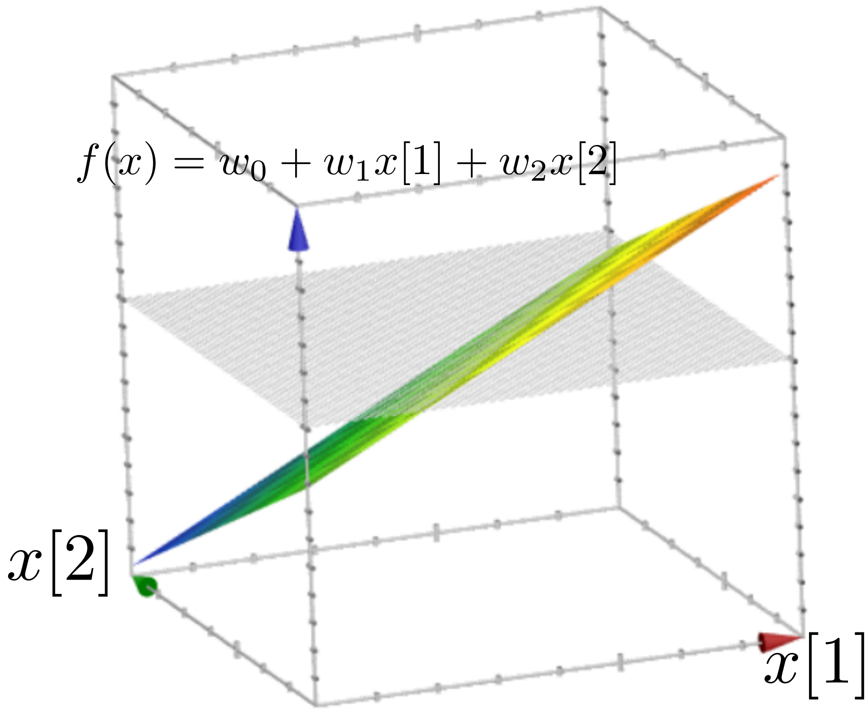

Polynomial

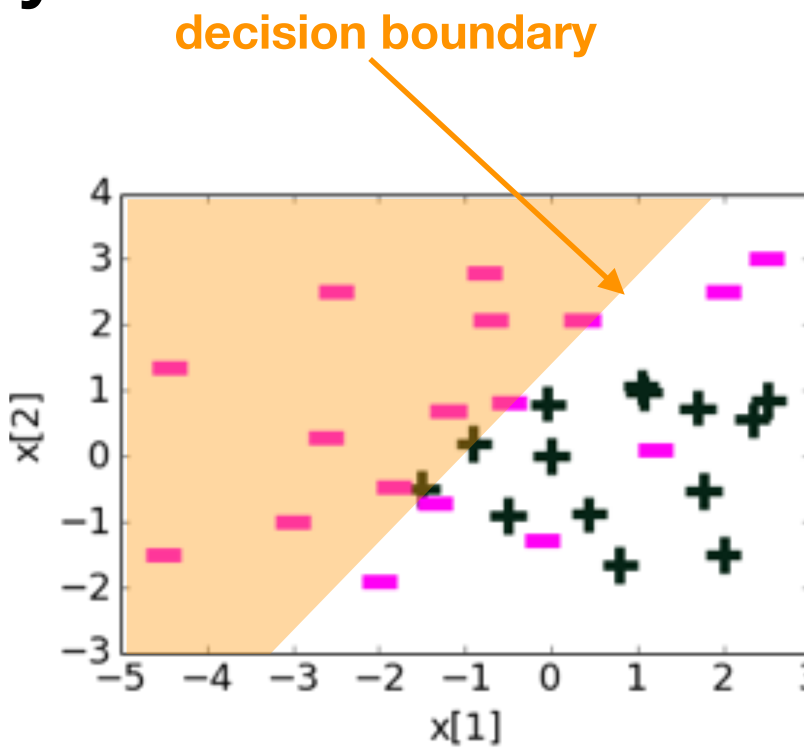
features

$$\begin{bmatrix} h_0(x) = 1 \\ h_1(x) = x[1] \\ h_2(x) = x[2] \\ h_3(x) = x[1]^2 \\ h_4(x) = x[2]^2 \\ \vdots \end{bmatrix}$$

- data: ***x*** in 2-dimensions, ***y*** in {+1,-1}
- features: polynomials
- model: linear on polynomial features
- $$f(x) \;=\; w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

# Learned decision boundary

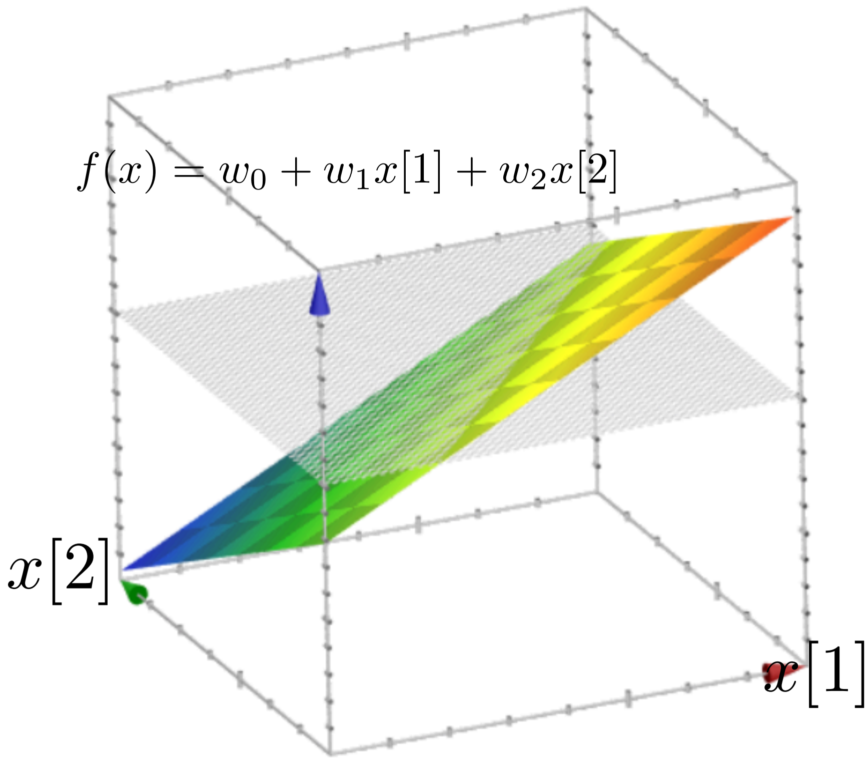


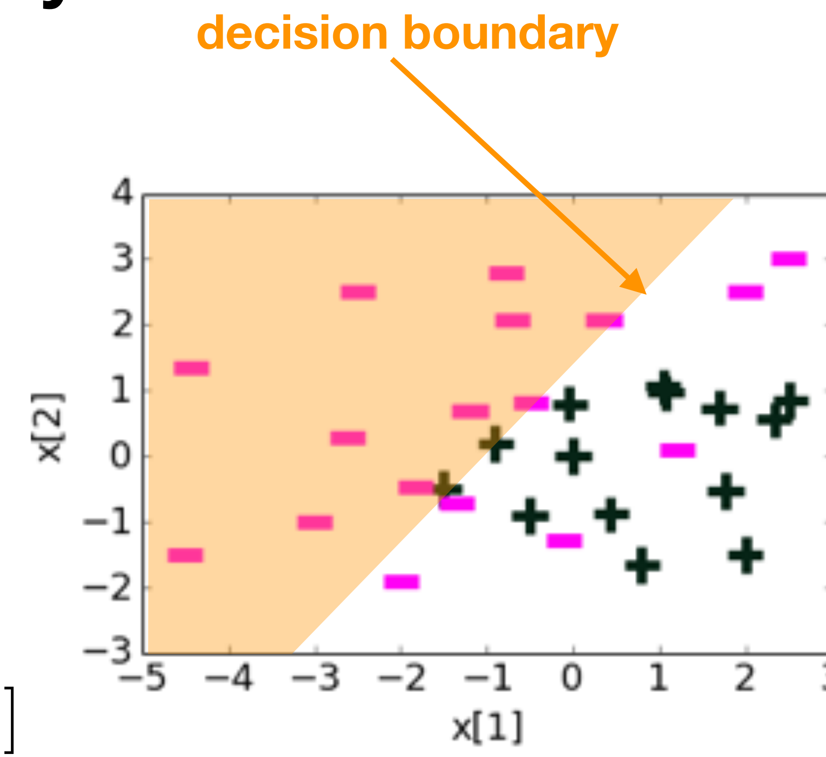




| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 0.23 |
| $h_1(x)$ | x[1] | 1.12 |
| $h_2(x)$ | x[2] | -1.07 |

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

# Learned decision boundary

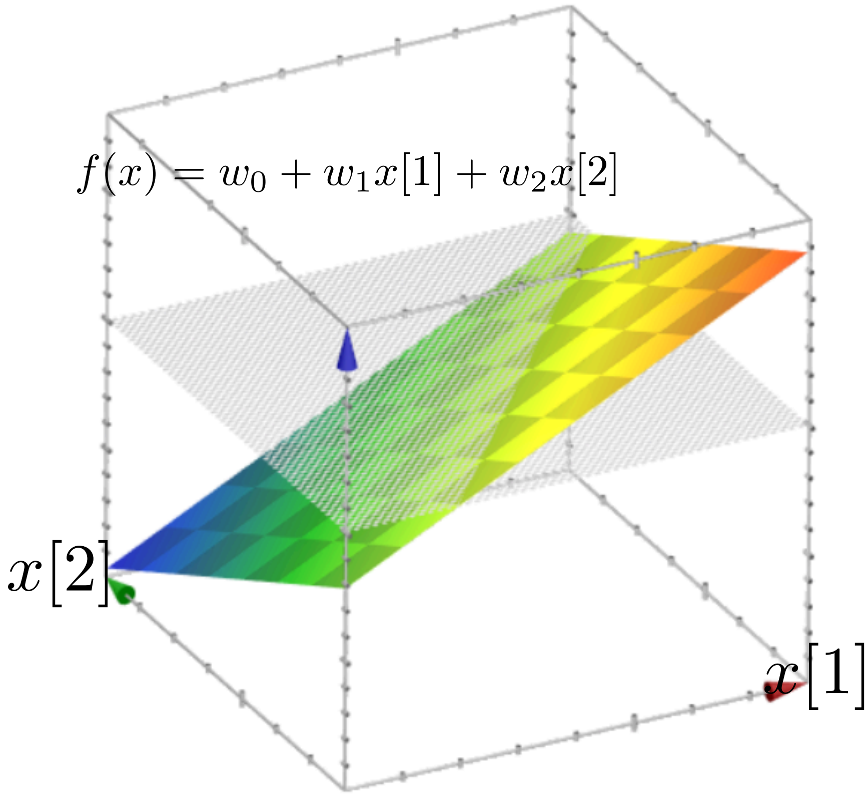


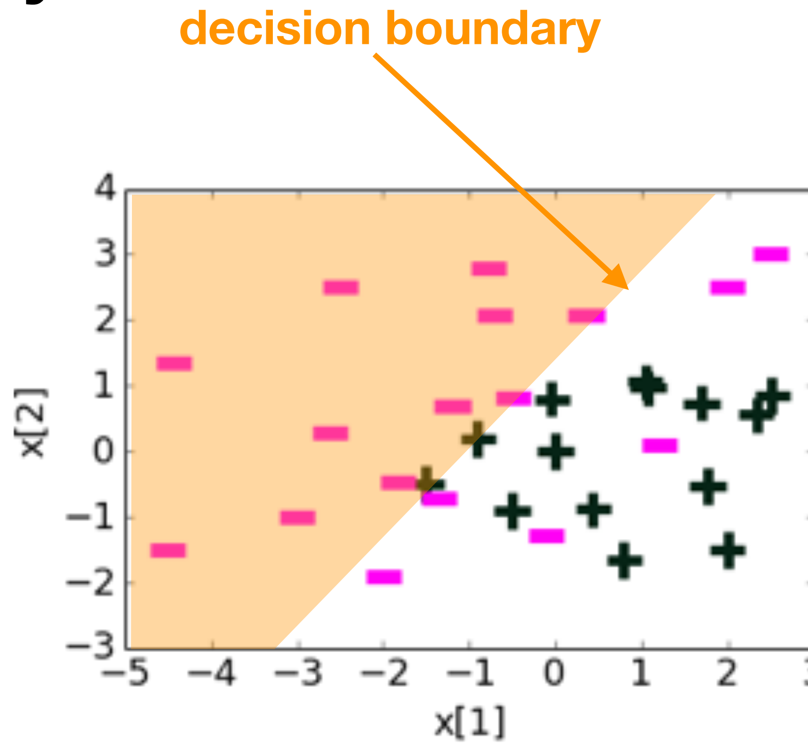




| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 0.23 |
| $h_1(x)$ | x[1] | 1.12 |
| $h_2(x)$ | x[2] | -1.07 |

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

# Learned decision boundary

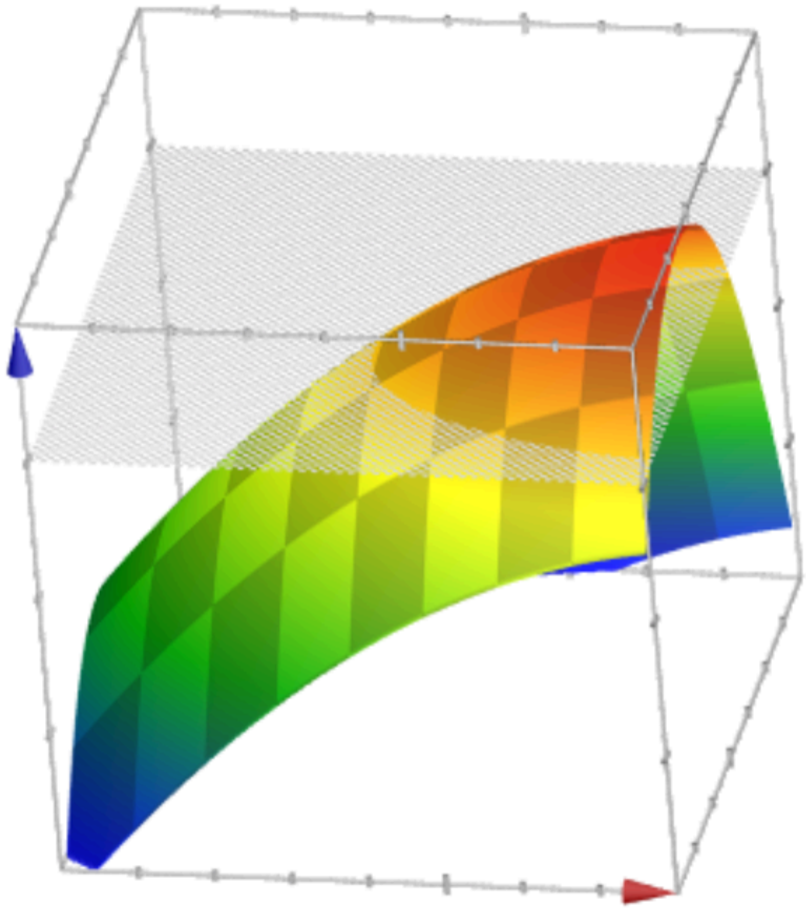


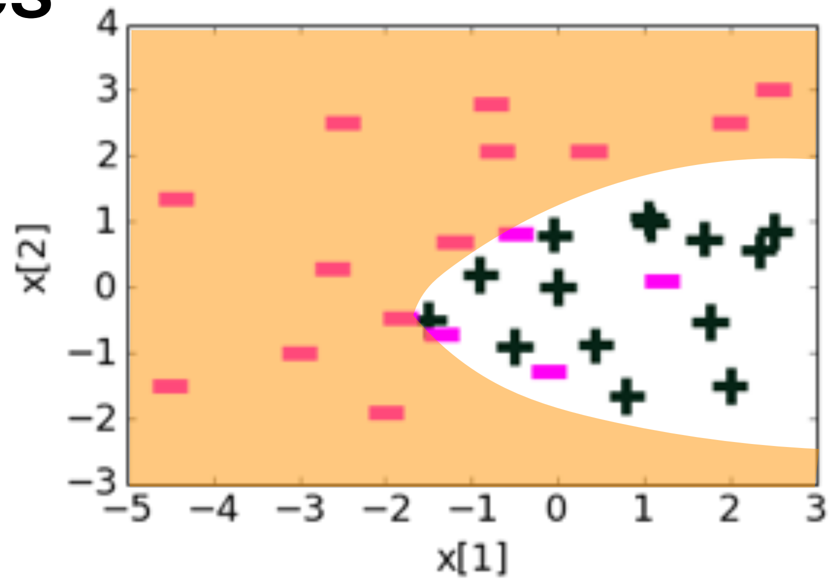




| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 0.23 |
| $h_1(x)$ | x[1] | 1.12 |
| $h_2(x)$ | x[2] | -1.07 |

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

# Adding quadratic features

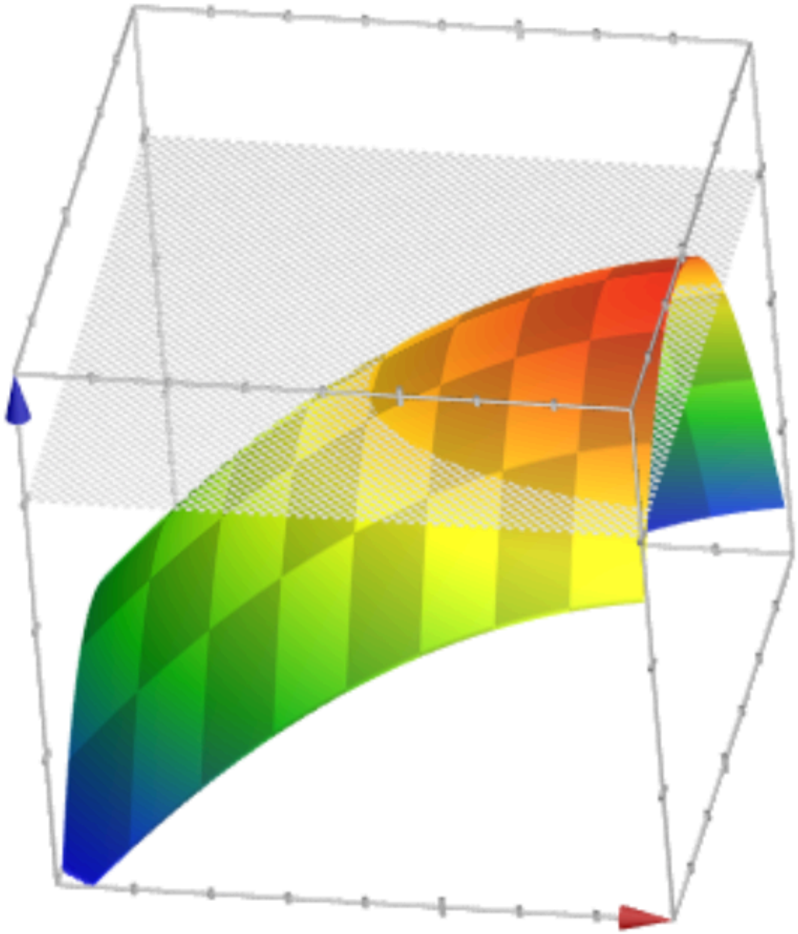


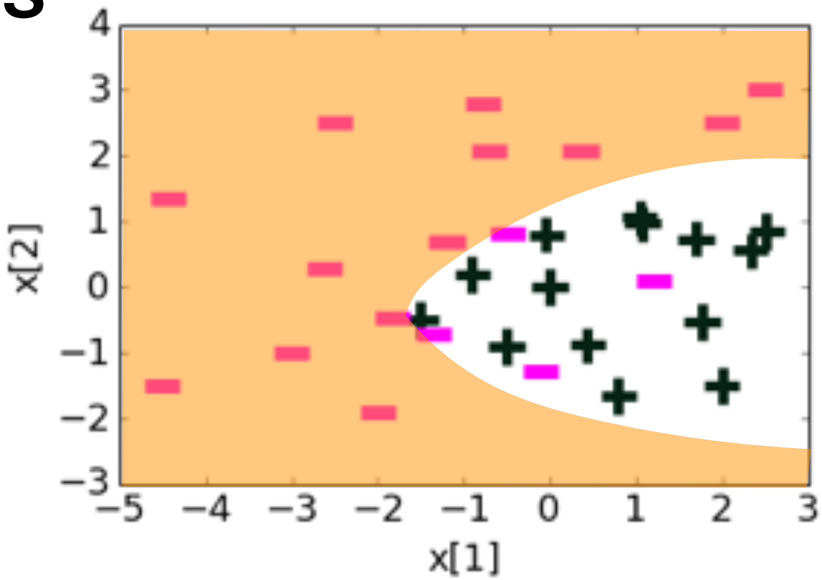



| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 1.68 |
| $h_1(x)$ | x[1] | 1.39 |
| $h_2(x)$ | x[2] | -0.59 |
| $h_3(x)$ | $(x[1])^2$ | -0.17 |
| $h_4(x)$ | $(x[2])^2$ | -0.96 |
| $h_5(x)$ | x[1]x[2] | Omitted |

- Adding more features gives more complex models
- Decision boundary becomes more complex

# Adding quadratic features

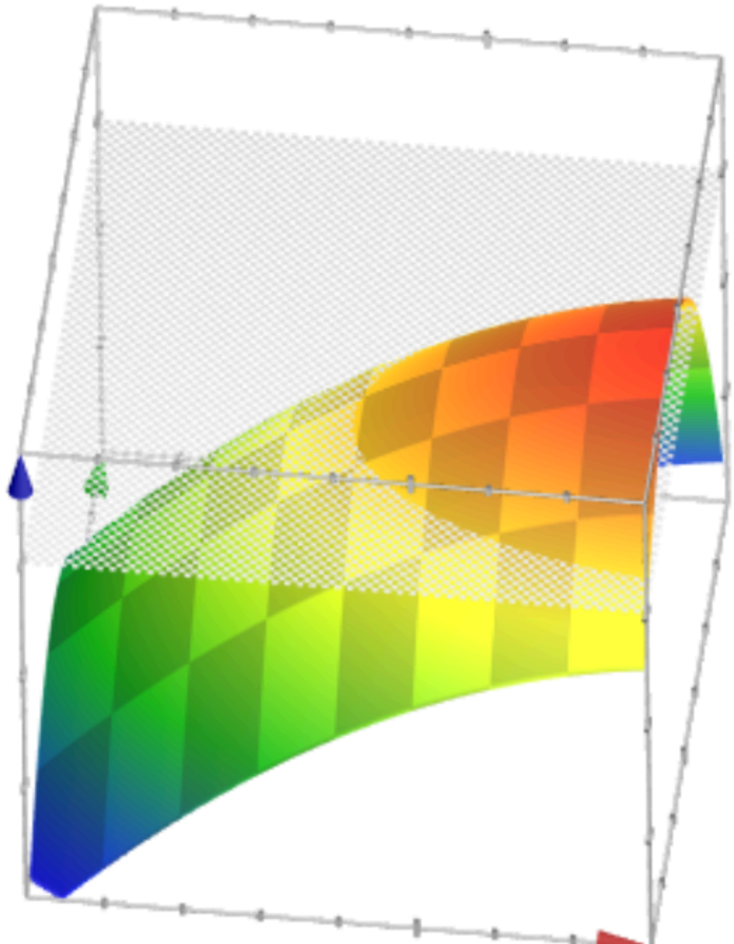


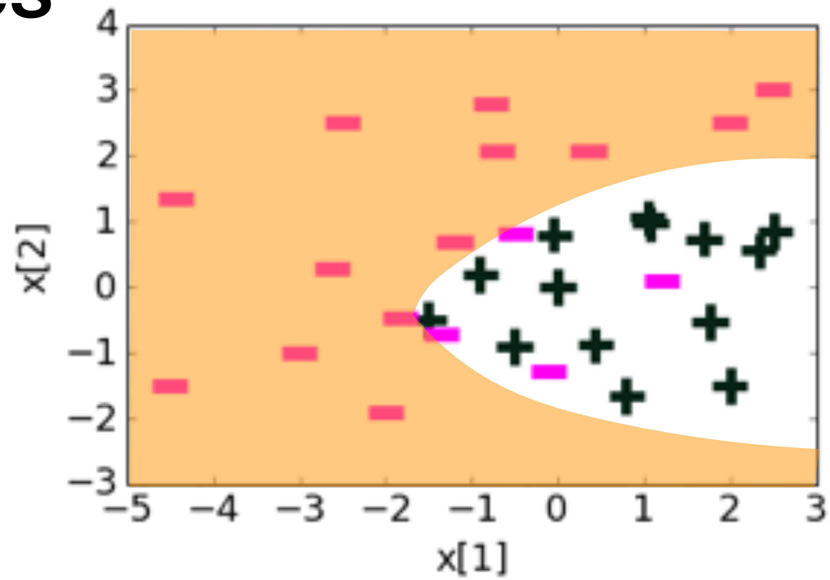



| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 1.68 |
| $h_1(x)$ | x[1] | 1.39 |
| $h_2(x)$ | x[2] | -0.59 |
| $h_3(x)$ | $(x[1])^2$ | -0.17 |
| $h_4(x)$ | $(x[2])^2$ | -0.96 |
| $h_5(x)$ | x[1]x[2] | Omitted |

- Adding more features gives more complex models
- Decision boundary becomes more complex

# Adding quadratic features

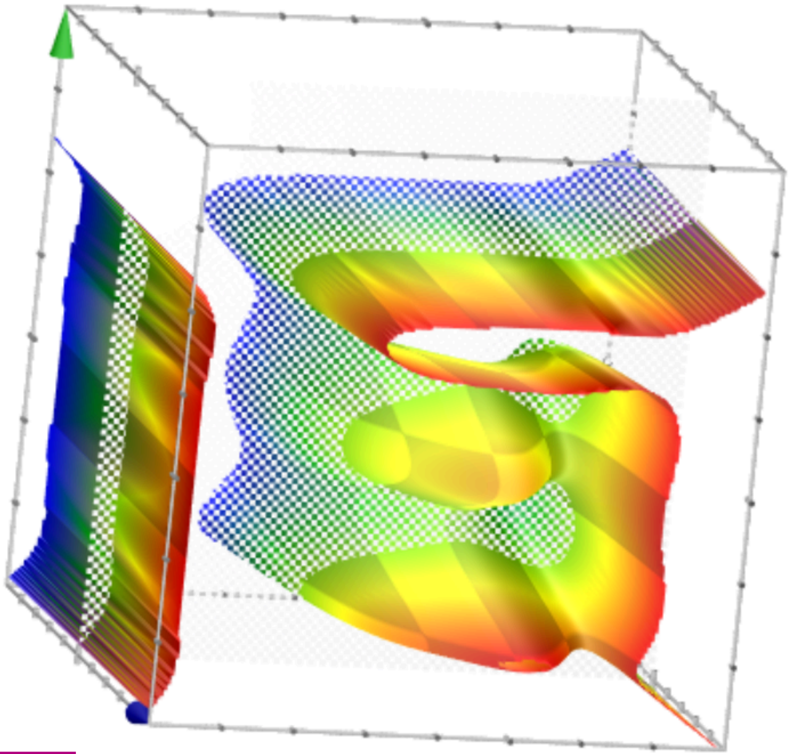




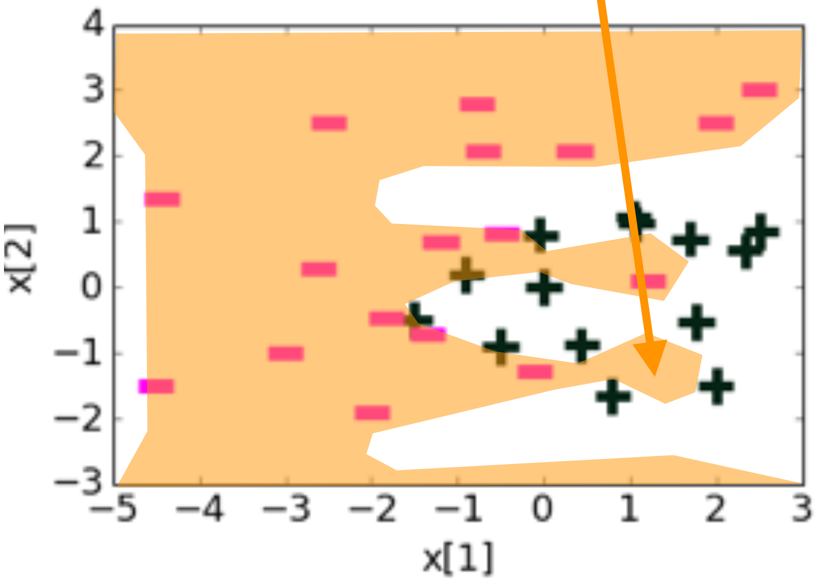

| Feature | Value | Coefficient |
|---|---|---|
| $h_0(x)$ | 1 | 1.68 |
| $h_1(x)$ | x[1] | 1.39 |
| $h_2(x)$ | x[2] | -0.59 |
| $h_3(x)$ | $(x[1])^2$ | -0.17 |
| $h_4(x)$ | $(x[2])^2$ | -0.96 |
| $h_5(x)$ | x[1]x[2] | Omitted |

- Adding more features gives more complex models
- Decision boundary becomes more complex

# Adding higher degree polynomial features



Overfitting leads to non-generalization

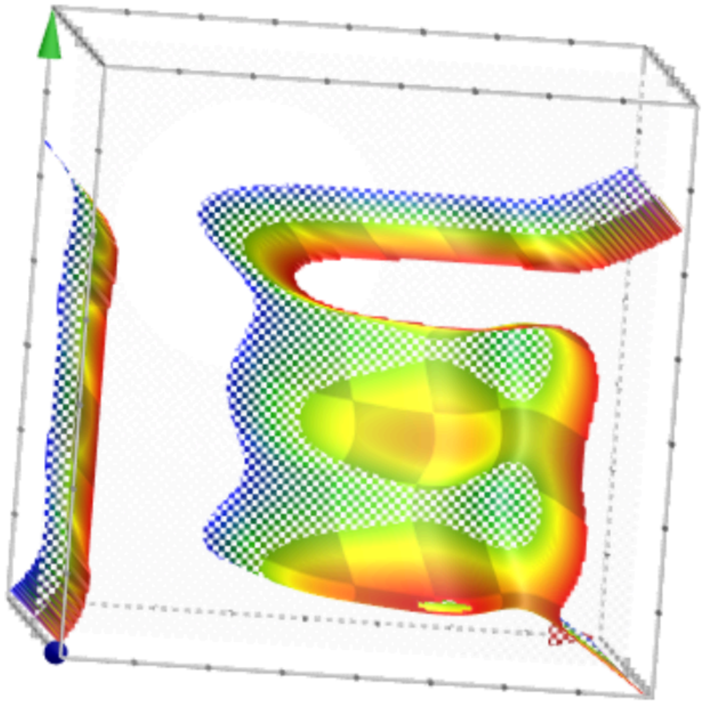


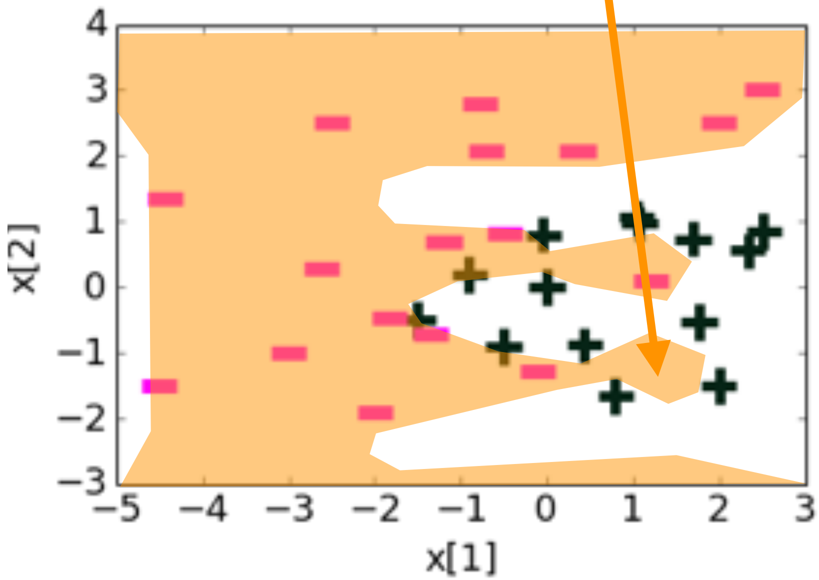

| Feature | Value | Coefficient learned |
|---|---|---|
| $h_0(x)$ | 1 | 21.6 |
| $h_1(x)$ | x[1] | 5.3 |
| $h_2(x)$ | x[2] | -42.7 |
| $h_3(x)$ | $(x[1])^2$ | -15.9 |
| $h_4(x)$ | $(x[2])^2$ | -48.6 |
| $h_5(x)$ | $(x[1])^3$ | -11.0 |
| $h_6(x)$ | $(x[2])^3$ | 67.0 |
| $h_7(x)$ | $(x[1])^4$ | 1.5 |
| $h_8(x)$ | $(x[2])^4$ | 48.0 |
| $h_9(x)$ | $(x[1])^5$ | 4.4 |
| $h_{10}(x)$ | $(x[2])^5$ | -14.2 |
| $h_{11}(x)$ | $(x[1])^6$ | 0.8 |
| $h_{12}(x)$ | $(x[2])^6$ | -8.6 |

Coefficient values getting large

# Adding higher degree polynomial features

Overfitting leads to non-generalization

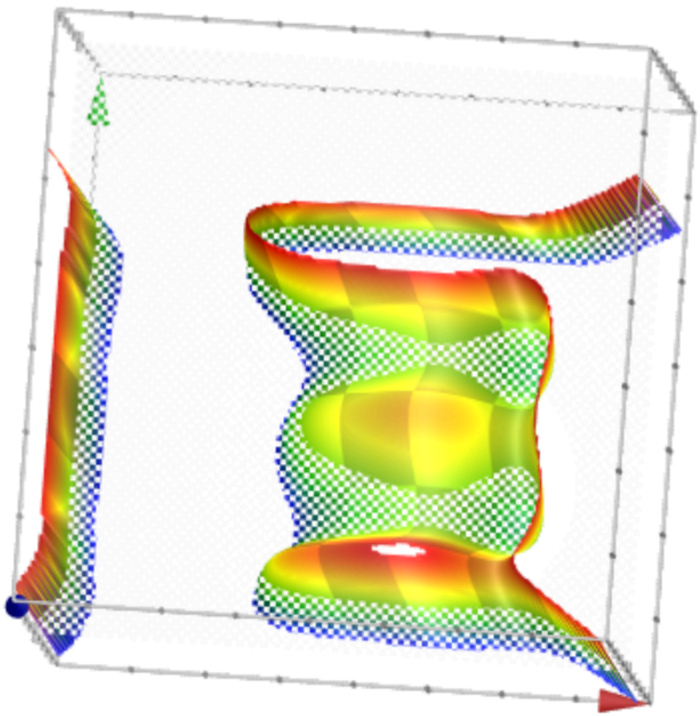




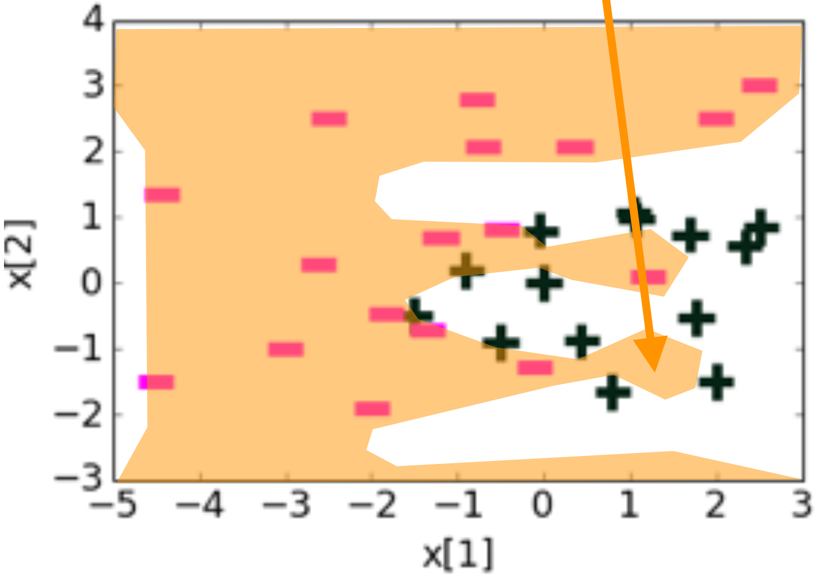

| Feature | Value | Coefficient learned |
|---|---|---|
| $h_0(x)$ | 1 | 21.6 |
| $h_1(x)$ | x[1] | 5.3 |
| $h_2(x)$ | x[2] | -42.7 |
| $h_3(x)$ | $(x[1])^2$ | -15.9 |
| $h_4(x)$ | $(x[2])^2$ | -48.6 |
| $h_5(x)$ | $(x[1])^3$ | -11.0 |
| $h_6(x)$ | $(x[2])^3$ | 67.0 |
| $h_7(x)$ | $(x[1])^4$ | 1.5 |
| $h_8(x)$ | $(x[2])^4$ | 48.0 |
| $h_9(x)$ | $(x[1])^5$ | 4.4 |
| $h_{10}(x)$ | $(x[2])^5$ | -14.2 |
| $h_{11}(x)$ | $(x[1])^6$ | 0.8 |
| $h_{12}(x)$ | $(x[2])^6$ | -8.6 |

Coefficient values getting large

# Adding higher degree polynomial features



Overfitting leads to non-generalization




| Feature | Value | Coefficient learned |
|---|---|---|
| $h_0(x)$ | 1 | 21.6 |
| $h_1(x)$ | x[1] | 5.3 |
| $h_2(x)$ | x[2] | -42.7 |
| $h_3(x)$ | $(x[1])^2$ | -15.9 |
| $h_4(x)$ | $(x[2])^2$ | -48.6 |
| $h_5(x)$ | $(x[1])^3$ | -11.0 |
| $h_6(x)$ | $(x[2])^3$ | 67.0 |
| $h_7(x)$ | $(x[1])^4$ | 1.5 |
| $h_8(x)$ | $(x[2])^4$ | 48.0 |
| $h_9(x)$ | $(x[1])^5$ | 4.4 |
| $h_{10}(x)$ | $(x[2])^5$ | -14.2 |
| $h_{11}(x)$ | $(x[1])^6$ | 0.8 |
| $h_{12}(x)$ | $(x[2])^6$ | -8.6 |

Coefficient values getting large

- Overfitting leads to very large values of

$$f(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

# Creating Features

- Feature mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-$k$ polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ ?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with $p$ where $d < n \ll p$

# Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-$k$ polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ ?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with $p$ where $d < n \ll p$

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)
  and if we can find a kernel for our feature map such that
  $K(x \,.\, x') = \phi(x) \cdot \phi(x')$

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)
  and if we can find a kernel for our feature map such that
  $K(x \,.\, x') = \phi(x) \cdot \phi(x')$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^{n}$ and instead only work with the kernel matrix of the training data $\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

# Ridge Linear Regression as Kernels

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda\|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

$$= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_{n\times n})^{-1}\mathbf{y}$$ (when $n < d$ via linear algebra)

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$

$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda\|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$

$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

- Hence, to make prediction on any future data points, all we need to know is

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$
  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$
$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

- Hence, to make prediction on any future data points, all we need to know is
  - $\mathbf{X}x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, and $\mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$

# Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$
$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$

- Hence, to make prediction on any future data points, all we need to know is
- $\mathbf{X}x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, and $\mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$
- Even if we run ridge linear regression on feature map $\phi(x) \in \mathbb{R}^p$, we only need to access the features via kernel $K(x_i, x_j)$ and $K(x_i, x_{\text{new}})$ and not the features $\phi(x_i)$

# Kernel (i.e., dot-product) of polynomial features

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point $x_i$, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point $x_i$, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point $x_i$, if we can make predictions (as we saw in the previous slide) by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time $dn$

# Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly $k$
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point $x_i$, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point $x_i$, if we can make predictions (as we saw in the previous slide) by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time $dn$
  - The features are **implicit** and accessed only via kernels, making it efficient

# The Kernel Trick

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\widehat{w} = \sum_{i=1}^n \alpha_i x_i} \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i} \text{ for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^{n} \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds $\alpha$ while accessing the data only via $\{x_i^T x_j\}$

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\widehat{w} = \sum_{i=1}^n \alpha_i x_i} \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

   Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds $\alpha$ while accessing the data only via $\{x_i^T x_j\}$
3. Substitute $x_i^T x_j$ with $K(x_i, x_j)$, and find $\alpha$ using the above algorithm from step 2.

# The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\widehat{w} = \sum_{i=1}^n \alpha_i x_i} \text{ for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

   Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds $\alpha$ while accessing the data only via $\{x_i^T x_j\}$
3. $\boxed{\text{Substitute } x_i^T x_j \text{ with } K(x_i, x_j)\text{, and find } \alpha}$ using the above algorithm from step 2.
4. Make prediction with $\widehat{y}_{\text{new}} = \sum_{i=1}^n \alpha_i K(x_i, x_{\text{new}})$

   (replacing $x_i^T x_{\text{new}}$ with $K(x_i, x_{\text{new}})$)

# The Kernel Trick for regularized least squares

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

(Step 1. Use a linear predictor)

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$= \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_\alpha ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{K} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$

# Examples of popular Kernels

- **Polynomials of degree exactly $k$**

$$K(x, x') = (x^T x')^k$$

- **Polynomials of degree up to $k$**

$$K(x, x') = (1 + x^T x')^k$$

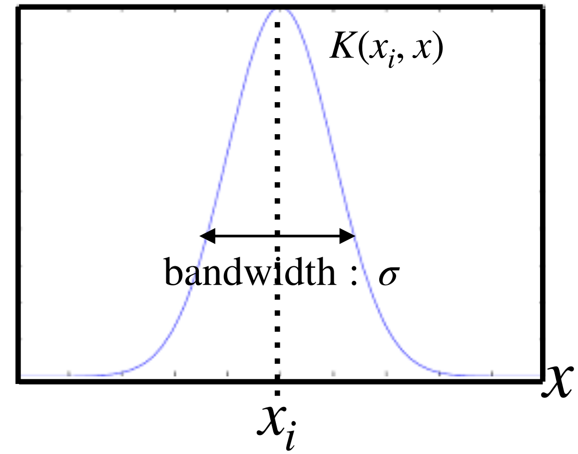
- **Gaussian (squared exponential) kernel**
  (a.k.a RBF kernel for Radial Basis Function)

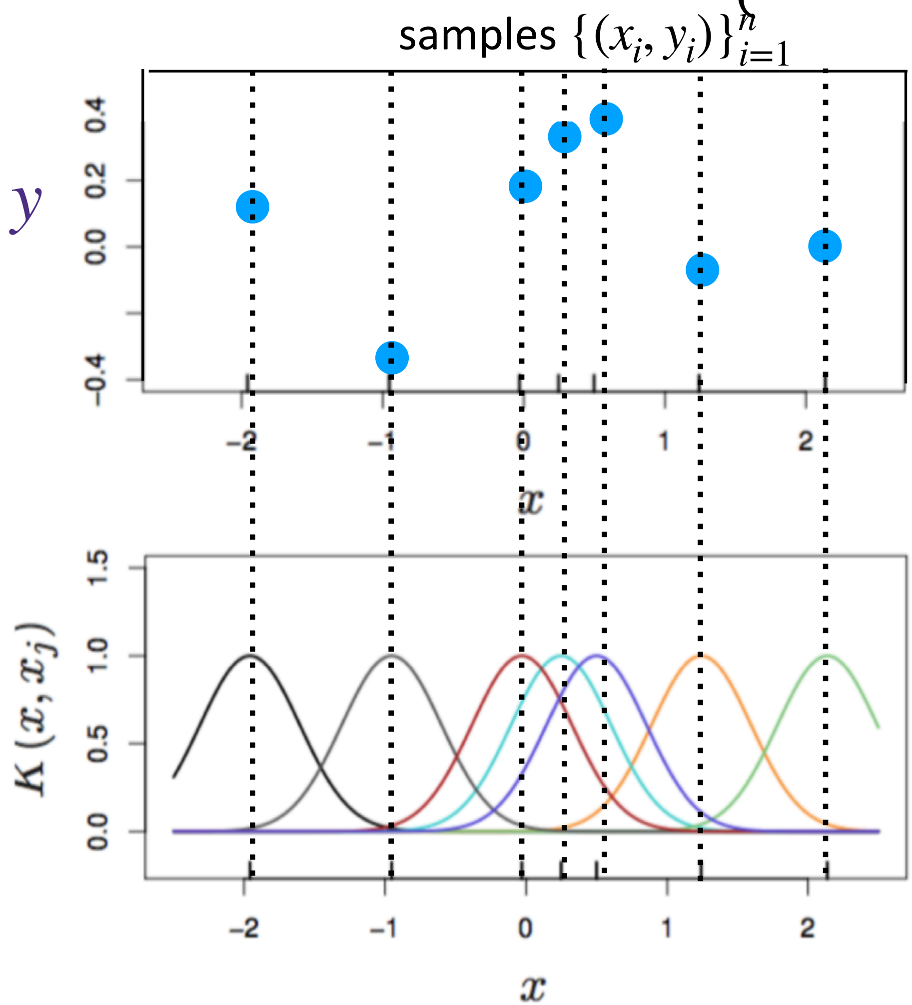
$$K(x, x') = \exp\Big( - \frac{\|x - x'\|_2^2}{2\sigma^2} \Big)$$

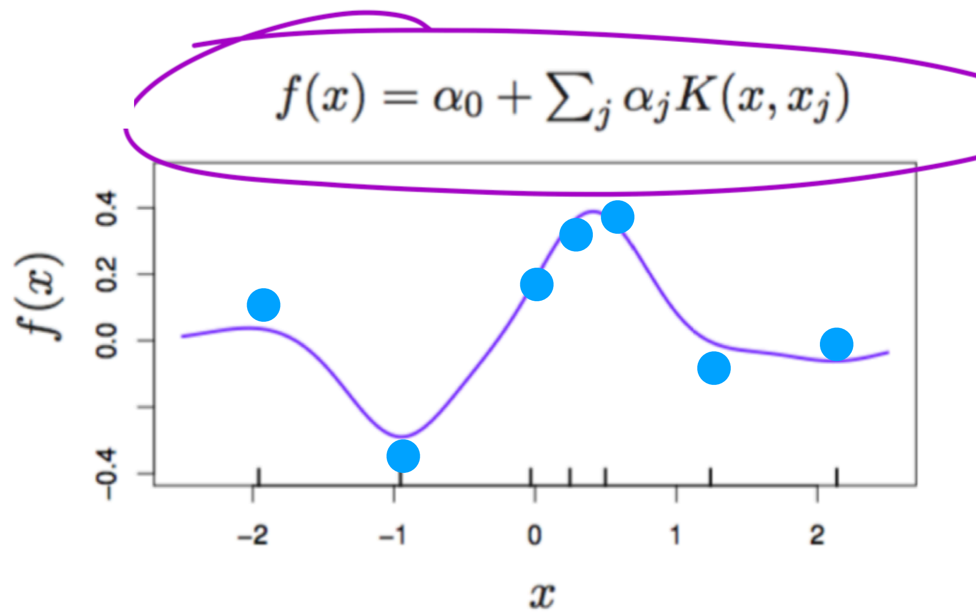
- **Sigmoid**

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$



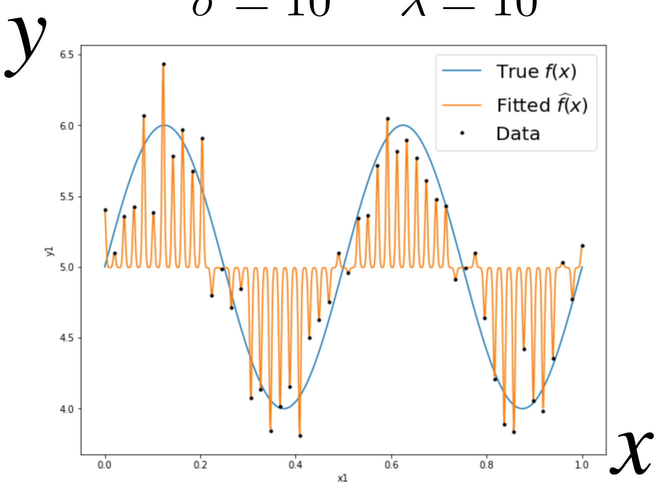




- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of $n$ kernel functions centered at each sample points

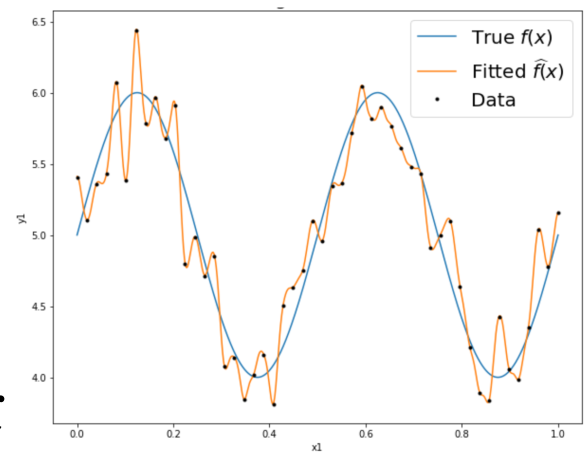
# RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathscr{L}(\alpha) \;=\; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 \;+\; \lambda \alpha^T K \alpha$
- The bandwidth $\sigma^2$ of the kernel regularizes the predictor, and the regularization coefficient $\lambda$ also regularizes the predictor

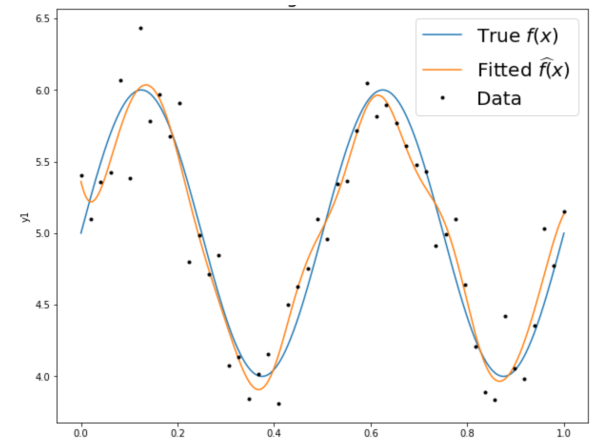
$\sigma = 10^{-3} \;\; \lambda = 10^{-4}$



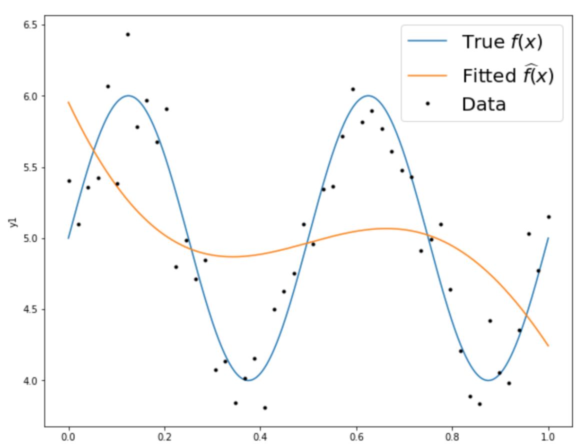

$\sigma = 10^{-2} \;\; \lambda = 10^{-4}$




$\sigma = 10^{-1} \;\; \lambda = 10^{-4}$

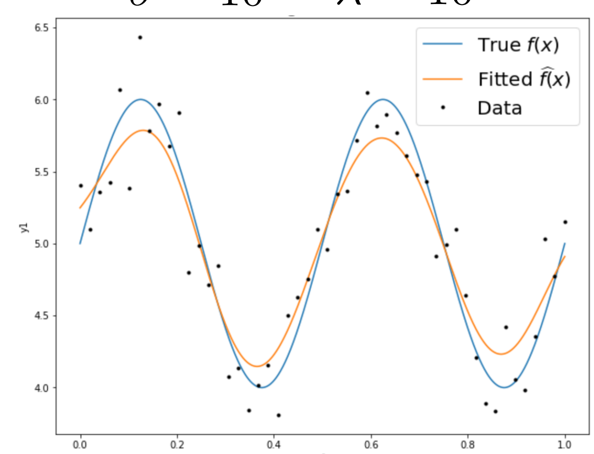



$\sigma = 10^{-0} \;\; \lambda = 10^{-4}$




$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

$\sigma = 10^{-1} \;\; \lambda = 10^{-0}$

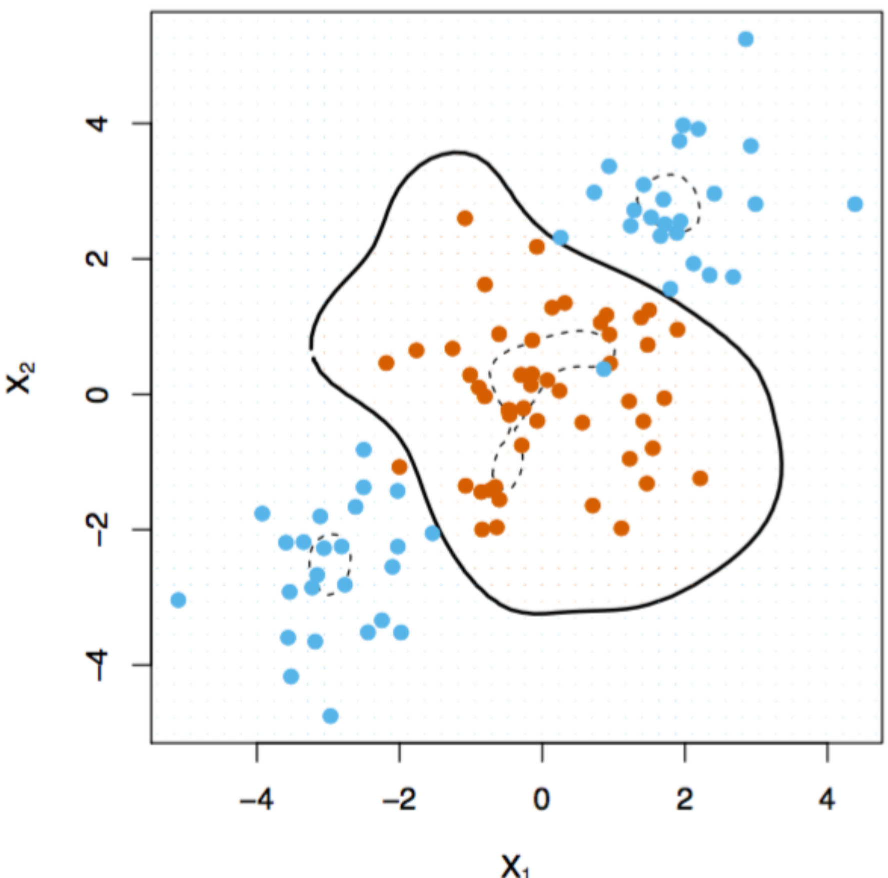


# RBF kernel for SVMs

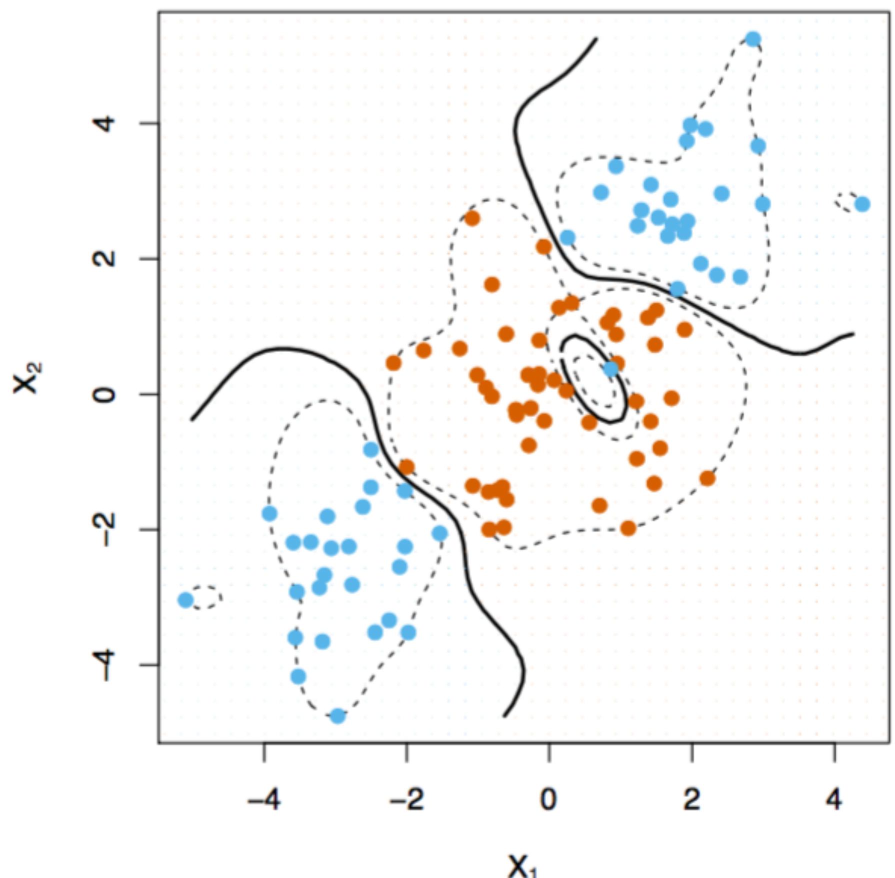
$$\widehat{w} = \arg\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + w^T x_i)\} + \lambda \|w\|_2^2$$

$$\widehat{\alpha}, \widehat{b} = \arg\min_{\alpha \in \mathbb{R}^n, b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} \alpha_j K(x_j, x_i))\} + \lambda \sum_{i=1, j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

Bandwidth $\sigma$ is large enough




Bandwidth $\sigma$ is small



# Bootstrap

# Confidence intervals

- Suppose you have training data $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some true distribution $P_{x,y}$
- We train a kernel ridge regressor, with some choice of a kernel $K : \mathbb{R}^{d\times d} \to \mathbb{R}$

$$\text{minmize}_\alpha \ \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K}\alpha$$

- The resulting predictor is

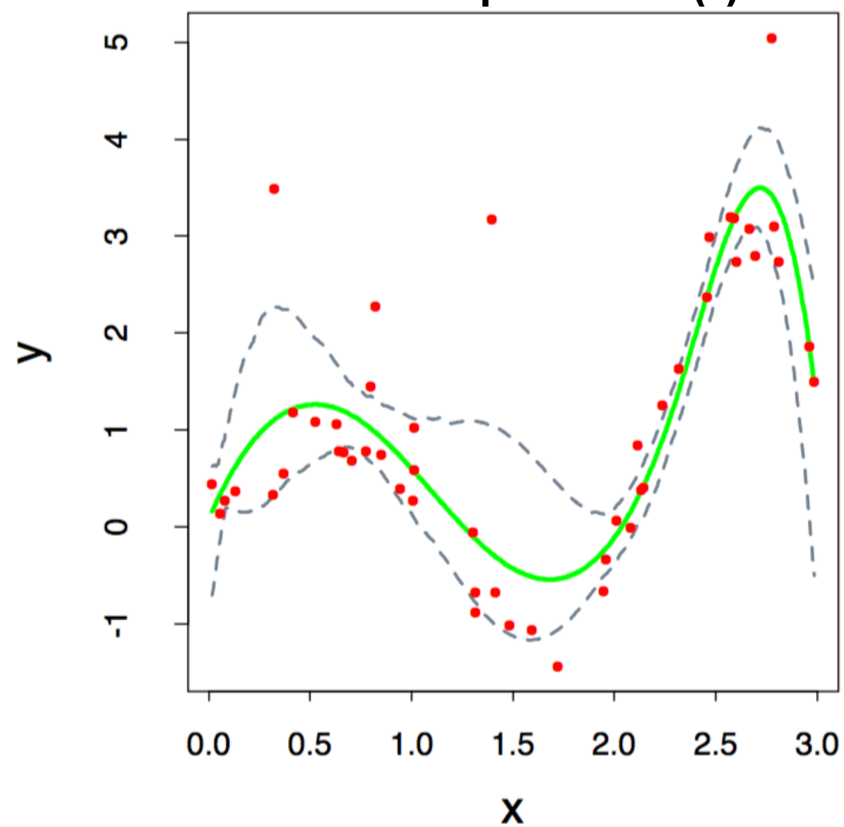
$$f(x) \ = \ \sum_{i=1}^n K(x_i, x)\hat{\alpha}_i,$$

where

$$\hat{\alpha} \ = \ (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \quad \in \mathbb{R}^n$$

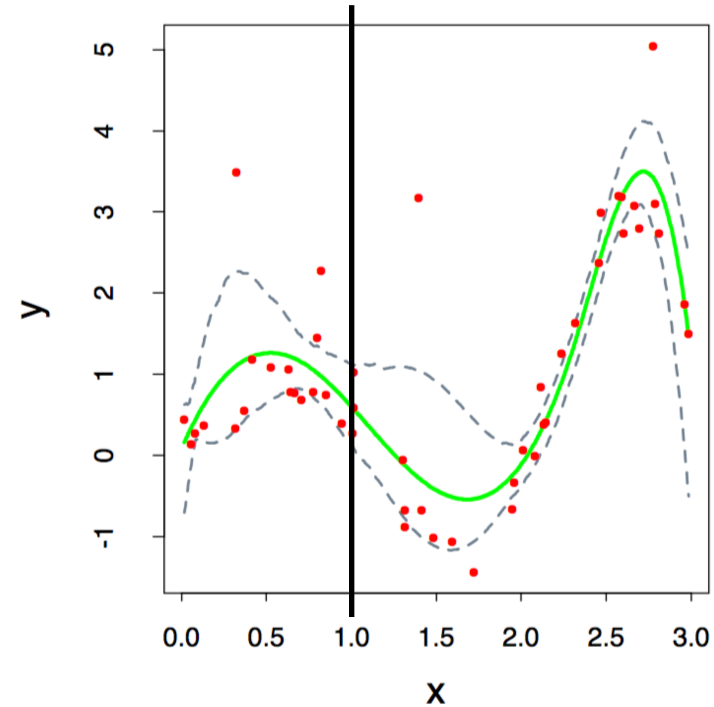
- We wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles

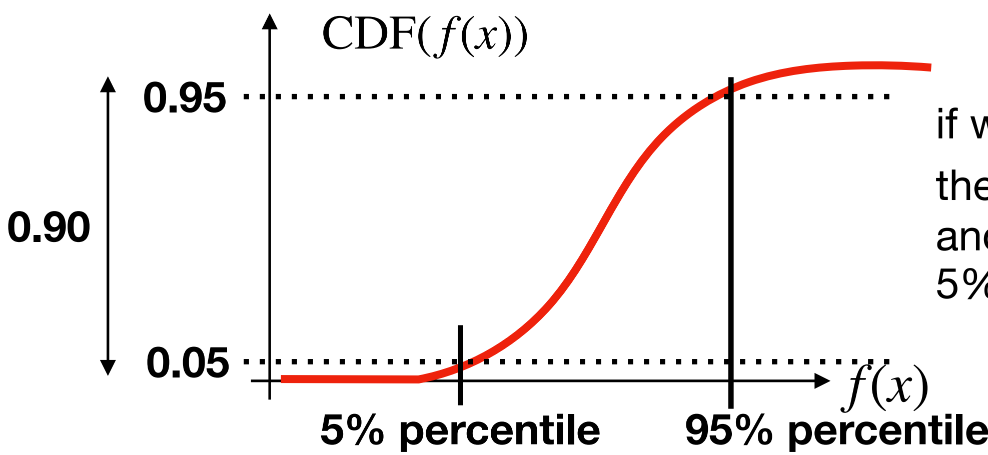
**Example of 5% and 95% percentile curves for predictor f(x)**



# Confidence intervals

- Let's focus on a single $x \in \mathbb{R}^d$
- Note that our predictor $f(x)$ is a random variable, whose randomness comes from the training data $S_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- If we know the statistics (in particular the CDF of the random variable $f(x)$) of the predictor, then the **confidence interval** with **confidence level 90%** is defined as



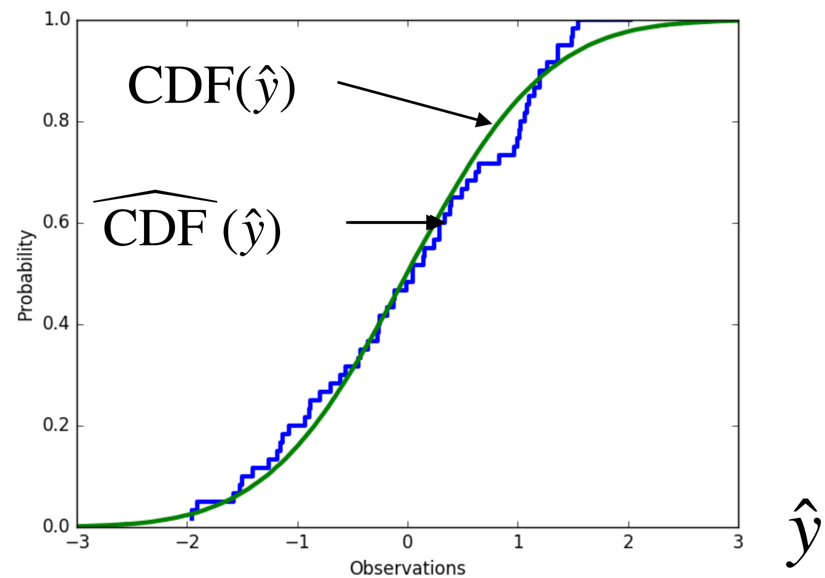

if we know the distribution of our predictor $f(x)$, the green line is the expectation $\mathbb{E}[f(x)]$ and the black dashed lines are the 5% and 95% percentiles in the figure above

- As we do not have the cumulative distribution function (CDF), we need to approximate them

# Confidence intervals

- Hypothetically, if we can sample as many times as we want, then we can train $B \in \mathbb{Z}^+$ i.i.d. predictors, each trained on $n$ fresh samples to get **empirical estimate of the CDF of** $\hat{y} \;=\; f(x)$

- For b=1,…,B
  - Draw $n$ fresh samples $\{(x_i^{(b)}, y_i^{(b)})\}_{i=1}^n$
  - Train a regularized kernel regression $\alpha^{*(b)}$
  - Predict $\hat{y}^{(b)} \;=\; \sum_{i=1}^{n} K(x_i^{(b)}, x)\alpha_i^{*(b)}$




- Let the empirical CDF of those B predictors $\{\hat{y}^{(b)}\}_{b=1}^B$ be $\widehat{\mathrm{CDF}}(\hat{y})$, defined as

$$\widehat{\mathrm{CDF}}(\hat{y}) \;=\; \frac{1}{B}\sum_{b=1}^{B} \mathbf{I}\{\,\hat{y}^{(b)} \le \hat{y}\,\}$$

- Compute the confidence interval using $\widehat{\mathrm{CDF}}(\hat{y})$

# Bootstrap

- As we cannot sample repeatedly (in typical cases), we use **bootstrap samples** instead
- Bootstrap is a general tool for assessing statistical accuracy
- We learn it in the context of confidence interval for trained models

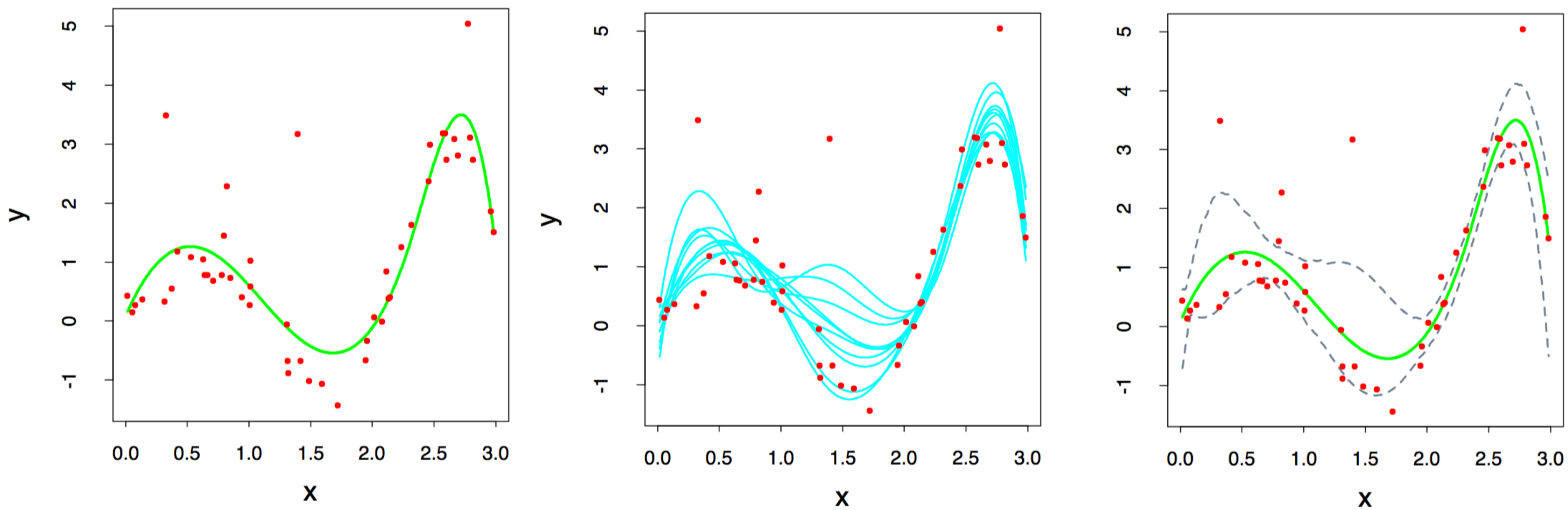
- A **bootstrap dataset** is created from the training dataset by taking $n$ (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^n$

- For b=1,…,B
  - Create a bootstrap dataset $S_{\text{bootstrap}}^{(b)}$
  - Train a regularized kernel regression $\alpha^{*(b)}$
  - Predict $\hat{y}^{(b)} = \sum_{i=1}^{n} K(x_i^{(b)}, x)\alpha_i^{*(b)}$

- Compute the empirical CDF from the bootstrap datasets, and compute the confidence interval

# bootstrap

**training a single predictor** | **multiple bootstrapped predictors** | **90% confidence interval**




Figures from Hastie et al