Support vector machine

Two different approaches to regression/classification

- Assume something about P(x,y)
- Find f which maximizes likelihood of training data | assumption
  - Often reformulated as minimizing loss

Versus

- Pick a loss function
- Pick a set of hypotheses H
- Pick f from H which minimizes loss on training data

Our description of logistic regression was the former

- **Learn**: f:**X** —>Y
  - **X** – features
  - **Y – target classes**

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

- **Expected loss of f:**

- **Bayes optimal classifier:**

- **Model of logistic regression:**

- **Loss function:**

Our description of logistic regression was the former

- **Learn**: f:**X —>**Y
  - **X** – features
  - **Y – target classes**

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

- **Expected loss of f:**

$$\mathbb{E}_{XY}[\mathbf{1}\{f(X) \neq Y\}] = \mathbb{E}_X[\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x]]$$

$$\mathbb{E}_{Y|X}[\mathbf{1}\{f(x) \neq Y\}|X = x] = 1 - P(Y = f(x)|X = x)$$

- **Bayes optimal classifier:**

$$f(x) = \arg\max_y \mathbb{P}(Y = y|X = x)$$

- **Model of logistic regression:**

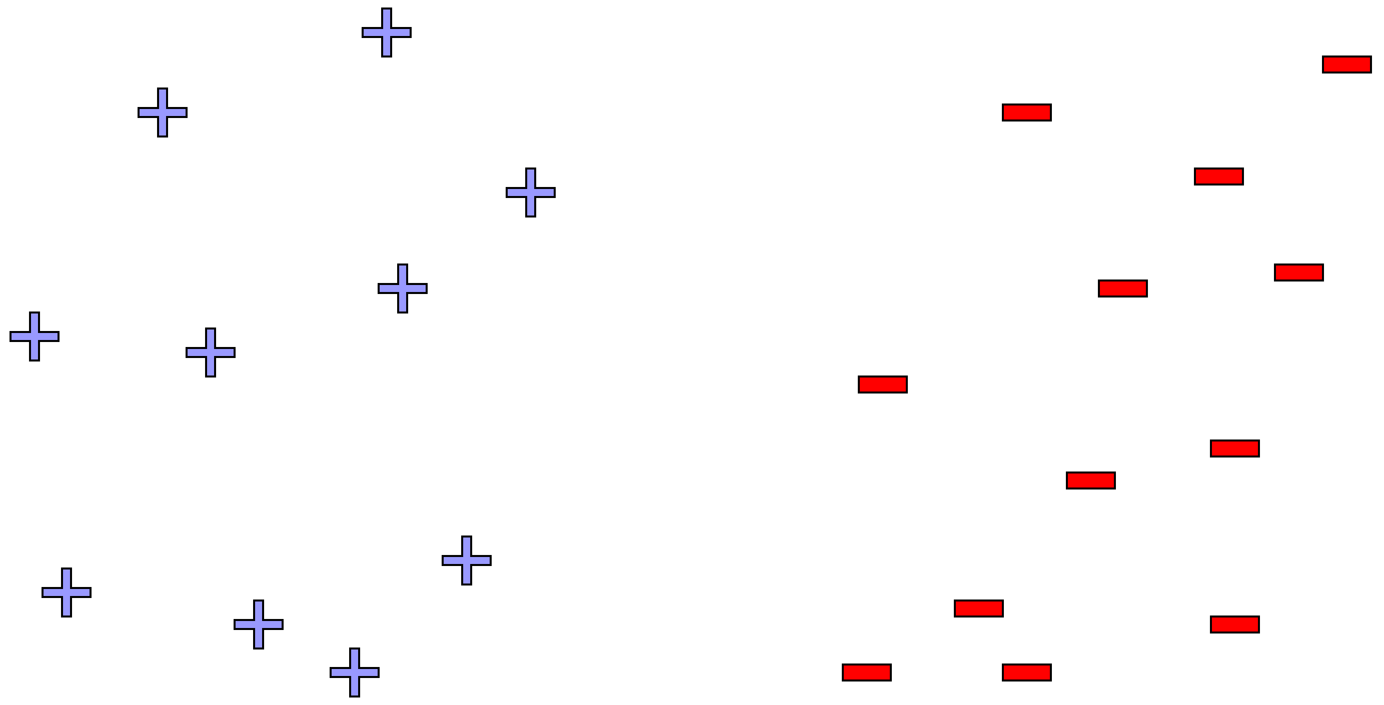
$$P(Y = y|x, w) = \frac{1}{1 + \exp(-y\, w^T x)}$$

- **Loss function:**

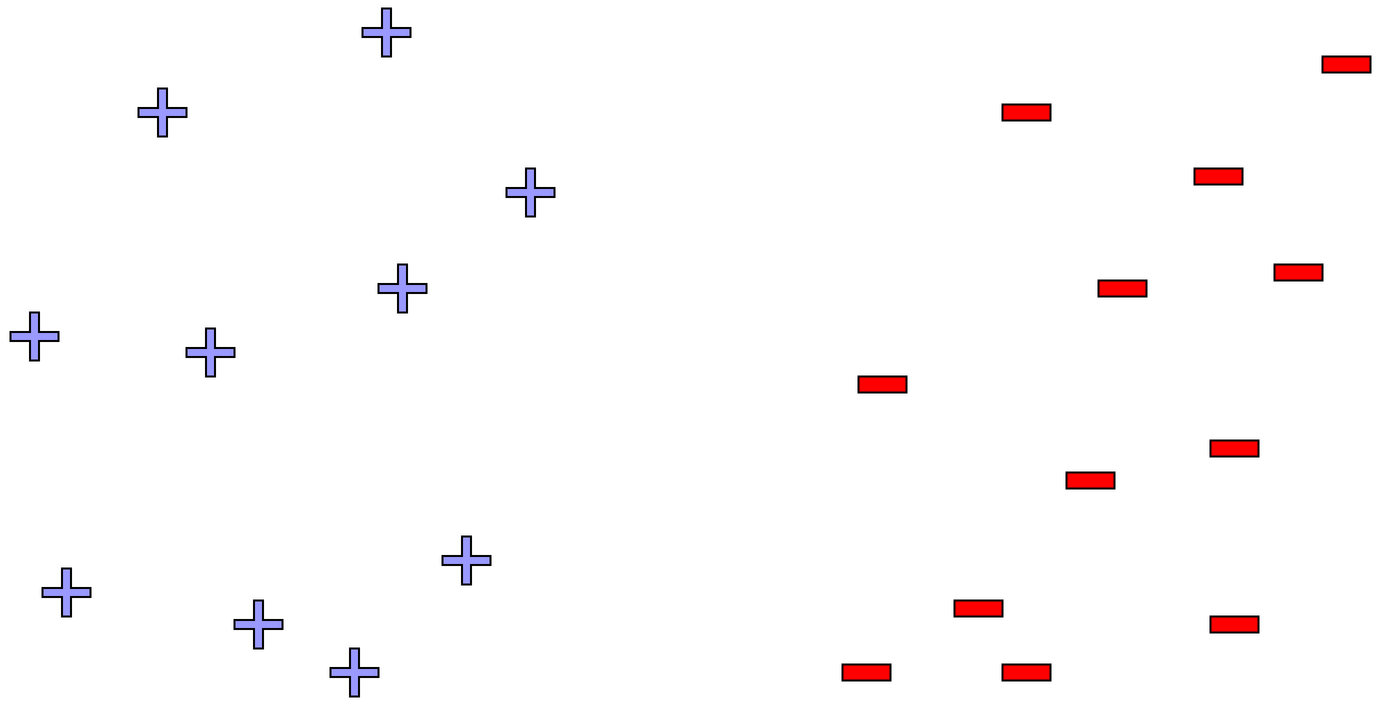
$$\ell(f(x), y) = \mathbf{1}\{f(x) \neq y\}$$

What if the model is wrong? What other ways can we pick linear decision rules?

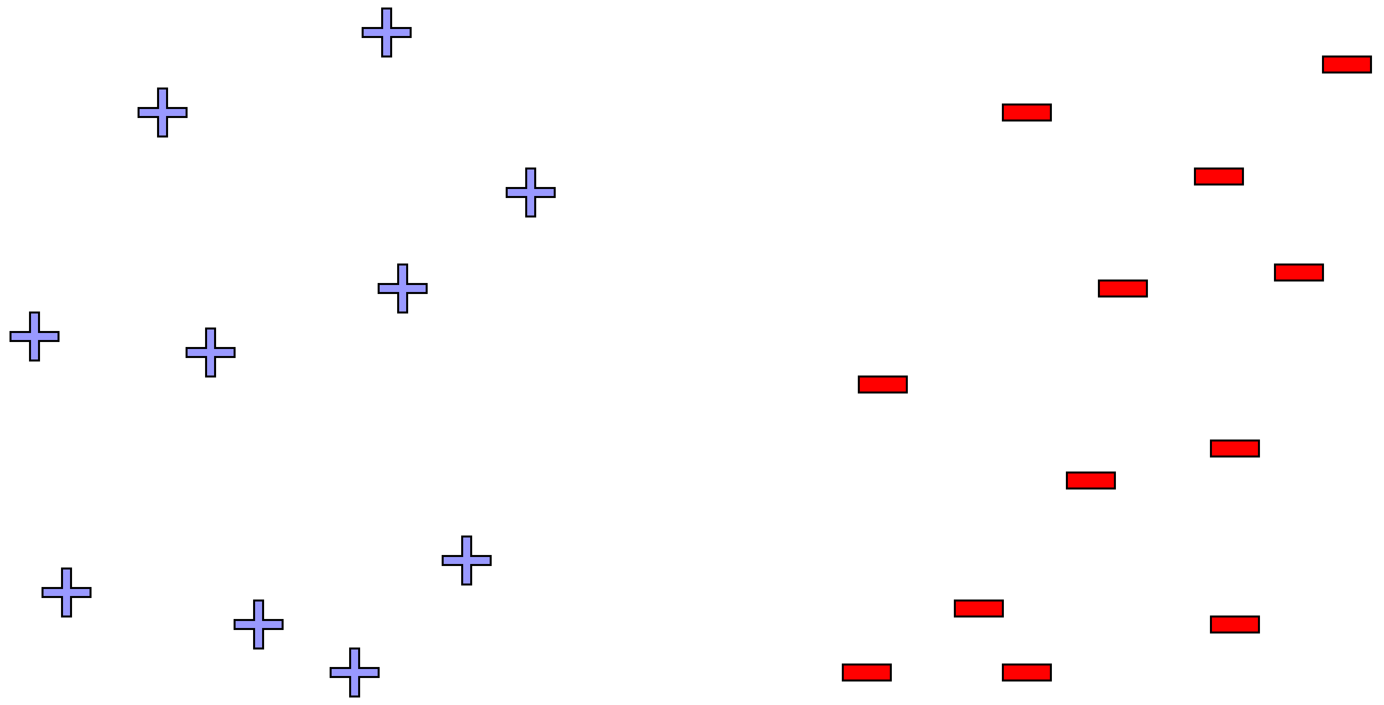
Linear classifiers – Which line is better?



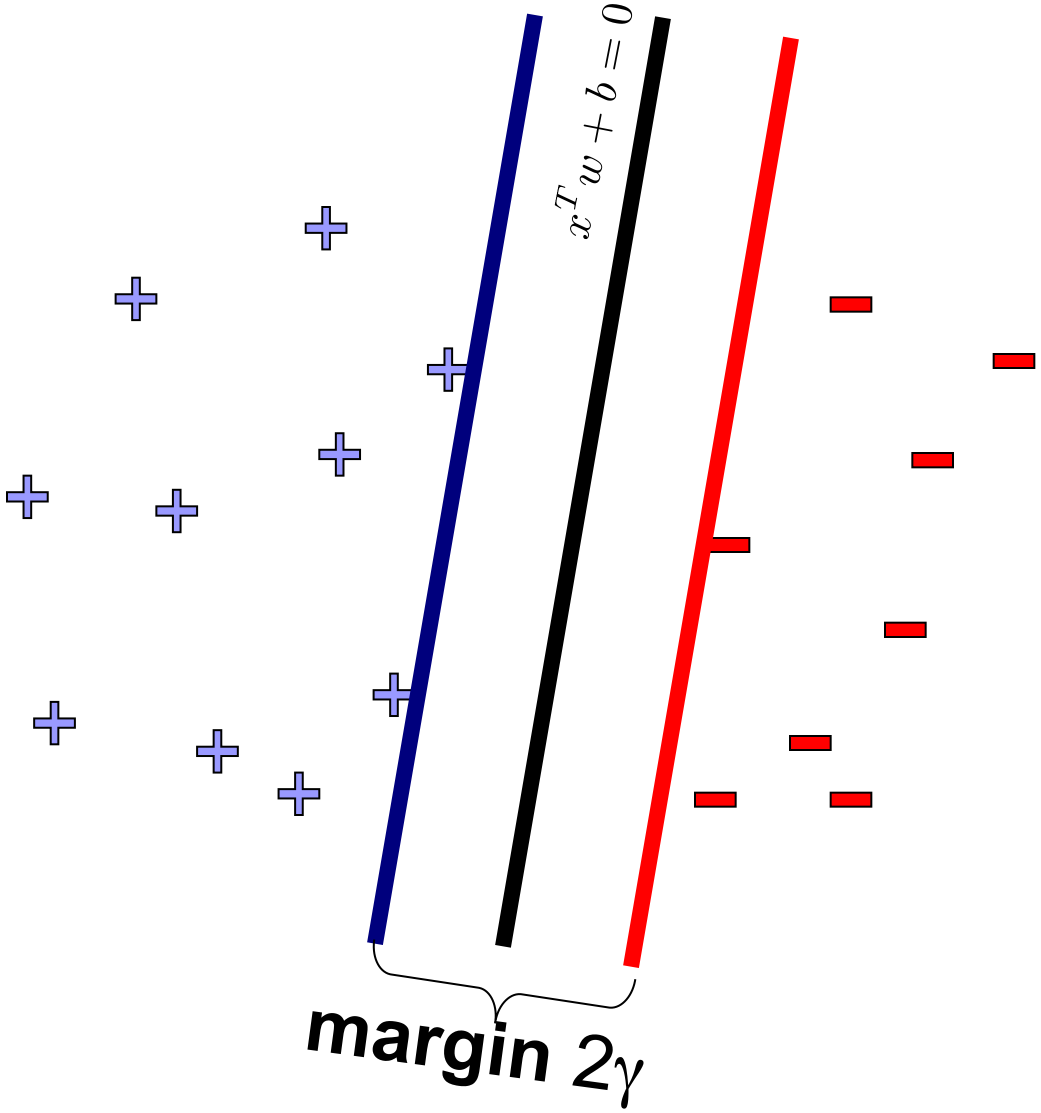
Linear classifiers – Which line is better?



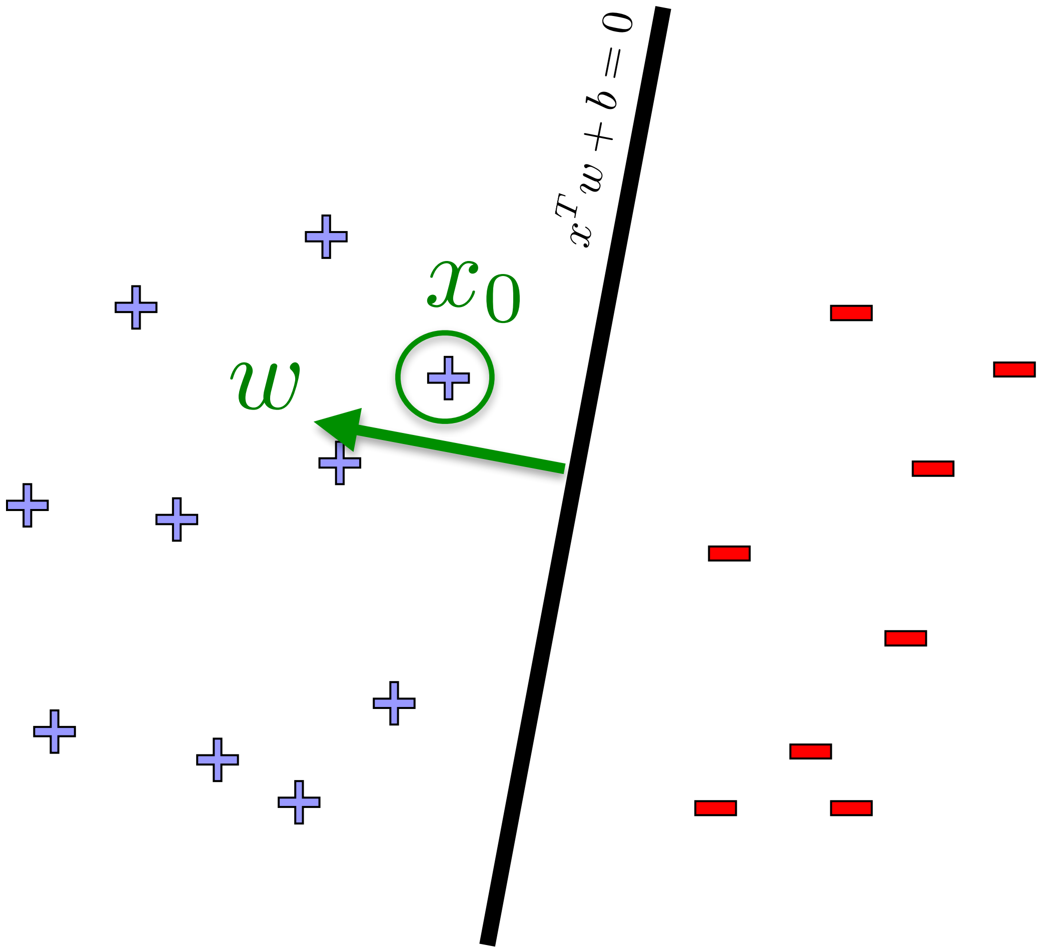
Linear classifiers – Which line is better?



Pick the one with the largest margin!

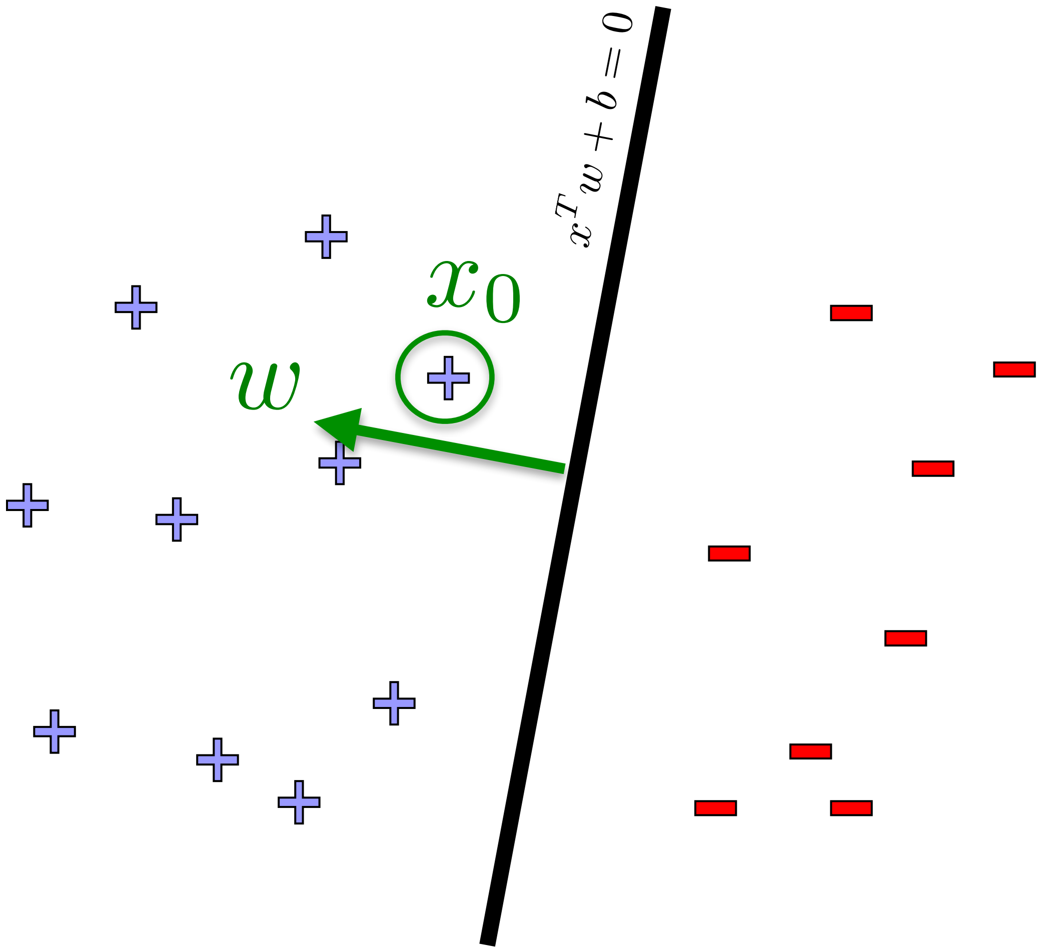


Pick the one with the largest margin!



Distance from x_0 to hyperplane defined by $x^T w + b = 0$?

Pick the one with the largest margin!



Distance from x_0 to hyperplane defined by $x^T w + b = 0$?

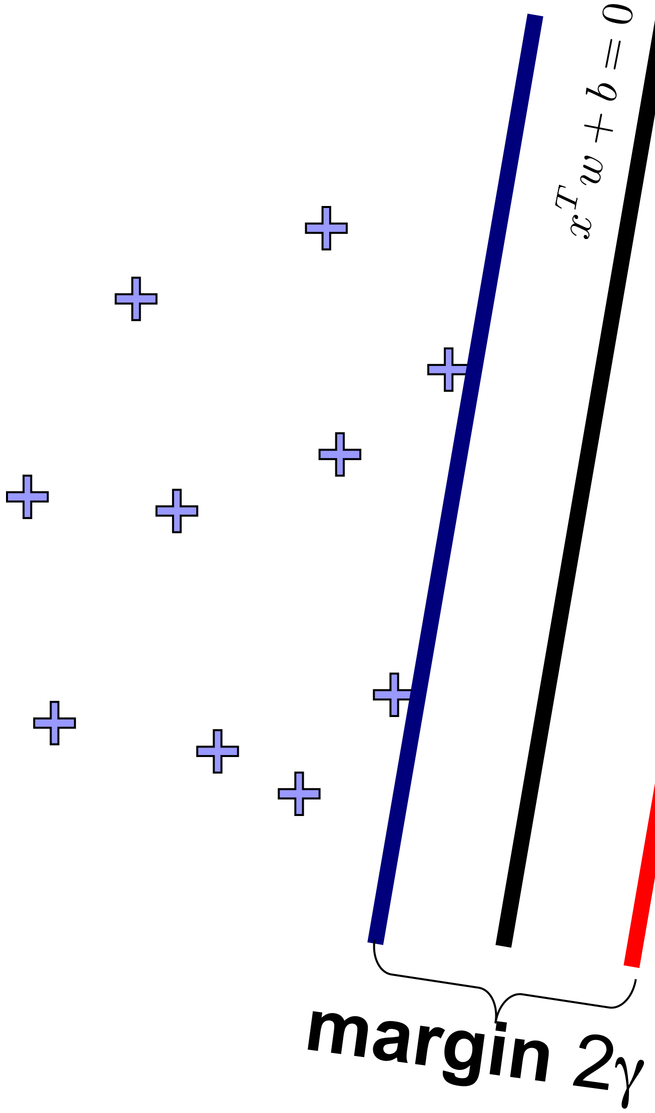
If $\widetilde{x}_0$ is the projection of x_0 onto the hyperplane then

$$||x_0 - \widetilde{x}_0||_2 = |(x_0^T - \widetilde{x}_0)^T \frac{w}{||w||_2}|$$

$$= \frac{1}{||w||_2} |x_0^T w - \widetilde{x}_0^T w|$$

$$= \frac{1}{||w||_2} |x_0^T w + b|$$

Pick the one with the largest margin!

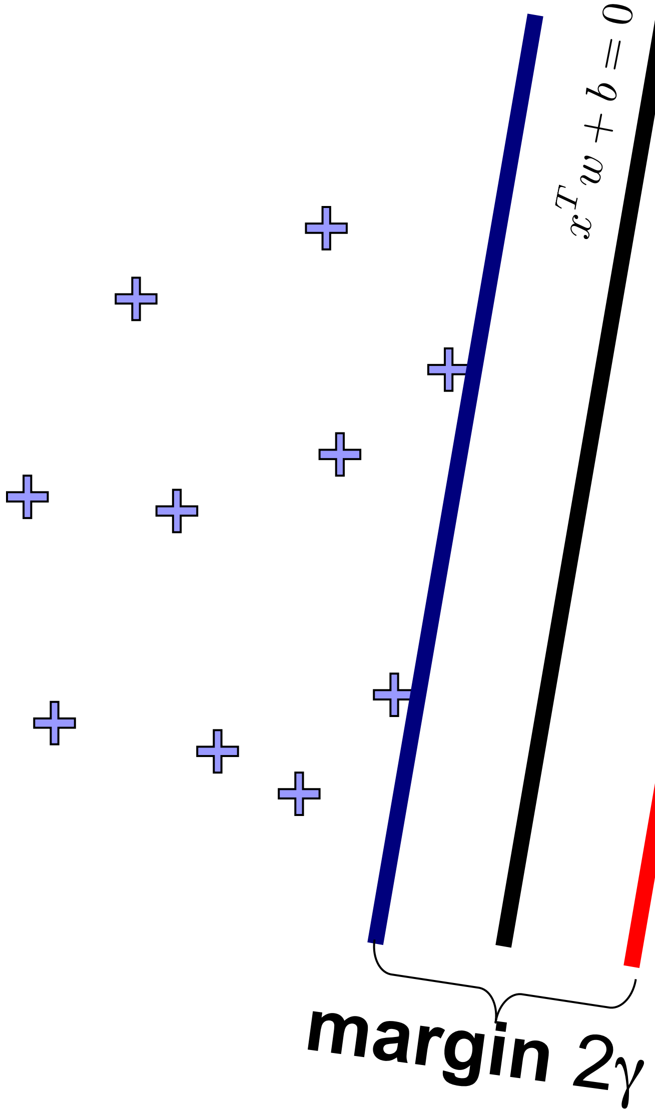


Distance of x_0 from hyperplane $x^T w + b$:

$$\frac{1}{||w||_2}(x_0^T w + b)$$

Optimal Hyperplane

Pick the one with the largest margin!



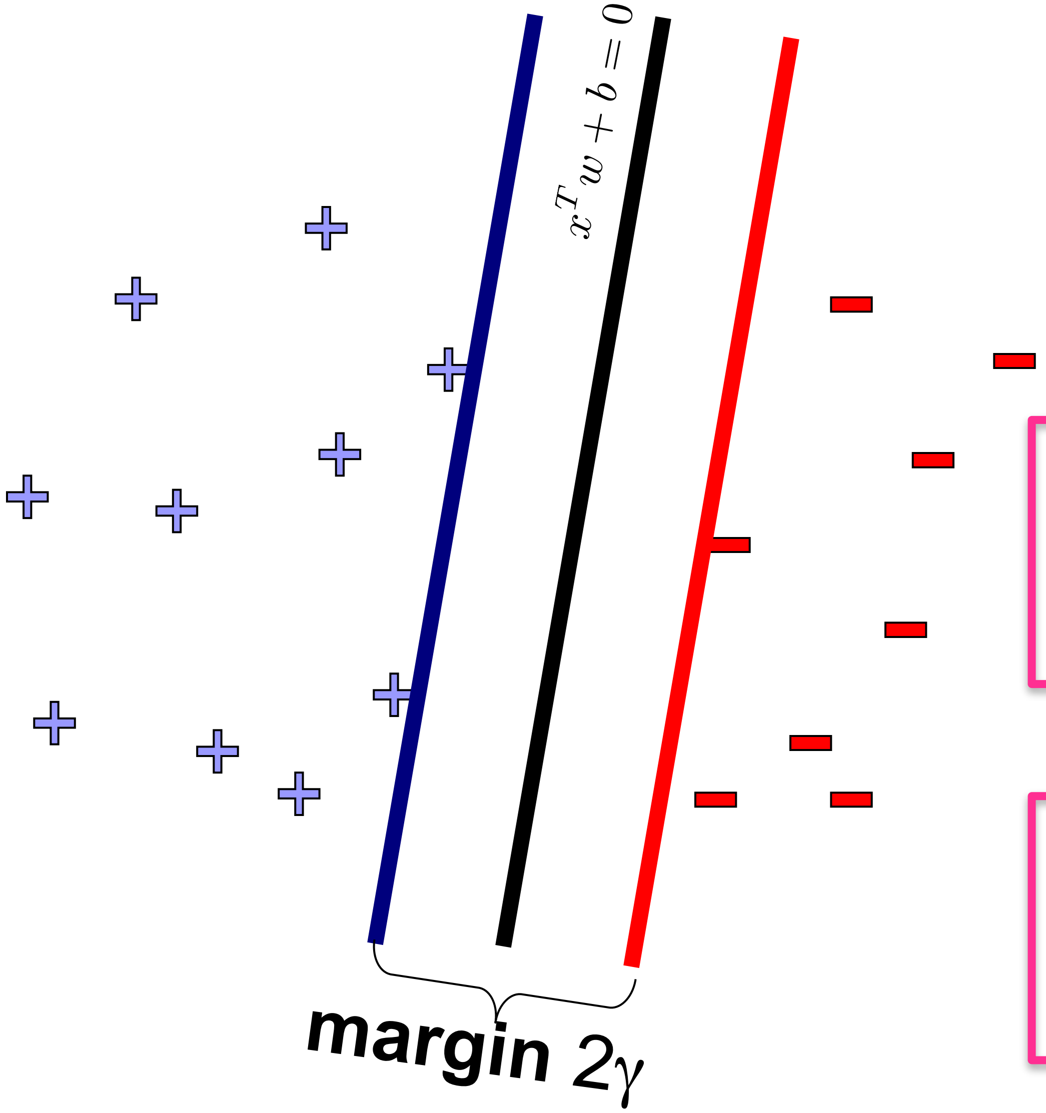
Distance of x_0 from hyperplane $x^T w + b$:

$$\frac{1}{||w||_2}(x_0^T w + b)$$

Optimal Hyperplane

$$\boxed{\begin{aligned}&\max_{w,b} \gamma \\ &\text{subject to } \frac{1}{||w||_2} y_i(x_i^T w + b) \geq \gamma \quad \forall i\end{aligned}}$$

Pick the one with the largest margin!



Distance of x_0 from hyperplane $x^T w + b$:

$$\frac{1}{||w||_2}(x_0^T w + b)$$

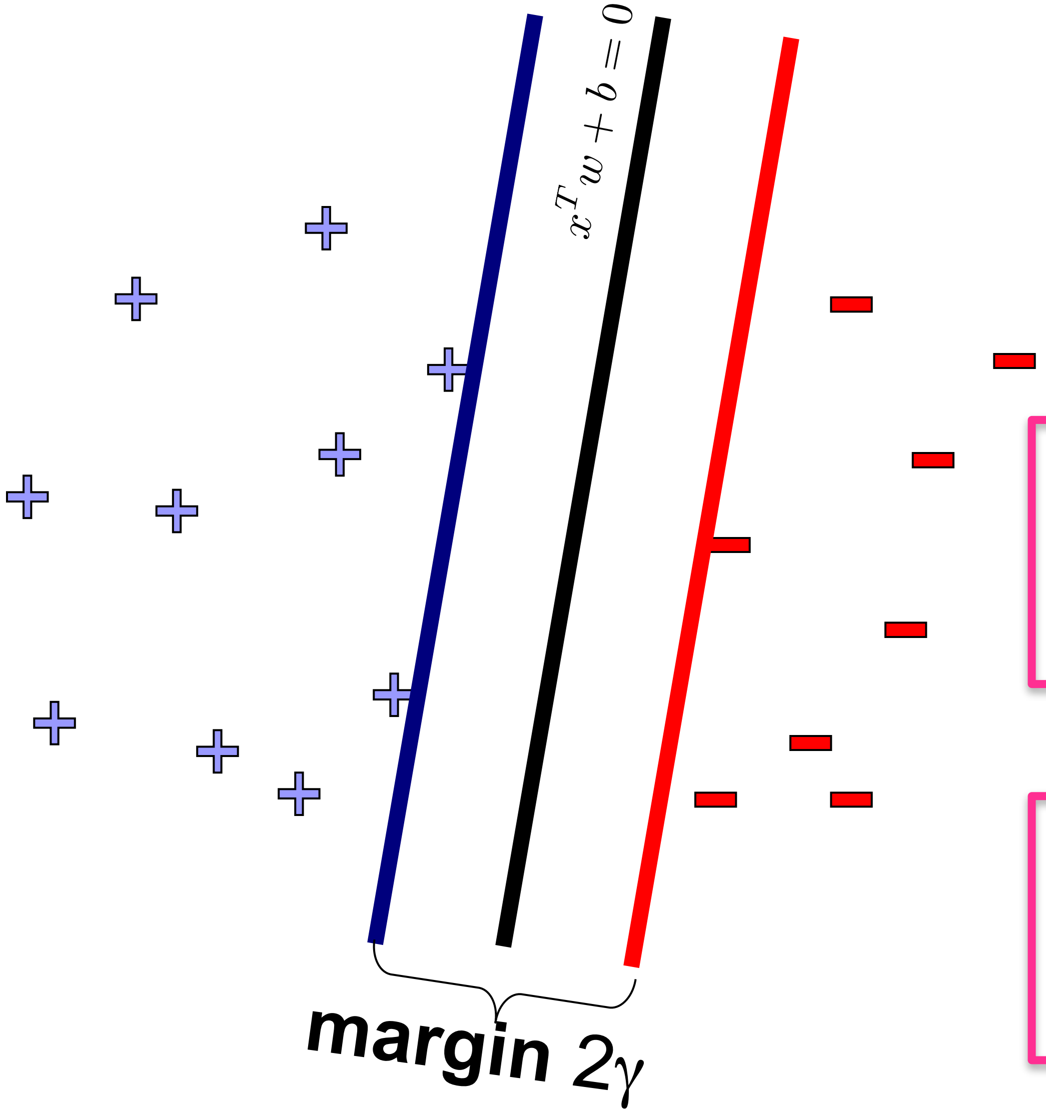
Optimal Hyperplane

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

$$\text{subject to } \frac{1}{||w||_2} y_i (x_i^T w + b) \geq \gamma \quad \forall i$$

Optimal Hyperplane (reparameterized)

Pick the one with the largest margin!



Distance of x_0 from hyperplane $x^T w + b$:

$$\frac{1}{||w||_2}(x_0^T w + b)$$

Optimal Hyperplane

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

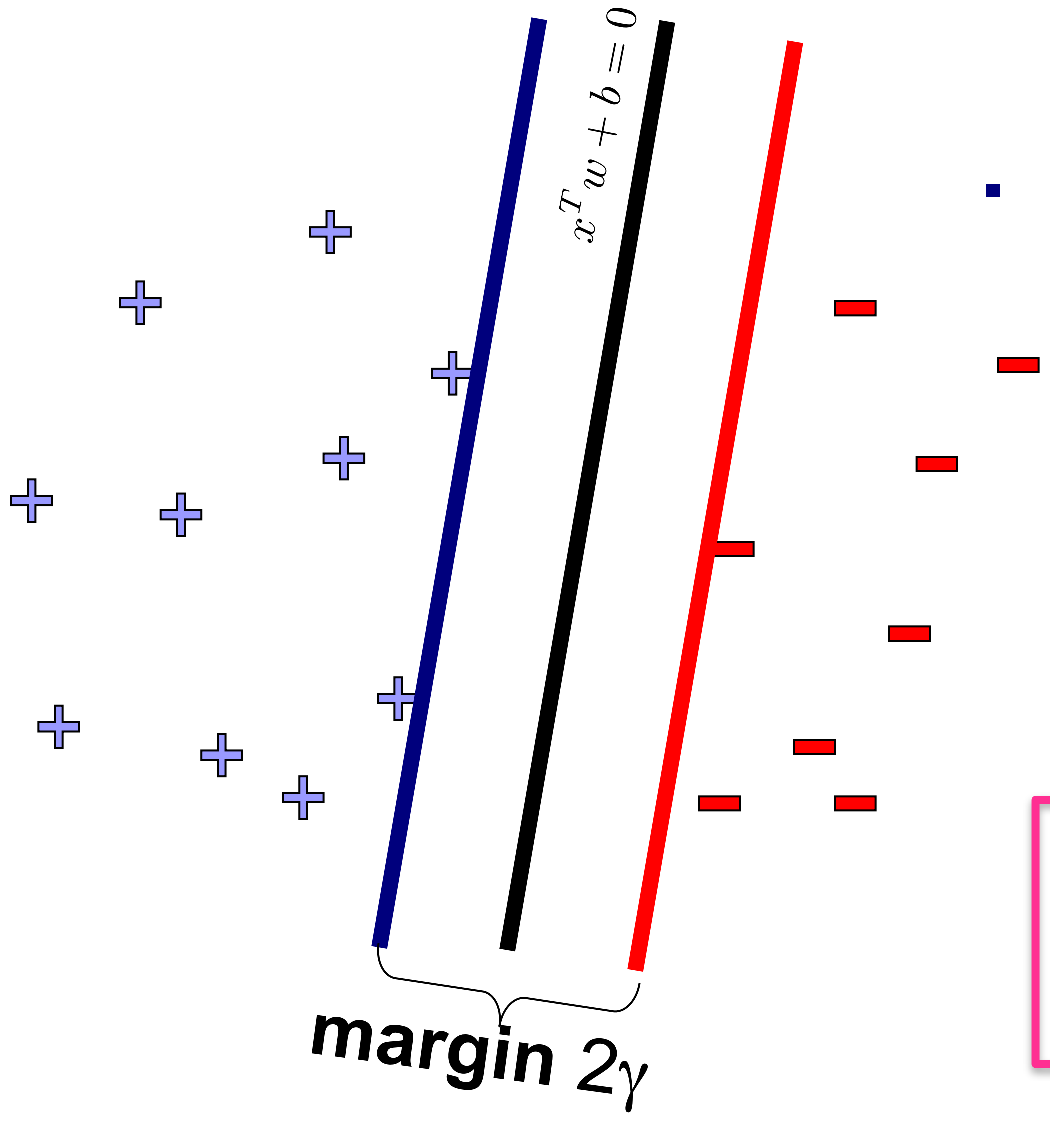
$$\text{subject to } \frac{1}{||w||_2} y_i(x_i^T w + b) \geq \gamma \quad \forall i$$

Optimal Hyperplane (reparameterized)

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

$$\text{subject to } y_i(x_i^T w + b) \geq 1 \quad \forall i$$

Pick the one with the largest margin!

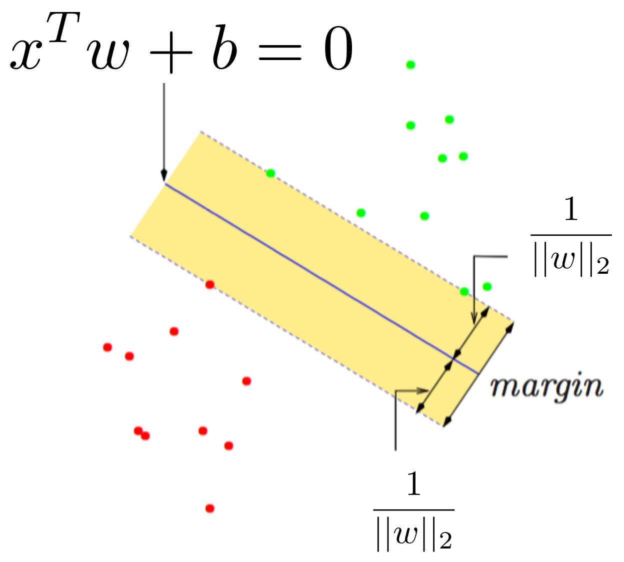


- Solve efficiently by many methods, e.g.,
  - quadratic programming (QP)
    - Well-studied solution algorithms
  - Stochastic gradient descent
  - Coordinate descent (in the dual)

Optimal Hyperplane (reparameterized)

$$\boxed{\begin{aligned}&\min_{w,b} ||w||_2^2\\&\text{subject to } y_i(x_i^T w + b) \geq 1 \quad \forall i\end{aligned}}$$

What are support vectors



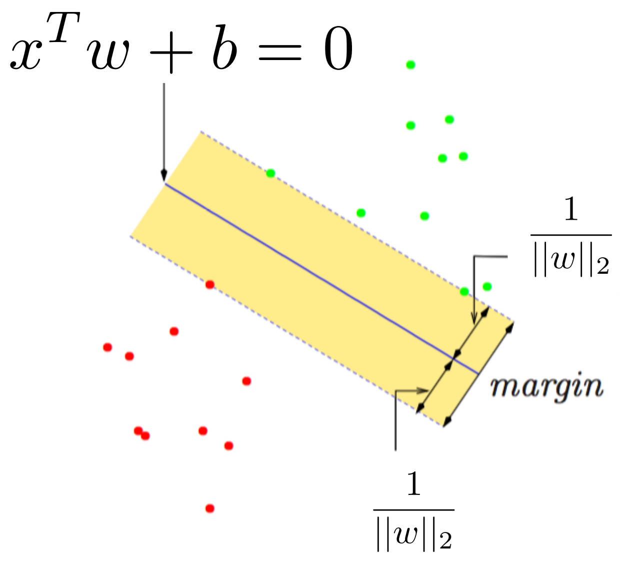
If data is linearly separable

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

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

Note: the solution of this can be written in terms of very few of the training points. These points are known as support vectors.

What if the data is not linearly separable?



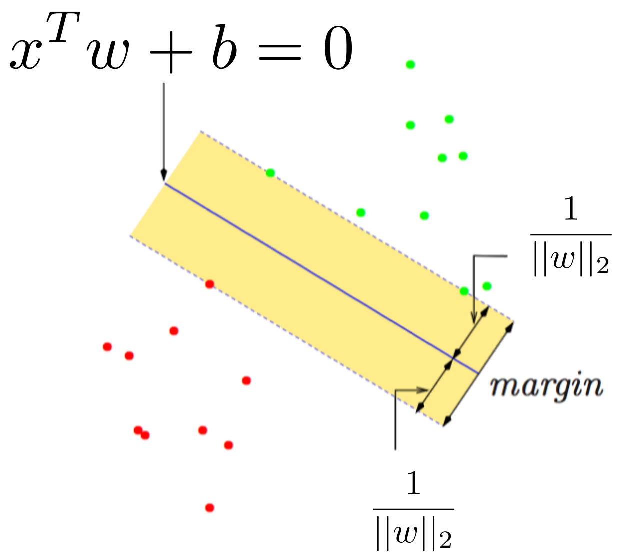
If data is linearly separable

$$\min_{w,b} ||w||_2^2$$
$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

If data is not linearly separable, some points don't satisfy margin constraint:

Two options:
1. Introduce slack to this optimization problem
2. Lift to higher dimensional space

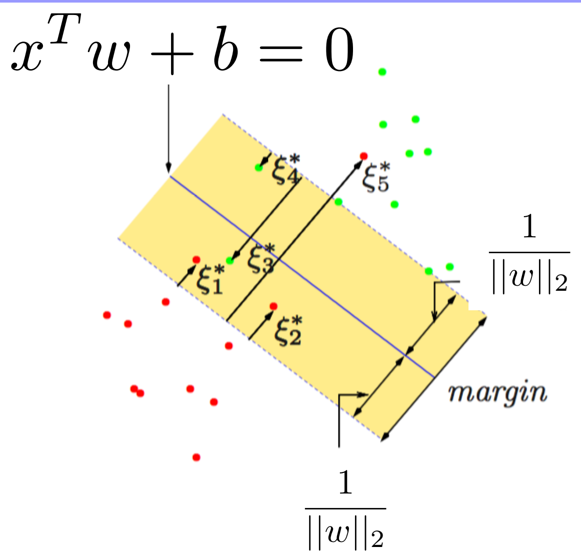
What if the data is not linearly separable?



If data is linearly separable:

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

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



If data is not linearly separable,

some points don't satisfy margin constraint:

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

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

$$\xi_i \geq 0, \sum_{j=1}^{n} \xi_j \leq \nu$$

SVM as penalization method

- Original quadratic program with linear constraints:

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

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

$$\xi_i \geq 0, \sum_{j=1}^{n} \xi_j \leq \nu$$

SVM as penalization method

- Original quadratic program with linear constraints:

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

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

$$\xi_i \geq 0, \sum_{j=1}^{n} \xi_j \leq \nu$$

- Using same constrained convex optimization trick as for lasso:

For any $\nu \geq 0$ there exists a $\lambda \geq 0$ such that the solution the following solution is equivalent:

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

SVMs: optimizing what?

SVM objective:

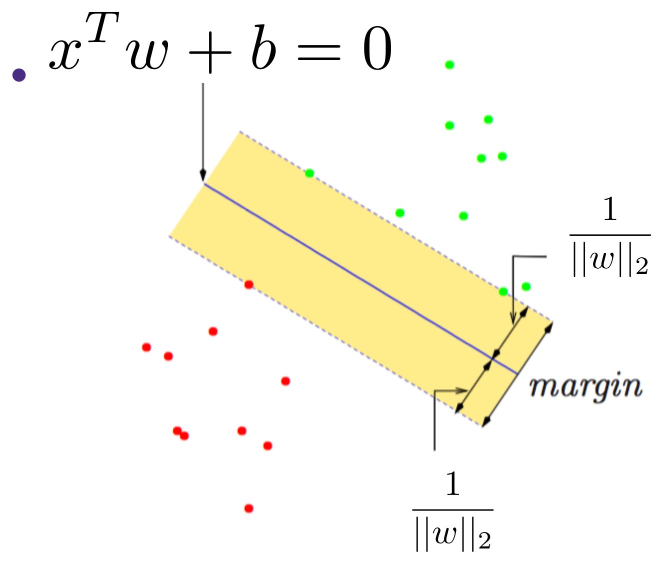
$$\boxed{\sum_{i=1}^{n} \max\{0, 1 - y_i(b + x_i^T w)\} + \lambda ||w||_2^2 \quad = \sum_{i=1}^{n} \ell_i(w, b)}$$

$$\nabla_w \ell_i(w, b) = \begin{cases} -x_i y_i + \frac{2\lambda}{n} w & \text{if } y_i(b + x_i^T w) < 1 \\ \frac{2\lambda}{n} & \text{otherwise} \end{cases}$$

$$\nabla_b \ell_i(w, b) = \begin{cases} -y_i & \text{if } y_i(b + x_i^T w) < 1 \\ 0 & \text{otherwise} \end{cases}$$

Kernel methods

What if the data is not linearly separable?



some points don't satisfy margin constraint:

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

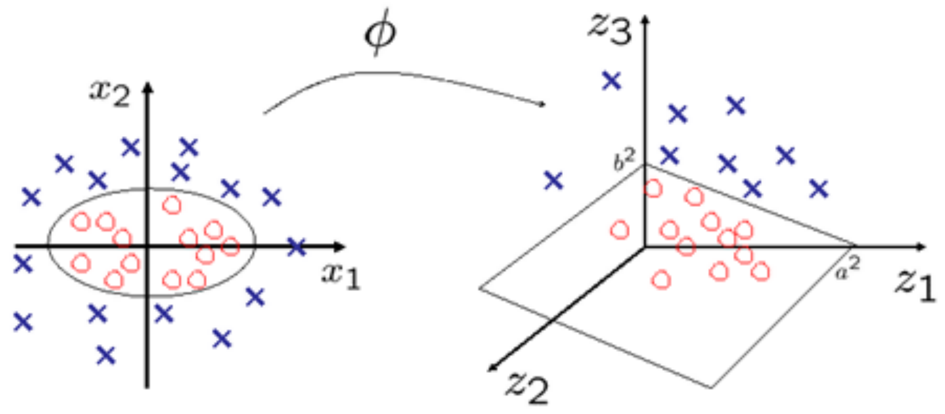
$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

Two options:

1. Introduce slack to this optimization problem
2. **Lift to higher dimensional space**

What if the data is not linearly separable?

Use features of features of features…



Creating Features

Transformed data:

$h : \mathbb{R}^d \to \mathbb{R}^p$ maps original features to a rich, possibly high-dimensional space

for d>1, generate

$$\{u_j\}_{j=1}^p \subset \mathbb{R}^d$$

$$h_j(x) = (u_j^T x)^2$$

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

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

in d=1:

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}$$

Creating Features

Transformed data:

$h : \mathbb{R}^d \to \mathbb{R}^p$ maps original features to a rich, possibly high-dimensional space

for d>1, generate

$$\{u_j\}_{j=1}^p \subset \mathbb{R}^d$$

$$h_j(x) = (u_j^T x)^2$$

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

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

in d=1: $$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}$$

Feature space can get really large really quickly!

Degree-d Polynomials

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 ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

So, if we can represent our algorithms/decision rules as dot products
and we can find a kernel for our feature map
then we can avoid explicitly dealing with ϕ(x).

Linear Regression as Kernels

Dot-product of polynomials

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree exactly d}$$

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

Dot-product of polynomials

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree exactly d}$$

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \qquad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

Feature space can get really large really quickly!

General d : Dimension of $\phi(u)$ is roughly p^d if $u \in \mathbb{R}^p$

Feature expansion can be written **implicitly** $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^p$

Examples of Kernels

- **Polynomials of degree exactly d**

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

- **Polynomials of degree up to d**

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$$

- **Gaussian (squared exponential) kernel**

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

- **Sigmoid**

$$K(u, v) = \tanh(\gamma \cdot u^T v + r)$$

The Kernel Trick

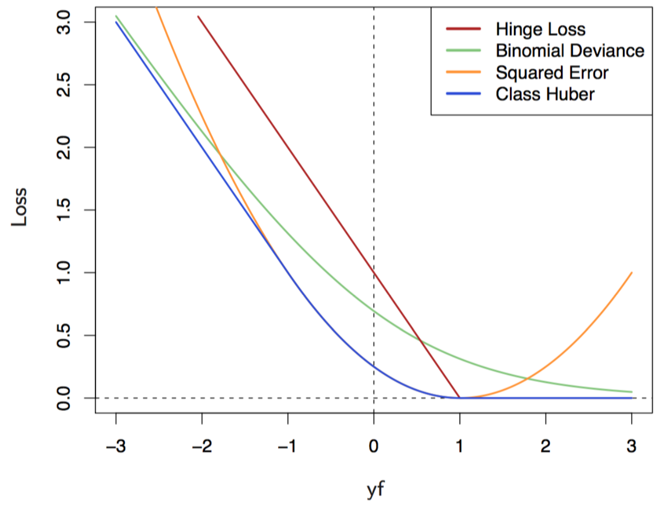
Pick a kernel K

For a linear predictor, show $w = \sum_i \alpha_i x_i$

Change loss function/decision rule to only access data through dot products

Substitute $K(x_i, x_j)$ for $x_i^T x_j$

Loss Functions



$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

- Loss functions: $$\sum_{i=1}^n \ell_i(w)$$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$

0/1 loss: $\ell_i(w) = \mathbb{I}[\text{sign}(y_i) \neq \text{sign}(x_i^T w)]$

Hinge Loss: $\ell_i(w) = \max\{0, 1 - y_i x_i^T w\}$

The Kernel Trick for regularized least squares

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

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

The Kernel Trick for regularized least squares

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

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

$$\begin{aligned}\widehat{\alpha} &= \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle \\ &= \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j) \\ &= \arg\min_\alpha ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha\end{aligned}$$

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

Why regularization?

Typically, $\mathbf{K} \succ 0$. What if $\lambda = 0$?

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

Why regularization?

Typically, $\mathbf{K} \succ 0$. What if $\lambda = 0$?

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

Unregularized kernel least squares can (over) fit **any data**!

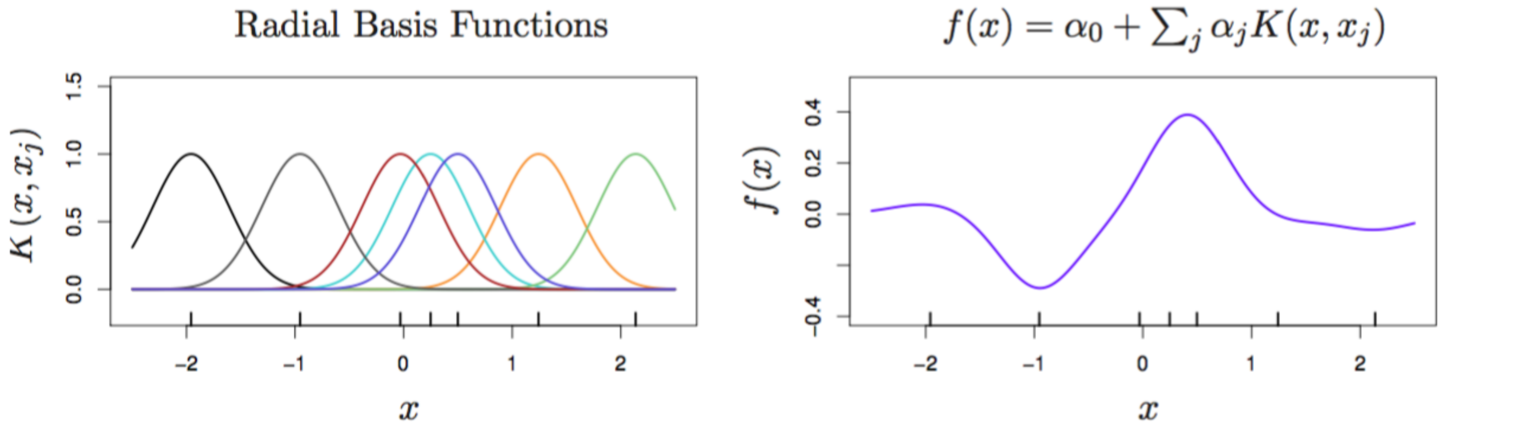
$$\widehat{\alpha} = \mathbf{K}^{-1}\mathbf{y}$$

The Kernel Trick for SVMs

RBF Kernel

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

This is like weighting “bumps” on each point

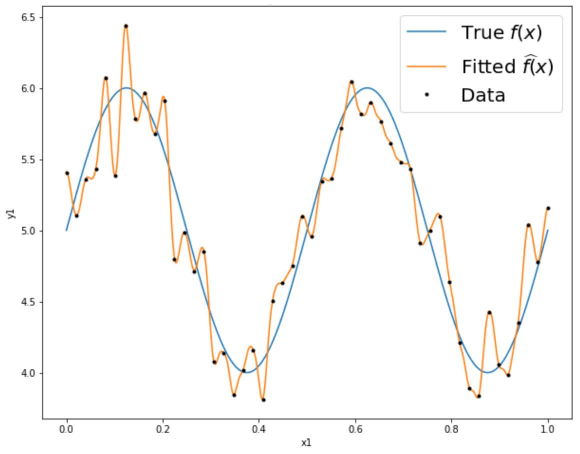


RBF Kernel

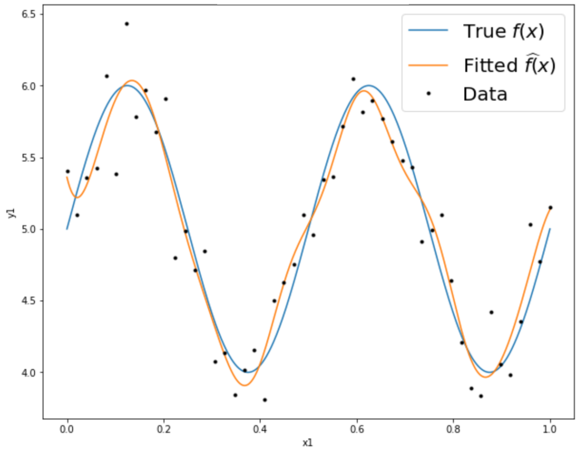
$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

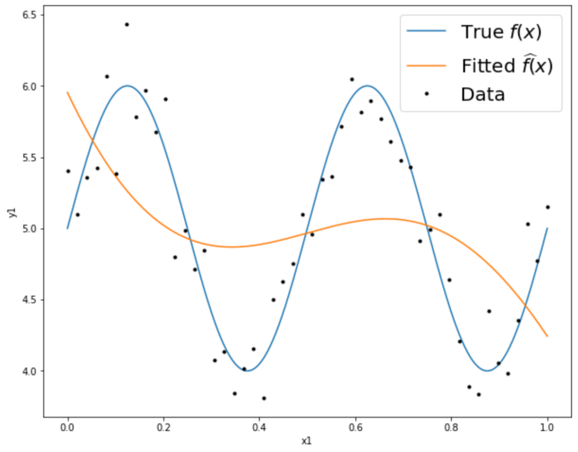
$\sigma = 10^{-2} \quad \lambda = 10^{-4}$



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



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



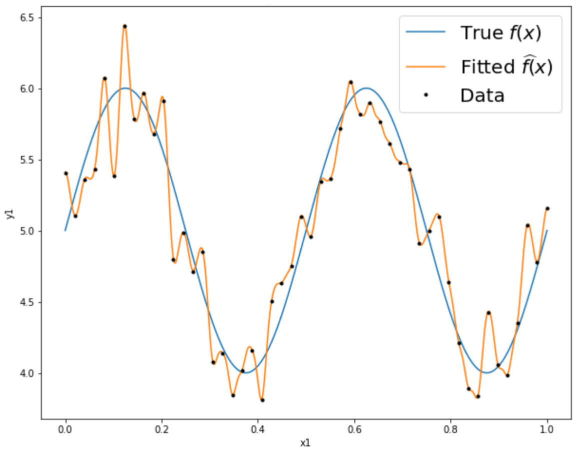
$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

RBF Kernel

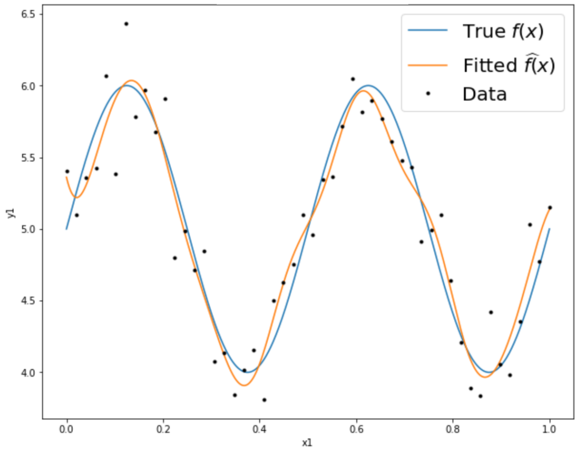
$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

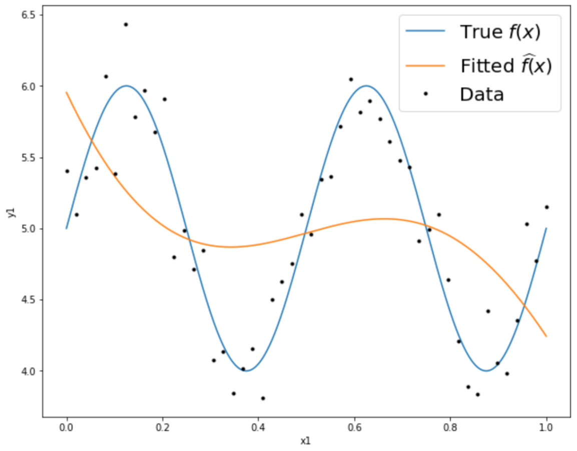
$\sigma = 10^{-2} \quad \lambda = 10^{-4}$



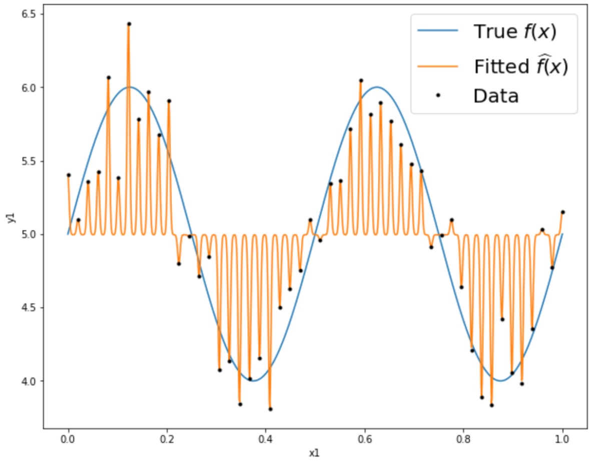
$\sigma = 10^{-1} \quad \lambda = 10^{-4}$



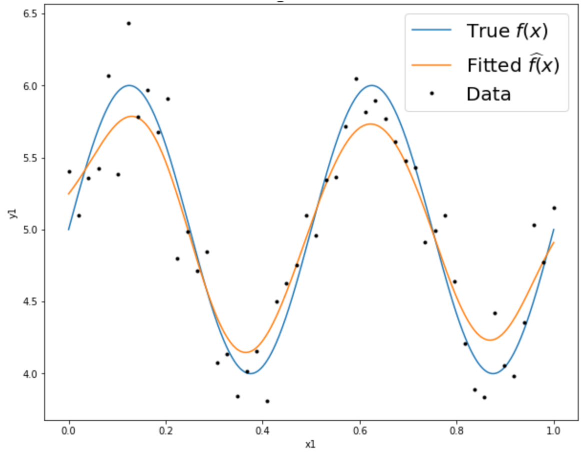
$\sigma = 10^{-0} \quad \lambda = 10^{-4}$



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



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



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