CSCI 567: Machine Learning

Vatsal Sharan

Spring 2026

Lecture 6, February 20

USC University of Southern California

Administrivia

- HW2 out, due in less than 1 week.
- Exam 1 in 2 weeks, more details on Ed

Recap

Regularized least squares

We looked at regularized least squares with non-linear basis:

$$\begin{aligned}\boldsymbol{w}^* &= \operatorname*{argmin}_{\boldsymbol{w}} F(\boldsymbol{w}) \\ &= \operatorname*{argmin}_{\boldsymbol{w}} \left(\|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2\right) \\ &= \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}\end{aligned}$$

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

This solution operates in the space $\mathbb{R}^M$ and M could be huge (and even infinite).

Regularized least squares solution: Another look

We realized that we can write,

$$\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

Thus the least square solution is **a linear combination of features of the datapoints**!

We calculated what $\boldsymbol{\alpha}$ should be,

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$$

where $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} \in \mathbb{R}^{n \times n}$ is the **kernel matrix**.

Kernel trick

The prediction of $\boldsymbol{w}^*$ on a new example $\boldsymbol{x}$ is

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Therefore, *only inner products in the new feature space matter!*

Kernel methods are exactly about computing inner products *without explicitly computing $\boldsymbol{\phi}$*. The exact form of $\boldsymbol{\phi}$ is inessential; *all we need to do is know the inner products $\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$.*

The kernel trick: Example 1

Consider the following polynomial basis $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^3$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

What is the inner product between $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{x}')$?

$$\begin{aligned} \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}') &= {x_1}^2{x_1'}^2 + 2x_1x_2x_1'x_2' + {x_2}^2{x_2'}^2 \\ &= (x_1x_1' + x_2x_2')^2 = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}')^2 \end{aligned}$$

Therefore, *the inner product in the new space is simply a function of the inner product in the original space*.

Kernel functions

Definition: a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a *kernel function* if there exists a function $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^M$ so that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}')$$

Popular kernels:

1. Polynomial kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}' + c)^M$$

   for $c \geq 0$ and M is a positive integer.

2. Gaussian kernel or Radial basis function (RBF) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0.$$

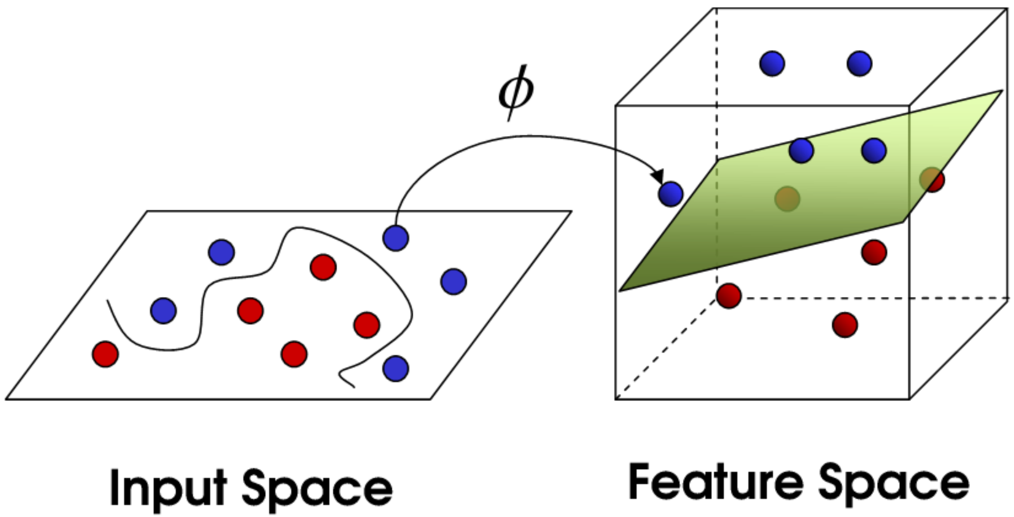
Prediction with kernels

As long as $\boldsymbol{w}^* = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$, prediction on a new example $\boldsymbol{x}$ becomes

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

This is known as a **non-parametric method**. Informally speaking, this means that there is no fixed set of parameters that the model is trying to learn (remember $\boldsymbol{w}^*$ could be infinite). Nearest-neighbors is another non-parametric method we have seen.

Classification with kernels



Similar ideas extend to the classification case, and we can predict using $\mathrm{sign}(\boldsymbol{w}^T\phi(\boldsymbol{x}))$.
Data may become linearly separable in the feature space!

We'll see this today.

Support vector machines (SVMs)

Why study SVM?

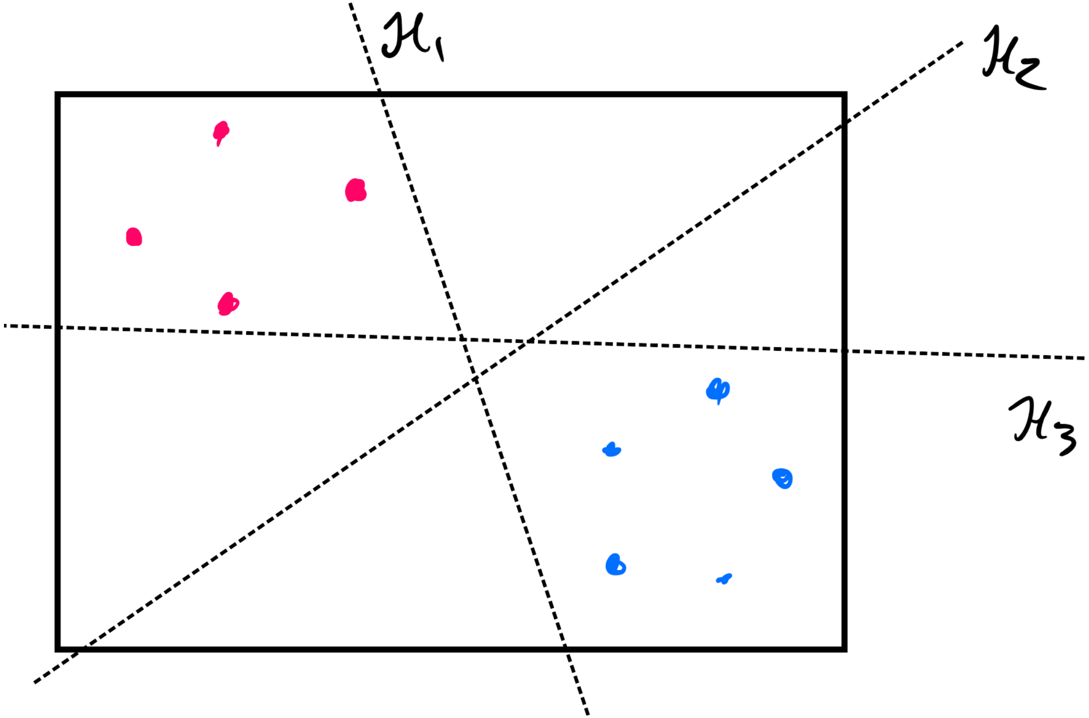
- One of the most commonly used classification algorithms
- Allows us to explore the concept of *margins* in classification
- Works well with the kernel trick
- Strong theoretical guarantees

We focus on **binary classification** here.

The *function class for SVMs is a linear function on a feature map ϕ applied to the datapoints*: $\mathrm{sign}(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b)$. Note, the bias term b is taken separately for SVMs, you'll see why.

Margins: separable case, geometric intuition

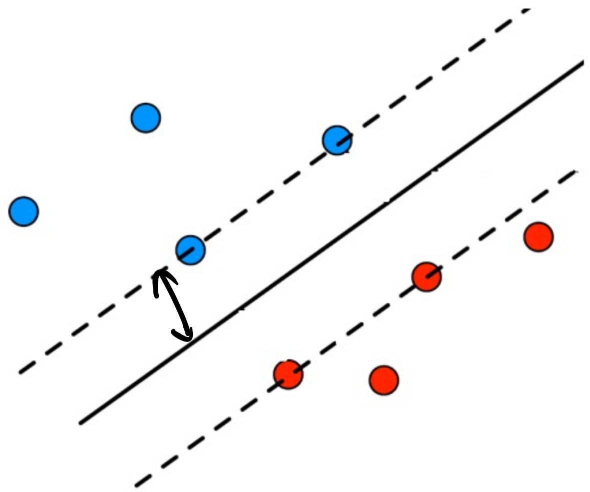
When data is **linearly separable**, there are infinitely many hyperplanes with zero training error:



Which one should we choose?

Margins: separable case, geometric intuition

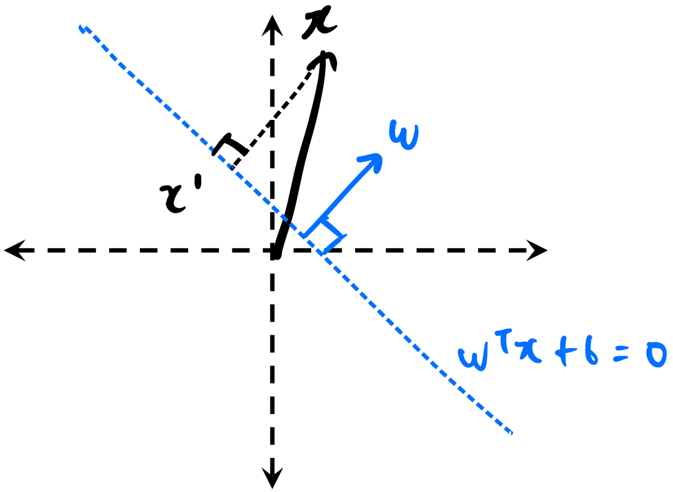
The further away the separating hyperplane is from the datapoints, the better.



Margin for linearly separable data: Distance from the hyperplane to the point closest to the hyperplane.

Formalizing geometric intuition: Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?



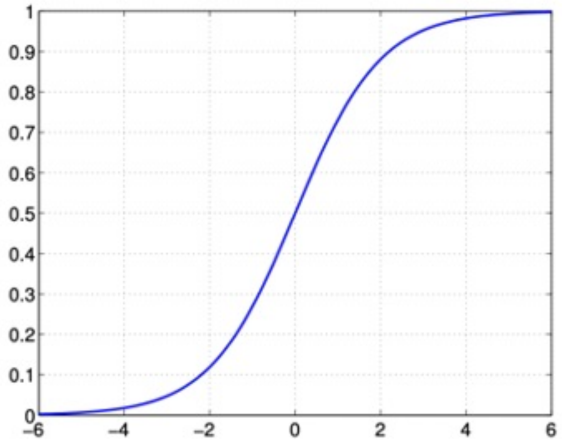
Assume the **projection** is $\boldsymbol{x}' = \boldsymbol{x} - \beta\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, then

$$0 = \boldsymbol{w}^{\mathrm{T}}\left(\boldsymbol{x} - \beta\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \beta\|\boldsymbol{w}\| + b \implies \beta = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b}{\|\boldsymbol{w}\|_2}.$$

Therefore the distance is $\|\boldsymbol{x} - \boldsymbol{x}'\|_2 = |\beta| = \frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b|}{\|\boldsymbol{w}\|_2}$.

For a hyperplane that correctly classifies $(\boldsymbol{x}, y)$, the distance becomes $\frac{y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b)}{\|\boldsymbol{w}\|_2}$.

Margins: functional motivation



$$\Pr[y = 1 \mid \boldsymbol{x}; \boldsymbol{w}] = \sigma(y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)) = \frac{1}{1+\exp(-y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b))}$$

If $y = 1$, want $w^T x + b >> 0$

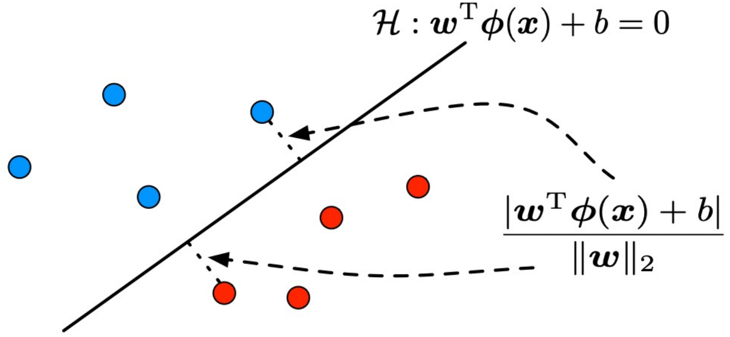
If $y = -1$, want $w^T x + b << 0$

$\therefore$ want $y(w^T x + b) >> 0$

Maximizing margin

Margin: the *smallest* distance from all training points to the hyperplane

$$\text{MARGIN OF } (\boldsymbol{w}, b) = \min_i \frac{y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|_2}$$



The intuition "**the further away the better**" translates to solving

$$\max_{\boldsymbol{w},b} \ \min_i \frac{y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|_2} = \max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)$$

Maximizing margin, rescaling

Note: rescaling $(\boldsymbol{w}, b)$ by multiplying both by some scalar does not change the hyperplane.

Decision boundary: $w^T \phi(x) + b = 0 \iff (10^6 w)^T \phi(x) + 10^6 b = 0$

multiply original (w, b) by $\frac{1}{\min_i (y_i (w^T \phi(x_i) + b))}$

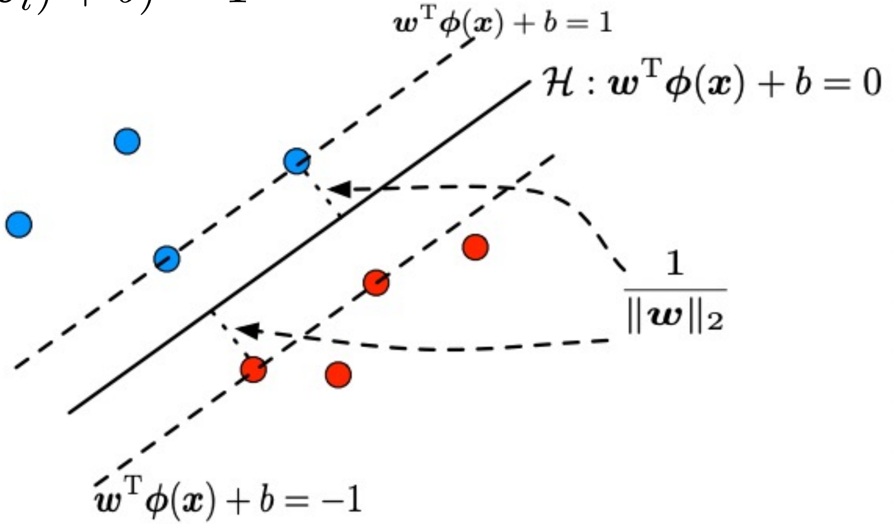
We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) = 1$

The margin then becomes

MARGIN OF $(\boldsymbol{w}, b)$

$$= \frac{1}{\|\boldsymbol{w}\|_2} \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)$$

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



SVM for separable data: "Primal" formulation

For a separable training set, we aim to solve

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) = 1$$

(this is non-convex)

This is equivalent to

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2$$
$$\text{s.t.} \quad y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1, \ \forall\, i \in [n]$$

Minimizing a convex function with convex constraints is convex

SVM is thus also called *max-margin* classifier. The constraints above are called *hard-margin* constraints.

General non-separable case

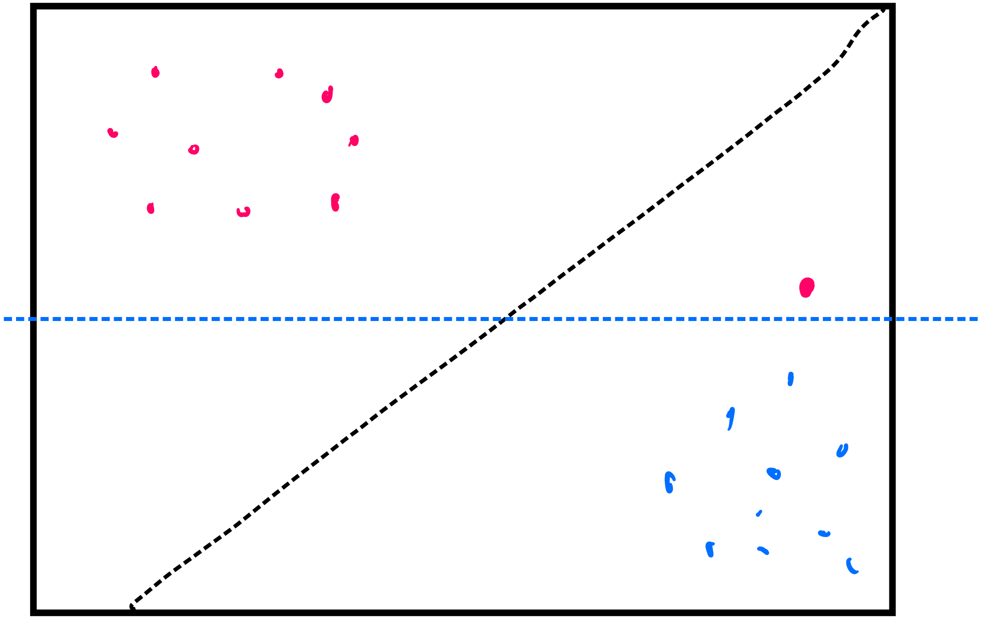
If data is not linearly separable, the previous constraint

$$y_i(\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i) + b) \geq 1, \quad \forall\, i \in [n]$$

Cannot even match $\mathrm{sign}(w^T\phi(x_i) + b) = y_i$ $\forall\, i \in [n]$

is obviously *not feasible*. What is the right thing to do?

Forcing classifier to classify all datapoints correctly may be bad



General non-separable case

If data is not linearly separable, the previous constraint $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)+b) \geq 1, \ \forall\, i \in [n]$ is not feasible. And more generally, forcing classifier to always classify all datapoints correctly may not be the best idea.

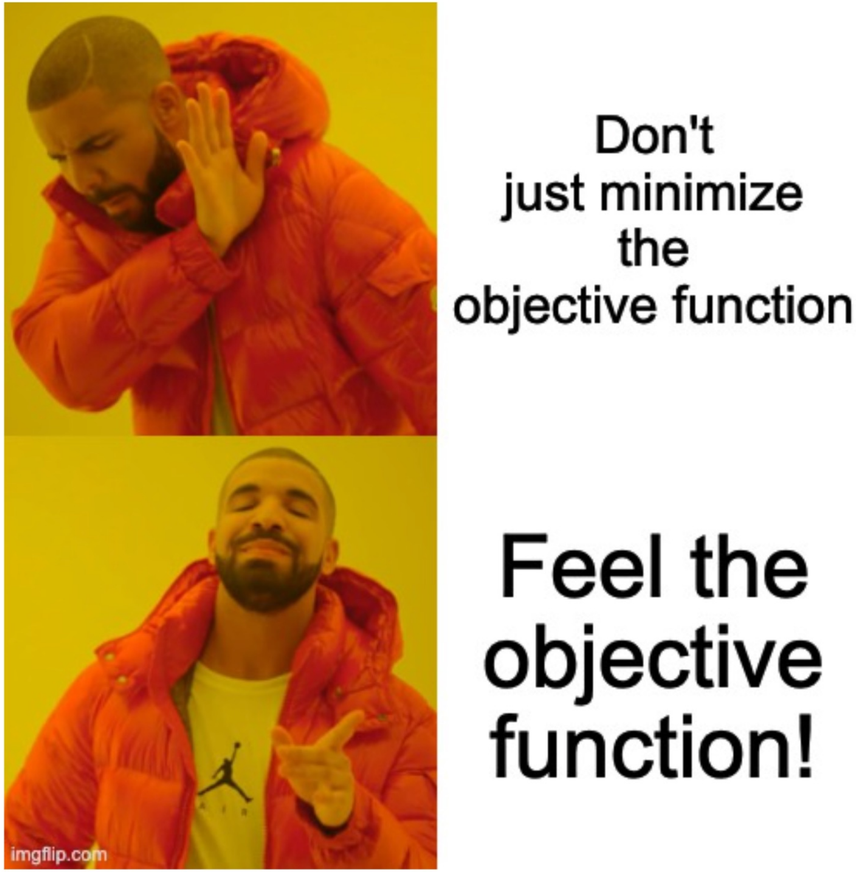
To deal with this issue, we relax the constraints to ℓ_1 **norm soft-margin** constraints:

$$\begin{aligned} & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n] \\ \Longleftrightarrow\ & 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \leq \xi_i, \quad \forall\, i \in [n] \end{aligned}$$

where we introduce **slack variables** $\xi_i \geq 0$.

Recall the hinge loss: $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$. In our case, $z = y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b)$.

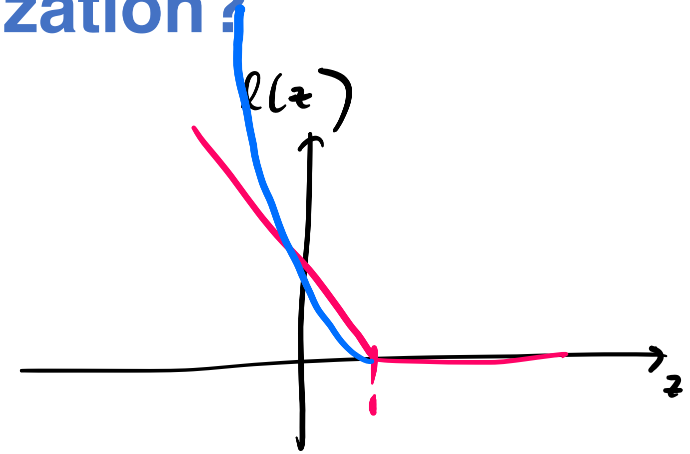
Aside: Why ℓ_1 penalization?



Aside: Why ℓ_1 penalization?

hinge loss $\ell(z) = \max(0, 1-z)$

squared hinge loss $\ell(z) = (\max(0, 1-z))^2$



What would be different?

x^2 grows much faster than x

squared hinge loss would really penalize getting some predictions very wrong

Aside: Why ℓ_1 penalization?

Because of this, absolute value loss can be more robust to outliers in data compared to squared loss.

A 1-D regression example: mean vs. median

If I have $x_1, x_2, \ldots, x_n$:

What is $w^*_{\ell_2} = \arg\min_w \sum_i (x_i - w)^2$? $\quad w^*_{\ell_2} = \frac{\sum_i x_i}{n}$

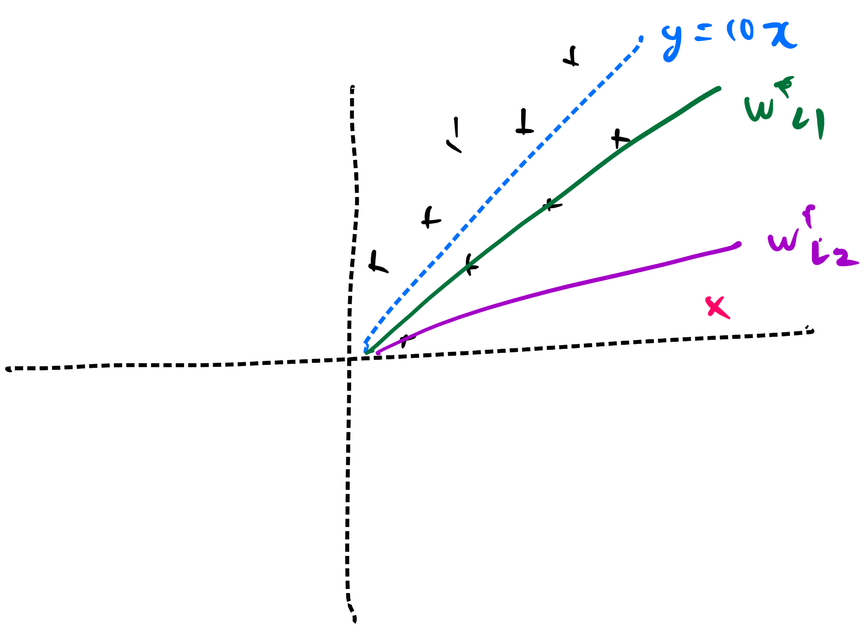
What is $w^*_{\ell_1} = \arg\min_w \sum_i |x_i - w|$? $\quad w^*_{\ell_1} = \text{median}(x_1, \ldots, x_n)$

Median is more robust to outliers than mean.

Aside: Why ℓ_1 penalization?

For 1-D regression

$y = 10x$ + noise



consider

$$w^*_{\ell_2} = \arg\min \sum_i (y_i - w x_i)^2$$

$$w^*_{\ell_1} = \arg\min \sum_i |y_i - w x_i|$$

Back to SVM: General non-separable case

If data is not linearly separable, the constraint $y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1, \ \forall\, i \in [n]$ is not feasible.

To deal with this issue, we relax the constraints to ℓ_1 **norm soft-margin** constraints:

$$y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \ \forall\, i \in [n]$$

where we introduce **slack variables** $\xi_i \geq 0$.

SVM General Primal Formulation

We want ξ_i to be as small as possible. The objective becomes

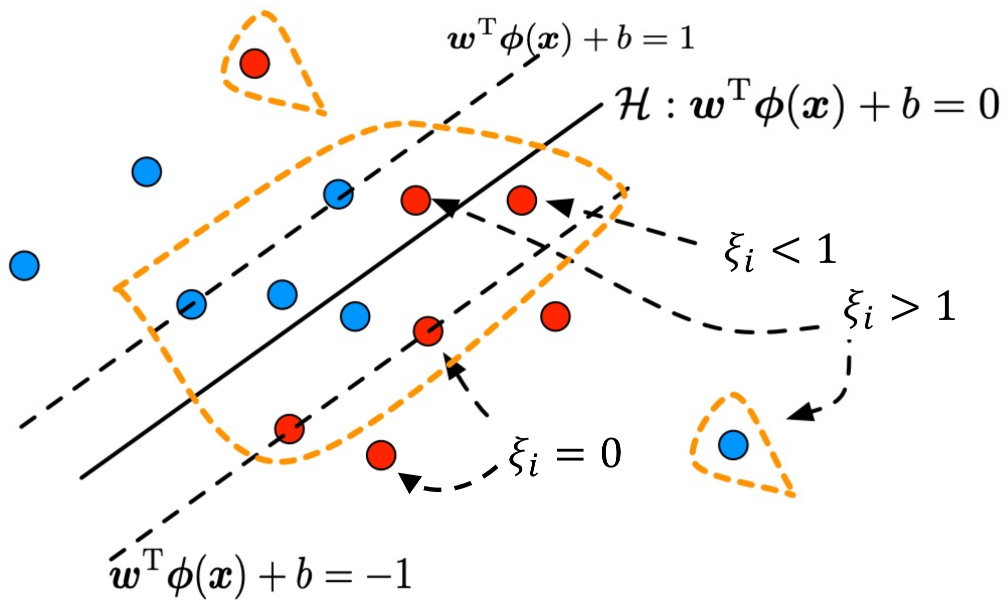
$$\min_{\boldsymbol{w}, b, \{\xi_i\}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C \sum_i \xi_i$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n]$$

$$\xi_i \geq 0, \quad \forall\, i \in [n]$$

where C is a hyperparameter to balance the two goals.

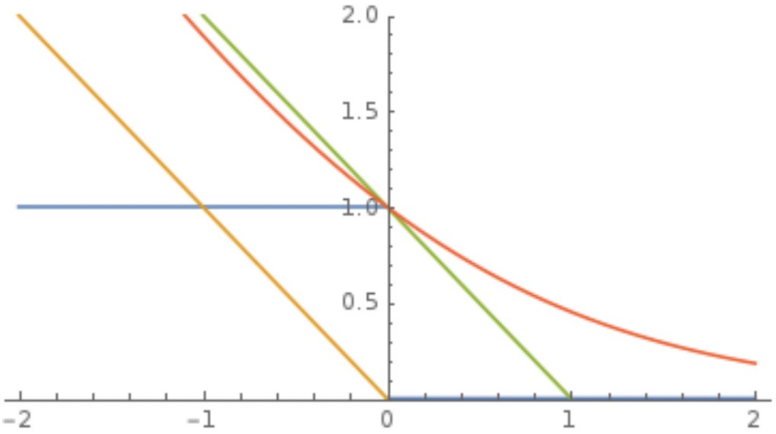
Understanding the slack conditions



- when $\xi_i^* = 0$, point is classified correctly and satisfies large margin constraint.
- when $\xi_i^* < 1$, point is classified correctly but does not satisfy large margin constraint.
- when $\xi_i^* > 1$, point is misclassified.

Primal formulation: Another view

In one sentence: **linear model with ℓ_2 regularized hinge loss**. Recall:



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\} \rightarrow$ Perceptron
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z)) \rightarrow$ logistic regression
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\} \rightarrow$ **SVM**

Primal formulation: Another view

For a linear model $(\boldsymbol{w}, b)$, this means

$$\min_{\boldsymbol{w},b} \sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_i \in \{-1, +1\}$
- a nonlinear mapping ϕ is applied
- the bias/intercept term b is used explicitly (why is this done?)

What is the relation between this formulation and the one which we just saw before?

Equivalent forms

The formulation

$$\min_{\boldsymbol{w},b,\{\xi_i\}} \quad C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \leq \xi_i, \quad \forall\, i \in [n]$$

$$\xi_i \geq 0, \quad \forall\, i \in [n]$$

In order to minimize $\sum_i \xi_i$

we should set ξ_i to be as small as possible:

is equivalent to

$$\min_{\boldsymbol{w},b,\{\xi_i\}} \quad C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} = \xi_i, \quad \forall\, i \in [n]$$

Equivalent forms

$$\min_{\boldsymbol{w},b,\{\xi_i\}} \quad C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} = \xi_i, \quad \forall\, i \in [n]$$

is equivalent to

$$\min_{\boldsymbol{w},b}\ C\sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

and

$$\min_{\boldsymbol{w},b} \sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

with $\lambda = 1/C$. *This is exactly minimizing ℓ_2 regularized hinge loss!*

Optimization

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_i\}} \quad & C \sum_i \xi_i + \frac{1}{2} \|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n] \\ & \xi_i \geq 0, \quad \forall\, i \in [n]. \end{aligned}$$

- it is a convex (in fact, a **quadratic**) problem
- thus can apply any convex optimization algorithms, e.g. SGD
- there are **more specialized and efficient** algorithms
- but usually we apply kernel trick, which requires solving the *dual problem*

SVMs:
Dual formulation
& Kernel trick

Recall SVM formulation for separable case,

$$\min_{w,b} \quad \frac{1}{2} \|w\|_2^2$$

$$\text{s.t.} \quad y_i (w^T \phi(x_i) + b) \geq 1 \quad \forall i \in [n]$$

Can we use the kernel trick??

Can we show that w^* is a linear combination of feature vectors $\phi(x_i)$??

How did we show this for regularized least squares?

By setting the gradient of $F(\boldsymbol{w}) = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$ to be $\mathbf{0}$:

$$\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{w}^* - \boldsymbol{y}) + \lambda\boldsymbol{w}^* = \mathbf{0}$$

we know

$$\boldsymbol{w}^* = \frac{1}{\lambda}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}^*) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

Thus the least square solution is **a linear combination of features of the datapoints**!

Is optimal predictor linear combination of feature vectors?

$$F(\boldsymbol{w}) = \underbrace{\sum_{i=1}^{n} \max\left\{0,\ 1 - y_i\left(\mathbf{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_i) + b\right)\right\}}_{\text{hinge loss}} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

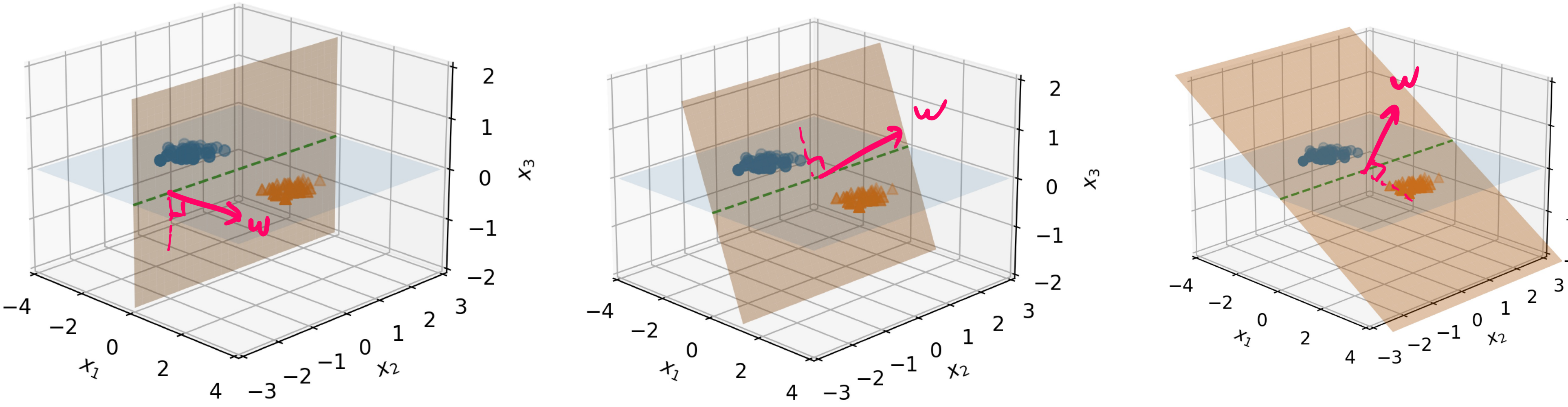
This is a convex problem. Therefore, gradient descent (GD) will find a minimizer with any initialization (for some suitable step size).

Exercise: Suppose we do GD with initialization $\boldsymbol{w}^{(0)} = \mathbf{0}$. Show that gradient descent iterates on $F(\boldsymbol{w})$ at any time step t satisfy $\boldsymbol{w}^{(t)} = \sum_{i=1}^{n} \alpha_i^{(t)} y_i \boldsymbol{\phi}(\boldsymbol{x}_i)$ for some $\alpha_i^{(t)}$.

For the SVM problem, $w^* = \sum_{i=1}^{n} \alpha_i^* y_i \phi(x_i)$

We can also geometrically understand why $\boldsymbol{w}^*$ should lie in the span of the data:

-ve +ve



$w_3 =$ $w_3 > 0$ $w_3 >> 0$

all datapoints have $x_3 = 0$

Kernelizing SVM

If $w = \sum_{i=1}^{n} \alpha_i y_i \phi(x_i)$, how can we use this?

$$\min_{w,b} \quad \frac{1}{2}\|w\|_2^2$$

s.t. $y_j(w^{\mathrm{T}}\phi(x_j) + b) \geq 1, \quad \forall\, j \in [n].$

$$\rightarrow \quad \min_{\alpha, b} \quad \frac{1}{2} \Big\| \sum_i \alpha_i y_i \phi(x_i) \Big\|_2^2$$

$$\text{s.t.} \quad y_j \Big(\Big(\sum_i \alpha_i y_i \phi(x_i) \Big)^T \phi(x_j) + b \Big) \geq 1$$

This is equivalent to

$$\min_{\alpha, b} \quad \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

$$\text{s.t.} \quad y_j \Big(\sum_i \alpha_i y_i \phi(x_i)^T \phi(x_j) + b \Big) \geq 1 \quad \forall\, i \in [n]$$

SVM: Dual form for separable case

With some optimization theory (Lagrange duality, not covered in this class), we can show this is equivalent to,

$$\max_{\{\alpha_i\}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_j)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0, \quad \forall\, i \in [n]$$

SVM: Dual form for separable case

Using the kernel function k for the mapping $\boldsymbol{\phi}$, we can kernelize this!

$$\max_{\{\alpha_i\}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad \alpha_i \geq 0, \quad \forall\, i \in [n]$$

No need to compute $\phi(\boldsymbol{x})$. This is also a **quadratic program** and many efficient optimization algorithms exist.

SVM: Dual form for the general case

For the primal for the general (non-separable) case:

$$\begin{aligned}
\min_{\boldsymbol{w}, b, \{\xi_i\}} \quad & C \sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\
\text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n] \\
& \xi_i \geq 0, \quad \forall\, i \in [n].
\end{aligned}$$

The dual is very similar,

$$\begin{aligned}
\max_{\{\alpha_i\}} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
\text{s.t.} \quad & \sum_{i=1}^n \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C, \quad \forall\, i \in [n].
\end{aligned}$$

Prediction using SVM

How do we predict given the solution $\{\alpha_i^*\}$ to the dual optimization problem?

Remember that,

$$\boldsymbol{w}^* = \sum_i \alpha_i^* y_i \boldsymbol{\phi}(\boldsymbol{x}_i) = \sum_{i:\alpha_i^*>0} \alpha_i^* y_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

A point with $\alpha_i^* > 0$ is called a "**support vector**". Hence the name SVM.

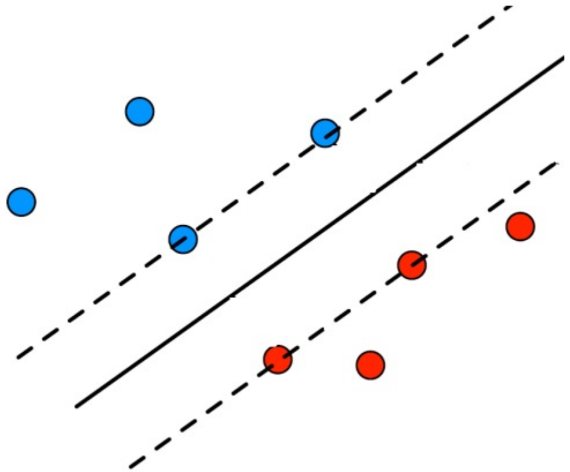
To make a prediction on any datapoint $\boldsymbol{x}$,

$$\begin{aligned}\text{sign}\left(\boldsymbol{w}^{*\mathrm{T}}\phi(\boldsymbol{x}) + b^*\right) &= \text{sign}\left(\sum_{i:\alpha_i^*>0} \alpha_i^* y_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) + b^*\right) \\ &= \text{sign}\left(\sum_{i:\alpha_i^*>0} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right).\end{aligned}$$

All we need now is to identify b^*.

How to compute bias term b^*?

We will only consider the separable case (general case is not too different):

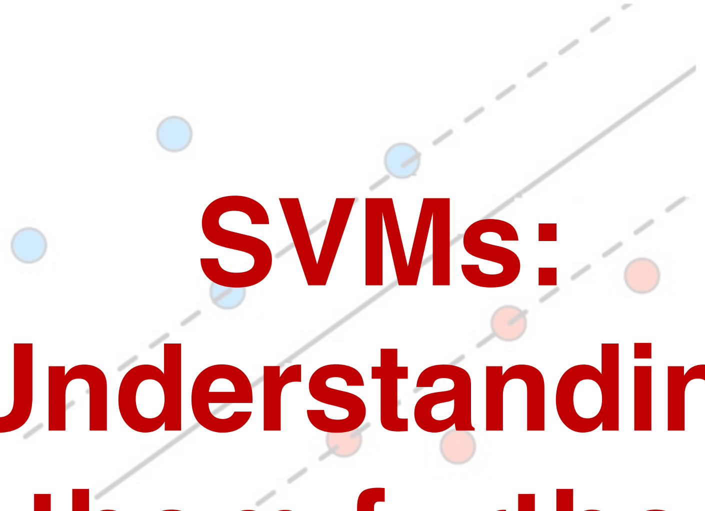


It can be shown (we will not cover in class), that in the separable case the support vectors lie on the margin.

$$y_i({w^*}^T\phi(x_i)+b^*)=1 \implies y_i^2({w^*}^T\phi(x_i)+b^*)=y_i$$

$$\implies (w^*)^T\phi(x_i)+b^*=y_i$$

$$\implies b^*=y_i-{w^*}^T\phi(x_i)=y_i-\sum_{j:\alpha_j^*>0}\alpha_j^* y_j k(x_j,x_i), \text{ for any } i \text{ s.t. } \alpha_i^*>0$$



SVMs: Understanding them further

Understanding support vectors

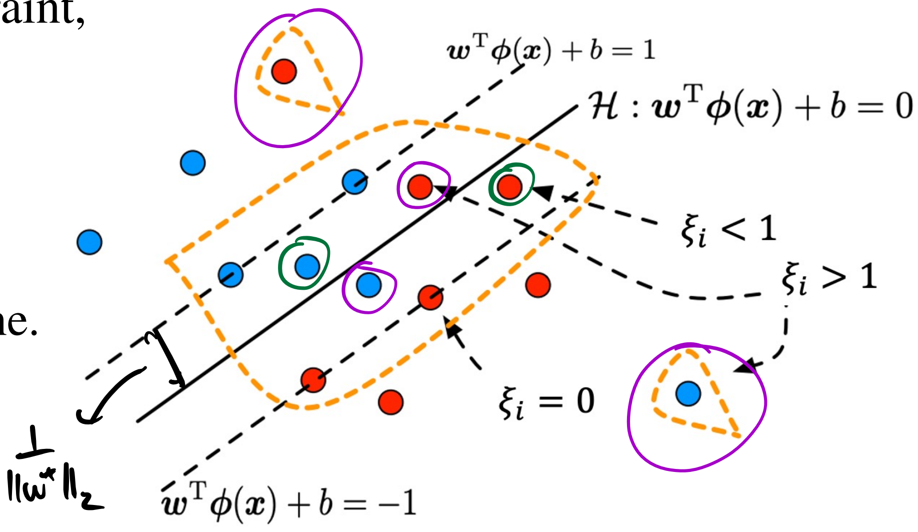
Support vectors are $\boldsymbol{\phi}(\boldsymbol{x}_i)$ such that $\alpha_i^* > 0$.

They are the set of points which satisfy one of the following:
(1) they are tight with respect to the large margin contraint,
(2) they do not satisfy the large margin contraint,
(3) they are misclassified.

- when $\xi_i^* = 0$, $y_i({\boldsymbol{w}^*}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b^*) = 1$,
and thus the point is $1/\|\boldsymbol{w}^*\|_2$ away from the hyperplane.

- when $\xi_i^* < 1$, the point is classified correctly
but does not satisfy the large margin constraint.

- when $\xi_i^* > 1$, the point is misclassified.



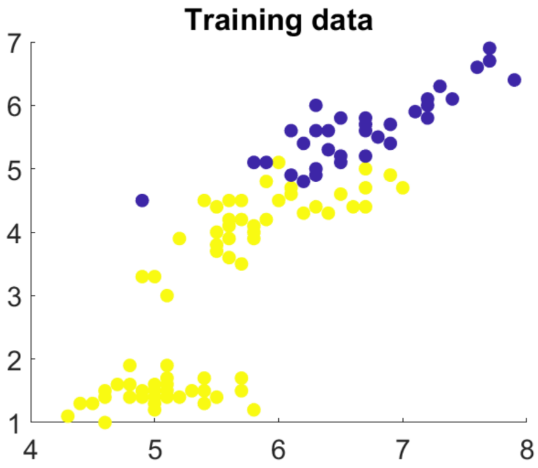
Support vectors (circled with the orange line) are the only points that matter!

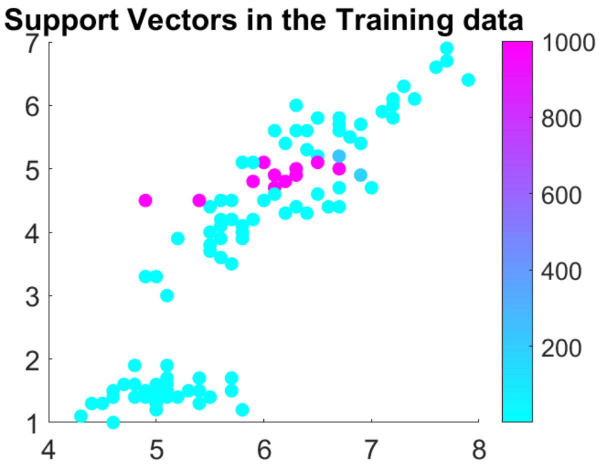
Understanding support vectors

One potential drawback of kernel methods: **non-parametric**, need to potentially keep all the training points.

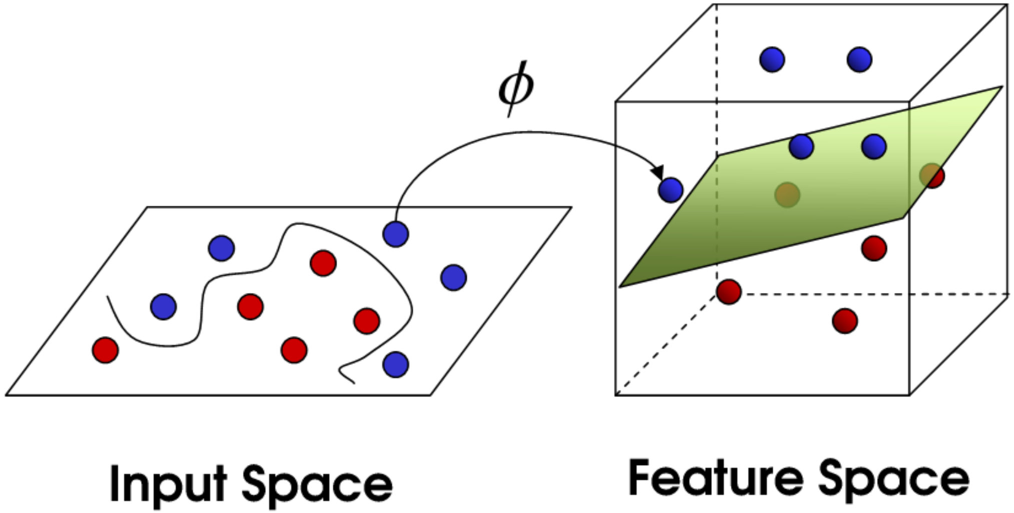
$$\operatorname{sign}\left(\boldsymbol{w}^{*\mathrm{T}}\phi(\boldsymbol{x})+b^*\right)=\operatorname{sign}\left(\sum_{i=1}^{n}\alpha_i^* y_i k(\boldsymbol{x}_i,\boldsymbol{x})+b^*\right).$$

For SVM though, very often #support vectors $= \left|\{i : \alpha_i^* > 0\}\right| \ll n$.



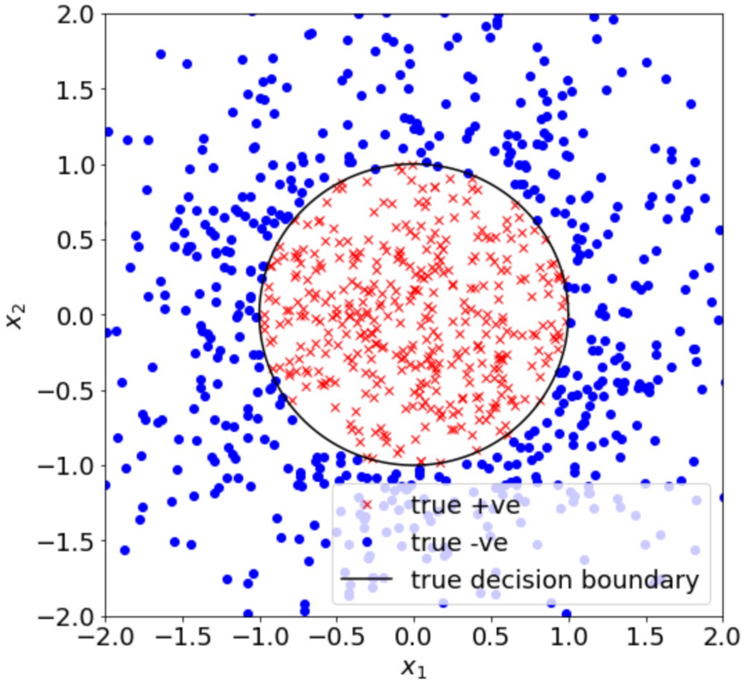


Examining the effect of kernels



Data may become linearly separable when lifted to the high-dimensional feature space!

Polynomial kernel: example



Switch to Colab

Gaussian kernel: example

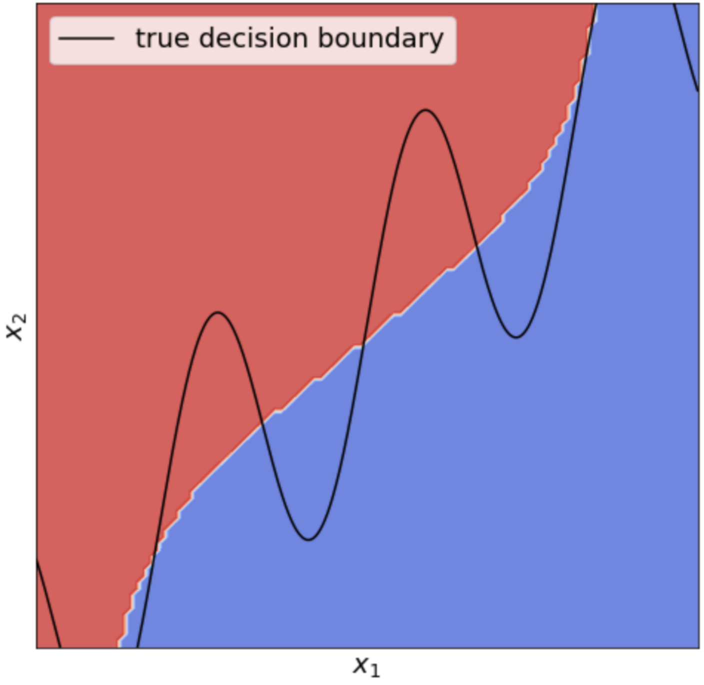
Gaussian kernel or Radial basis function (RBF) kernel

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

for some $\sigma > 0$. This is also parameterized as,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2\right)$$

for some $\gamma > 0$.



What does the decision boundary look like?
What is the effect of γ?

Note that the prediction is of the form

$$\operatorname{sign}\left(\boldsymbol{w}^{*\mathrm{T}}\phi(\boldsymbol{x}) + b^*\right) = \operatorname{sign}\left(\sum_{i:\alpha_i^*>0} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b^*\right).$$

Switch to Colab

SVM: Summary of mathematical forms

SVM: **max-margin linear classifier**

Primal (equivalent to minimizing ℓ_2 regularized hinge loss):

$$\begin{aligned}
\min_{\boldsymbol{w}, b, \{\xi_i\}} \quad & C \sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\
\text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n] \\
& \xi_i \geq 0, \quad \forall\, i \in [n].
\end{aligned}$$

Dual (kernelizable, reveals what training points are support vectors):

$$\begin{aligned}
\max_{\{\alpha_i\}} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_j) \\
\text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C, \quad \forall\, i \in [n].
\end{aligned}$$

Multiclass classification

Setup

Recall the setup:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^d$
- output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- goal: learn a mapping $f : \mathbb{R}^d \to [\mathsf{C}]$

Examples:

- recognizing digits ($\mathsf{C} = 10$) or letters ($\mathsf{C} = 26$ or 52)
- predicting weather: sunny, cloudy, rainy, etc
- predicting image category: ImageNet dataset ($\mathsf{C} \approx 20K$)

Linear models: Binary to multiclass

Step 1: *What should a linear model look like for multiclass tasks?*

Note: a linear model for binary tasks (switching from $\{-1,+1\}$ to $\{1,2\}$)

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0 \\ 2 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} < 0 \end{cases}$$

can be written as

$$\begin{aligned} f(\boldsymbol{x}) &= \begin{cases} 1 & \text{if } \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \geq \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} \\ 2 & \text{if } \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} > \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \end{cases} \\ &= \operatorname*{argmax}_{k\in\{1,2\}} \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x} \end{aligned}$$

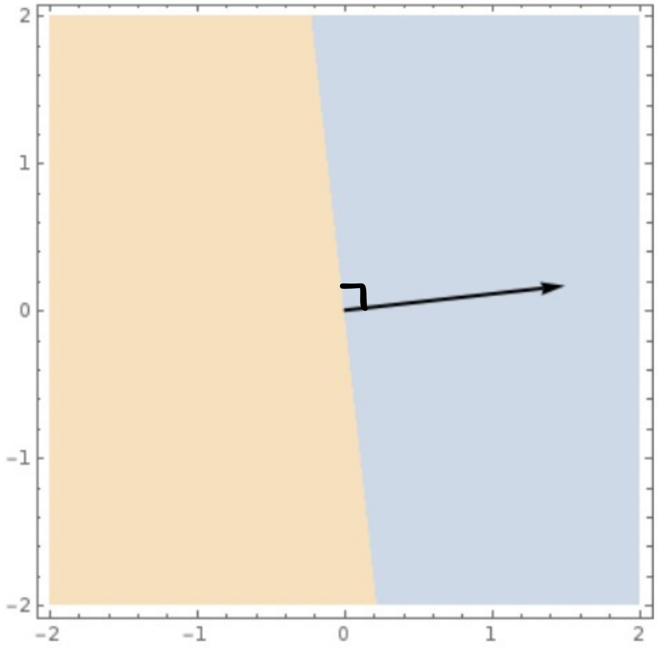
w_1 & w_2 s.t.

$w_1 - w_2 = w$

for any $\boldsymbol{w}_1, \boldsymbol{w}_2$ s.t. $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$

Think of $\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$ as **a score for class k**.

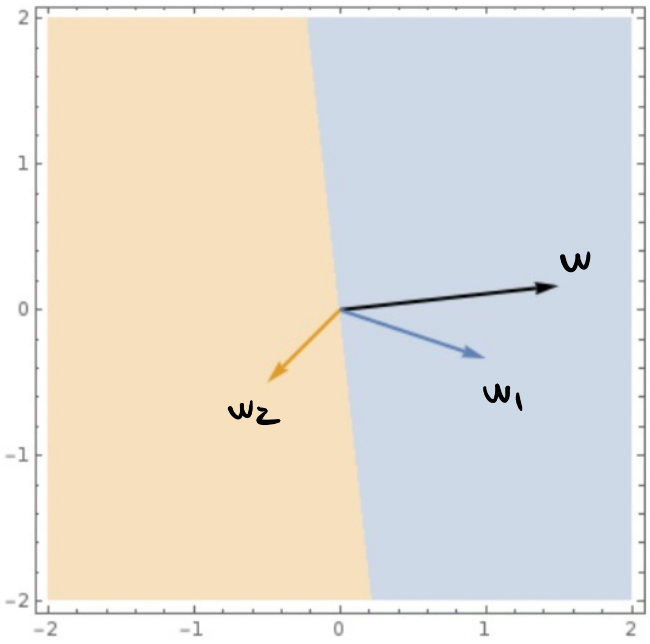
Linear models: Binary to multiclass



$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6})$$

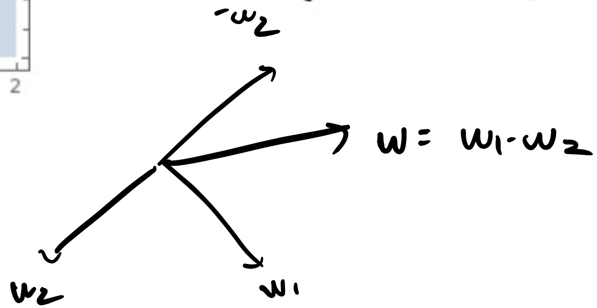
- Blue class:
  $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0\}$
- Orange class:
  $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} < 0\}$

Linear models: Binary to multiclass

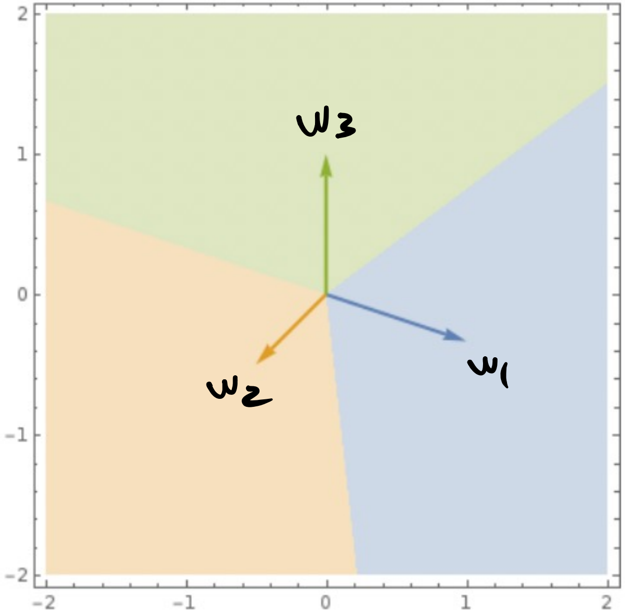


$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6}) = \boldsymbol{w}_1 - \boldsymbol{w}_2$$
$$\boldsymbol{w}_1 = (1, -\tfrac{1}{3})$$
$$\boldsymbol{w}_2 = (-\tfrac{1}{2}, -\tfrac{1}{2})$$

- Blue class:
  $\{\boldsymbol{x} : 1 = \mathrm{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \mathrm{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$



Linear models: Binary to multiclass



$$\boldsymbol{w}_1 = (1, -\tfrac{1}{3})$$
$$\boldsymbol{w}_2 = (-\tfrac{1}{2}, -\tfrac{1}{2})$$
$$\boldsymbol{w}_3 = (0, 1)$$

- Blue class:
  $\{\boldsymbol{x} : 1 = \operatorname{argmax}_k \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \operatorname{argmax}_k \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Green class:
  $\{\boldsymbol{x} : 3 = \operatorname{argmax}_k \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

Function class: Linear models for multiclass classification

$$\mathcal{F} = \left\{ f(\boldsymbol{x}) = \operatorname*{argmax}_{k \in [\mathsf{C}]} \; \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\mathsf{C}} \in \mathbb{R}^d \right\}$$

$$= \left\{ f(\boldsymbol{x}) = \operatorname*{argmax}_{k \in [\mathsf{C}]} \; (\boldsymbol{W}\boldsymbol{x})_k \mid \boldsymbol{W} \in \mathbb{R}^{\mathsf{C} \times d} \right\}$$

Next, lets try to generalize the loss functions. Focus on the logistic loss today.