CSCI 567: Machine Learning

Vatsal Sharan

Fall 2022

Lecture 5, Sep 22

USC University of Southern California

Administrivia

- HW2 due in about a week.
- Quiz 1 in 2 weeks.

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 $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^3$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(\begin{array}{c} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{array} \right)$$

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

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

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 $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is called a *kernel function* if there exists a function $\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^M$ 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

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

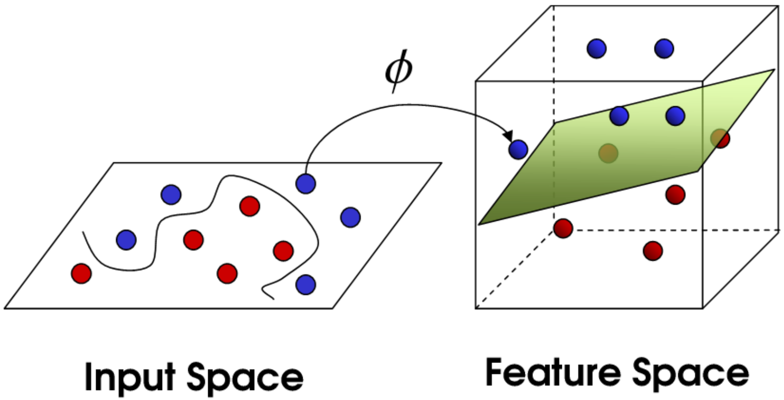
Prediction with kernels

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

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

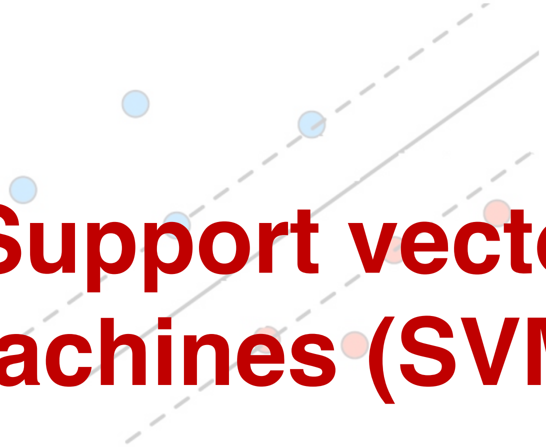
This is known as a **non-parametric method**. Informally speaking, this means that there is no fixed set of parameters that the model is trying to learn (remember $\boldsymbol{w}^*$ could be infinite). Nearest-neighbors is another non-parametric method we have seen.

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)

1.1 Why study SVM?

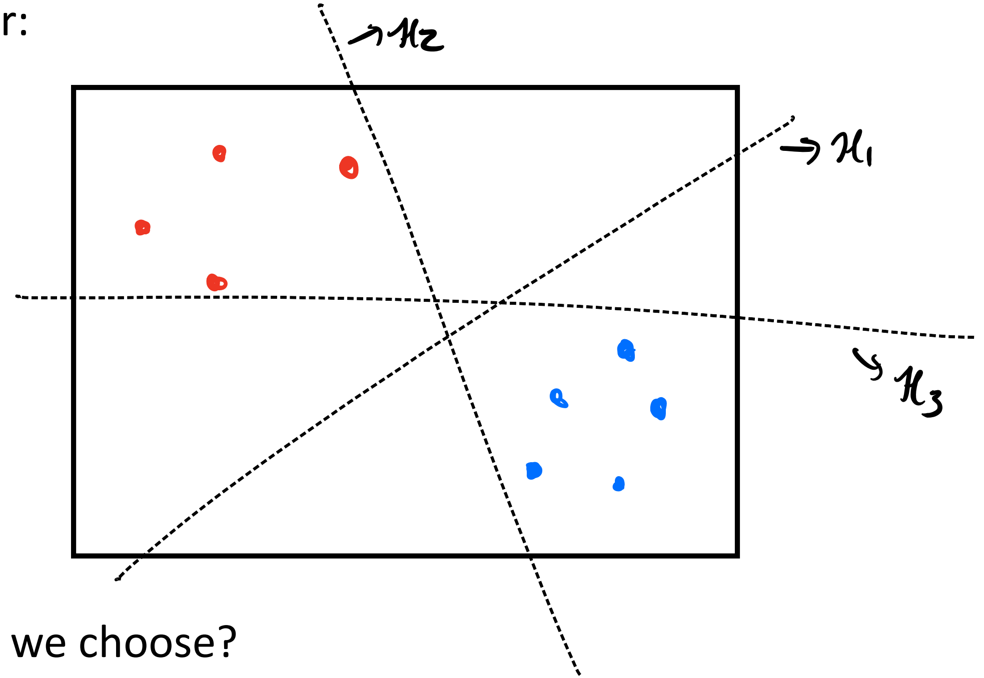
- One of the most commonly used classification algorithms
- Allows us to explore the concept of *margins* in classification
- Works well with the kernel trick
- Strong theoretical guarantees

We focus on **binary classification** here.

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

1.2 Margins: separable case, geometric intuition

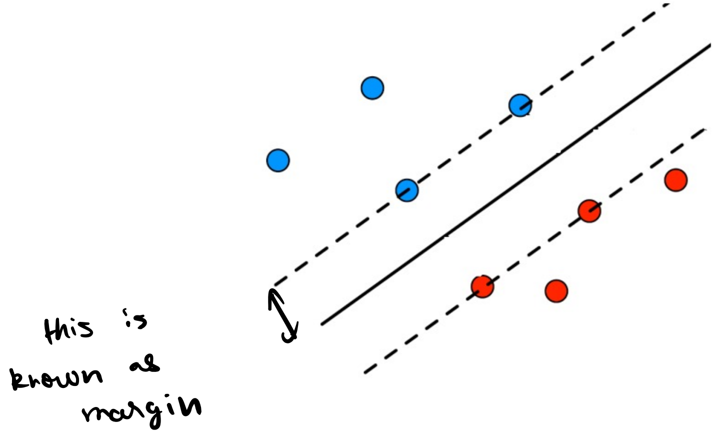
When data is **linearly separable**, there are infinitely many hyperplanes with zero training error:



Which one should we choose?

1.2 Margins: separable case, geometric intuition

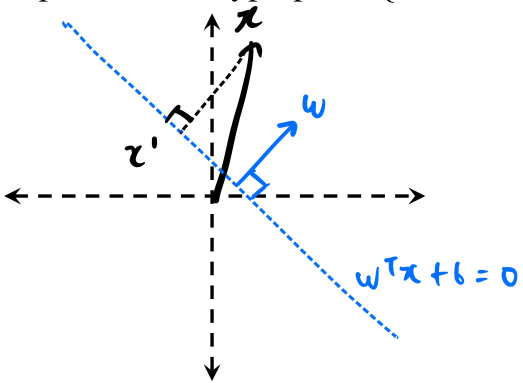
The further away the separating hyperplane is from the datapoints, the better.



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

1.2 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

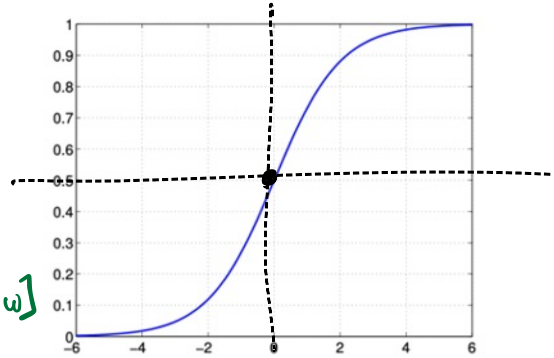
$$0 = \boldsymbol{w}^{\mathrm{T}}\left(\boldsymbol{x} - \beta \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \beta\|\boldsymbol{w}\| + b \implies \beta = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b}{\|\boldsymbol{w}\|_2}.$$

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

$\operatorname{sign}(w^T x + b) = y$

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

1.2 Margins: functional motivation



This should be $\Pr[y=1 \mid x; w]$

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

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

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

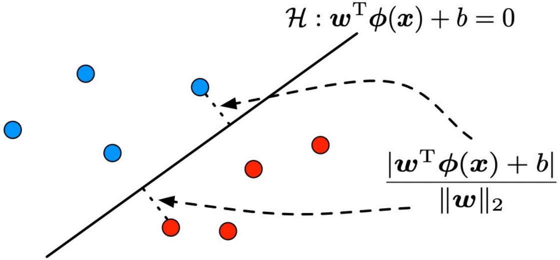
$\therefore$ want $y(w^T x + b) \gg 0$

1.3 Maximizing margin

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

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

data
(x_i, y_i), $i \in \{1, \dots, n\}$



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

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

1.3 Maximizing margin, rescaling

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

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

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

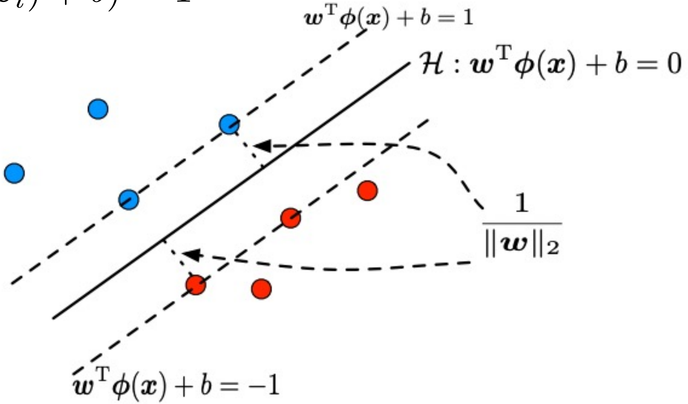
We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_i y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + b) = 1$

The margin then becomes

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

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

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



1.4 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

This is convex! Minimizing a convex function with convex constraint is convex.

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

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

1.5 General non-separable case

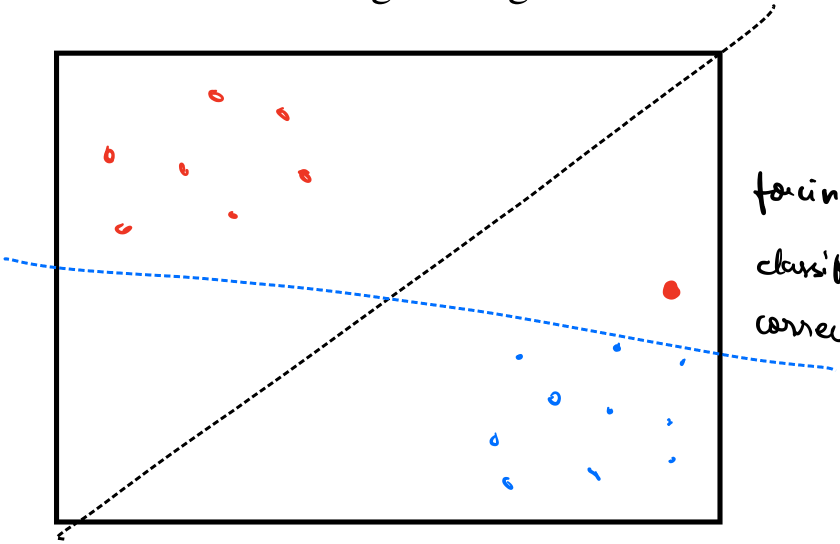
If data is not linearly separable, the previous constraint

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

Can't even match sign$(w^T \phi(x_i) + b)$ & y $\forall$ $i \in [n]$, if not linearly separable.

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

Even if data is linearly separable, should we always separate it?



forcing classifier to classify all datapoints correctly might not be good.

1.5 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.

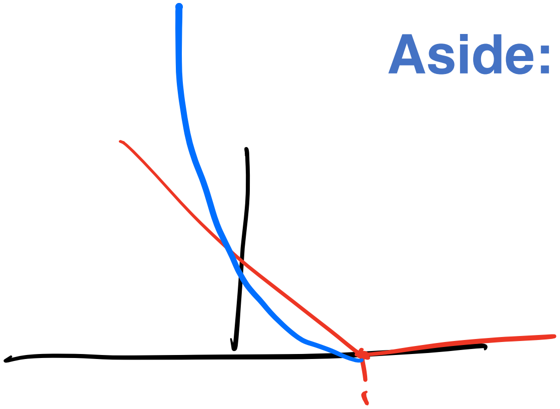
To deal with this issue, we relax the constraints to ℓ_1 **norm soft-margin** constraints:

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

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?



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 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 $w^*_{L2} = \arg\min_w \sum_i (x_i - w)^2$? $\quad w^*_{L2} = \frac{\sum x_i}{n}$

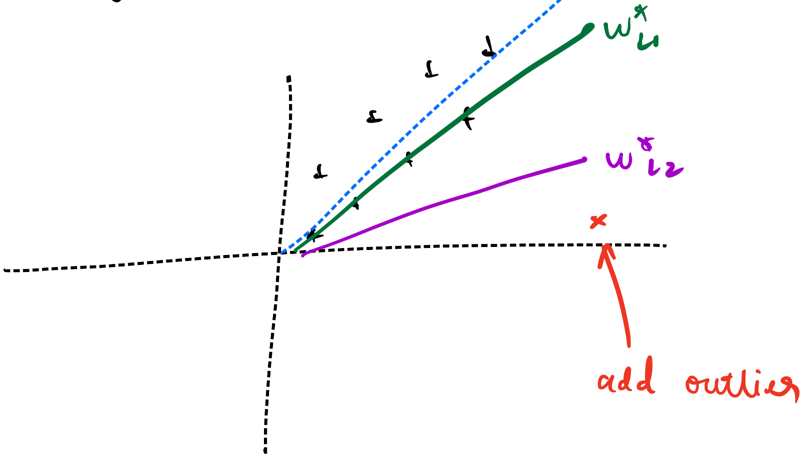
median is more robust to outliers than mean

what is $w^*_{L1} = \arg\min_w \sum_i |x_i - w|$? $\quad w^*_{L1} = \text{median}(x_1, \ldots, x_n)$

Aside: Why ℓ_1 penalization?

For 1-D regression.

$y = 10x +$ noise



consider

$$w^*_{L2} = \arg\min \sum_i (y_i - w x_i)^2$$

$$w^*_{L1} = \arg\min \sum_i |y_i - w x_i|$$

1.5 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** $\xi_i \geq 0$.

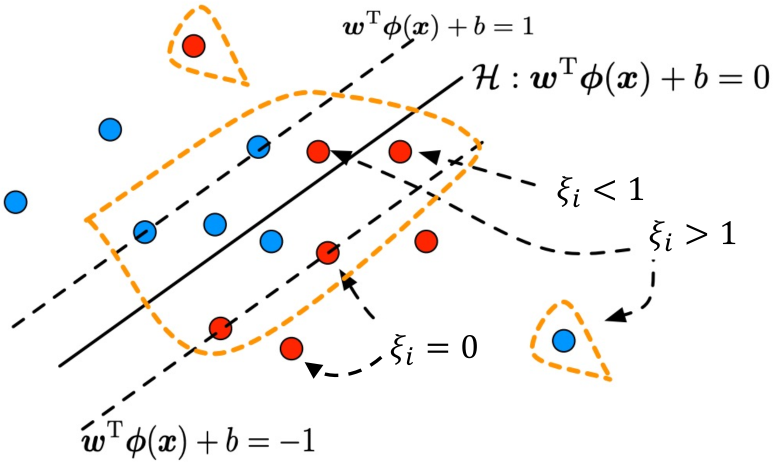
1.5 SVM General Primal Formulation

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

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

where C is a hyperparameter to balance the two goals.

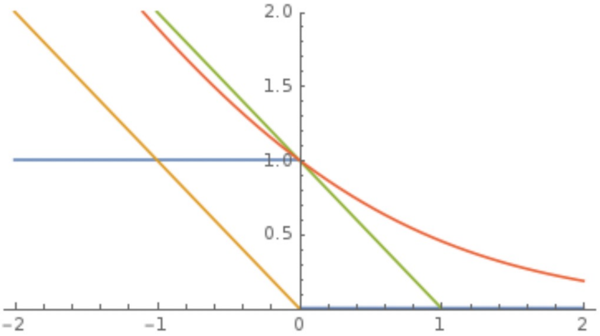
1.6 Understanding the slack conditions



- 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.

1.7 Primal formulation: Another view

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



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

1.7 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?

1.7 Equivalent forms

The formulation

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

In order to minimize $\sum_i \xi_i$

we should set ξ_i to be as small as possible, which is:

is equivalent to

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

1.7 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!*

1.8 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 :

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

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

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**!

2.1 Kernelizing SVM

Claim: For the SVM problem, $w^* = \sum_i \alpha_i^* y_i \phi(x_i)$

Informal Proof:

formulation as a linear model with l_2 regularized hinge loss:

$$F(w) = \min_{w,b} \underbrace{\sum_{i=1}^{n} \max\{0, 1 - y_i (w^T \phi(x_i) + b)\}}_{\text{hinge loss}} + \frac{\lambda}{2} \|w\|_2^2$$

This is a convex problem $\therefore$ GD will find a minimizer with any initialization (for some appropriate step size).

Recall $\ell_{hinge}(z) = \max(0, 1-z)$

$$\frac{\partial F(w)}{\partial w} = \sum_{i=1}^{n} \left(\frac{\partial \ell_{hinge}(z)}{\partial z} \bigg|_{z = y_i(w^T\phi(x_i)+b)} \left(-y_i \phi(x_i)\right) \right) + \lambda w^{(t)}$$

$$w^{(0)} \leftarrow 0$$

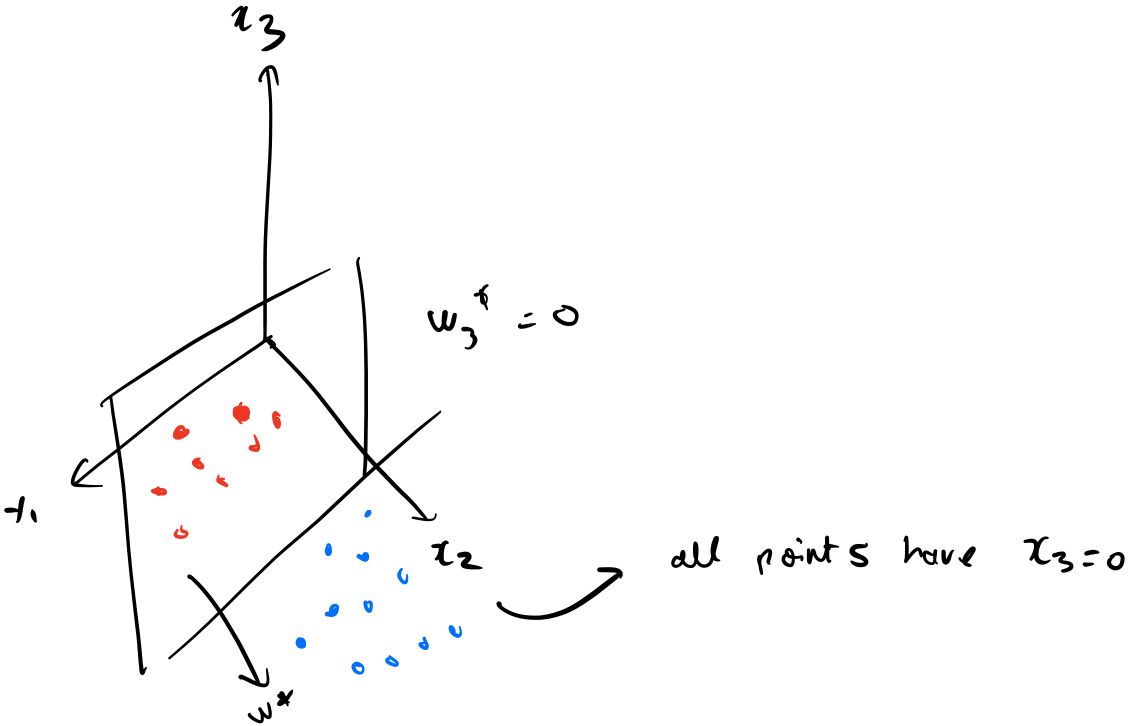
$$w^{(t+1)} \leftarrow w^{(t)} - \eta \sum_{i=1}^{n} \left(\frac{\partial \ell_{hinge}(z)}{\partial z} \bigg|_{z = y_i(w^T\phi(x_i)+b)} \left(-y_i \underline{\underline{\phi(x_i)}}\right) + \lambda w^{(t)} \right)$$

$\therefore$ $w^{(t)}$ always lie in span of $\phi(x_i)$

$$w^{(t)} = \sum \alpha_i^{(t)} y_i \phi(x_i) \quad \forall t, \text{ for some } \alpha_i^{(t)}$$

$$\therefore \quad w^* = \sum_i \alpha_i^* y_i \phi(x_i) \text{ for some } \alpha_i^*$$

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



If $w = \sum_i \alpha_i y_i \phi(x_i)$, how can we use this?

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

$$\longrightarrow \min_{\alpha, b} \quad \frac{1}{2}\left\|\sum_i \alpha_i y_i \phi(x_i)\right\|_2^2$$
$$\text{s.t.} \quad y_j\left(\sum_i \alpha_i y_i \phi(x_i)^T \phi(x_j) + b\right) \geq 1$$

This is equivalent to,

$$\min_{\alpha, b} \quad \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j))$$
$$\text{s.t.} \quad y_j\left(\sum_i \alpha_i y_i \phi(x_i)^T \phi(x_j) + b\right) \geq 1 \quad \forall i \in [n].$$

2.2 SVM: Dual form for separable case

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

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

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

2.2 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.

2.3 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,

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

2.4 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.

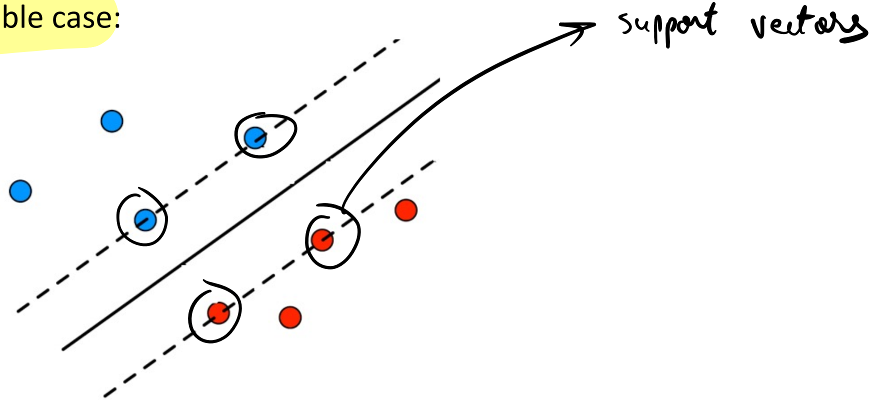
To make a prediction on any datapoint $\boldsymbol{x}$,

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

All we need now is to identify b^*.

2.5 Bias term b^*

First, let's consider the separable case:



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

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

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

$$\Rightarrow b^*=y_i-w^{*T}\phi(x_i) \quad \text{for any } i \text{ s.t. } \alpha_i^*>0.$$

2.5 Bias term b^*

General (non-separable case):

For any support vector $\boldsymbol{\phi}(\boldsymbol{x}_i)$ with $0 < \alpha_i^* < C$, it can be shown that $1 = y_i({\boldsymbol{w}^*}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b^*)$ (i.e. that support vector lies on the margin). Therefore, as before,

$$b^* = y_i - {\boldsymbol{w}^*}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) = y_i - \sum_{j=1}^{n} \alpha_j^* y_j k(\boldsymbol{x}_j, \boldsymbol{x}_i).$$

In practice, often *average* over all i with $0 < \alpha_i^* < C$ to stabilize computation.

With $\boldsymbol{\alpha}^*$ and b^* in hand, we can make a prediction on any datapoint $\boldsymbol{x}$,

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

SVMs: Understanding them further

3.1 Understanding support vectors

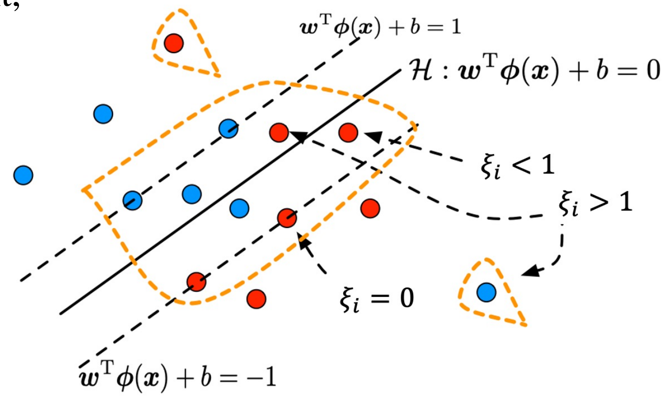
Support vectors are $\boldsymbol{\phi}(\boldsymbol{x}_i)$ such that $\alpha_i^* > 0$.

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.



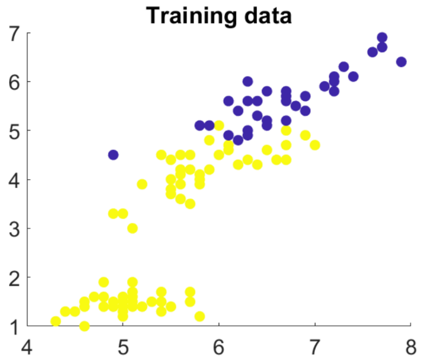
Support vectors (circled with the orange line) are the only points that matter!

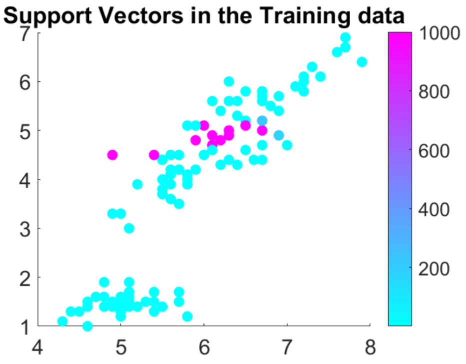
3.1 Understanding support vectors

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

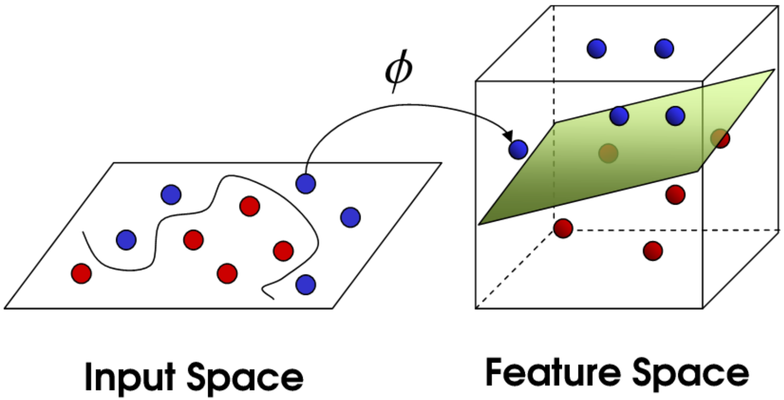
$$\mathrm{sign}\left(\boldsymbol{w}^{*\mathrm{T}}\phi(\boldsymbol{x}) - b^*\right) = \mathrm{sign}\left(\sum_{i=1}^{n} \alpha_i^* y_i k(\boldsymbol{x}_i, \boldsymbol{x}) - b^*\right).$$

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



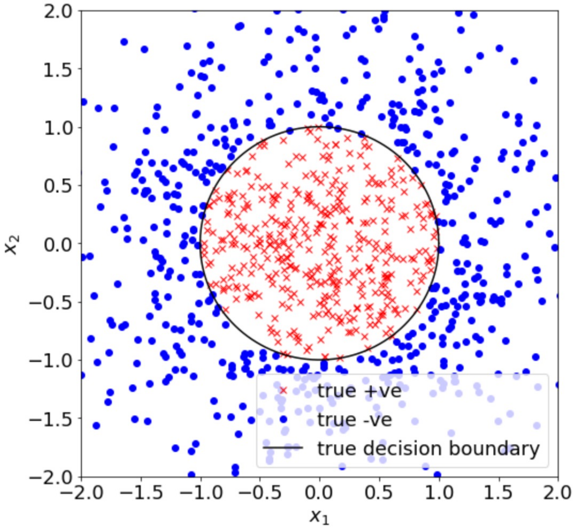


3.2 Examining the effect of kernels



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,

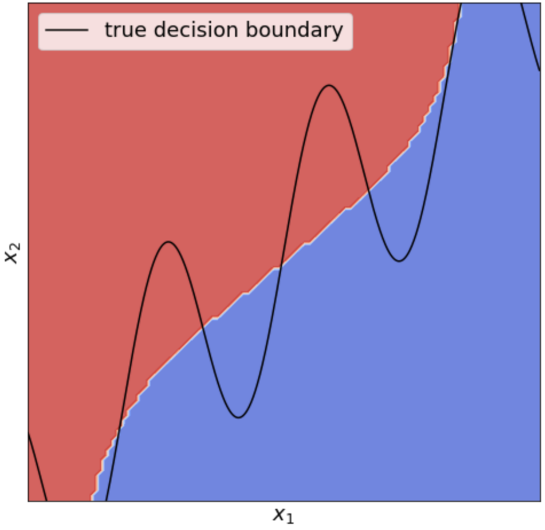
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2\right)$$

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

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



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):

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