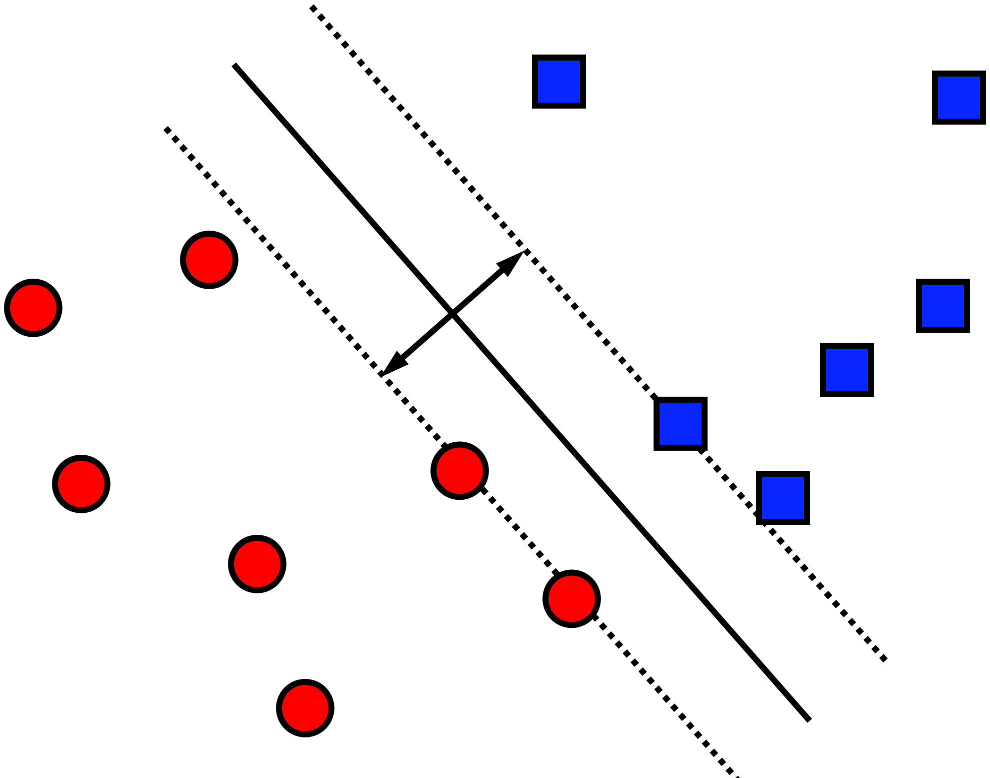
Support Vector Machines



Support Vector Machines

- Early ideas developed in 1960s and 1970s by **Vladimir Vapnik** and Alexey Chervonenkis in USSR
- Major developments in 1990s by Vapnik and many others (Corinna Cortes, Bernhard Schölkopf, ...)
- Originally developed for **binary classification** tasks
- Elegant theory
  - Clear notion of model ‘capacity’
  - Generalization bounds
- Works very well in practice

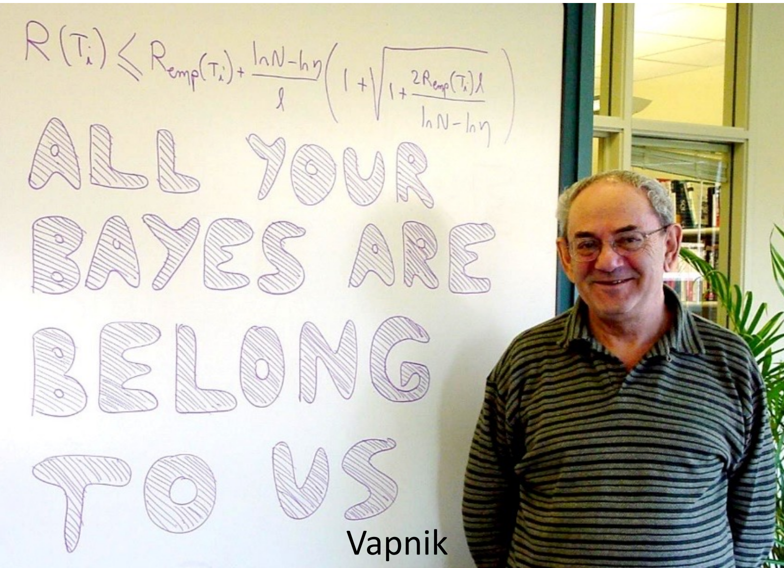


Image credit: Yann LeCun

Using Support Vector Machines

sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

Parameters:

? **C** : ***float, optional (default=1.0)***

Penalty parameter C of the error term.

Lots of terminology and concepts specific to SVMs.
To use SVMs effectively should know what they mean!

? **kernel** : ***string, optional (default='rbf')***

Specifies the kernel type to be used in the algorithm. It must be one of 'linear' ?, 'poly' ?, 'rbf' ?, 'sigmoid', 'precomputed' or a callable.

? **degree** : ***int, optional (default=3)***

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

? **gamma** : ***float, optional (default='auto')***

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

? **coef0** : ***float, optional (default=0.0)***

Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

Using Support Vector Machines

sklearn.svm.SVC

1.4.6. Kernel functions

Reading documentation isn't enough to understand what these things mean, or why SVMs are expressed this way!

The *kernel function* can be any of the following:

- ? linear: $\langle x, x' \rangle$.
- ? polynomial: $(\gamma \langle x, x' \rangle + r)^d$. d is specified by keyword `degree`, r by `coef0`.
- ? rbf: $\exp(-\gamma \|x - x'\|^2)$. γ is specified by keyword `gamma`, must be greater than 0.
- sigmoid $(\tanh(\gamma \langle x, x' \rangle + r))$, where r is specified by `coef0`.

1.4.6.1. Custom Kernels

You can define your own kernels by either giving the kernel as a python function or by precomputing the Gram matrix. ?

Using Support Vector Machines

`sklearn.svm`.SVC

None of this makes sense without building it up piece-by-piece...

1.4.7.1. SVC

?

Given training vectors $x_i \in \mathbb{R}^p$, i=1,…, n, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

$$\text{subject to } y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$

$$\zeta_i \geq 0, i = 1, \ldots, n$$

...so that is what we will do!

👨‍🏫

Its dual is

???

😭 😭 😭 😭

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$\text{subject to } y^T \alpha = 0$$

$$0 \leq \alpha_i \leq C, i = 1, \ldots, n$$

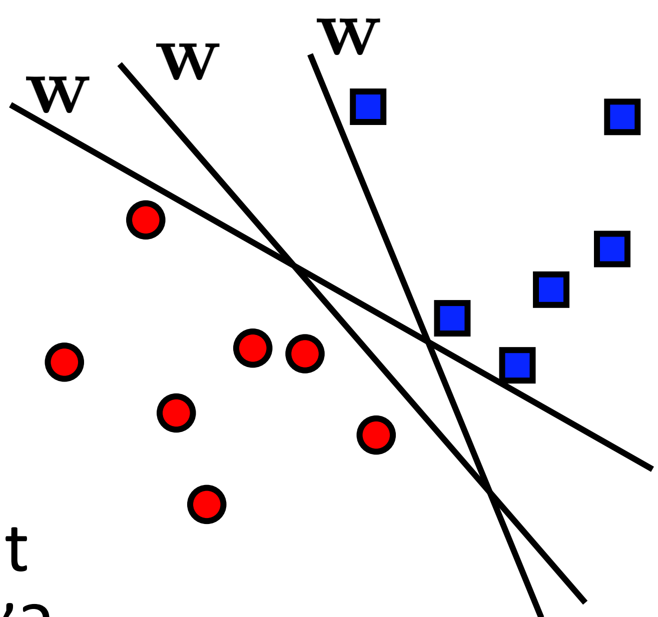
? ?

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$ where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

?

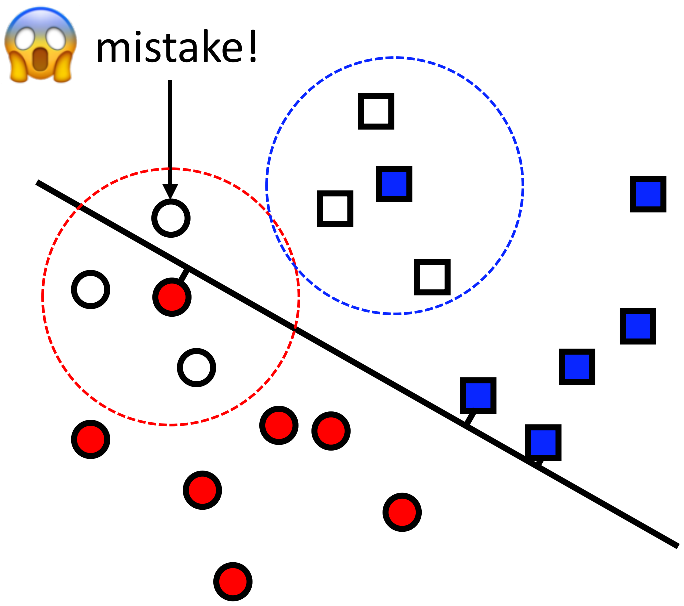
Linear Discriminant Functions

- If data is linearly separable, can choose from among many possible separating hyperplanes
- Is any choice of $\mathbf{w}$ better than the rest? In what sense?
- Unregularized logistic regression <u>does not care</u>; all equally good
- Regularized logistic regression <u>cares</u>, but does biasing each w_i toward zero give a hyperplane that satisfies useful definition of "best"?



Choosing a hyperplane

- Suppose we choose a hyperplane that passes <u>close to the training data</u>
- BUT training data is just a small subsample of all possible data.
- New class samples likely to be ‘near’ training data of that class
- We’re setting ourselves up to make mistakes on test data!

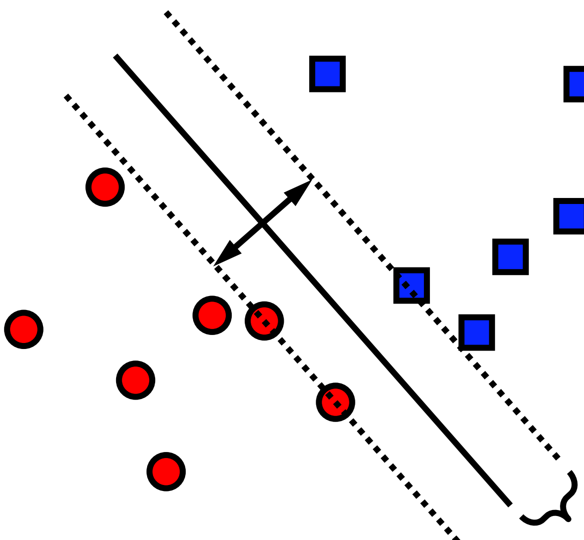


Notice that this concern hinges on an assumption that the data distribution is somehow “smooth,” where the presence of a sample of class *C* indicates a higher probability of observing class *C* “nearby” in feature space.

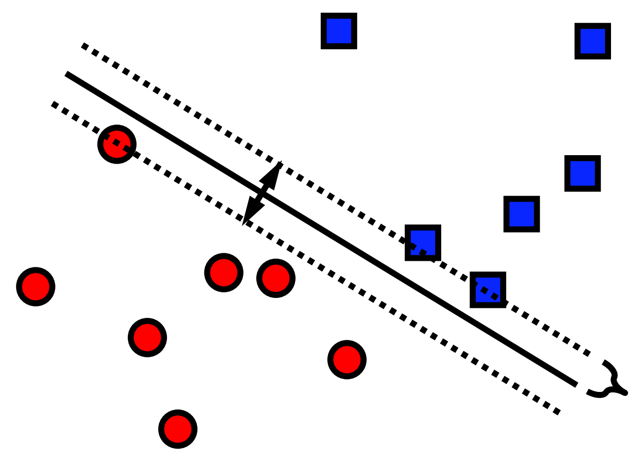
Maximum margin principle

- **Idea:** seek the separating hyperplane that has *maximum margin* from the training samples

more likely to generalize



less likely to generalize

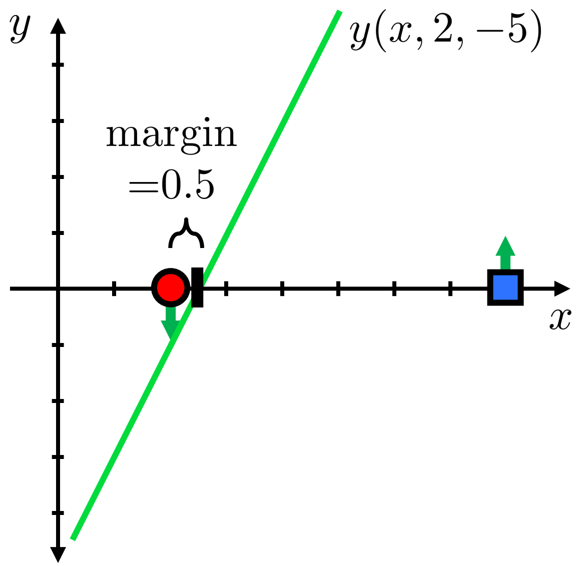


margin

= distance of nearest point(s) to hyperplane

Preview of geometric intuition behind SVM training formulation:

Let $y(x, a, b) = ax + b$ be a "linear discriminant", or a "decision function" for a 1-dimensional classification task. Predict positive class when $y(x) \geq 0$



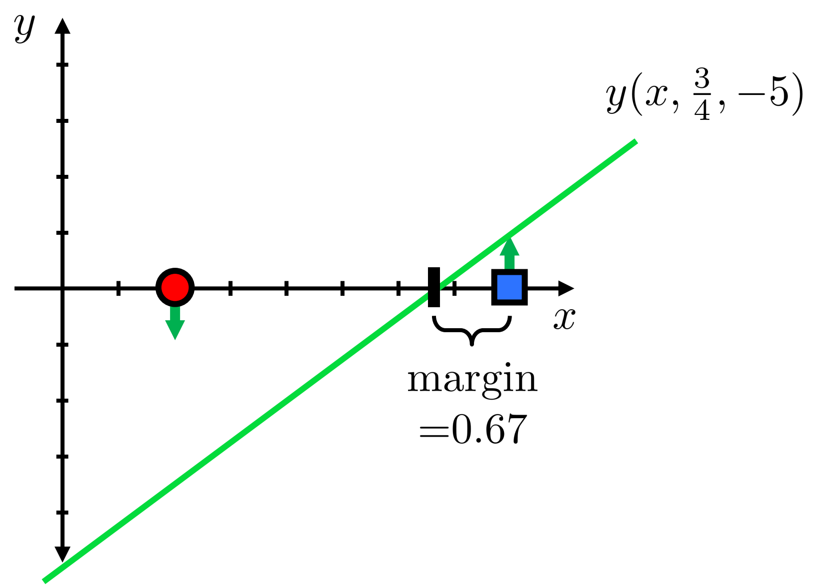
Put little "pegs" on the training data.
Positive examples get an "upward" peg.
Negative examples get a "downward" peg.

Doesn't matter what height of pegs is.
Assume they have height = 1.

Constraint: linear discriminant must pass *above* all upward pegs, and must pass *below* all the downward pegs

Preview of geometric intuition behind SVM training formulation:

Let $y(x, a, b) = ax + b$ be a "linear discriminant", or a "decision function" for a 1-dimensional classification task. Predict positive class when $y(x) \geq 0$



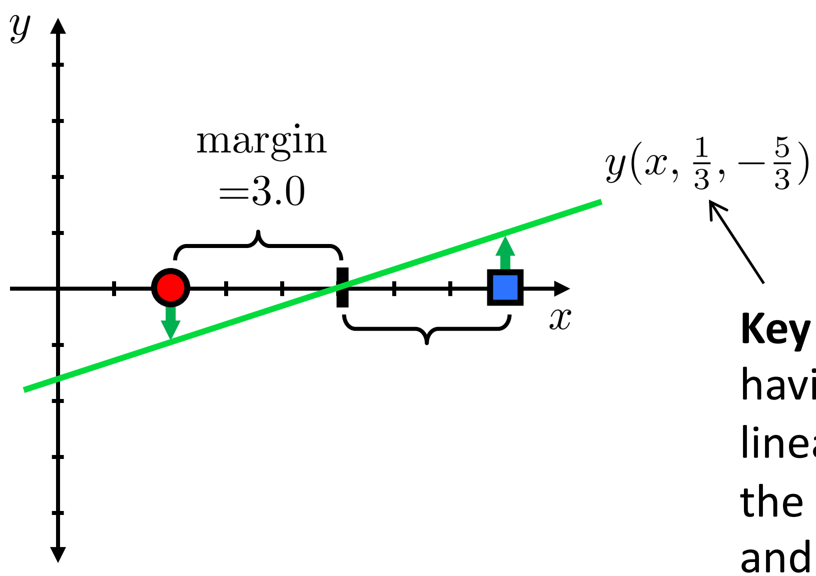
Different combinations of (a, b) can satisfy these "above/below the pegs" constraints.

Each choice has a different margin.

Each choice may perform differently on test data than the other choices.

Preview of geometric intuition behind SVM training formulation:

Let $y(x, a, b) = ax + b$ be a "linear discriminant", or a "decision function" for a 1-dimensional classification task. Predict positive class when $y(x) \geq 0$



Key idea of SVMs: the choice of (a, b) having *smallest slope* $|a|$ is the unique linear discriminant having $y(x) = 0$ at the *midpoint* between the closest positive and negative examples, and therefore has the *maximum margin*.

SVMs prefer *this choice* among all others.

Hyperplane geometry

- How to express distance of a point to a hyperplane, *i.e.* the magnitude of the *margin*?

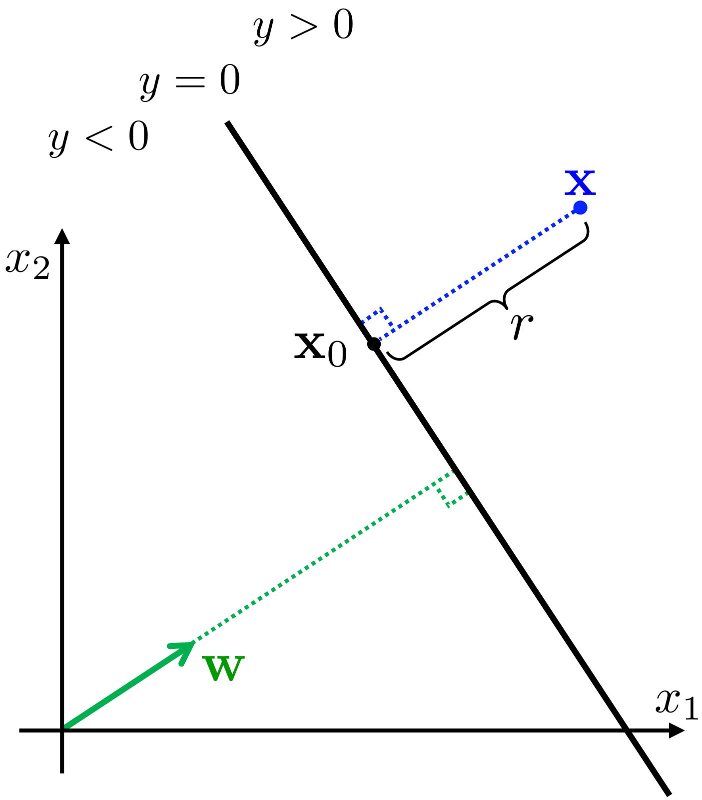
Linear discriminant is $y(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T\mathbf{x} + b$, or $y(\mathbf{x})$ for short. Hyperplane is $y(\mathbf{x}) = 0$.

Point $\mathbf{x}$ can be written as $\mathbf{x} = \mathbf{x}_0 + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$ where $\mathbf{x}_0$ is projection onto plane and $\|\mathbf{w}\| = \sqrt{\mathbf{w}^T\mathbf{w}}$ is length of $\mathbf{w}$.

Since $y(\mathbf{x}_0) = 0$ we have

$$\mathbf{w}^T\left(\mathbf{x} - r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + b = 0 \quad \Rightarrow \quad r = \frac{\mathbf{y}(\mathbf{x})}{\|\mathbf{w}\|}$$

signed distance to hyperplane!



('r' is then really $r(\mathbf{x}, \mathbf{w}, b)$)

Classification from hyperplane

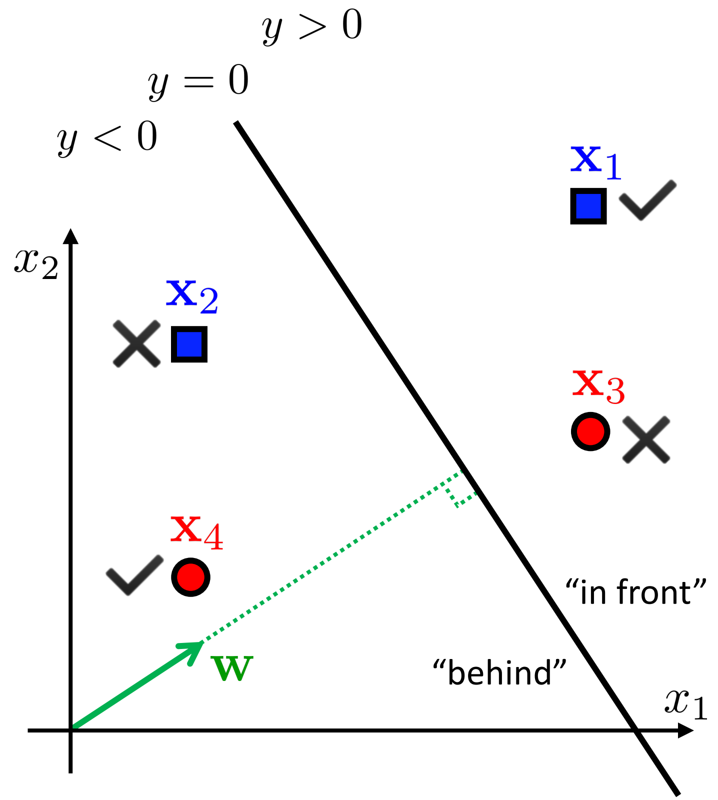
- Given training set $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ where class labels $t_i \in \{-1, +1\}$.
- Can classify new point $\mathbf{x}$ using *sign* of signed distance:

$$\hat{t} = \operatorname{sign}\left(\frac{y(\mathbf{x}, \mathbf{w}, b)}{\|w\|}\right) = \operatorname{sign}\left(y(\mathbf{x}, \mathbf{w}, b)\right) = \operatorname{sign}\left(\mathbf{w}^T\mathbf{x} + b\right)$$

predicted class label

$$\text{where } \operatorname{sign}(y) = \begin{cases} +1 & y > 0 \\ 0 & y = 0 \\ -1 & y < 0 \end{cases}$$

means "not sure"

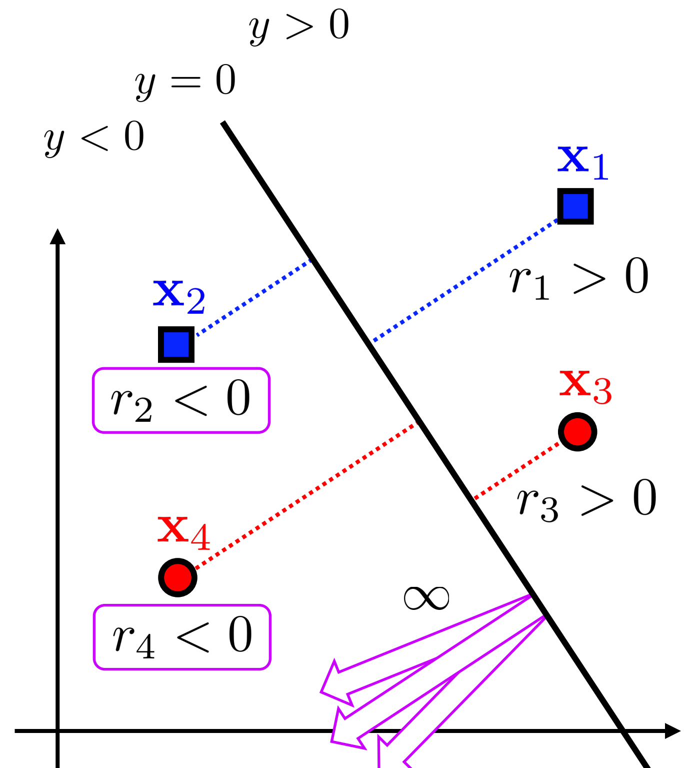


Maximizing "the margin" (1st attempt)

Given training set $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$
where class labels $t_i \in \{-1, +1\}$.

Can we define the "margin" to be the smallest signed distance to all points, and try to maximize it?

$$\max_{\mathbf{w},b} \Big[\underbrace{\min_{i=1..N} r_i}_{\text{margin?}} \Big] \qquad r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$



NO! This is "make all points be *in front* of the hyperplane as far as possible."

As far as possible is $+\infty$, and we're not even using t_i 

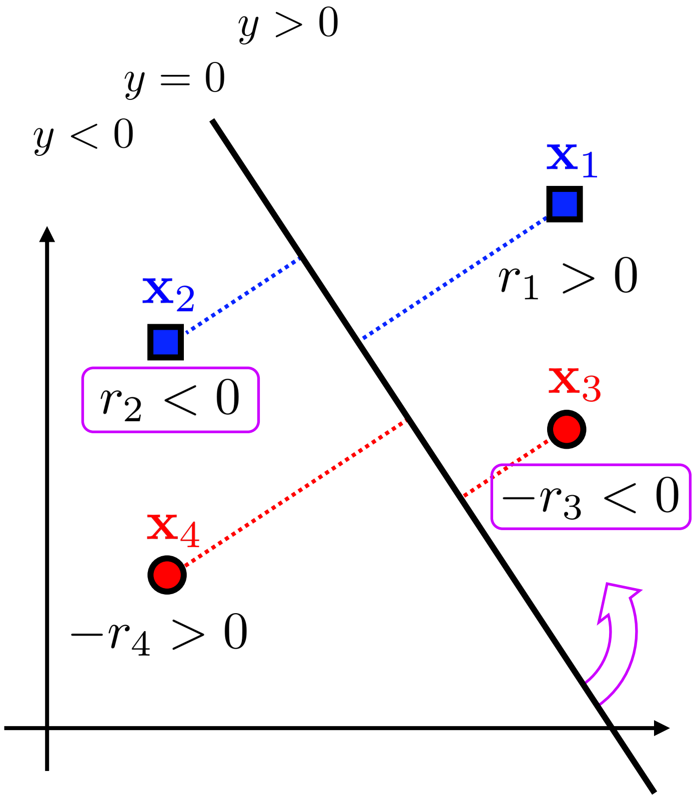
Maximizing "the margin" (correct)

Given training set $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$
where class labels $t_i \in \{-1, +1\}$.

Fix: Make $t_i = -1$ cases be ***behind*** the hyperplane as far as possible.

$$\max_{\mathbf{w},b} \Big[\underbrace{\min_{i=1..N} t_i r_i}_{\text{margin!}} \Big] \qquad r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$

YES! Margin is negative if *any* point is not in the halfspace assigned by t_i.

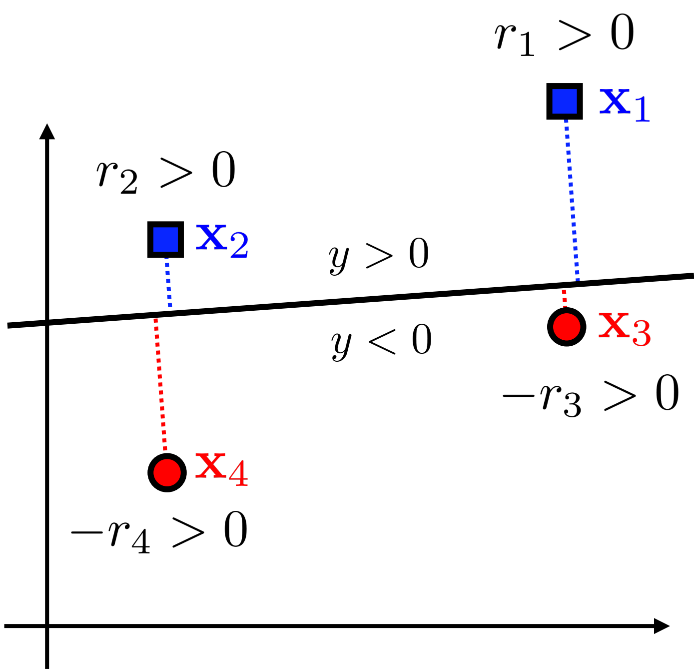


Maximizing "the margin" (correct)

Given training set $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$
where class labels $t_i \in \{-1, +1\}$.

Fix: Make $t_i = -1$ cases be ***behind***
the hyperplane as far as possible.

$$\max_{\mathbf{w},b} \Big[\underbrace{\min_{i=1..N} t_i r_i}_{\text{margin!}} \Big] \qquad r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$



<u>**YES!**</u> Margin < 0 if any misclassified.

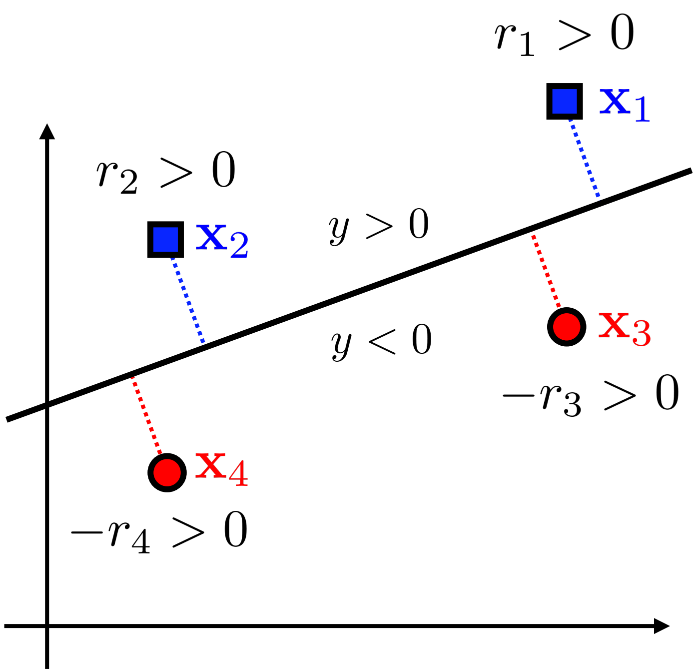
<u>**YES!**</u> Margin becomes positive when *all* data is separated.

Maximizing "the margin" (correct)

Given training set $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$
where class labels $t_i \in \{-1, +1\}$.

Fix: Make $t_i = -1$ cases be ***behind*** the hyperplane as far as possible.

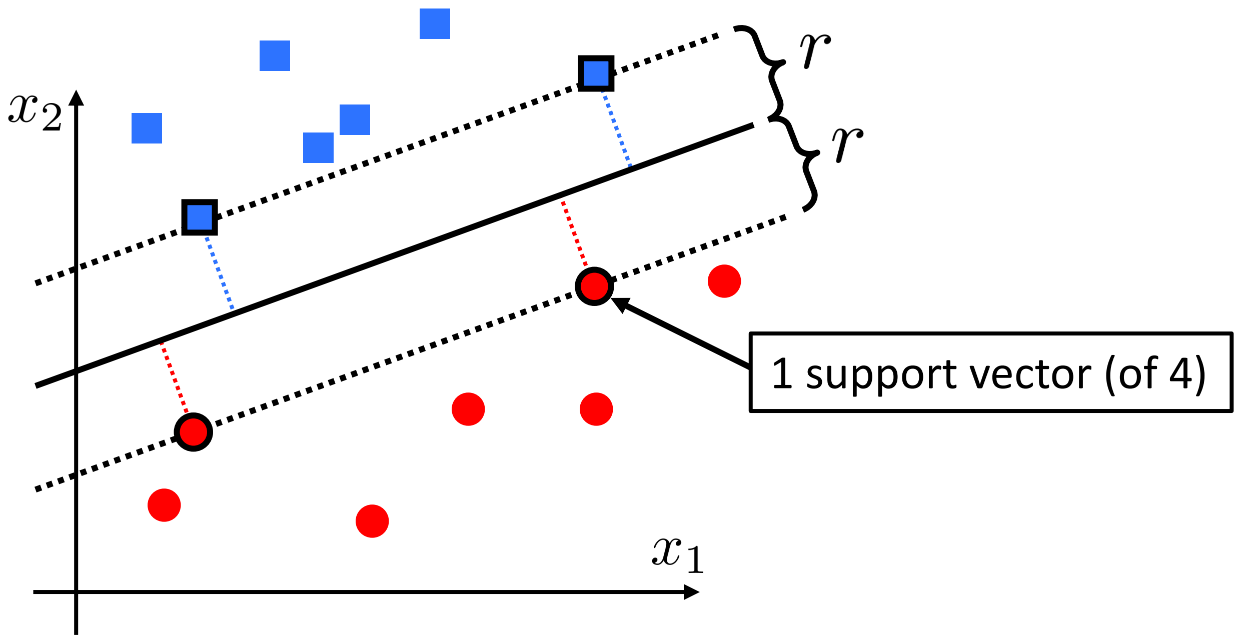
$$\max_{\mathbf{w},b} \Big[\underbrace{\min_{i=1..N} t_i r_i}_{\text{margin!}} \Big] \qquad r_i = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$$



- **YES!** Margin < 0 if any misclassified.
- **YES!** Margin > 0 if data fully separated.
- **YES!** Margin is bounded from above.

Support Vectors

Data point $\mathbf{x}_i$ is a *"support vector"* if no other data point has strictly smaller distance to the hyperplane



The support vectors are sufficient to determine the hyperplane. Other points are irrelevant!

Towards constrained programming

Goal: Learn classifier by solving this max-min problem:

$$\max_{\mathbf{w},b}\left[\min_{i=1..N}\frac{t_i\left(\mathbf{w}^T\mathbf{x}_i+b\right)}{\|\mathbf{w}\|}\right]$$

Strategy: Formulate as *constrained programming*, so that we can use powerful optimization algorithms.

Idea: Express the $\min_{i=1..N}$ with a set of *N* constraints:

$$\max_{\mathbf{w},b,r} r \quad \text{such that} \quad r \leq \frac{t_i\left(\mathbf{w}^T\mathbf{x}_i+b\right)}{\|\mathbf{w}\|} \quad \text{for } i=1,\ldots,N$$

introduce new variable $r \in \mathbb{R}$ to be *margin*

OK! But these constraints are non-linear in $\mathbf{w}$. Can we make them linear somehow?

Why aim for *linear* constraints?

Because we can use faster optimization algorithms!

Hopefully quadratic program optimizers! (faster "solvers")

CVXOPT User's Guide

Quadratic Programming

The function `qp` is an interface for quadratic programs. It provides the option of using the quadratic programming solver from MOSEK.

`cvxopt.solvers.qp`(*P*, *q* [, *G*, *h* [, *A*, *b* [, *solver* [, *initvals*]]]])

Solves the pair of primal and dual convex quadratic programs

$$\begin{array}{ll} \text{minimize} & (1/2)x^T P x + q^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b \end{array}$$

← x will represent our $[\mathbf{w} \quad b]$ parameters

already handles *linear* inequality constraints

https://cvxopt.org/userguide/coneprog.html#quadratic-programming

Solvers and scripting (programming) languages [edit]

Name	
AMPL	A popular modeling language for large-scale mathematical opti
CPLEX	Popular solver with an API (C, C++, Java, .Net, Python, Matlab
Excel Solver Function	A nonlinear solver adjusted to spreadsheets in which function e
GAMS	A high-level modeling system for mathematical optimization
Gurobi	Solver with parallel algorithms for large-scale linear programs,
IPOPT	Ipopt (Interior Point OPTimizer) is a software package for large
Maple	General-purpose programming language for mathematics. Solv
MATLAB	A general-purpose and matrix-oriented programming-language
Mathematica	A general-purpose programming-language for mathematics, inc
MOSEK	A solver for large scale optimization with API for several langua

Towards *quadratic* programming

First, move non-linear $\|\mathbf{w}\|$ term out of the denominator

$$\max_{\mathbf{w},b,r}\ r\ \text{ such that }\ r\|\mathbf{w}\| \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\ \text{ for } i = 1,\ldots,N$$

🤔 Non-linear in $\mathbf{w}, r$.
But can we somehow
make this side linear?

Linear in $\mathbf{w}, b$! 😀
($t_i, \mathbf{x}_i$ are constants)

Observation: The scale of $\mathbf{w}, b$ is *arbitrary* in this formulation, since $y(\mathbf{x}, \alpha\mathbf{w}, \alpha b) = \alpha(\mathbf{w}^T\mathbf{x} + b)$ defines the exact same decision boundary for any $\alpha > 0$, and likewise $\|\alpha\mathbf{w}\| = \alpha\|\mathbf{w}\|$.

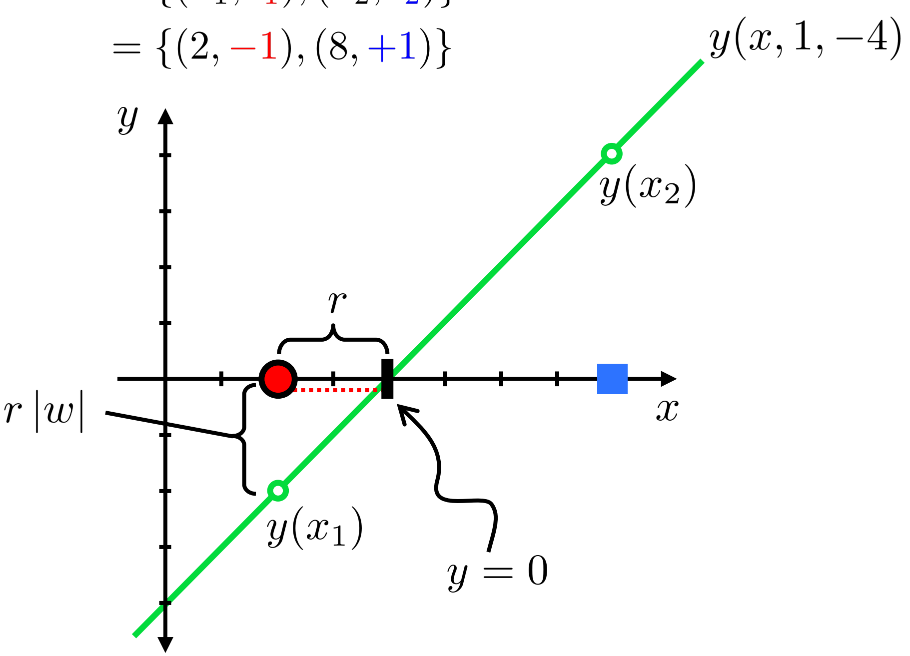
This means it's OK to *restrict our search* space to only $\mathbf{w}, b$ for which $\|\mathbf{w}\| = (\text{something})$. Still max-margin!

Towards quadratic programming

Idea: Use this “degree of freedom” in $\mathbf{w}, b$ to search only solutions where $r\,\|\mathbf{w}\|$ takes some *constant* value.

1D example $y(x, w, b) = wx + b$

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$

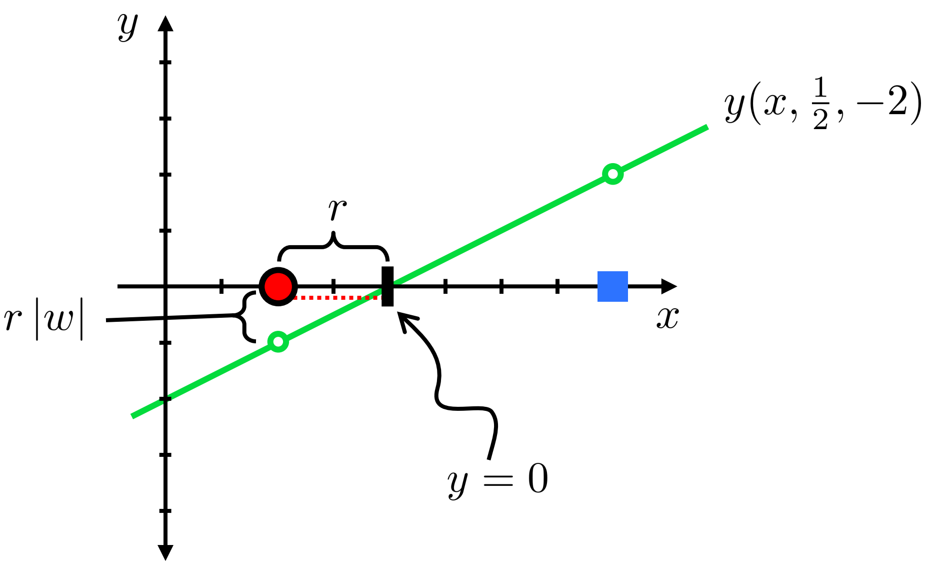


Towards quadratic programming

Idea: Use this “degree of freedom” in $\mathbf{w}, b$ to search only solutions where $r\,\|\mathbf{w}\|$ takes some *constant* value.

1D example $y(x, w, b) = wx + b$

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$

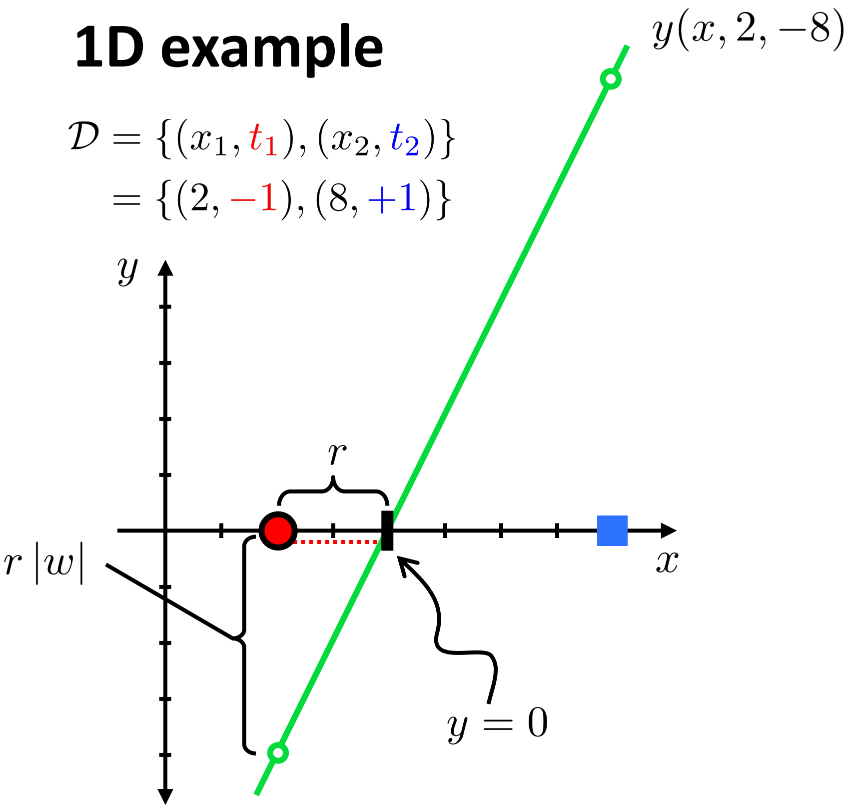


Towards quadratic programming

Idea: Use this "degree of freedom" in $\mathbf{w}, b$ to search only solutions where $r\,\|\mathbf{w}\|$ takes some *constant* value.

1D example

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$

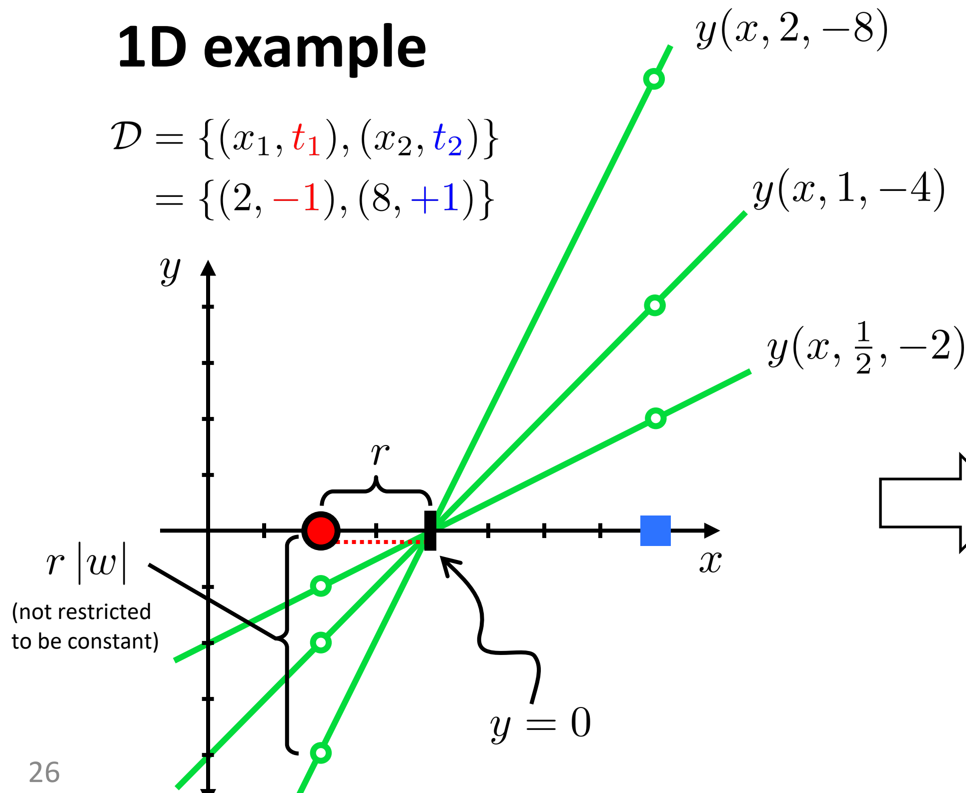


Towards quadratic programming

Idea: Use this "degree of freedom" in $\mathbf{w}, b$ to search only solutions where $r\,\|\mathbf{w}\|$ takes some *constant* value.

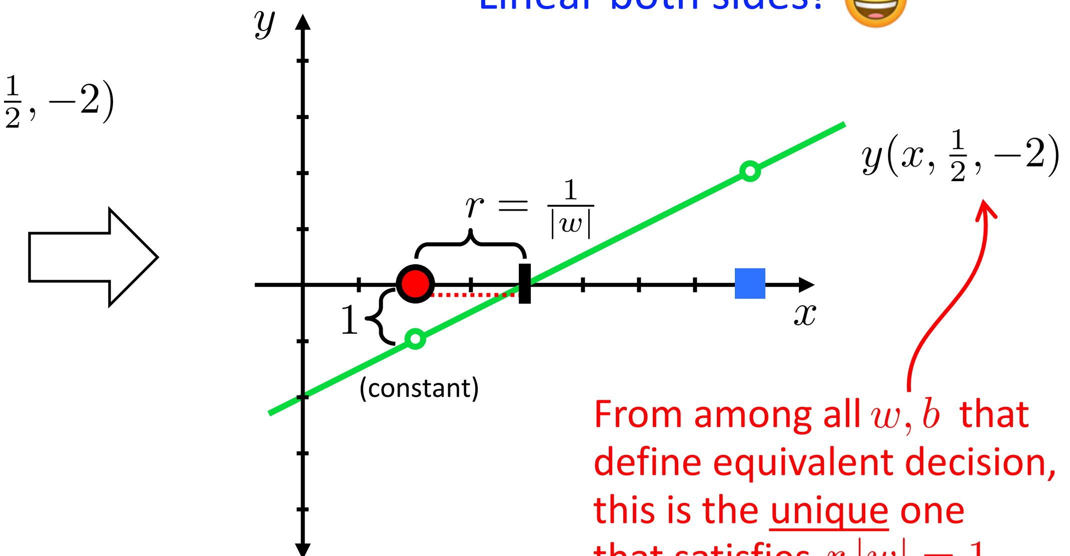
1D example

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$



Choose $r\,\|\mathbf{w}\| = 1$ arbitrarily. Now we automatically consider only $\mathbf{w}, b$ for which $1 \leq t_i y(\mathbf{x}_i, \mathbf{w}, b)$ for all *i*.

Linear both sides! 😀

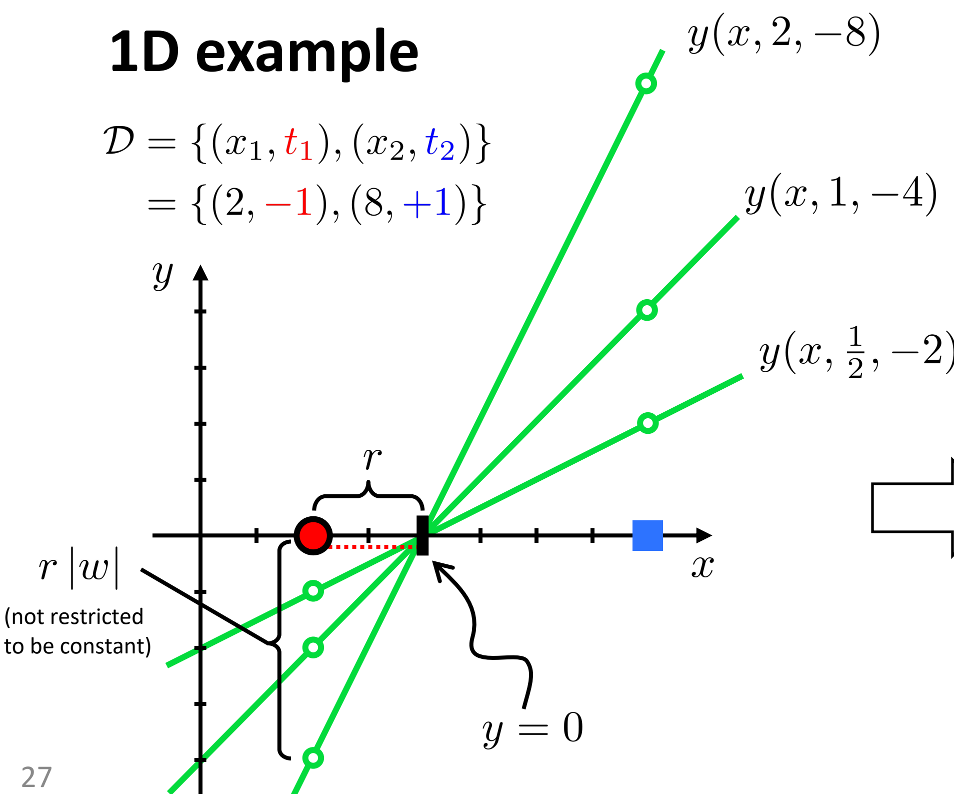


Towards quadratic programming

Idea: Use this "degree of freedom" in $\mathbf{w}, b$ to search only solutions where $r \|\mathbf{w}\|$ takes some *constant* value.

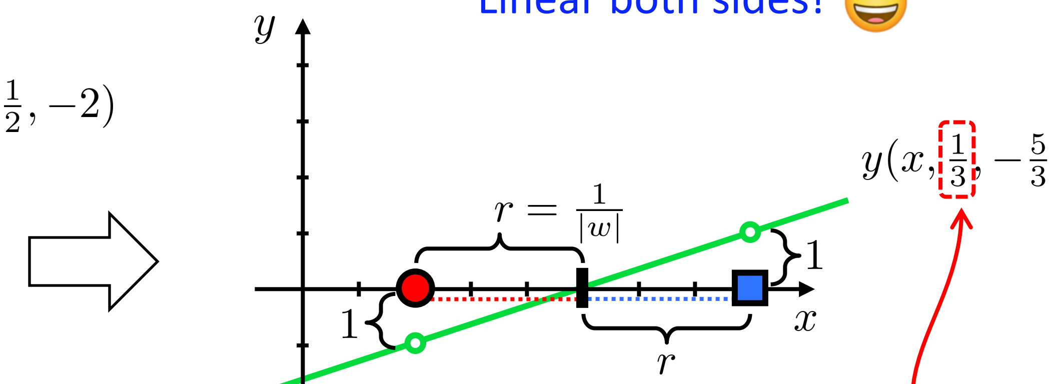
1D example

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$



Choose $r \|\mathbf{w}\| = 1$ arbitrarily. Now we automatically consider only $\mathbf{w}, b$ for which $1 \leq t_i y(\mathbf{x}_i, \mathbf{w}, b)$ for all *i*.

Linear both sides! 😀



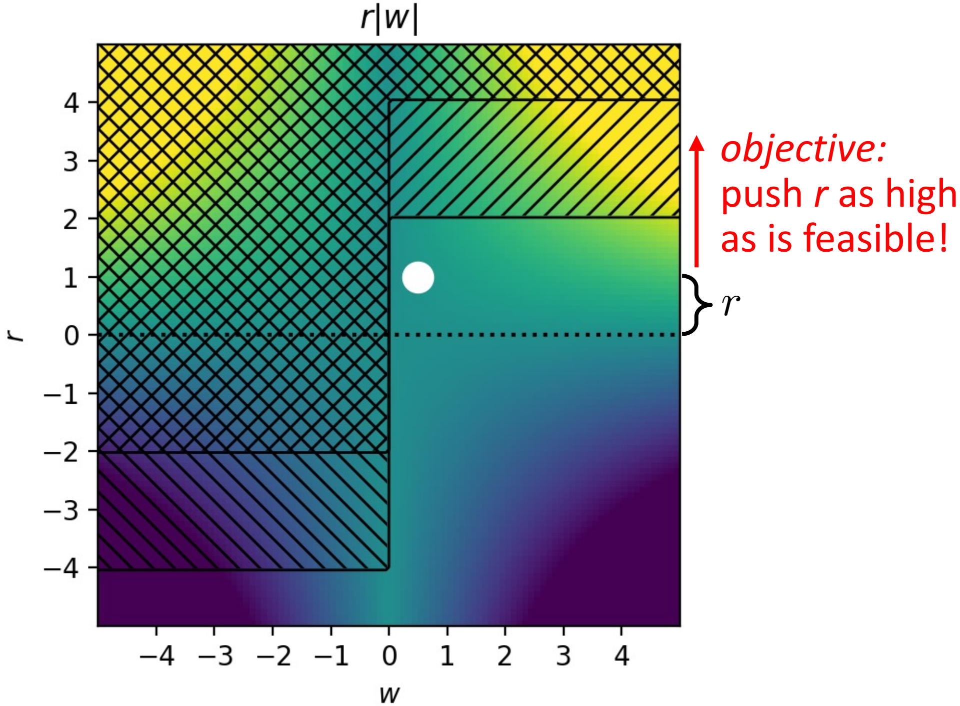
Notice: with $r|w| = 1$ constraint, the maximum margin r happens when $y(x)$ is "flattest" possible, *i.e.* when $|w|$ smallest possible!

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$

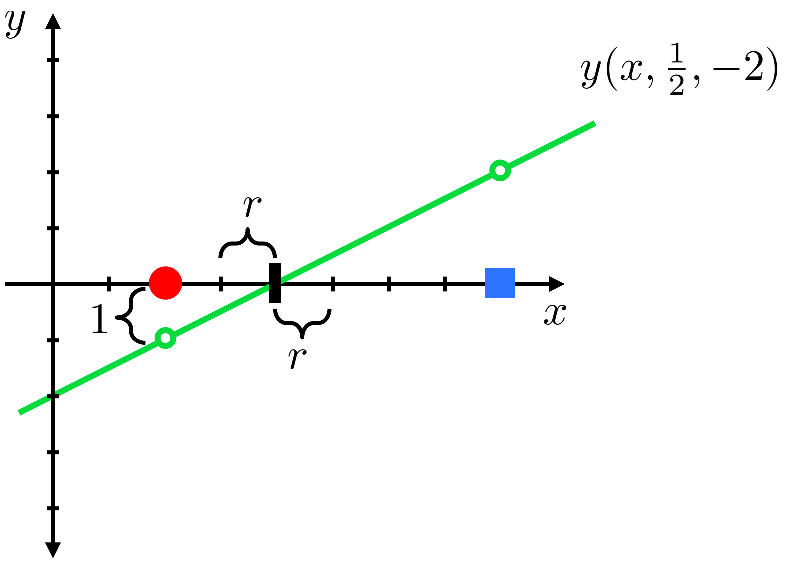
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \;\; r\,|w| \leq -2w - b \quad \text{(not tight)}$$
$$r\,|w| \leq \;\; 8w + b \quad \text{(not tight)}$$



where $b = -4w$ (intercept held constant at 4)

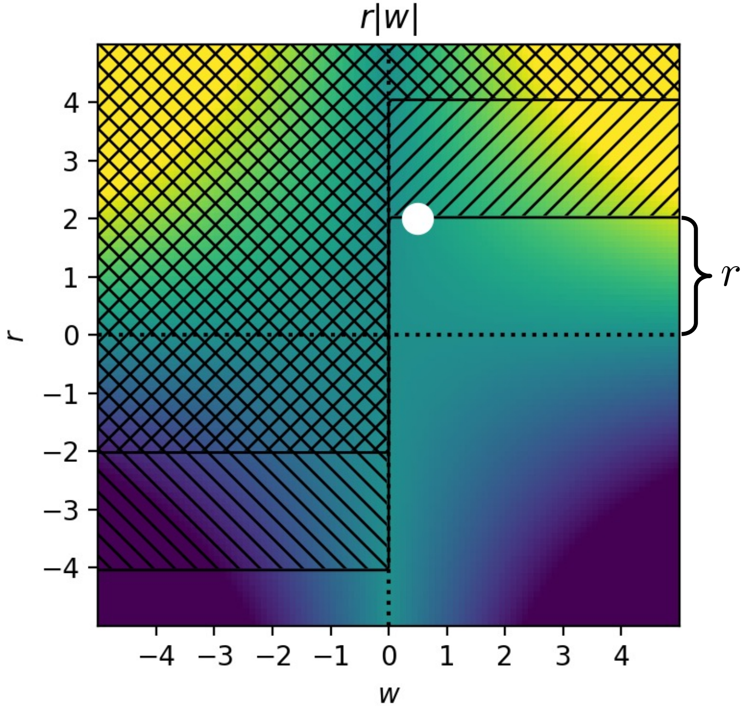


$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

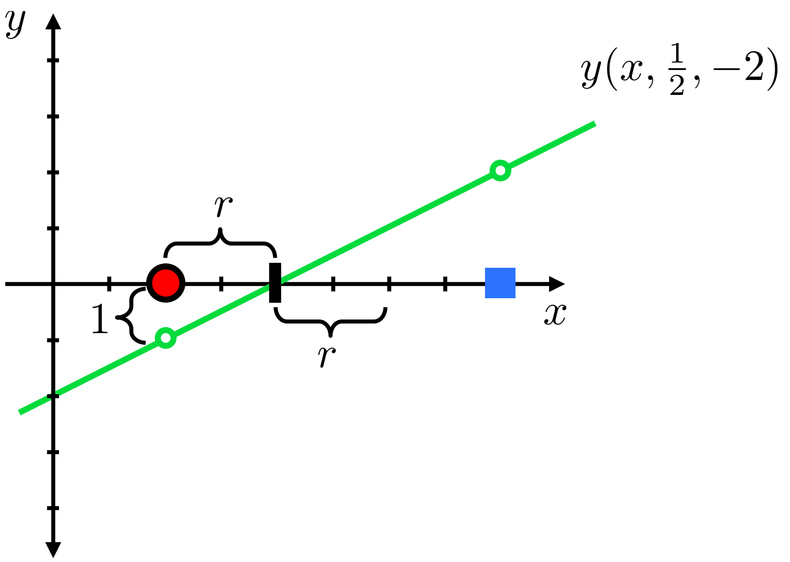
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \; r\,|w| \leq -2w - b \quad \text{(tight)}$$
$$r\,|w| \leq \;\; 8w + b \quad \text{(not tight)}$$



where $b = -4w$ (intercept held constant at 4)

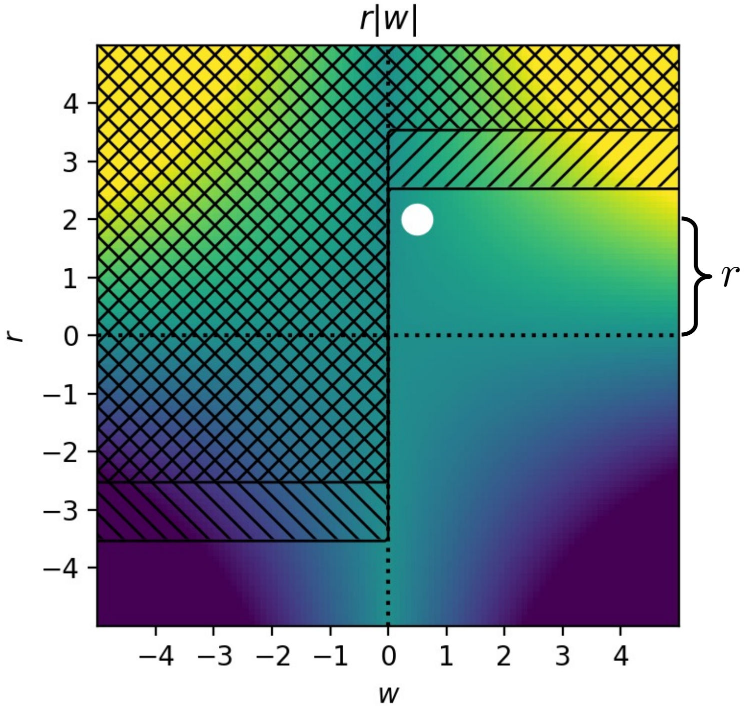


$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

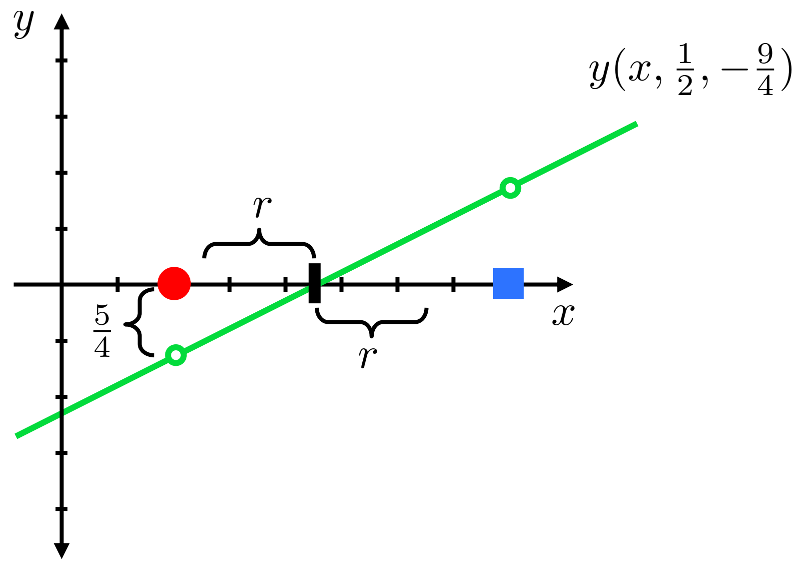
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \; r\,|w| \le -2w - b \quad \text{(not tight)}$$
$$r\,|w| \le \;\; 8w + b \quad \text{(not tight)}$$



where $b = -\frac{9}{4}w$ (intercept held constant at 4.5)

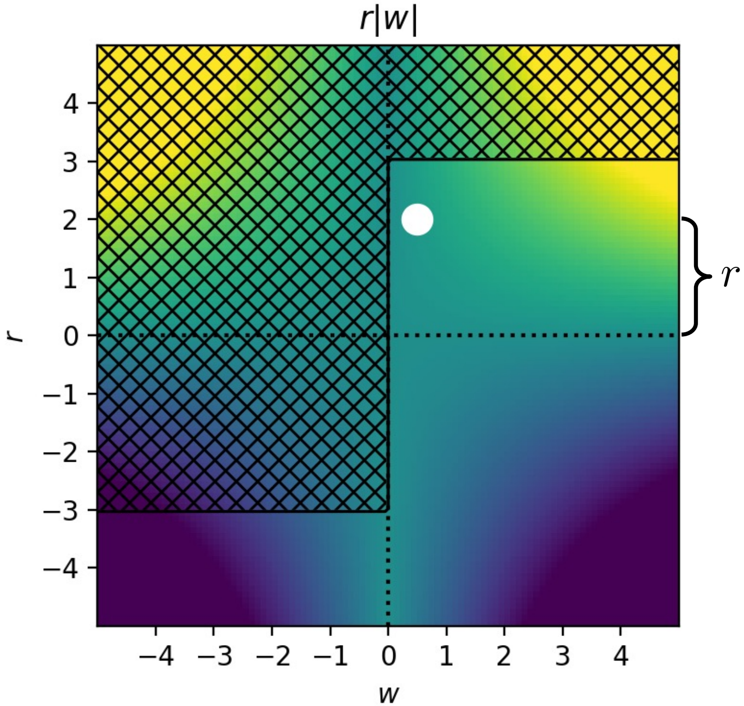


$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

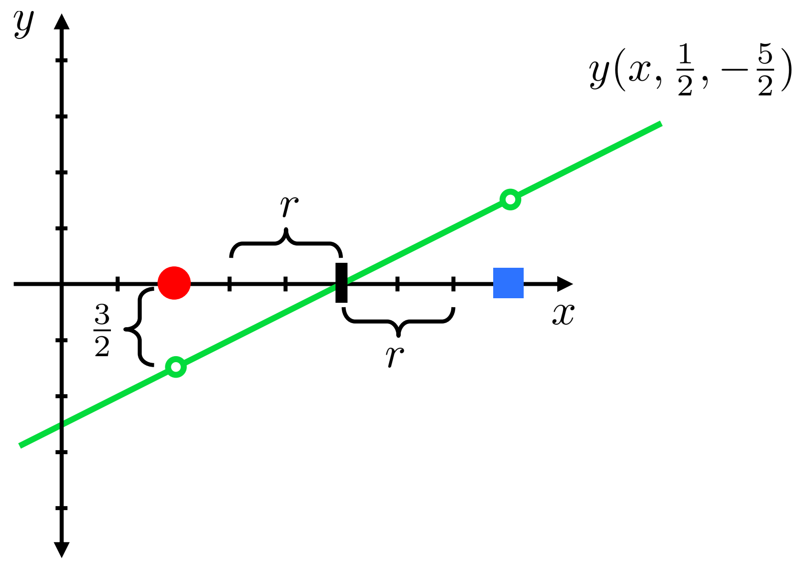
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \quad r\,|w| \le -2w - b \quad \text{(not tight)}$$
$$r\,|w| \le 8w + b \quad \text{(not tight)}$$



where $b = -5w$ (intercept held constant at 5)

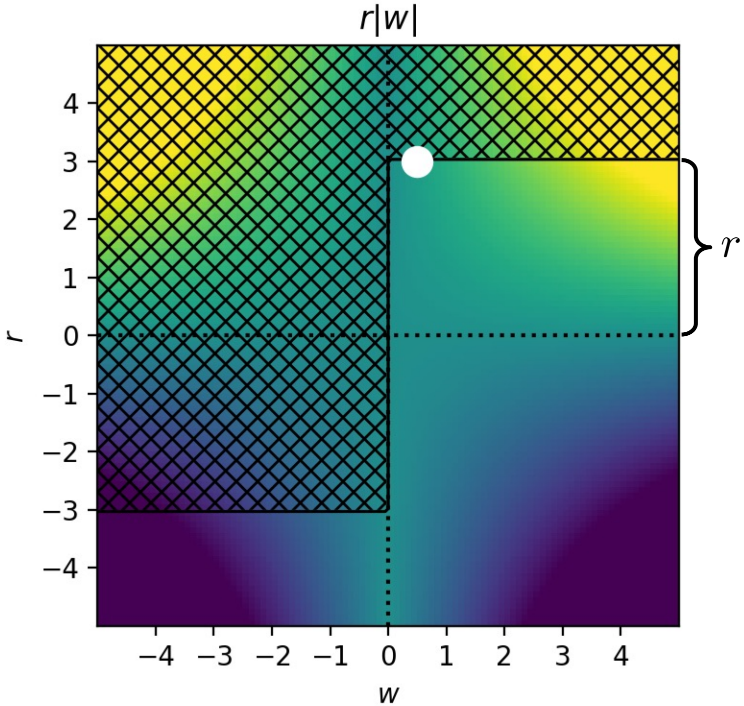


$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

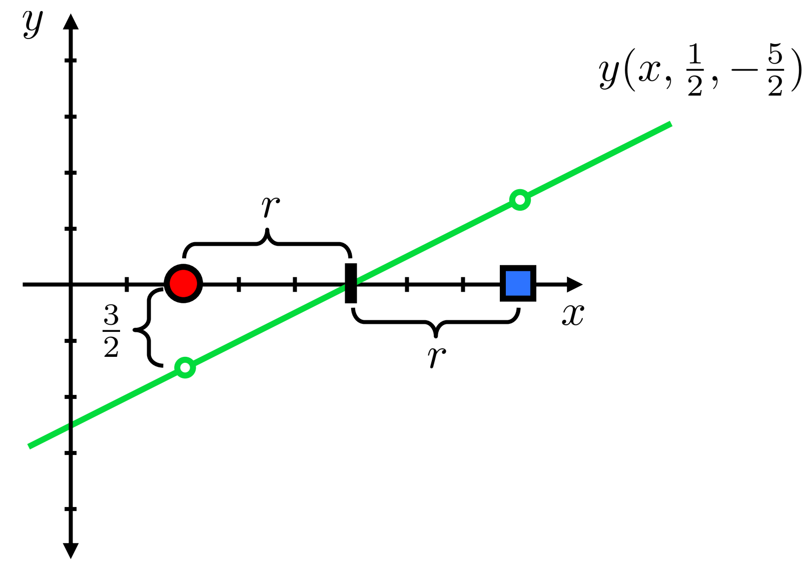
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \; r\,|w| \leq -2w - b \quad \textbf{(tight)}$$
$$r\,|w| \leq 8w + b \quad \textbf{(tight)}$$



where $b = -5w$ (intercept held constant at 5)

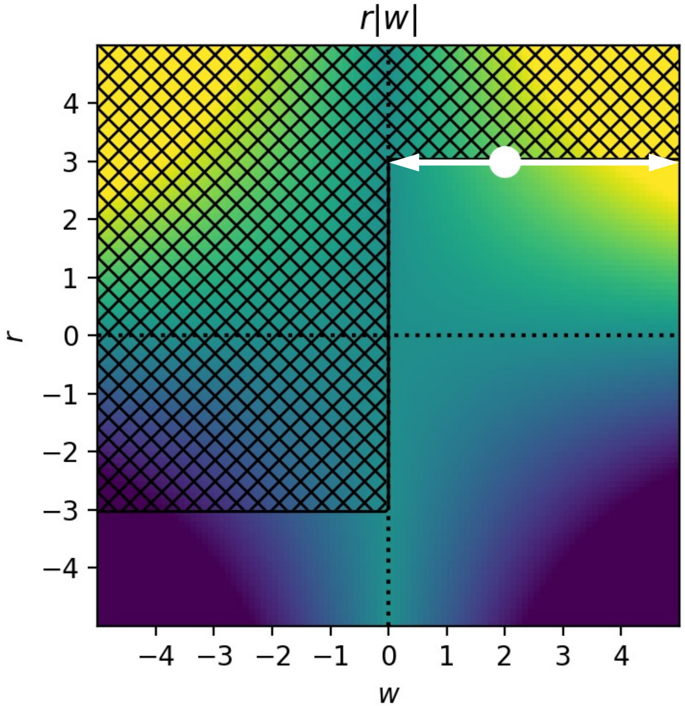


$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$
$= \{(2, -1), (8, +1)\}$

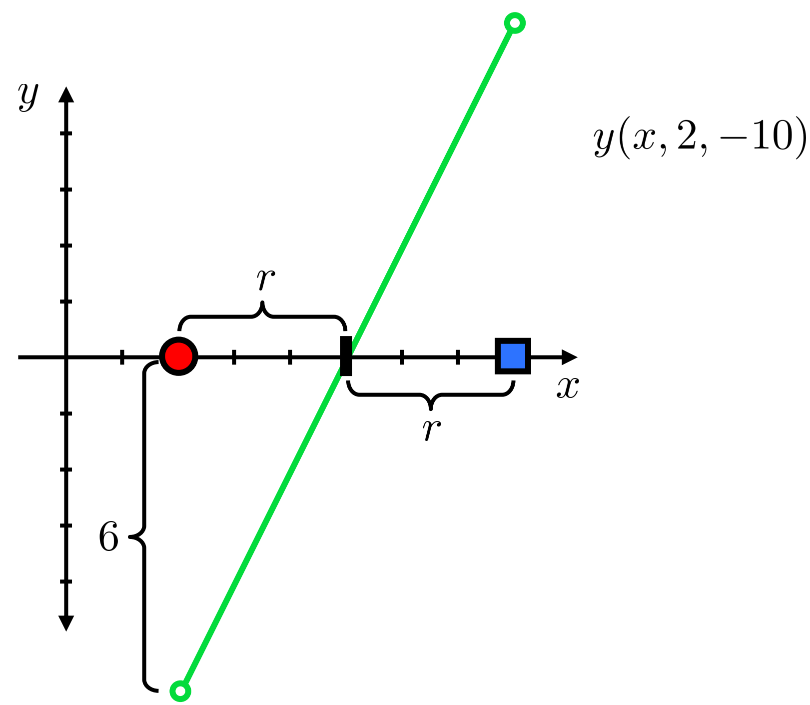
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \quad r\,|w| \le -2w - b \quad \text{(tight)}$$
$$r\,|w| \le 8w + b \quad \text{(tight)}$$



Rescaling w, b gives equivalent solutions!



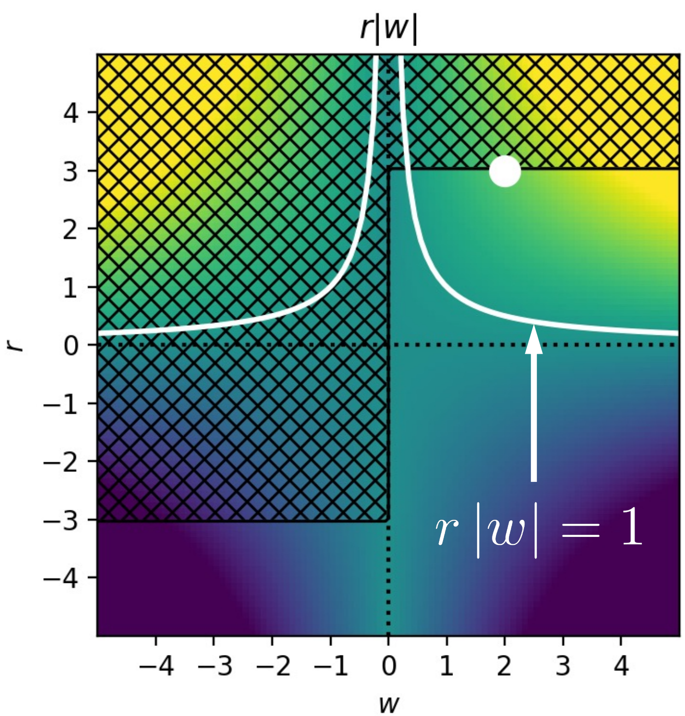
where $b = -5w$ (intercept held constant at 5)

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

Towards quadratic programming

Can we understand what we did using our toy 1D example?

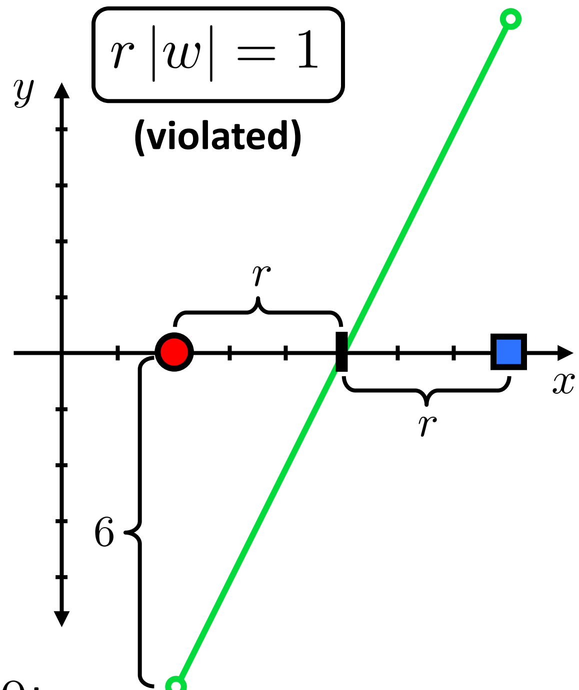
$$\max_{w,b,r} \; r$$
$$\text{s.t.} \; r\,|w| \le -2w - b \quad \text{(tight)}$$
$$r\,|w| \le 8w + b \quad \text{(tight)}$$
$$r\,|w| = 1 \quad \text{(violated)}$$



where $b = -5w$ (intercept held constant at 5)

For every $r > 0$ there is now a unique w, b corresponding.

BUT if data not separable, now infeasible, since can't have $r \le 0$!

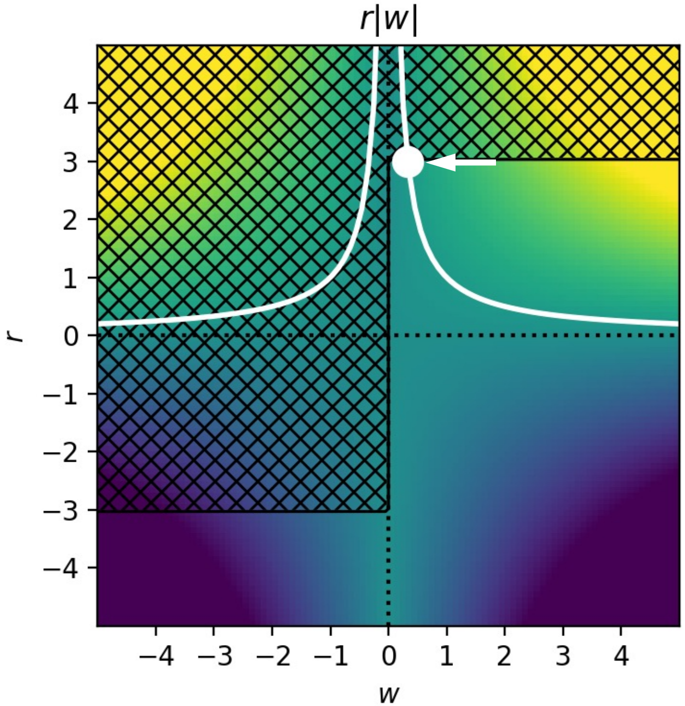


$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\}$$
$$= \{(2, -1), (8, +1)\}$$

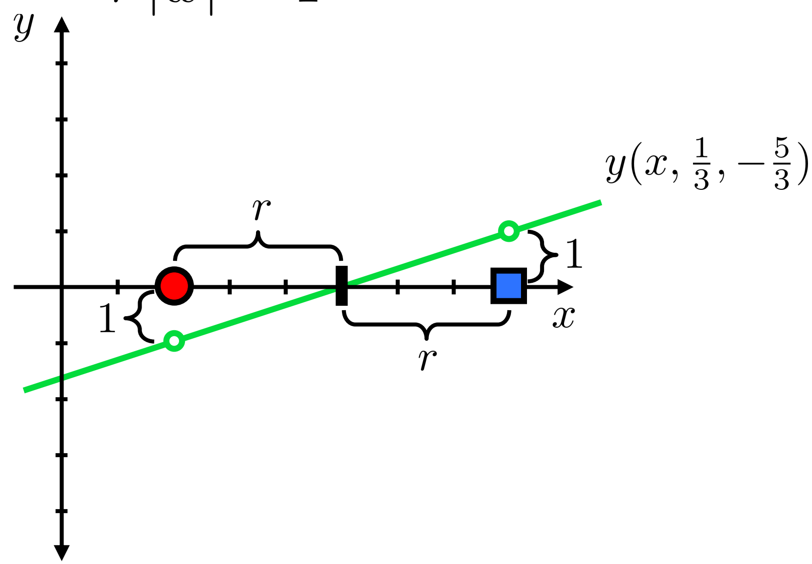
Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\max_{w,b,r} \; r$$
$$\text{s.t.} \; r\,|w| \leq -2w - b \quad \text{(tight)}$$
$$r\,|w| \leq \;\; 8w + b \quad \text{(tight)}$$
$$r\,|w| = 1$$



we choose to optimize *only* over subspace satisfying $r\,|w| = 1$



where $b = -5w$ (intercept held constant at 5)

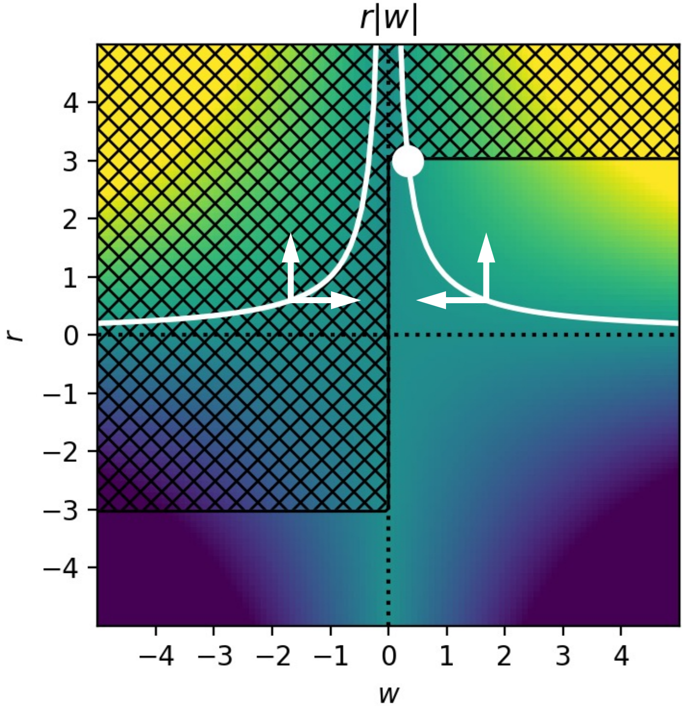
$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

Towards quadratic programming

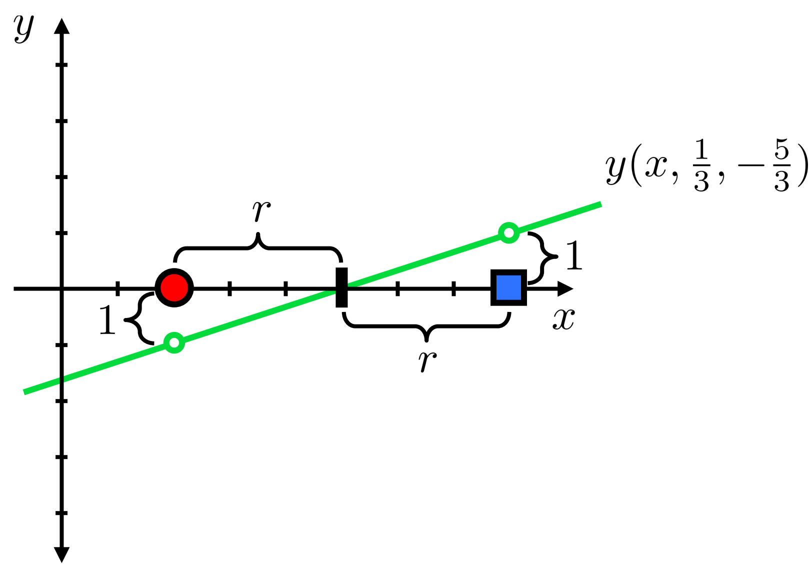
Can we understand what we did using our toy 1D example?

$$\min_{w,b} \ \frac{1}{2}w^2$$
$$\text{s.t.} \ 1 \leq -2w - b$$
$$1 \leq \ 8w + b$$

Linear SVM formulation for our 1D toy problem!!



maximizing r equivalent to minimizing $|w|$, so equivalent to minimizing w^2



where $b = -5w$ (intercept held constant at 5)

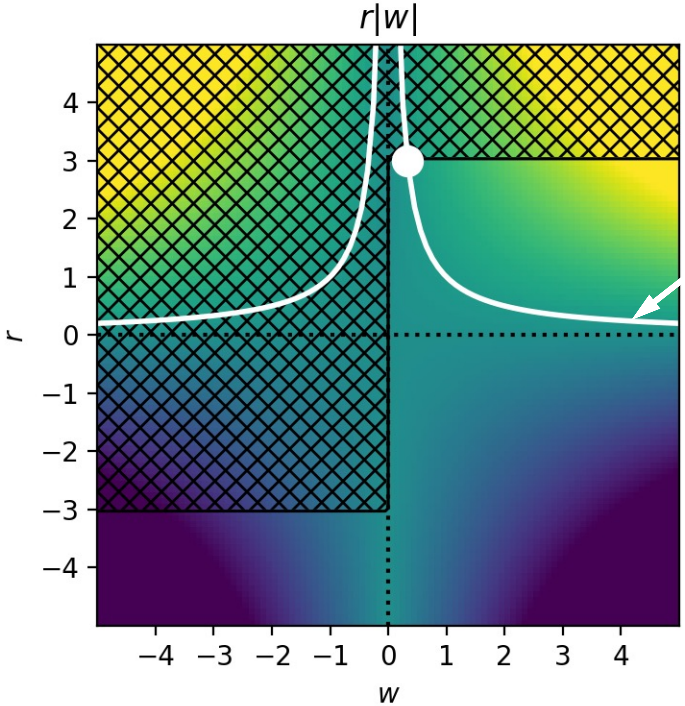
$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

Towards quadratic programming

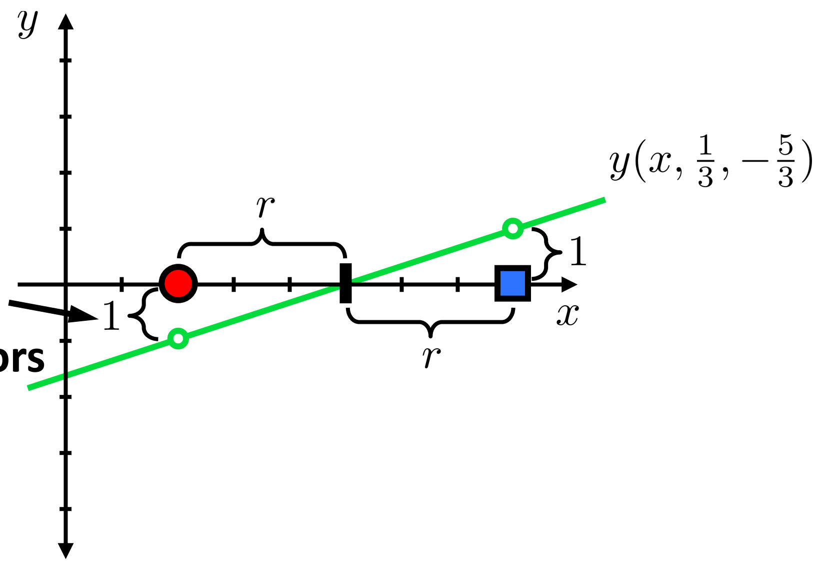
Can we understand what we did using our toy 1D example?

$$\min_{w,b} \ \frac{1}{2}w^2 \quad \text{s.t.} \quad 1 \leq -2w - b, \quad 1 \leq 8w + b$$

Linear SVM formulation for our 1D toy problem!!



optimizing over this subspace **forces $y = \pm 1$ at support vectors** *by definition*



where $b = -5w$ (intercept held constant at 5)

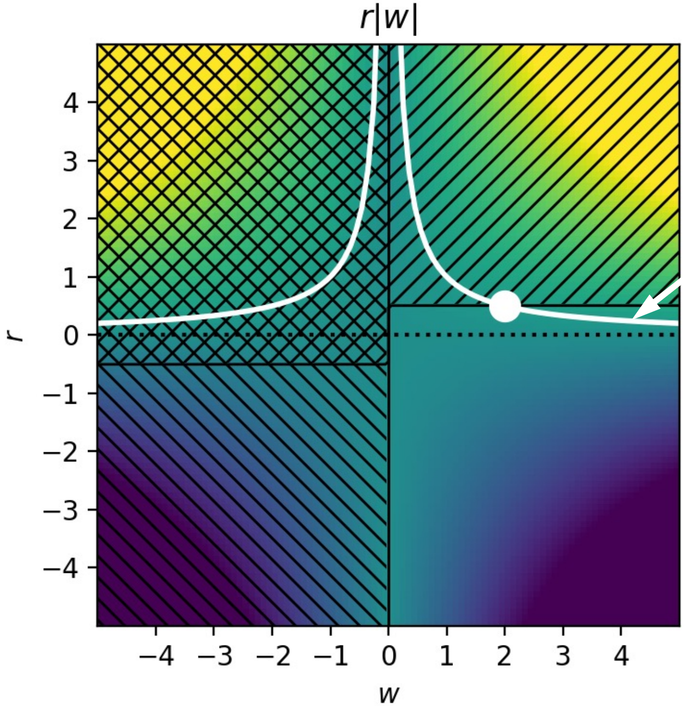
$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

Towards quadratic programming

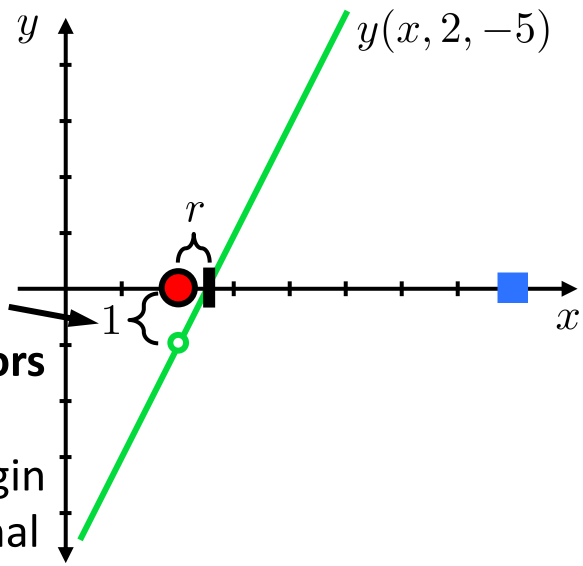
Can we understand what we did using our toy 1D example?

$$\min_{w,b} \frac{1}{2}w^2$$
$$\text{s.t. } 1 \leq -2w - b$$
$$1 \leq 8w + b$$

Linear SVM formulation for our 1D toy problem!!



optimizing over this subspace **forces** $y = \pm 1$ **at support vectors** *by definition*, even when margin is not yet maximal



where $b = -\frac{5}{2}w$ (intercept held constant at 2.5)

$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

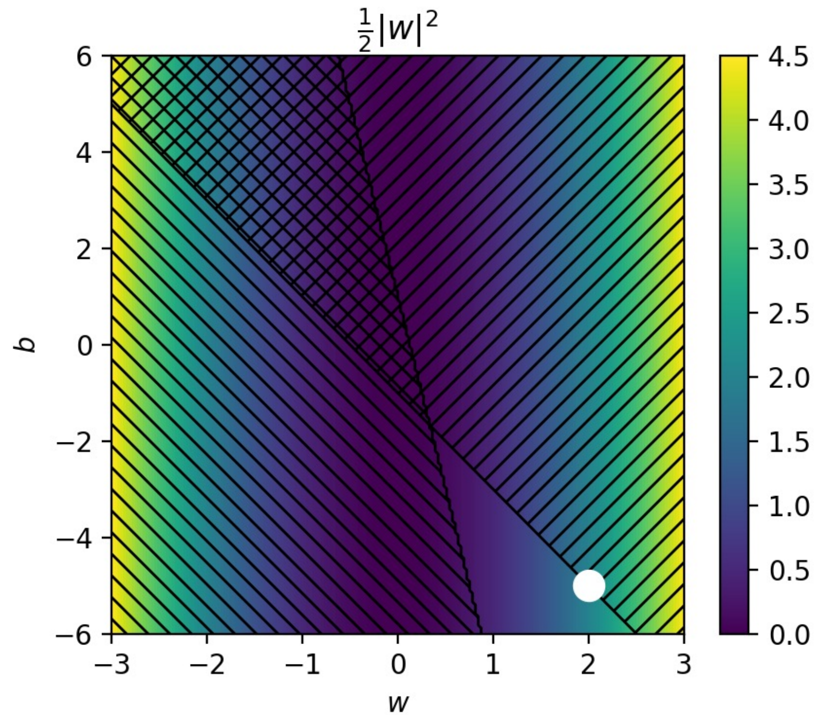
Towards quadratic programming

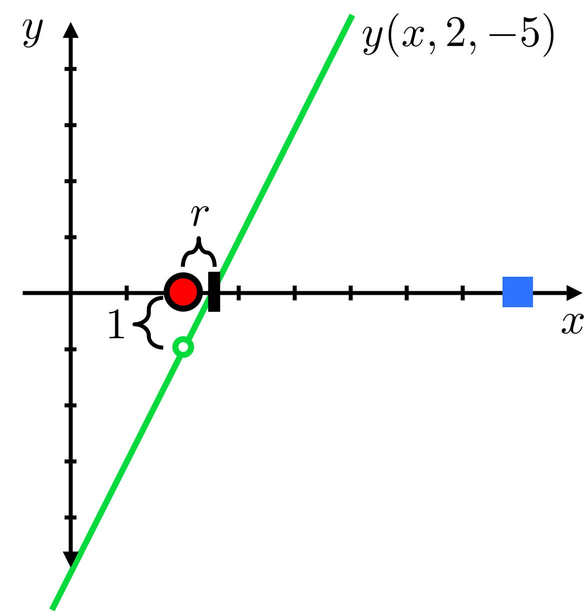
Can we understand what we did using our toy 1D example?

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^2 \\ \text{s.t.} \quad & 1 \le -2w - b \\ & 1 \le \quad 8w + b \end{aligned}$$

(tight)

(not tight)





$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

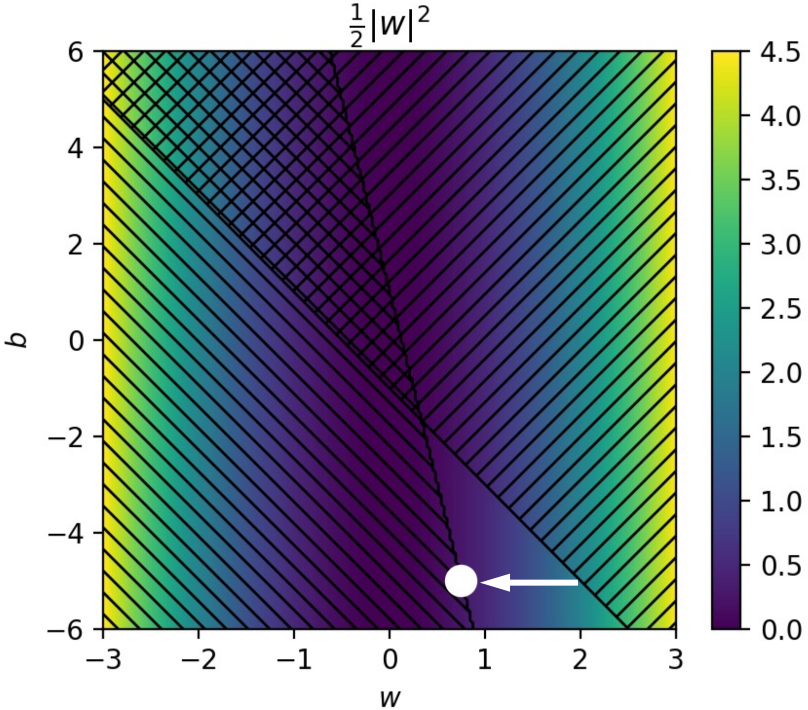
Towards quadratic programming

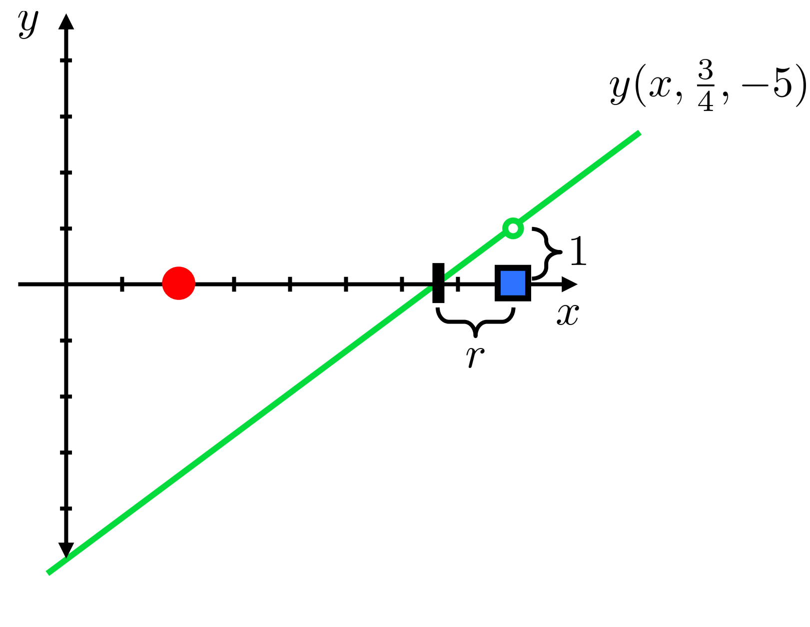
Can we understand what we did using our toy 1D example?

$$\min_{w,b} \ \frac{1}{2}w^2$$
$$\text{s.t.} \ 1 \leq -2w - b$$
$$1 \leq 8w + b$$

(not tight)

(tight)



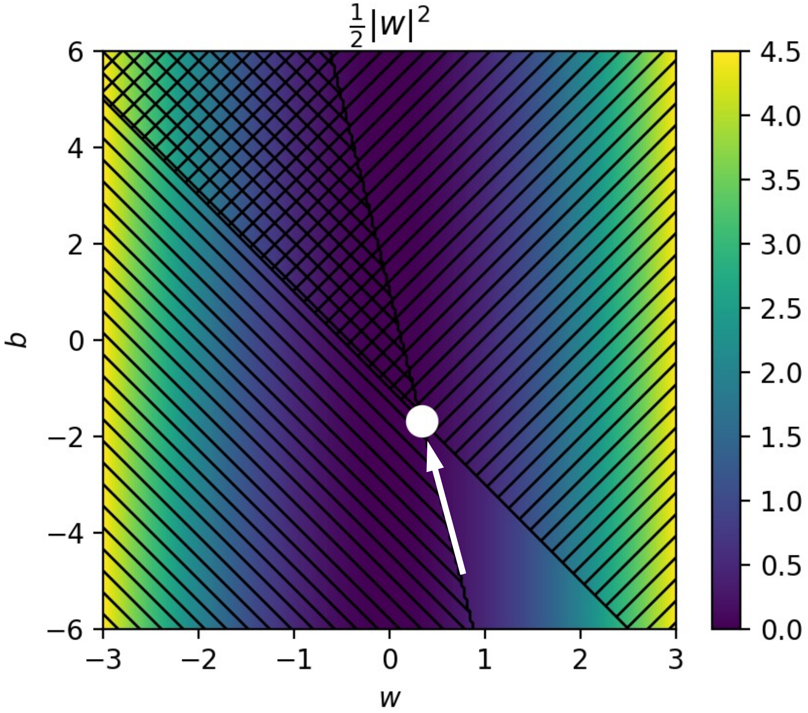


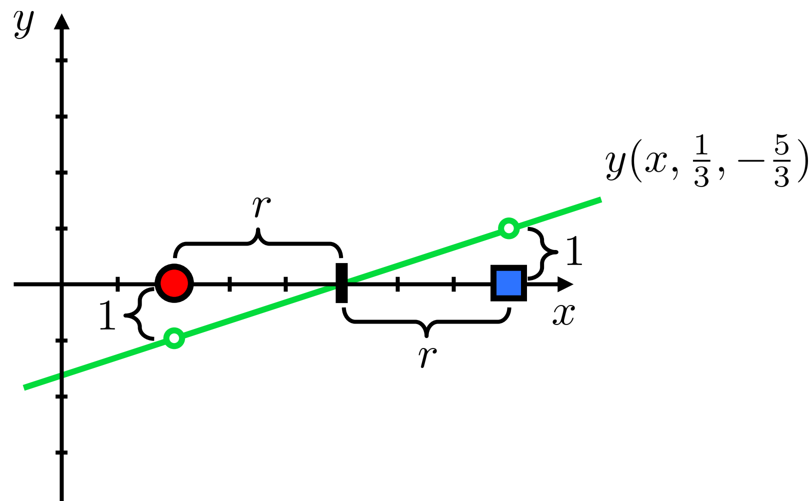
$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2)\} = \{(2, -1), (8, +1)\}$$

Towards quadratic programming

Can we understand what we did using our toy 1D example?

$$\min_{w,b} \ \frac{1}{2}w^2$$
$$\text{s.t. } 1 \leq -2w - b \quad \textbf{(tight)}$$
$$1 \leq 8w + b \quad \textbf{(tight)}$$





Notice the training objective $\frac{1}{2}w^2$ is convex!
Also true of $\frac{1}{2}\|\mathbf{w}\|^2$ in higher dimensions.
That means we can find a global optimum!

So what have we done?

Linear SVM for separable data

$$\max_{\mathbf{w},b,r} r \text{ such that } r\|\mathbf{w}\| \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \text{ for } i = 1, \ldots, N$$

$$r\|\mathbf{w}\| = 1$$ (we *added* this constraint)

which simplifies to

$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|} \text{ such that } 1 \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \text{ for } i = 1, \ldots, N$$

which is equivalent to

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \text{ such that } 1 \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \text{ for } i = 1, \ldots, N$$

💪 which we can apply quadratic programming solvers to!!

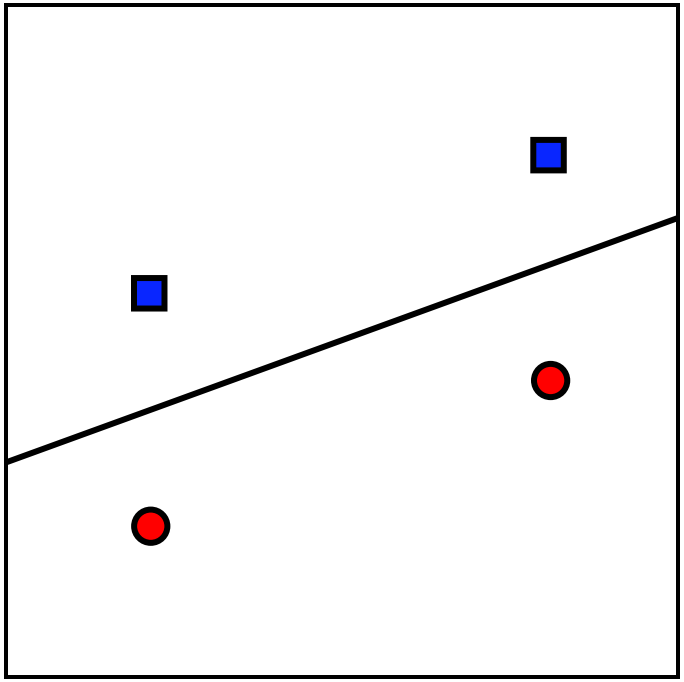
Linear SVM with Hard Margin

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad 1 \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \quad \forall\, i = 1, \ldots, N$$

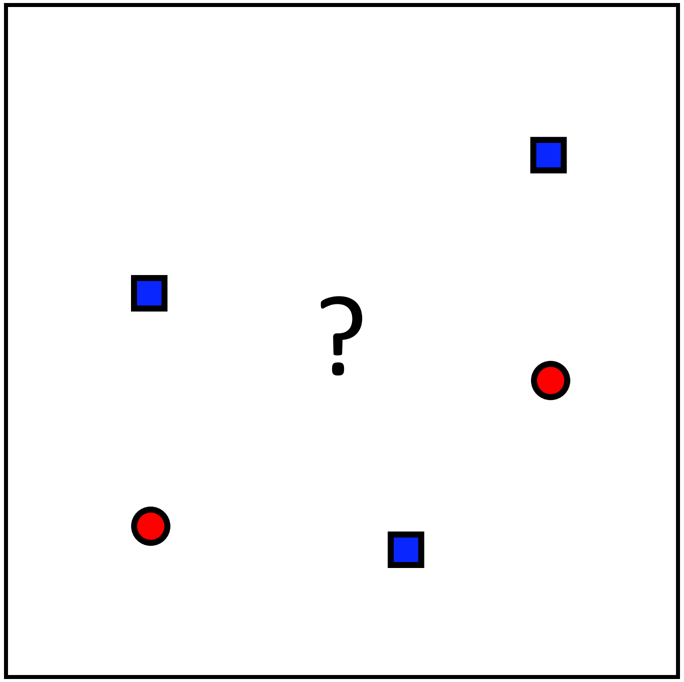
- This is called a hard margin linear SVM formulation.
- If data non-separable, then no $\mathbf{w}, b$ can satisfy all $1 \leq t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)$ simultaneously.
    - Their intersection in $(\mathbf{w}, b)$-space is an *empty set.*
- In that case, a quadratic programming solver will report the problem instance as being '*infeasible*'
    - No useful $\mathbf{w}, b$ will be computed.
    - This is what we "gave up" by assuming $r = \frac{1}{\|\mathbf{w}\|}$

What about *non*-separable data?

separable

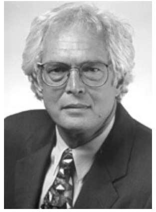


non-separable

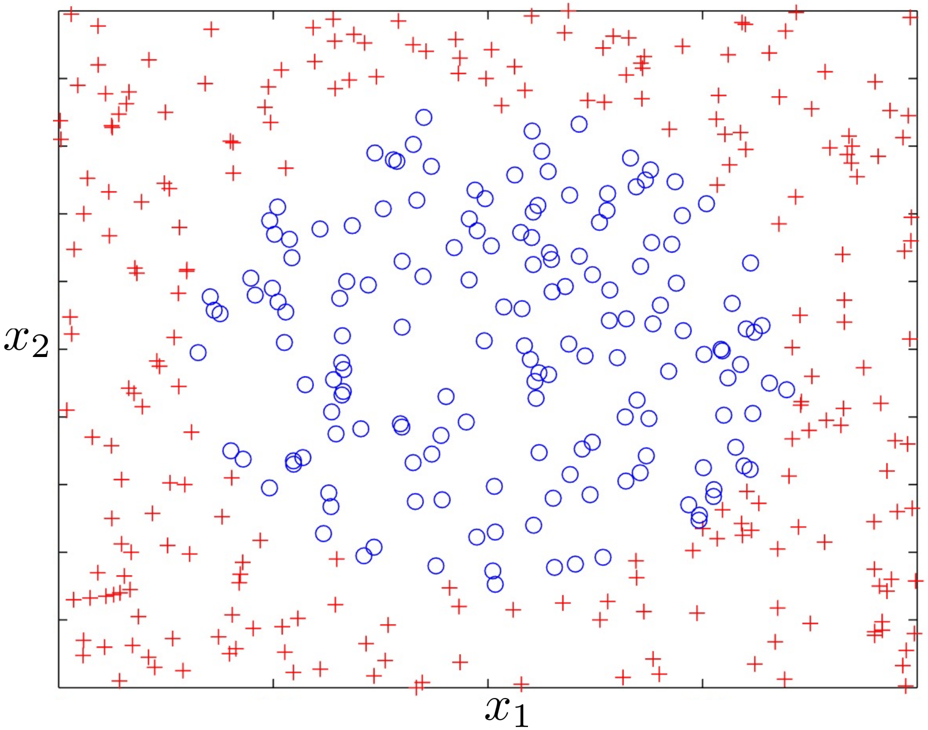


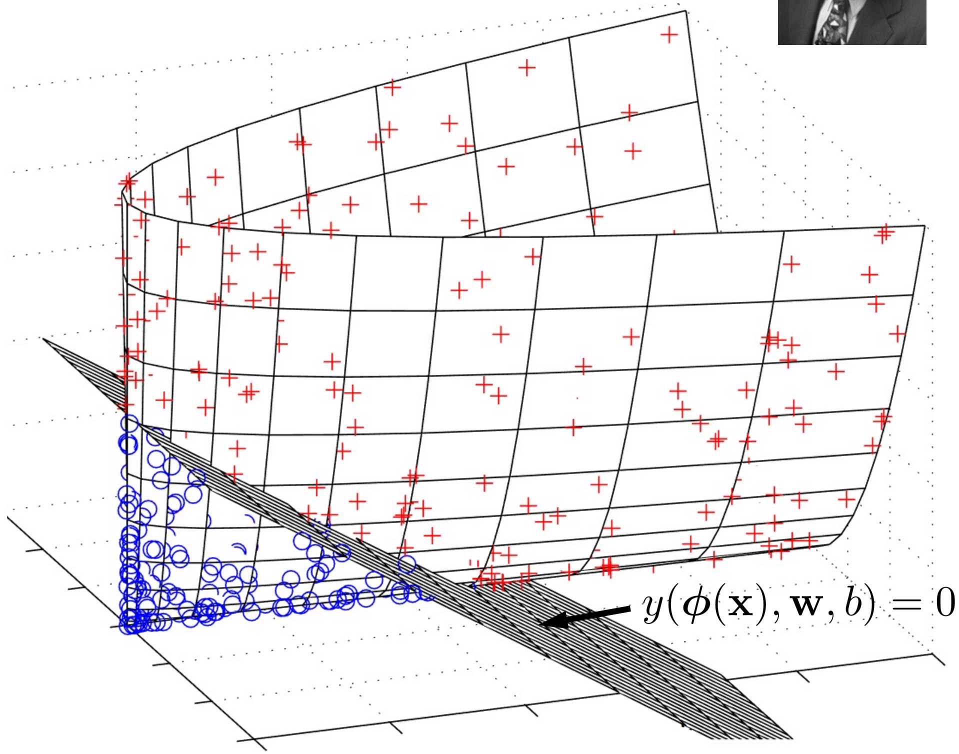
What about *non*-separable data?

Cover's theorem



- **Option 1:** increase the dimensionality via some non-linear feature transformation





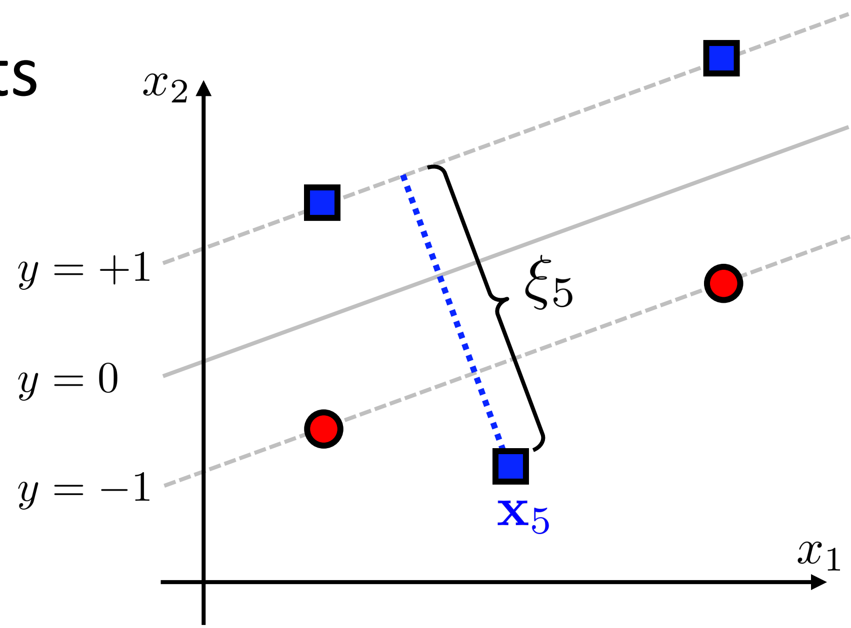
$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}^T$$

Image source: Wikimedia Commons

What about *non*-separable data?

- **Option 2:** introduce an SVM formulation that merely <u>penalizes</u> non-separation, rather than <u>forbidding</u> it.
  - Doesn't magically make data separable, but at least gives us a useful solution $\mathbf{w}, b$ when data is non-separable!
- **Idea:** allow margin constraints to be violated, but introduce variable $\xi_i \geq 0$ to measure *how violated* constraint i is, if at all.
- Each constraint becomes:

$$1 - \xi_i \leq t_i \left(\mathbf{w}^T \mathbf{x}_i + b \right)$$



ξ = "ksi"

Linear Soft Margin SVM

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{subject to} \quad 1-\xi_i \le t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right),$$

$$\xi_i \ge 0 \quad \forall\, i = 1,\ldots,N$$

- Now, for every possible $\mathbf{w}, b$ there exists a setting of **slack variables** ξ_i that make the constraints feasible.
- There is also a 'force' of strength $C > 0$ pushing each slack variable ξ_i to be small (*encourages* constraint *i*).
  - As $C \to \infty$, tightens to data, reducing to hard-margin SVM
- Still a quadratic program with linear constraints!

Non-Linear Soft-Margin SVM

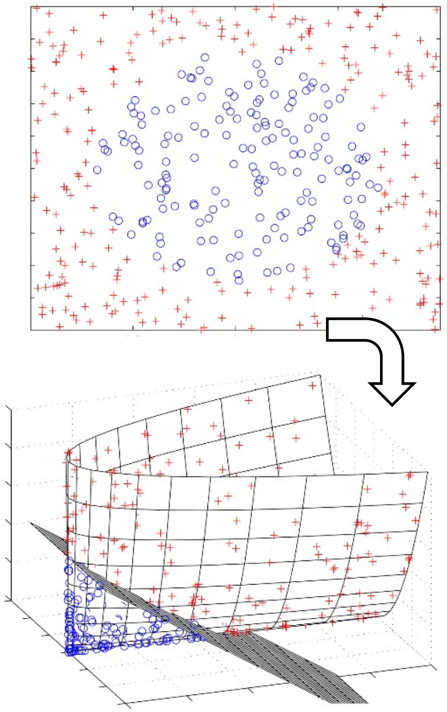
- **Idea:** apply non-linear transformation to features like we did for linear models.

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_D \end{bmatrix}^T$$

$$\boldsymbol{\phi} = \begin{bmatrix} \phi_1(\mathbf{x}) & \phi_2(\mathbf{x}) & \cdots & \phi_M(\mathbf{x}) \end{bmatrix}^T$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_M \end{bmatrix}^T$$

- Replace features! Easy! Are we done yet?



$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \; \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\text{subject to} \quad 1 - \xi_i \leq t_i \left(\mathbf{w}^T \boldsymbol{\phi}_i + b\right), \quad \xi_i \geq 0 \quad \forall\, i = 1, \ldots, N$$

where we precompute all

$$\boldsymbol{\phi}_i = \boldsymbol{\phi}(\mathbf{x}_i)$$

before formulating the actual SVM instance

The SVM formulations so far don't scale with number of features

- Suppose we want to use LOTS of features, and then tune regularization term C to prevent over-fitting, rather than hard-limiting our features.
- Example: Polynomial basis with all cross-terms

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$$

If we want polynomials up to degree d from our D original features, new dimension M is $O(D^d)$!

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & x_1^2 & x_1 x_2 & x_2^2 & \cdots & x_1^2 x_2^3 & x_1 x_2^4 & x_2^5 \end{bmatrix}^T$$

- To specify our SVM training objective we must explicitly build this entire $N \times M$ matrix inside the computer!

$$\mathbf{\Phi} = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix}$$

Towards a scalable SVM formulation

Sketch of the plan:

1. Write an equivalent "dual" formulation of our current SVM training problem (the "primal").
2. Write our original hyperplane variables $\mathbf{w}, b$ in terms of the new "dual variables" $\mathbf{a}$.
3. Explain the "kernel trick" and how by optimizing over dual variables we avoid computing $\mathbf{\Phi}$ matrix.
4. Show that we can recover optimal $\mathbf{w}, b$ from the optimal $\mathbf{a}$ values after optimization completes.

1. Write dual of hard-margin SVM

Primal formulation of hard-margin SVM training (rearranged).

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \ 1 - t_i y(\mathbf{x}_i) \leq 0 \quad \forall\, i = 1, \ldots, N$$

$$\min_{\mathbf{w},b} \ \max_{\mathbf{a}\geq 0} \ \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} a_i (1 - t_i y(\mathbf{x}_i))$$

Equivalent formulation of hard-margin SVM training.

We have introduced “Lagrange multipliers” $\mathbf{a} = \begin{bmatrix} a_1 & \cdots & a_N \end{bmatrix}$, one for each constraint of form $f(\mathbf{w}, b) \leq 0$ in the primal.

Remember: $y(\mathbf{x})$ is really $y(\mathbf{x}, \mathbf{w}, b) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b$, so a function of $\mathbf{w}, b$.

1. Write dual of hard-margin SVM

Why are these equivalent problems?

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad 1 - t_i y(\mathbf{x}_i) \leq 0 \quad \forall\, i = 1, \ldots, N$$

$$\min_{\mathbf{w},b} \max_{\mathbf{a}\geq 0} \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} a_i (1 - t_i y(\mathbf{x}_i))$$

$$1 - t_i y(\mathbf{x}_i) \leq 0 \quad \Leftrightarrow \quad \max_{a_i \geq 0} a_i (1 - t_i y(\mathbf{x}_i)) = 0$$

$$1 - t_i y(\mathbf{x}_i) > 0 \quad \Leftrightarrow \quad \max_{a_i \geq 0} a_i (1 - t_i y(\mathbf{x}_i)) = +\infty$$

If the primal is feasible, the dual cannot be at a minimum unless $\mathbf{w}, b$ satisfy *all* $\leq$ constraints.

You do not need to understand "Slater's condition" for this course, just take it on faith

1. Write dual of hard-margin SVM

If data separable, primal is *strictly* feasible ("Slater's condition") ...

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad 1 - t_i y(\mathbf{x}_i) \leq 0 \quad \forall\, i = 1, \ldots, N$$

"for fixed $\mathbf{w}, b$ maximize over $\mathbf{a}$"

$$\min_{\mathbf{w},b} \max_{\mathbf{a}\geq 0} \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} a_i \left(1 - t_i y(\mathbf{x}_i)\right)$$

By "Slater's condition," can swap min-max for max-min and *still* be equivalent!

$$\max_{\mathbf{a}\geq 0} \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} a_i \left(1 - t_i y(\mathbf{x}_i)\right)$$

"for fixed $\mathbf{a}$ minimize over $\mathbf{w}, b$"

2. Write $\mathbf{w}$ in terms of dual vars $\mathbf{a}$ for hard-margin SVM

For a fixed setting of dual variables $\mathbf{a}$, can the optimal setting $\mathbf{w}^*$ be expressed in closed form?

Let $\ell(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} a_i \left(1 - t_i y(\mathbf{x}_i)\right)$

Then $$\nabla_{\mathbf{w}} \ell(\mathbf{w}, b, \mathbf{a}) = \nabla_{\mathbf{w}} \left[\frac{1}{2}\|\mathbf{w}\|^2\right] + \sum_{i=1}^{N} \nabla_{\mathbf{w}} \left[a_i \left(1 - t_i \underbrace{y(\mathbf{x}_i)}_{\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b}\right)\right]$$

$$= \mathbf{w} - \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

Setting gradient to zero gives $\mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$ Yes!

2. Write b in terms of dual vars $\mathbf{a}$ for hard-margin SVM

For a fixed setting of dual variables $\mathbf{a}$, can the optimal setting b^* be expressed in closed form?

Take $\frac{\partial \ell}{\partial b}(\mathbf{w}, b, \mathbf{a}) = \frac{\partial \ell}{\partial b}\left[\frac{1}{2}\|\mathbf{w}\|^2\right] + \sum_{i=1}^{N} \frac{\partial \ell}{\partial b}\left[a_i\left(1 - t_i y(\mathbf{x}_i)\right)\right]$

$y(\mathbf{x}_i)$: $\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b$

$$= 0 - \sum_{i=1}^{N} a_i t_i$$

Setting derivative to zero gives an additional constraint on the dual problem:

$$\sum_{i=1}^{N} a_i t_i = 0$$

Not expression for b^* itself, but dual variables must satisfy *this* for b^* to be feasible.

2. Simplifying the dual formulation

Use $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b$ and separate the sums of $\ell(\mathbf{w}, b, \mathbf{a})$

$$\ell(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} a_i - \sum_{i=1}^{N} a_i t_i \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) - \sum_{i=1}^{N} a_i t_i b$$

$$= \sum_{i=1}^{N} a_i + \frac{1}{2}\mathbf{w}^T \left(\mathbf{w} - 2\sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i) \right) - (0)b$$

$$\sum_{i=1}^{N} a_i t_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

$$= \sum_{i=1}^{N} a_i + \frac{1}{2}\mathbf{w}^T (\mathbf{w} - 2\mathbf{w})$$

$$= \sum_{i=1}^{N} a_i - \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$= \sum_{m=1}^{M} \left(\sum_{i=1}^{N} a_i t_i \phi_m(\mathbf{x}_i) \right) \left(\sum_{j=1}^{N} a_j t_j \phi_m(\mathbf{x}_j) \right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j t_i t_j \left(\sum_{m=1}^{M} \phi_m(\mathbf{x}_i) \phi_m(\mathbf{x}_j) \right)$$

$$= \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j t_i t_j \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j)$$

Defined in terms of only dual variables and inner products!

2. Final dual formulation of hard-margin SVM training

Dual formulation of hard-margin SVM training, **final form**:

$$\max_{\mathbf{a} \geq 0} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j t_i t_j \underbrace{k(\mathbf{x}_i, \mathbf{x}_j)}_{\text{kernel function}}$$

$$\text{subject to } \sum_{i=1}^{N} a_i t_i = 0$$

where $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$ for finite-dimensional feature spaces, or more generally $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ for possibly infinite-dimensional feature space $\phi(\cdot)$.

Still equivalent to primal! ***Still a quadratic program!***

this is why we really went to the trouble of deriving 'dual'

Most importantly, expressed in terms of a kernel, not features!

3. The "Kernel trick"

How does an SVM in terms of $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ rather than $\phi(\mathbf{x}_i)$ help us to 'scale' better?

Reason: We can now train our SVM one of two ways:

$$(N \times M) \quad \mathbf{\Phi} = \begin{bmatrix} \phi_1(\mathbf{x}_1) & \cdots & \phi_M(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \cdots & \phi_M(\mathbf{x}_2) \\ \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \cdots & \phi_M(\mathbf{x}_N) \end{bmatrix}$$

← **For primal formulation.** Good when $N \gg M$, *i.e.* fewer features than training points.

or

$$(N \times N) \quad \mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

The "Gram matrix"

For dual formulation. Good when $N \ll M$, *i.e.* more features than training points, including $M = \infty$, which is the case for the popular "Gaussian kernel"!

3. The "Kernel trick"

Computing $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$ doesn't require us to explicitly compute $\phi(\mathbf{x})$ or $\phi(\mathbf{x}')$, can pre-simplify!

Example: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^2$ ← "Polynomial kernel" of degree 2 with coefficient 1

If $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ then $k(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + 1)^2$

whereas $\phi(\mathbf{x}) = \begin{bmatrix} 1 & \sqrt{2} x_1 & \sqrt{2} x_2 & \sqrt{2} x_1 x_2 & x_1^2 & x_2^2 \end{bmatrix}^T$

is the feature transformation that k corresponds to.

In other words: can just compute the pre-simplified expression $(x_1 x_1' + x_2 x_2' + 1)^2$ directly (the "trick") without ever creating vectors $\phi(\mathbf{x})$ or $\phi(\mathbf{x}')$.

4. Making a prediction

Suppose we *do* find a setting $\mathbf{a} = \begin{bmatrix} a_1 & \cdots & a_N \end{bmatrix}$ that solves the dual SVM formulation.

Then what? How to use $\mathbf{a}$ to make an actual *prediction*?

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b$$

$$= \left(\sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i) \right)^T \boldsymbol{\phi}(\mathbf{x}) + b$$

$$= b + \sum_{i=1}^{N} a_i t_i k(\mathbf{x}_i, \mathbf{x})$$

$$\mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

(from earlier)

Prediction is just a <u>weighted sum of kernel evaluations</u> between $\mathbf{x}$ and training data! Each $\mathbf{x}_i$ influences y in direction t_i with strength proportional to weight a_i and similarity measure $k(\mathbf{x}_i, \mathbf{x})$.

4. Making a prediction

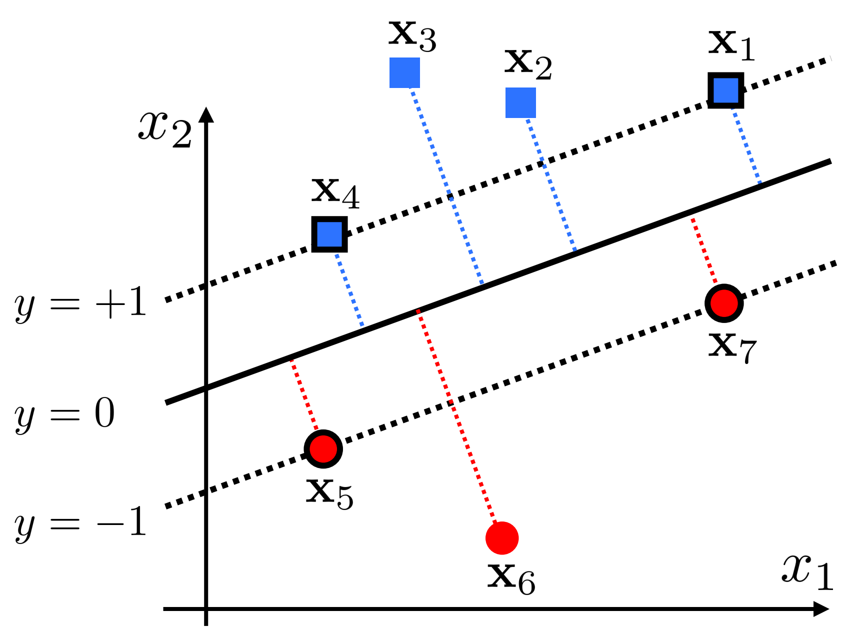
$1 \leq t_i y(\mathbf{x}_i)$

Duality theory tells that constraint i in the primal is **tight** (support vector!) if and only if $a_i > 0$ in the dual.

$$y(\mathbf{x}) = \sum_{i \in \mathcal{S}} a_i t_i k(\mathbf{x}_i, \mathbf{x}) + b \quad \text{where } \mathcal{S} = \{i \,:\, a_i > 0\}$$

Therefore, more specifically:
Prediction is weighted sum of kernel evaluations between $\mathbf{x}$ and the support vectors only!

After training, support vectors need to be remembered, but all other data (with $a_i = 0$) can be discarded!



dual vars

$a_1 > 0$
$a_2 = 0$
$a_3 = 0$
$a_4 > 0$
$a_5 > 0$
$a_6 = 0$
$a_7 > 0$

This SVM only needs to remember 4 data points after training.

4. Making a prediction

Final detail: how do we solve for the intercept b^*?

Observation: any support vector $\mathbf{x}_i$ satisfies $1 = t_i y(\mathbf{x}_i)$

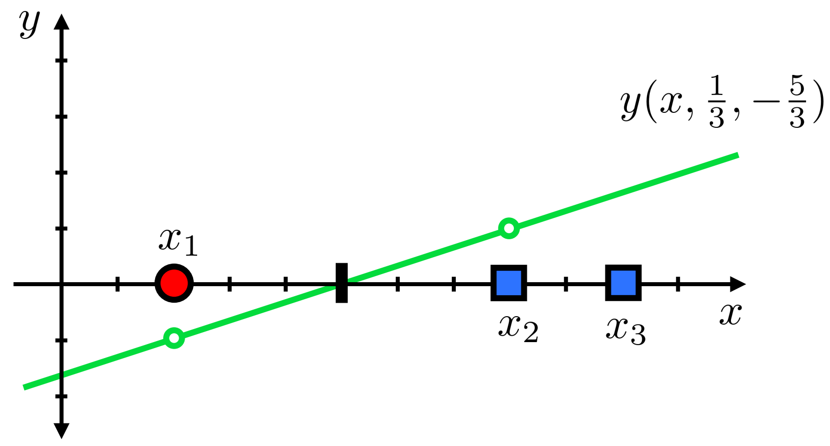
(tight)

$$1 = t_i \left(b + \sum_{j \in S} a_j t_j k(\mathbf{x}_i, \mathbf{x}_j) \right)$$

$$\Rightarrow \boxed{b = t_i - \sum_{j \in \mathcal{S}} a_j t_j k(\mathbf{x}_i, \mathbf{x}_j)} \quad \text{for any choice } i \in \mathcal{S}$$

Therefore, the optimal dual variables $\mathbf{a}^*$ determine the optimal primal variables $\mathbf{w}^*, b^*$.

1D Linear Example



$$\mathcal{D} = \{(x_1, t_1), (x_2, t_2), (x_3, t_3)\}$$
$$= \{(2, -1), (8, +1), (10, +1)\}$$

Primal (hard-margin)

$$\min_{w,b} \frac{1}{2}w^2$$
$$\text{s.t.}\quad 1 \leq -2w - b$$
$$1 \leq 8w + b$$
$$1 \leq 10w + b$$

Dual

$t_i t_j k(x_i, x_j) = t_i t_j x_i x_j$

$$\max_{\mathbf{a} \geq 0} \mathbf{1}^T\mathbf{a} - \frac{1}{2}\mathbf{a}^T \begin{bmatrix} 4 & -16 & -20 \\ -16 & 64 & 80 \\ -20 & 80 & 100 \end{bmatrix} \mathbf{a}$$
$$\text{s.t.}\quad -a_1 + a_2 + a_3 = 0$$

$$\mathbf{a}^* = \begin{bmatrix} \frac{1}{18} & \frac{1}{18} & 0 \end{bmatrix}^T$$
$$\mathcal{S} = \{1, 2\}$$

$$w^* = a_1 t_1 x_1 + a_2 t_2 x_2 = \frac{1}{3}$$
$$b^* = t_1 - a_1 t_1 x_1 x_1 - a_2 t_2 x_2 x_1 = -\frac{5}{3}$$

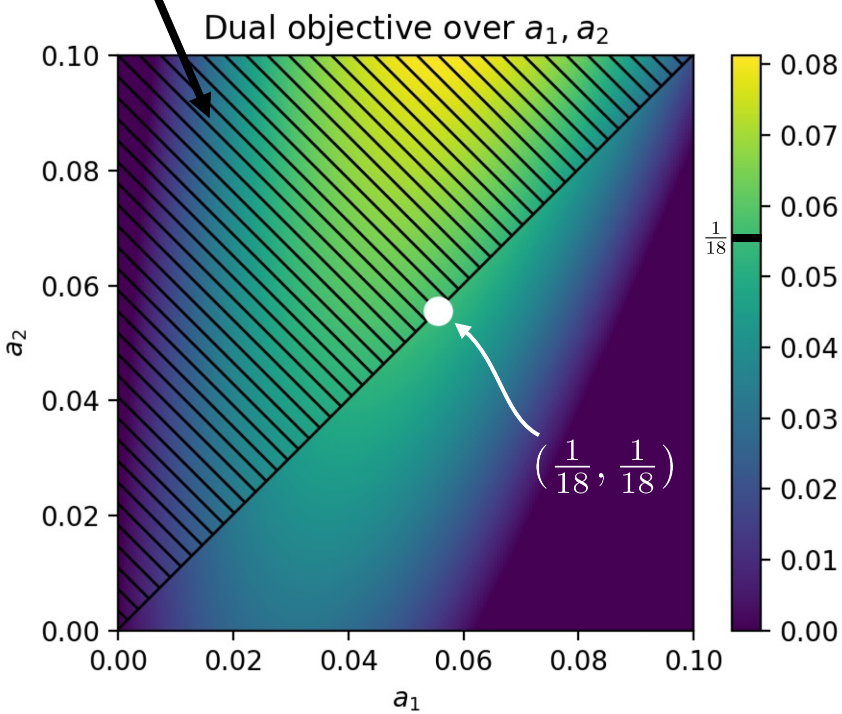
(here we chose to compute *b* with respect to support vector 1)

1D Linear Example (closer look)

Primal objective value for $w^* = \frac{1}{3}$ is $\frac{1}{2}(\frac{1}{3})^2 = \frac{1}{18}$

Dual objective for $\mathbf{a}^*$ is also $\frac{1}{18}$ ("strong duality")

forbidden by constraint $a_3 \geq 0$



Dual (all separate terms)

$$\max_{\mathbf{a} \geq 0} \; a_1 + a_2 + a_3 - 2a_1^2 + 16a_1a_2 + 20a_1a_3 - 32a_2^2 - 80a_2a_3 - 50a_3^2$$

$$\text{s.t.} \quad a_3 = a_1 - a_2$$

$$\mathbf{a}^* = \begin{bmatrix} \frac{1}{18} & \frac{1}{18} & 0 \end{bmatrix}^T$$

Popular kernel functions

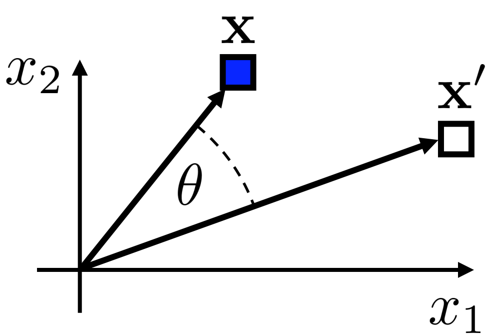
Linear kernel

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- Reduces problem to a Linear SVM.
- Larger value when points by are 'aligned' when treated as vectors

$$\mathbf{x}^T \mathbf{x}' = \|\mathbf{x}\| \, \|\mathbf{x}'\| \cos\theta$$

(bigger when vectors large and aligned)



- Corresponds to $\phi(\mathbf{x}) = \mathbf{x}$

Popular kernel functions

***Polynomial kernel** of degree d with coefficient c*

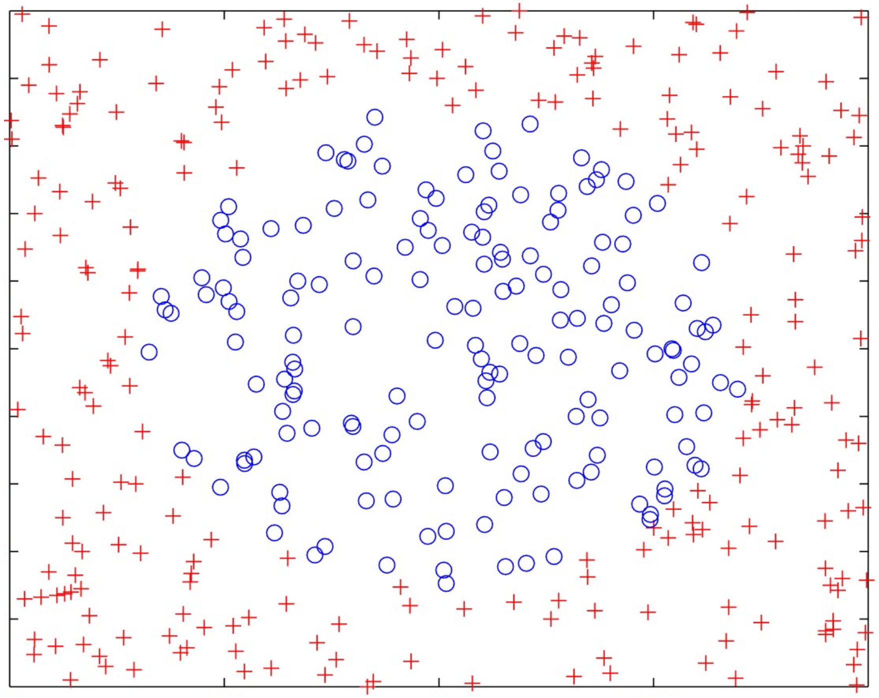
$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d \quad \text{where } c \geq 0, d \in \{1, 2, \ldots\}$$

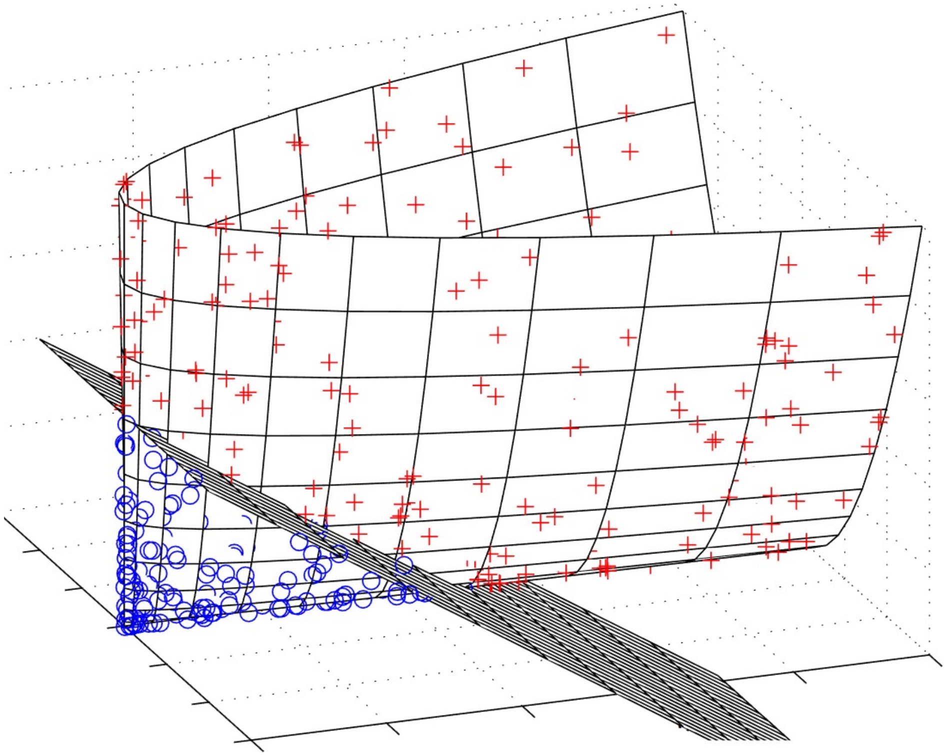
- Popular kernel. Typically use $d = 2$ (up to quadratic).
- Coefficient c scales the low-order terms relative to the highest-order terms.
- For $\mathbf{x} \in \mathbb{R}^D$, $d = 2$ corresponds to features:

$$\boldsymbol{\phi}(\mathbf{x}) = \underbrace{\begin{bmatrix} c & \sqrt{2c}x_1 & \cdots & \sqrt{2c}x_D & \sqrt{2}x_1x_2 & \cdots & \sqrt{2}x_1x_D \\ \sqrt{2}x_2x_3 & \cdots & \sqrt{2}x_2x_D & \cdots & \sqrt{2}x_{D-1}x_D & x_1^2 & \cdots & x_D^2 \end{bmatrix}^T}_{\text{vector of dimension } M = \binom{D+d}{d} \quad (D = 100, d = 4 \ \Rightarrow \ 4.6 \text{ M features!})}$$

Popular kernel functions

Polynomial kernel *of degree* $d = 2$ *, coefficient* $c = 0$





Recall this example.
It was a quadratic kernel!

$$\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & \sqrt{2}x_1x_2 & x_2^2 \end{bmatrix}^T$$

Unlike the Gaussian kernel for "kernel densities," we don't normalize this version because SVMs do not use the kernel as a density.

Popular kernel functions

Also known as Radial Basis Function (RBF) kernel

Gaussian kernel *with spread coefficient* γ

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right) \quad \text{where } \gamma > 0$$

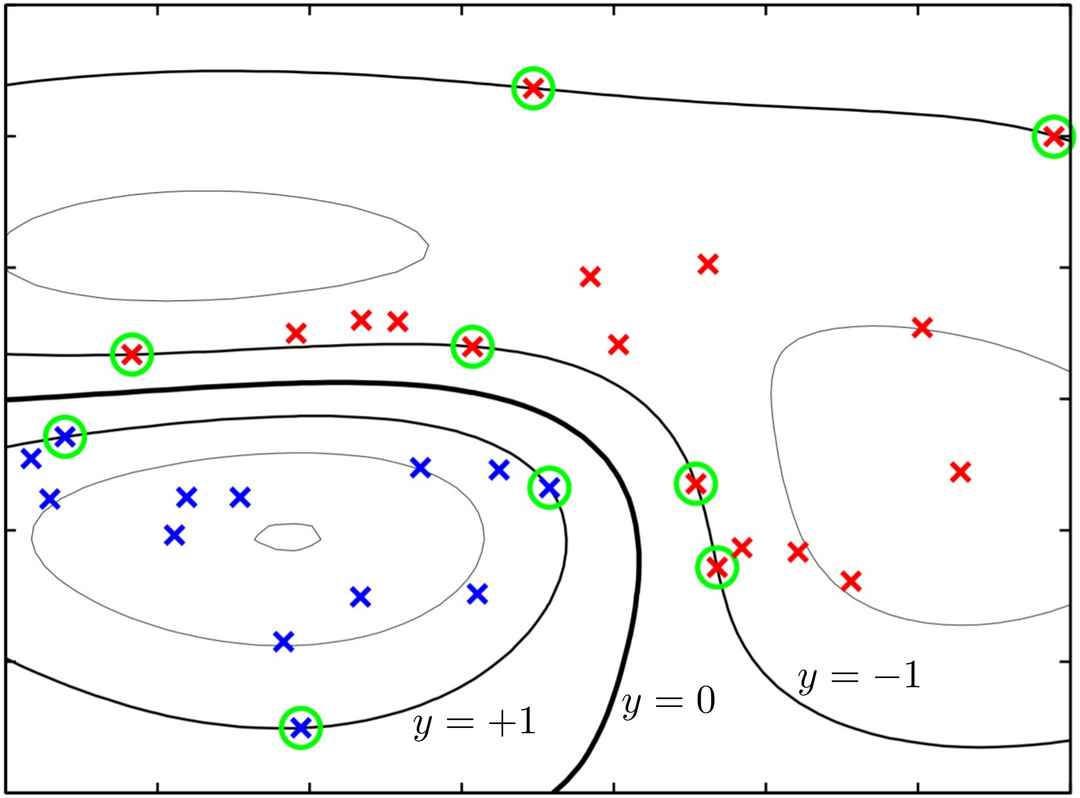
- Default for sci-kit learn! `class sklearn.svm. SVC (C=1.0, kernel='rbf',`
- Coefficient γ controls how far away a training point $\mathbf{x}_i$ can influence the prediction for a new point $\mathbf{x}$.
- For $\mathbf{x} \in \mathbb{R}^D$, corresponds to feature transformation to infinite-dimensional space $\phi(\mathbf{x}) \in \mathbb{R}^\infty$, where the output feature in dimension d involves polynomial kernel of degree d.

You do not need to understand how the infinite-dimensional thing works.

Gaussian kernel

Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

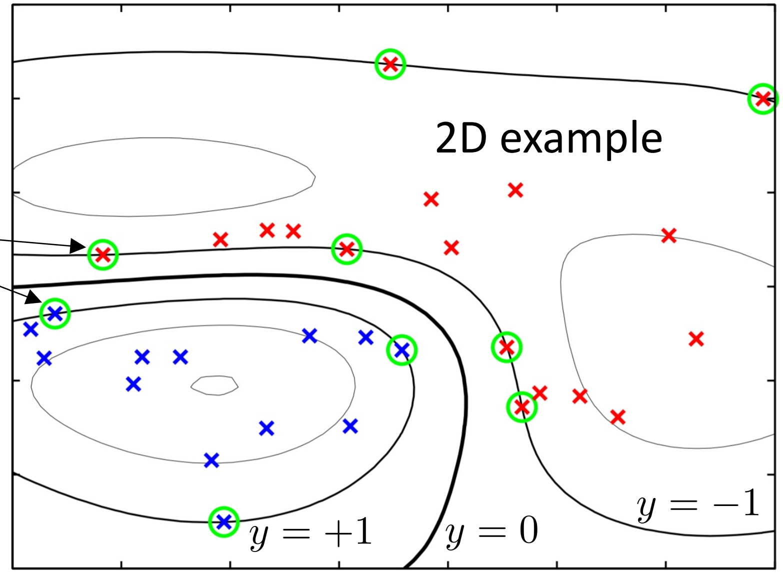
O = support vectors



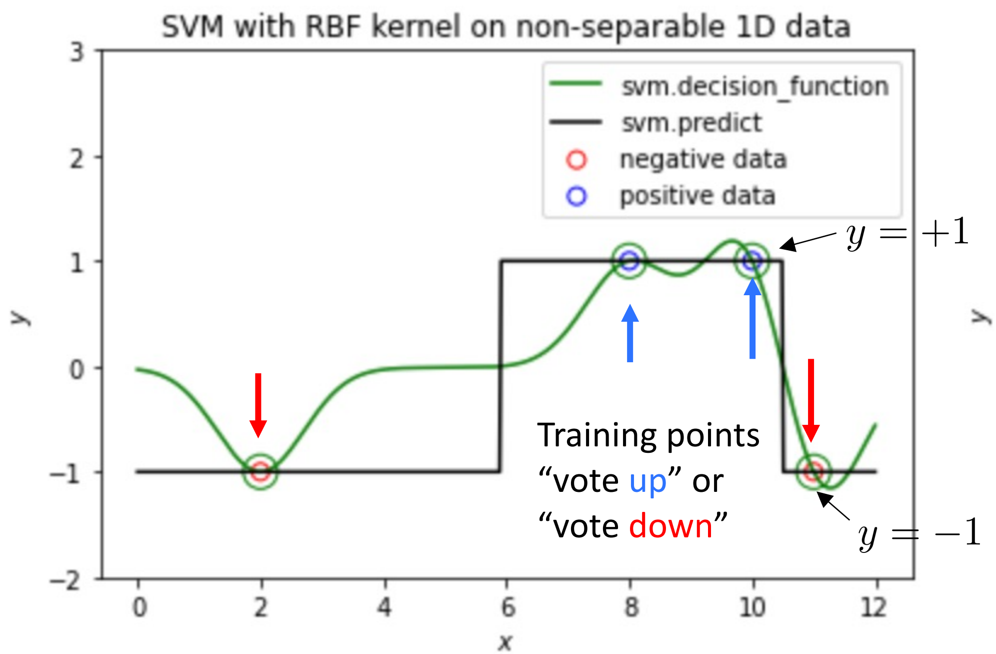
Data non-separable in two dimensions, but separable in the infinite-dimensional space of Gaussian kernel!

Image credit: Christopher M. Bishop

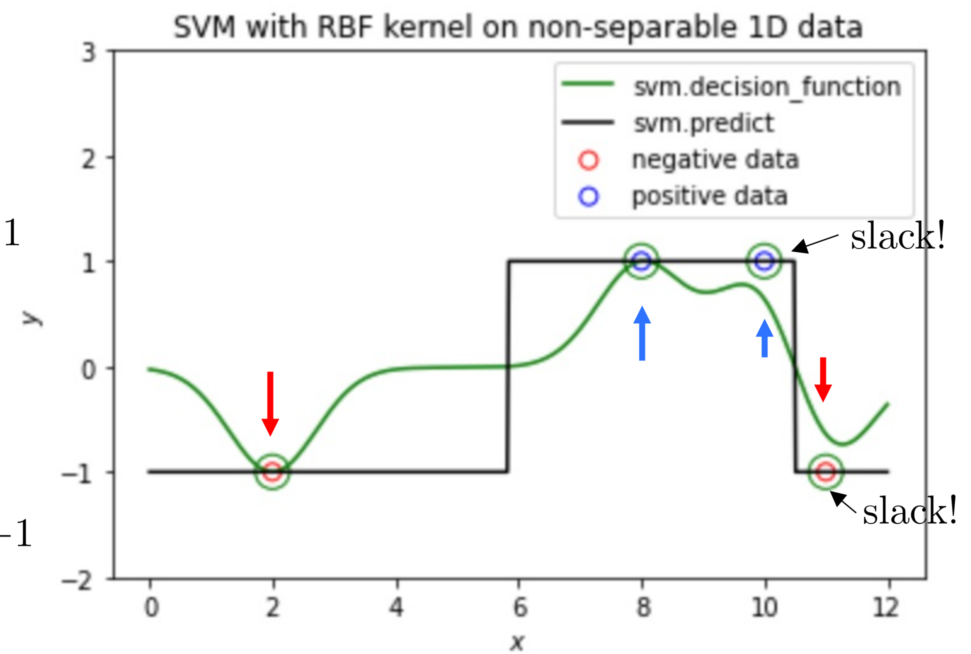
For hard-margin, linear discriminant always takes value {-1, +1} at each support vector



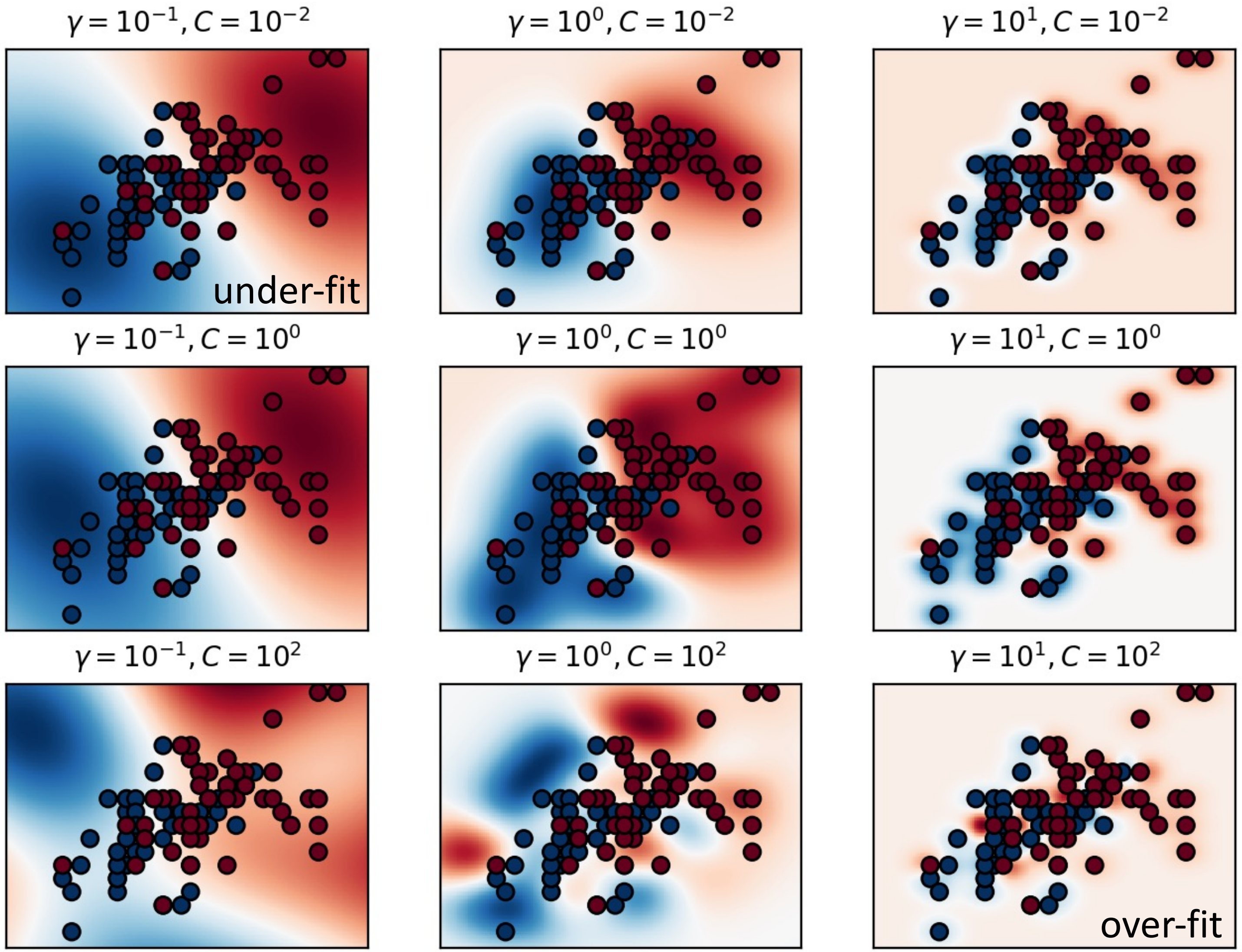
1D example, hard-margin (C = infinity)



1D example, soft-margin (C = 1.0)



Gamma vs *C* for regularization



https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Techniques for Constructing New Kernels.

from Bishop, 6.2

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

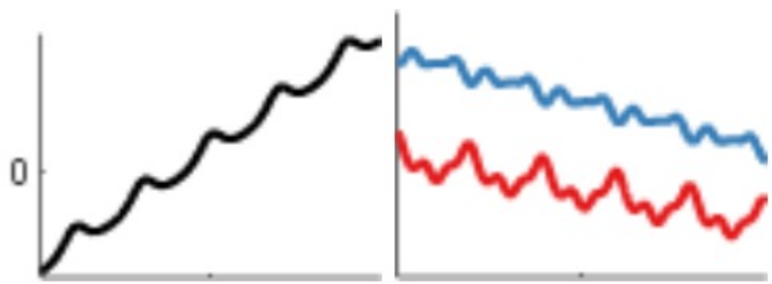
$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

More on kernels to come

- We’ll revisit kernels again when studying Gaussian Processes

Linear plus Periodic



A linear kernel plus a periodic results in functions which are periodic with increasing mean as we move away from the origin.

From https://www.cs.toronto.edu/~duvenaud/cookbook/

SVM Summary

- Advantages:
  - Good generalization principle, theoretical justification
  - Can be formulated as convex quadratic program
  - Can use domain expertise to design good kernels
  - Kernel framework very flexible (vectors, sets, strings)
  - Scales to large (or even infinite) feature spaces
  - Predicts from sparse subset of data (non-parametric)
- Disadvantages:
  - Can be slow to train, sensitive to params, hard to predict
  - Sensitive to feature normalization
- And of course, like any model, can over/under-fit.

Much more to SVMs!

- We explicitly covered:
  - linear hard-margin SVM primal for binary classification
  - linear soft-margin SVM primal for binary classification
  - non-linear hard-margin SVM primal for binary classification
  - non-linear hard-margin SVM dual for binary classification
- We did not cover:
  - soft-margin SVM dual for binary classification (doable!)
  - Hinge-loss formulation of SVM
  - SVM for multi-class classification (k-way etc)
  - SVM for regression
  - Vapnik-Chervonenkis theory (VC theory)
    - VC dimension
    - Generalization bounds

PRML Readings

§4.1.0 Discriminant functions

§4.1.1 Two classes

§6.0.0 Kernel Methods

§6.2.0 Constructing Kernels

- (only up to and including equation 6.23)

§7.0.0 Sparse Kernel Machines

§7.1.0 Maximum Margin Classifiers

§7.1.1 Overlapping class distributions

- (only up to and including equation 7.21, *i.e.*, primal formulation only)