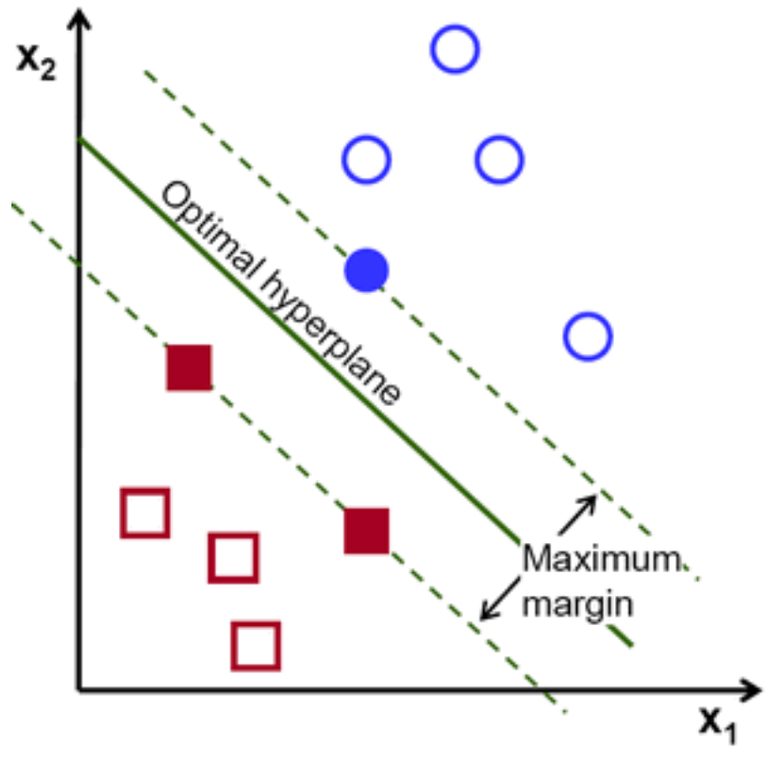
Machine Learning

A Geometric Approach



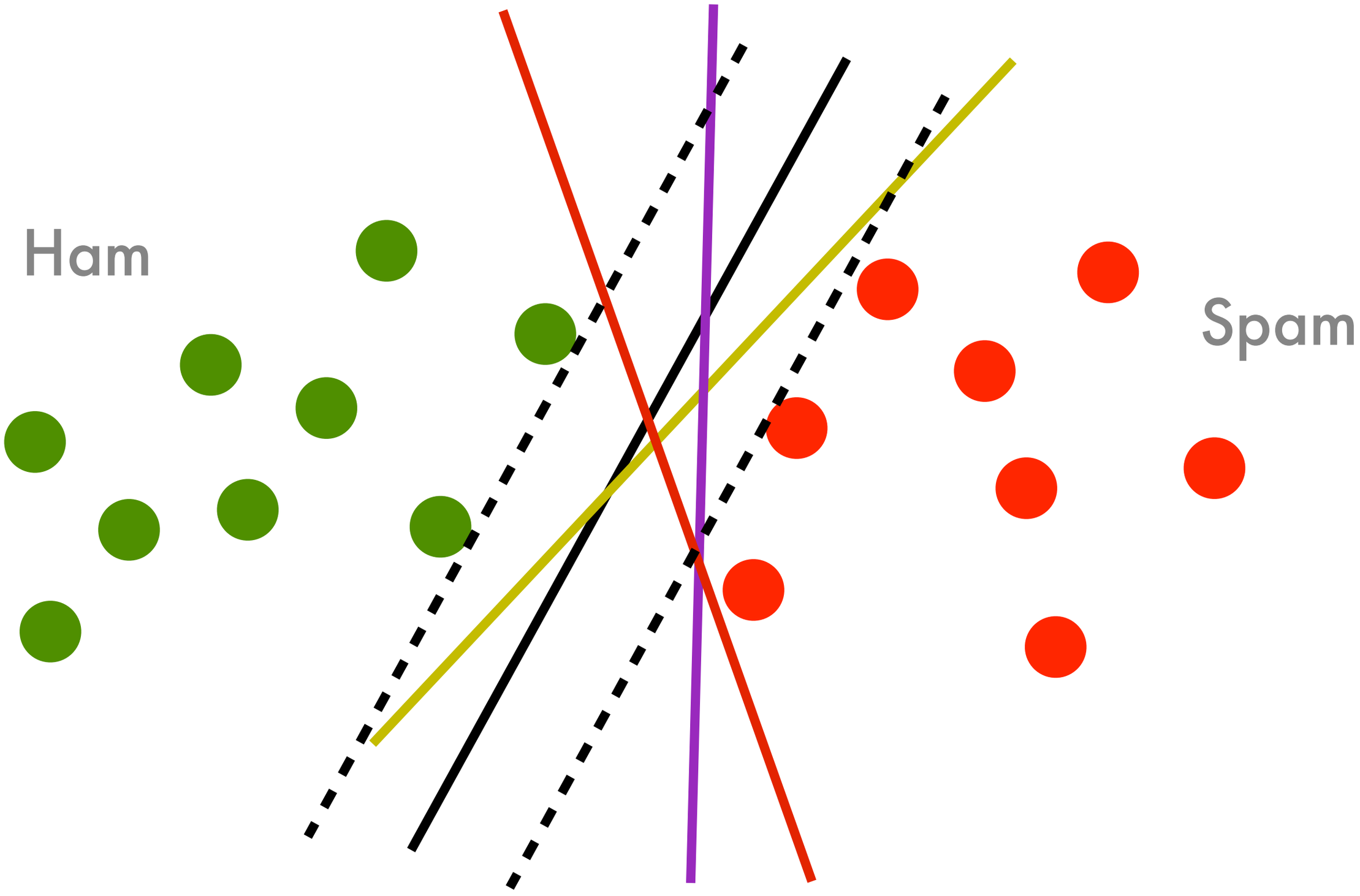
CIML book
Chap 7.7

Linear Classification: Support Vector Machines (SVM)

Professor Liang Huang

some slides from Alex Smola (CMU)

Linear Separator



From Perceptron to SVM

batch

1997
Cortes/Vapnik
SVM

+kernels

+soft-margin

+max margin

1964
Vapnik
Chervonenkis

online approx.
max margin

subgradient descent

2007–2010*
Singer group
Pegasos

minibatch

online

2003
Crammer/Singer
MIRA

2006
Singer group
aggressive

conservative updates

1959
Rosenblatt
invention

1962
Novikoff
proof

1999
Freund/Schapire
voted/avg: revived

inseparable case

2002
Collins
structured

2005*
McDonald/Crammer/Pereira
structured MIRA

fall of USSR

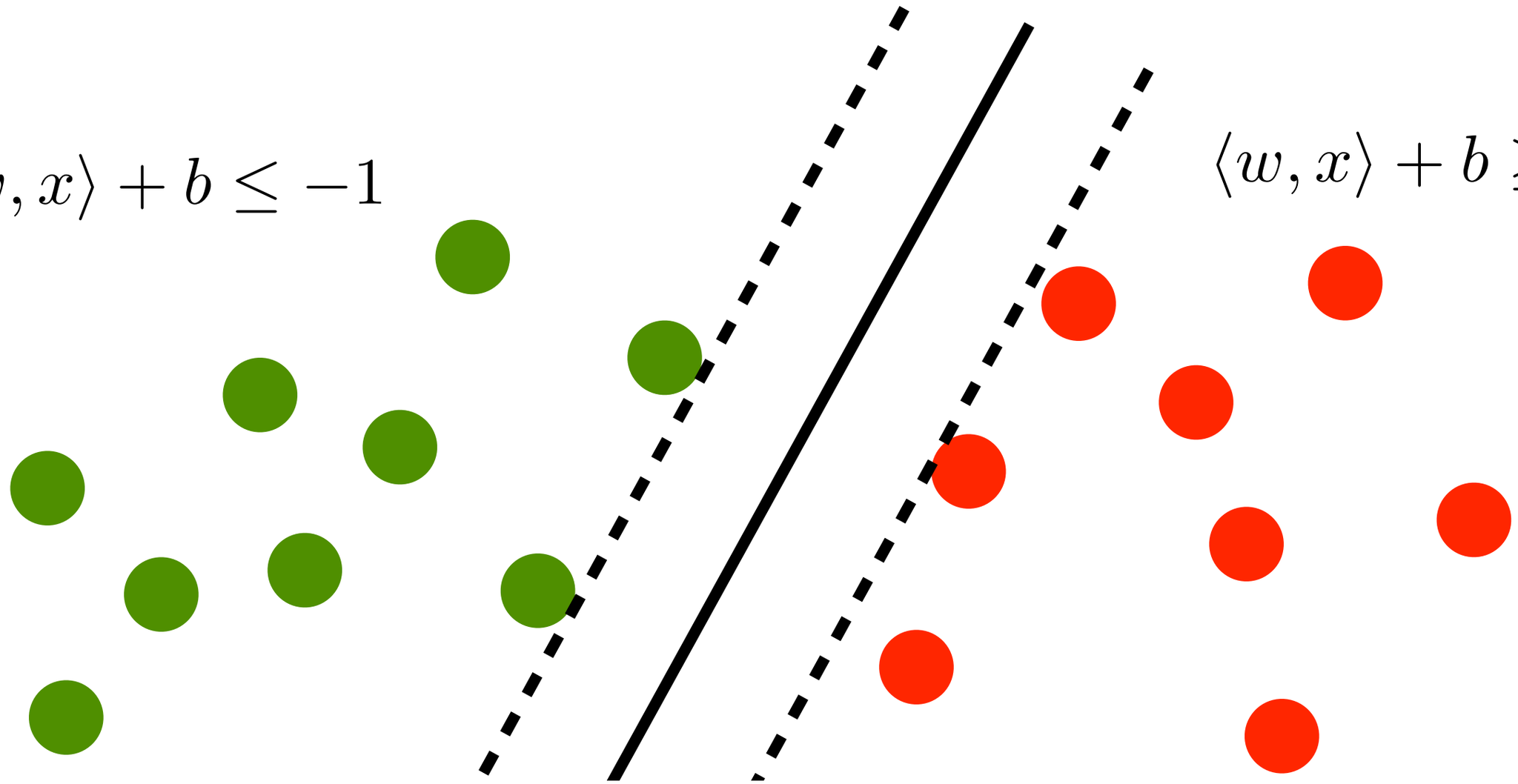
Vladimir N. Vapnik and Alexey Ya. Chervonenkis. On a class of perceptrons. *Automation and Remote Control*, 25(1):103–109, 1964. Russian original: В. Н. Вапник, А. Я. Червоненкис. Об одном классе персептронов. *Автоматика и телемеханика*, 25(1):112–120, 1964; with English summary entitled "On a perceptron class". The original article submitted on 21 February 1963.

*mentioned in lectures but optional
(others papers all covered in detail)

AT&T Research

ex-AT&T and students

Large Margin Classifier



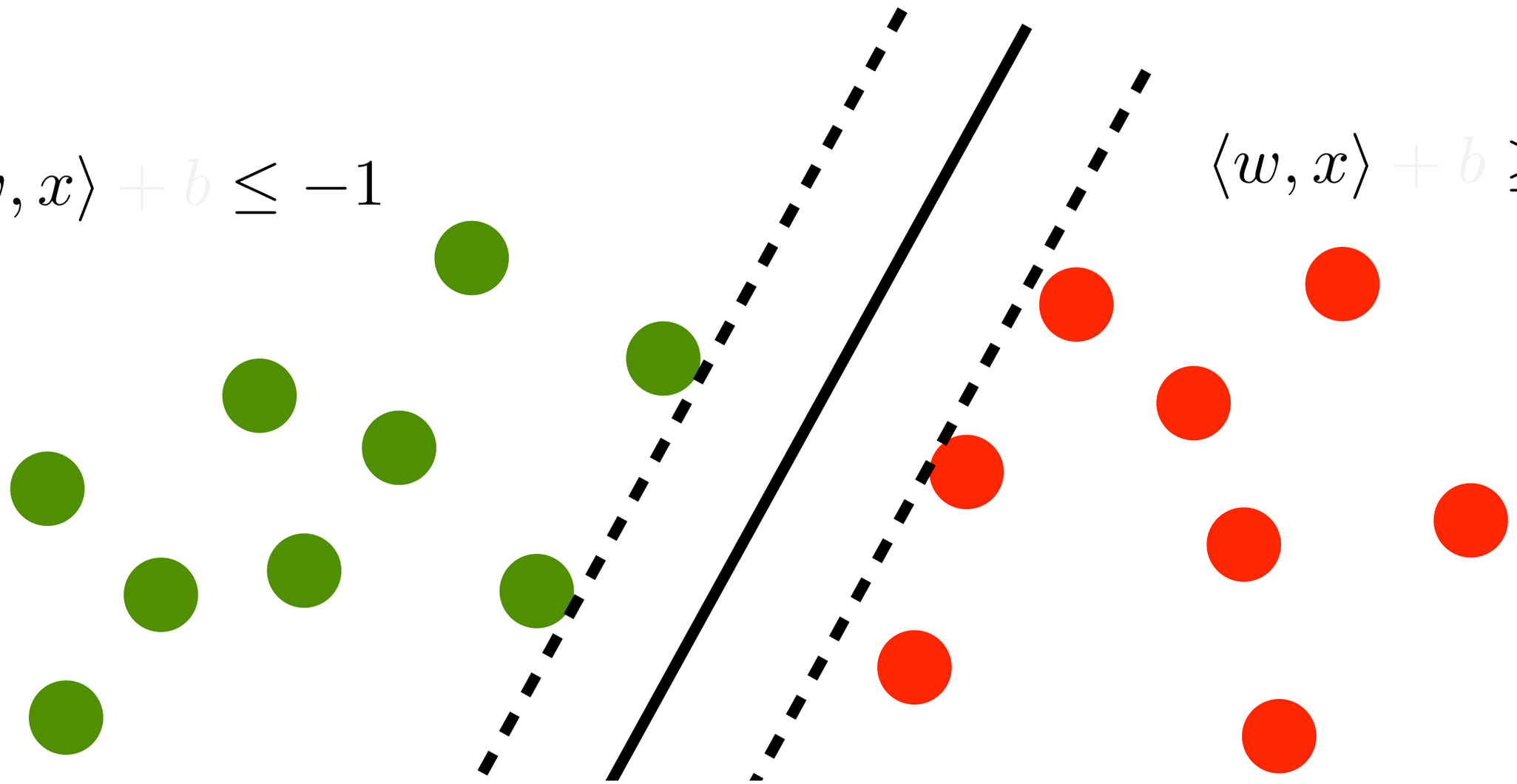
linear function

$$f(x) = \langle w, x \rangle + b$$

Large Margin Classifier

$\langle w, x \rangle + b \leq -1$

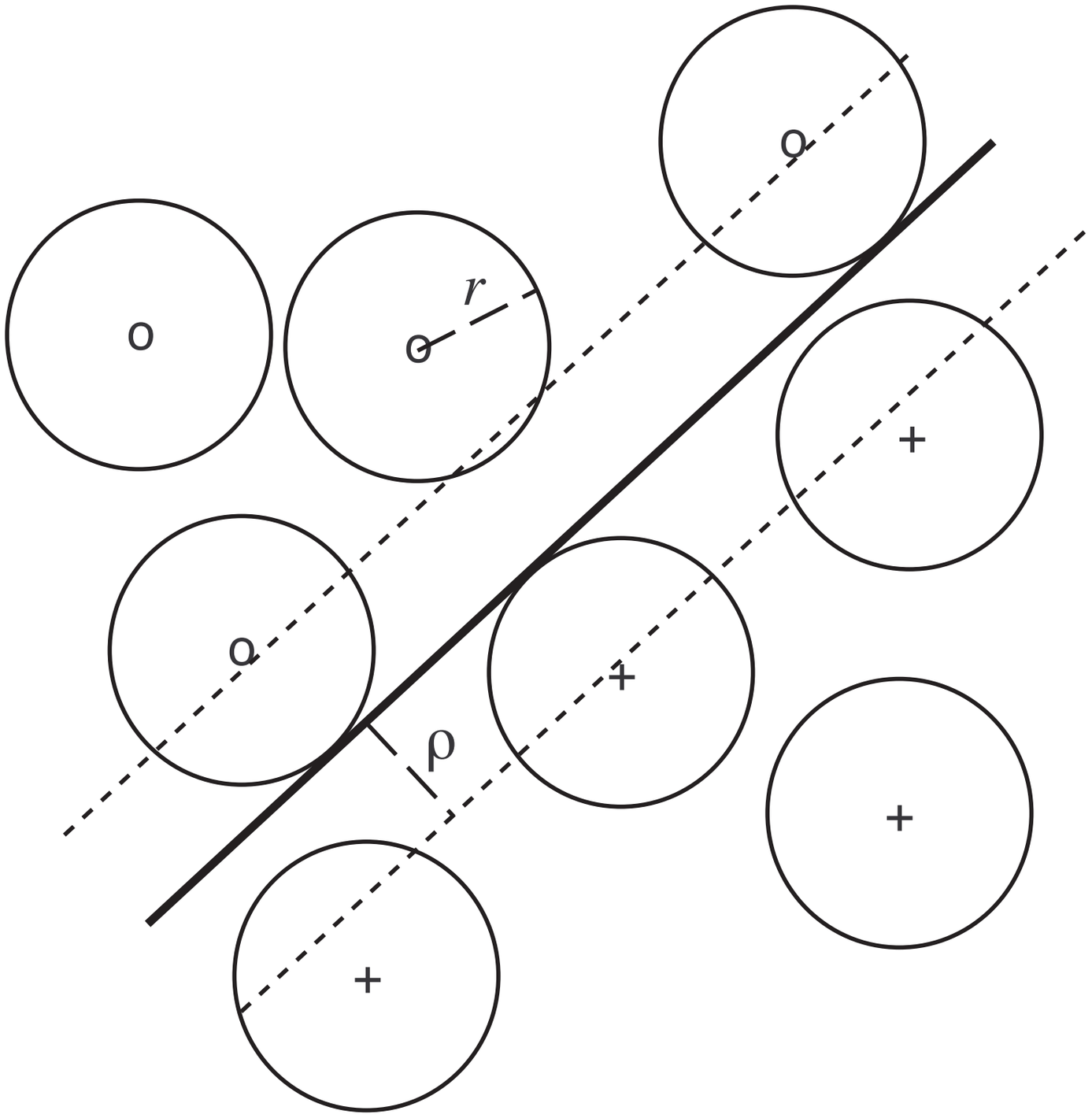
$\langle w, x \rangle + b \geq 1$



linear function

$f(x) = \langle w, x \rangle + b$

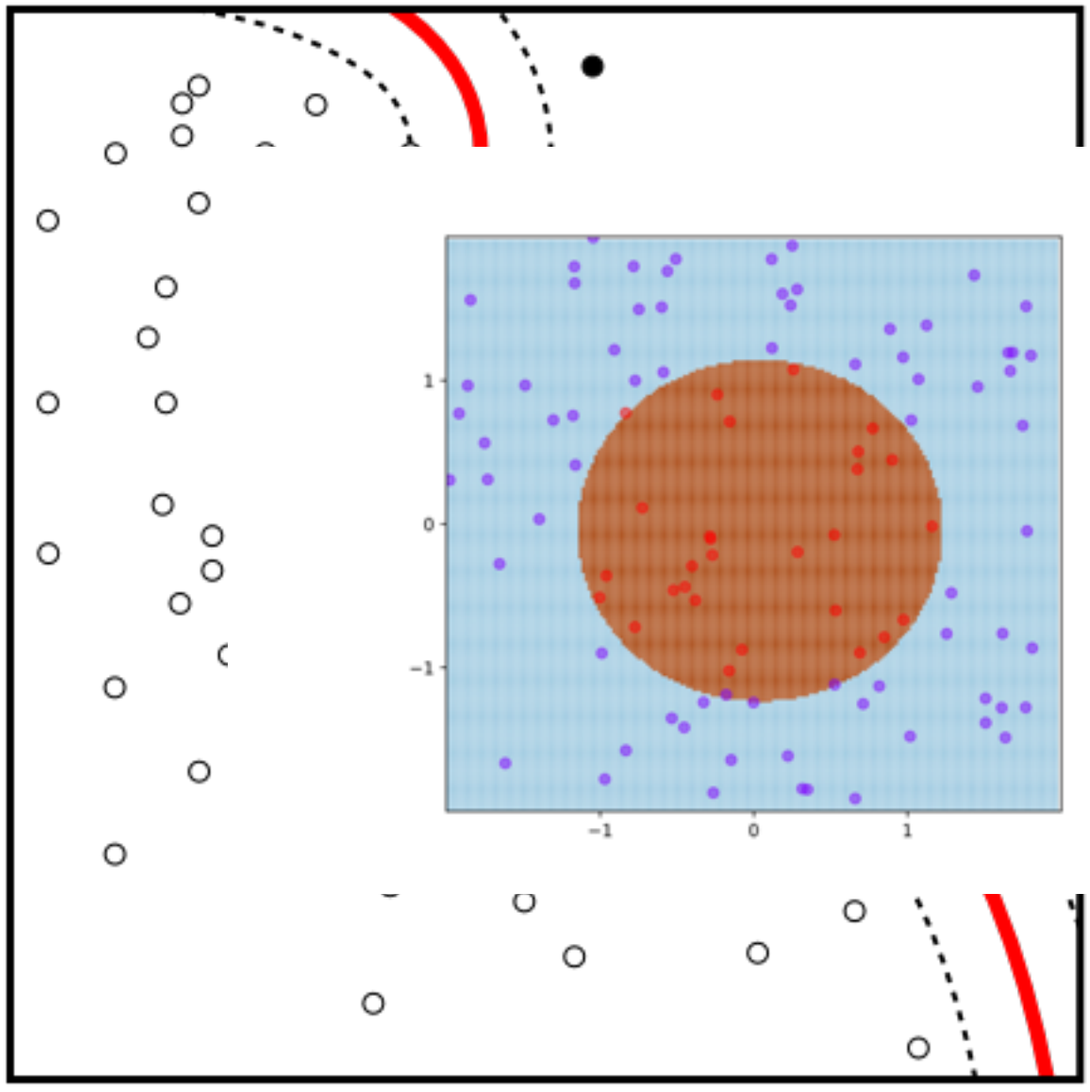
Why large margins?

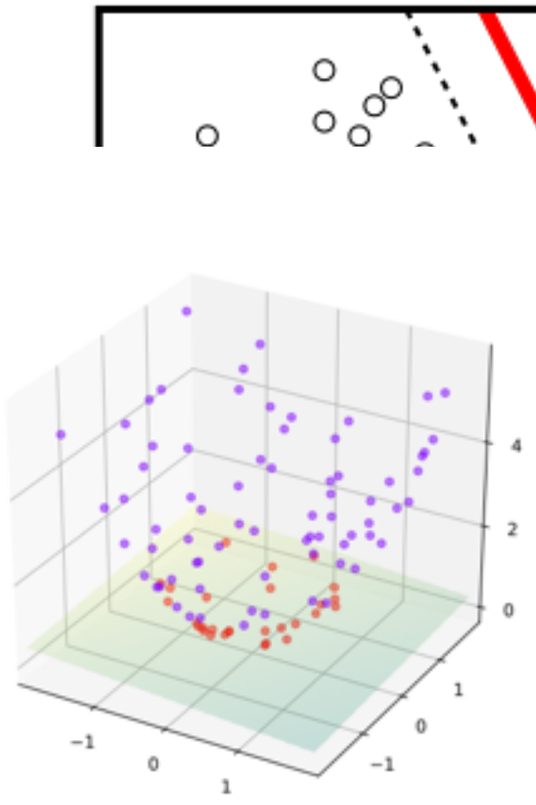


- Maximum robustness relative to uncertainty
- Symmetry breaking
- Independent of correctly classified instances
- Easy to find for easy problems

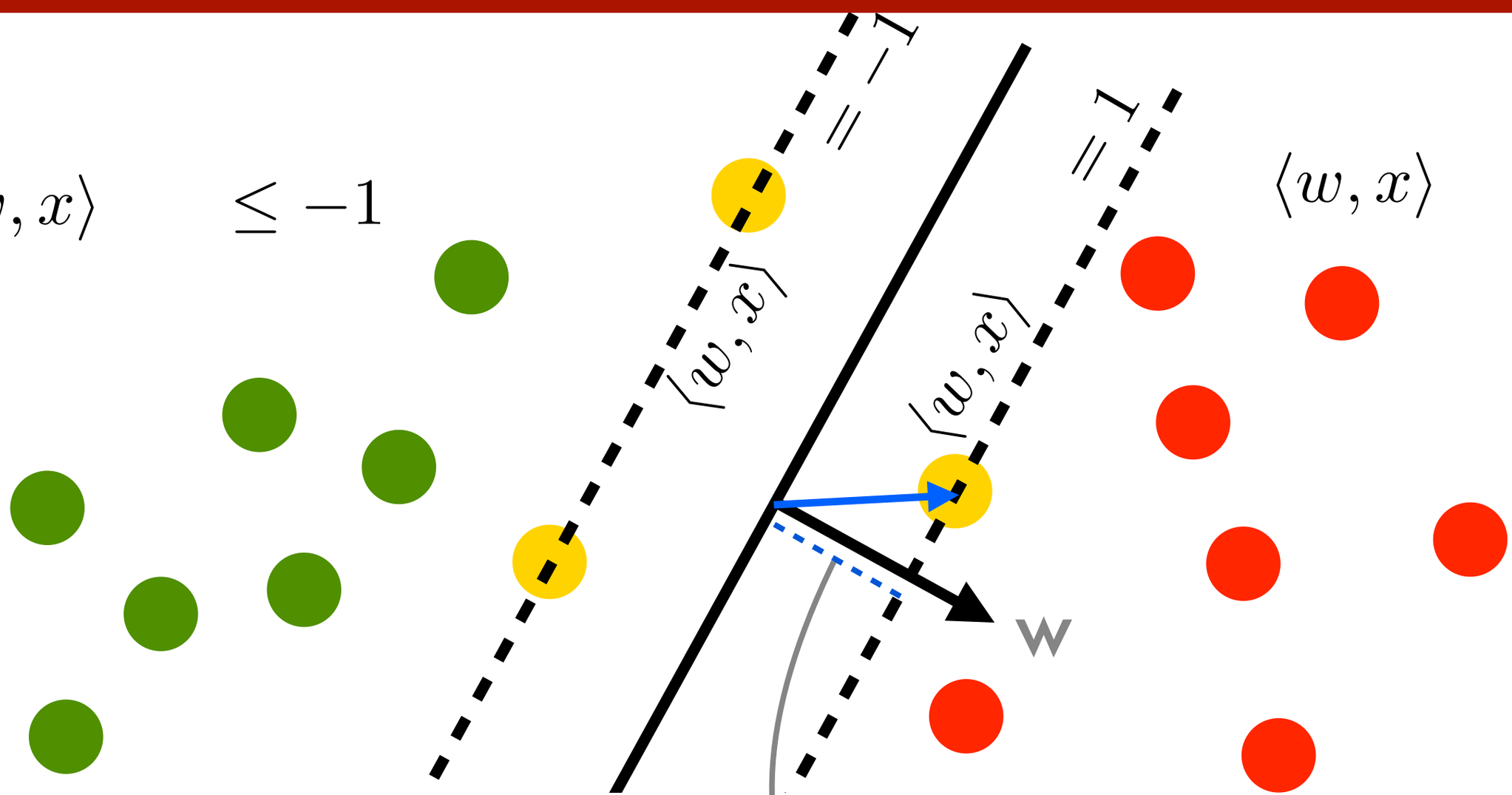
Feature Map Φ

- SVM is often used with kernels





Large Margin Classifier



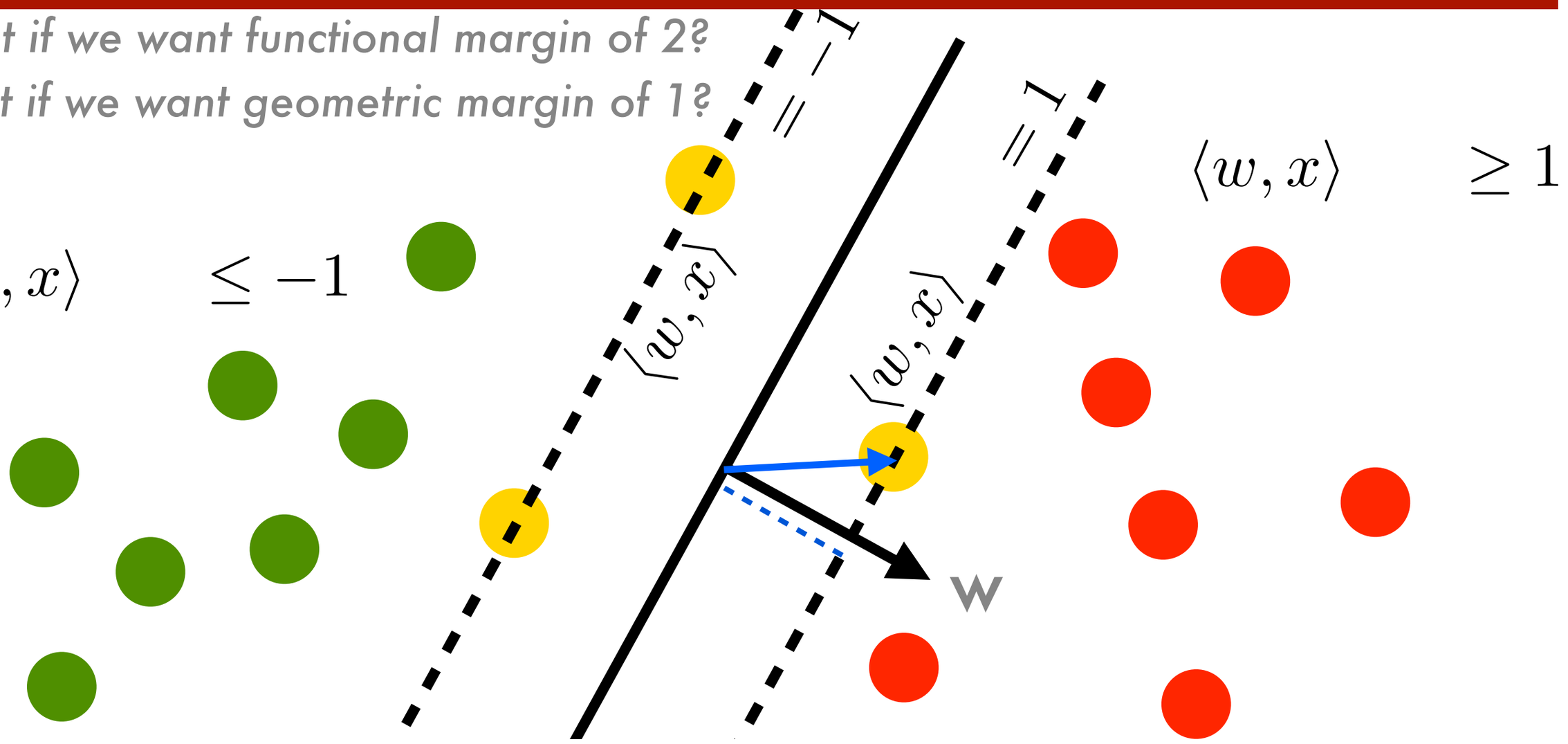
functional margin: $y_i(\mathbf{w} \cdot x_i)$

geometric margin: $\dfrac{y_i(\mathbf{w} \cdot x_i)}{\|\mathbf{w}\|} = \dfrac{1}{\|\mathbf{w}\|}$

Large Margin Classifier

Q1: what if we want functional margin of 2?

Q2: what if we want geometric margin of 1?

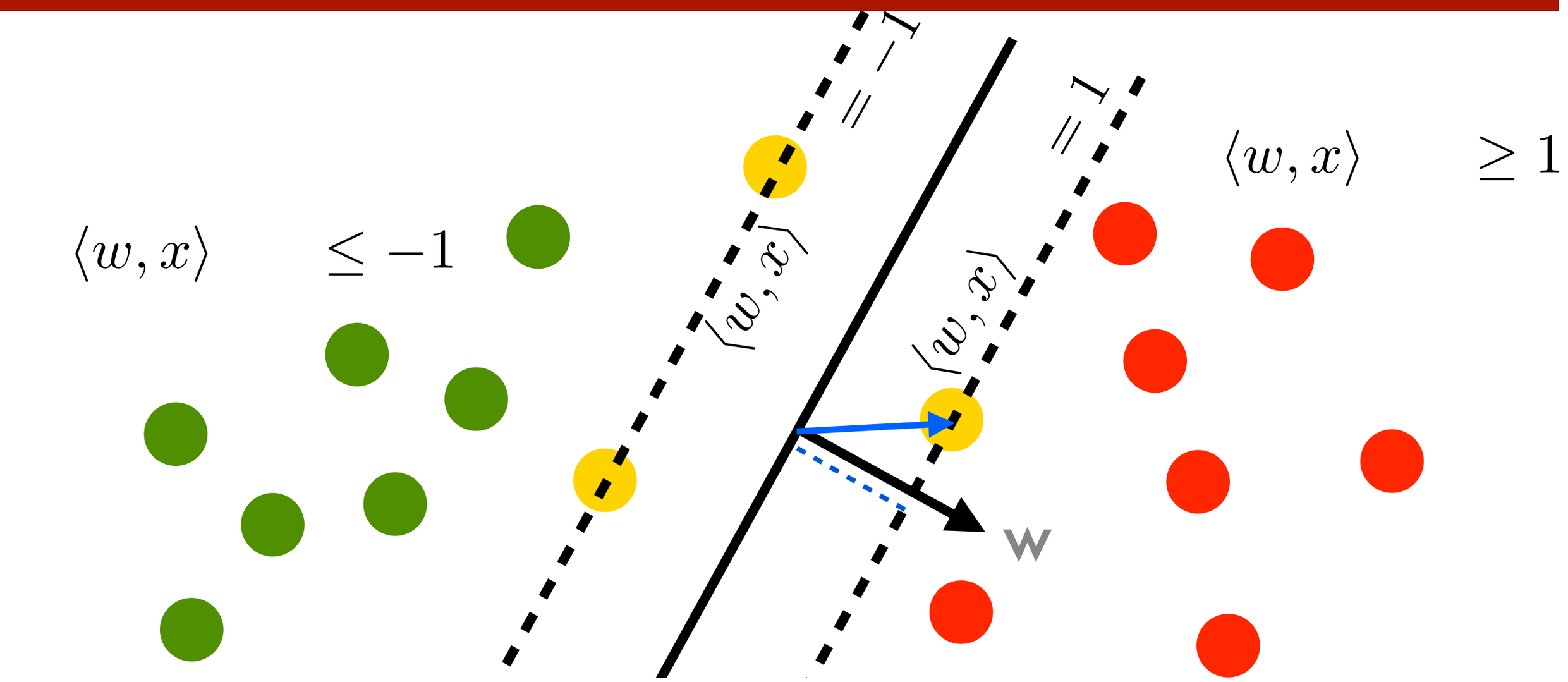


SVM objective (max version):

$$\max_{\mathbf{w}} \frac{1}{\|\mathbf{w}\|} \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

max. geometric margin
s.t. functional margin
is at least 1

Large Margin Classifier



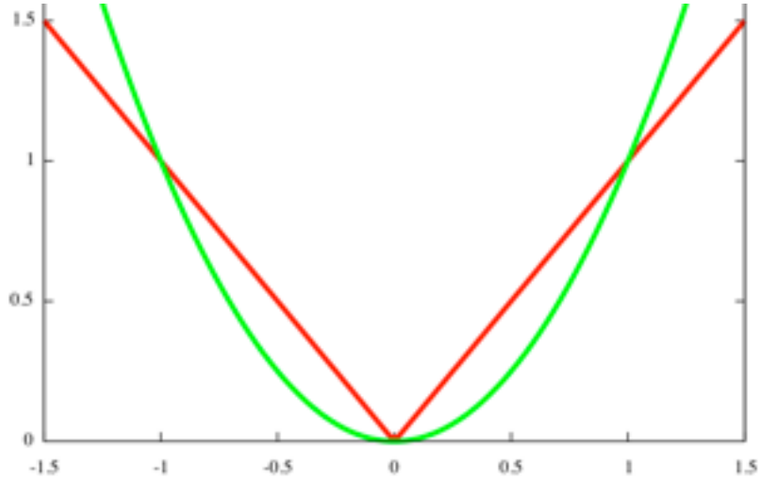
SVM objective (min version):

$$\min_{\mathbf{w}} \|\mathbf{w}\| \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

interpretation: small models generalize better

min. weight vector
s.t. functional margin
is at least 1

Large Margin Classifier



SVM objective (min version):

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

|w| not differentiable, $|w|^2$ is.

min. weight vector
s.t. functional margin
is at least 1

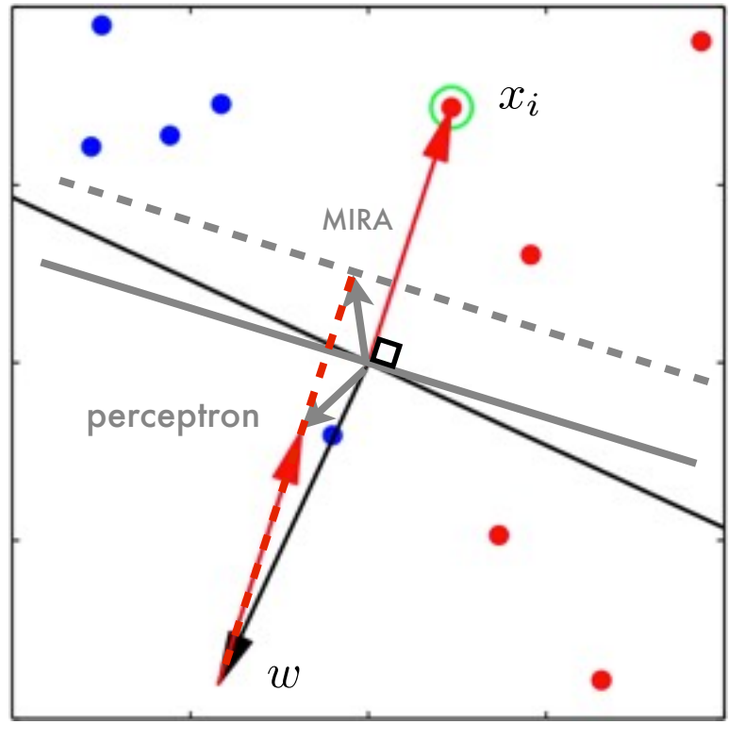
SVM vs. MIRA

- SVM: min weight vector to enforce functional margin of at least 1 on ALL EXAMPLES
- MIRA: min weight change to enforce functional margin of at least 1 on THIS EXAMPLE
- MIRA is 1-step or online approximation of SVM
- Aggressive MIRA→SVM as $p \to 1$

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

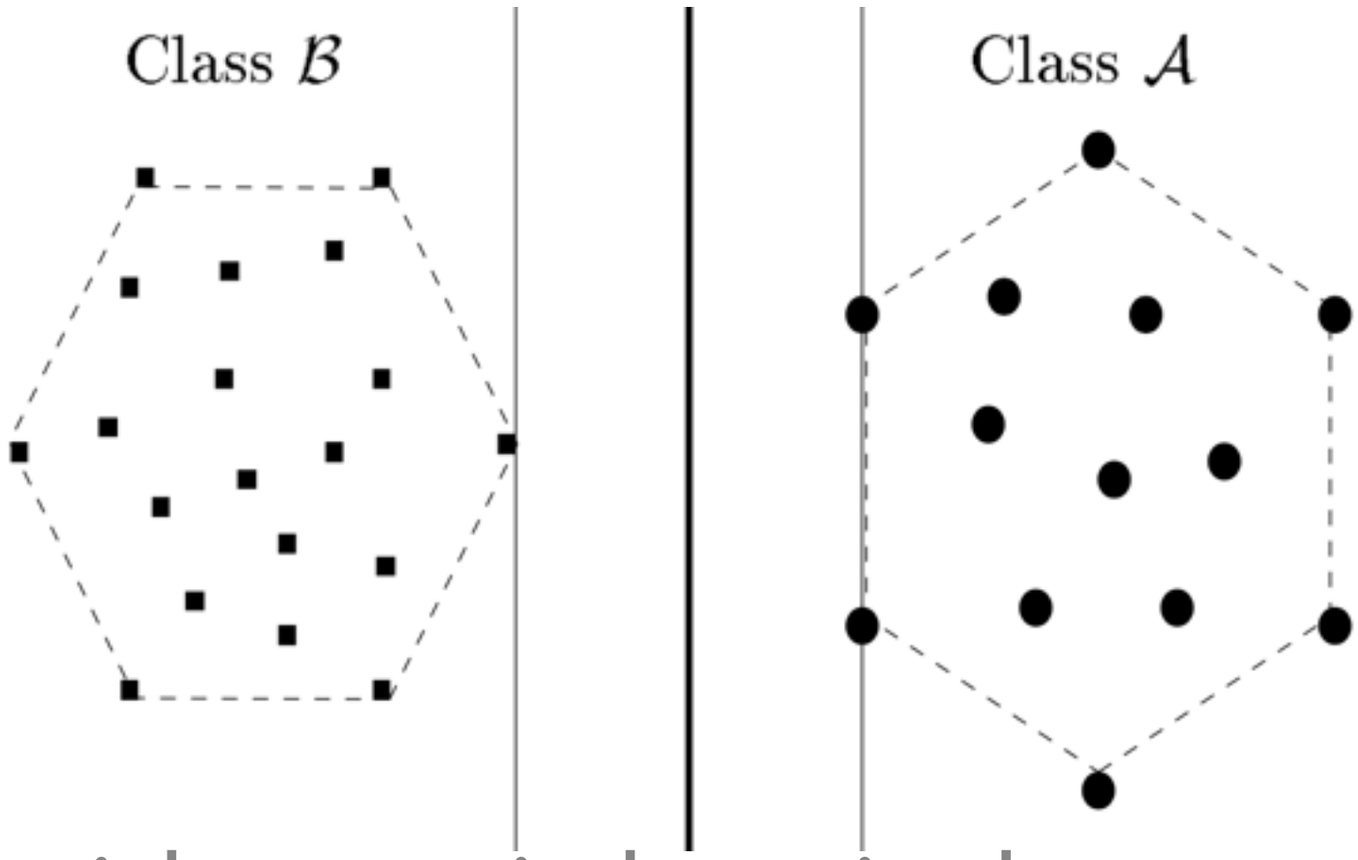
$$\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}\|^2$$

$$\text{s.t. } \mathbf{w}' \cdot \mathbf{x} \geq 1$$

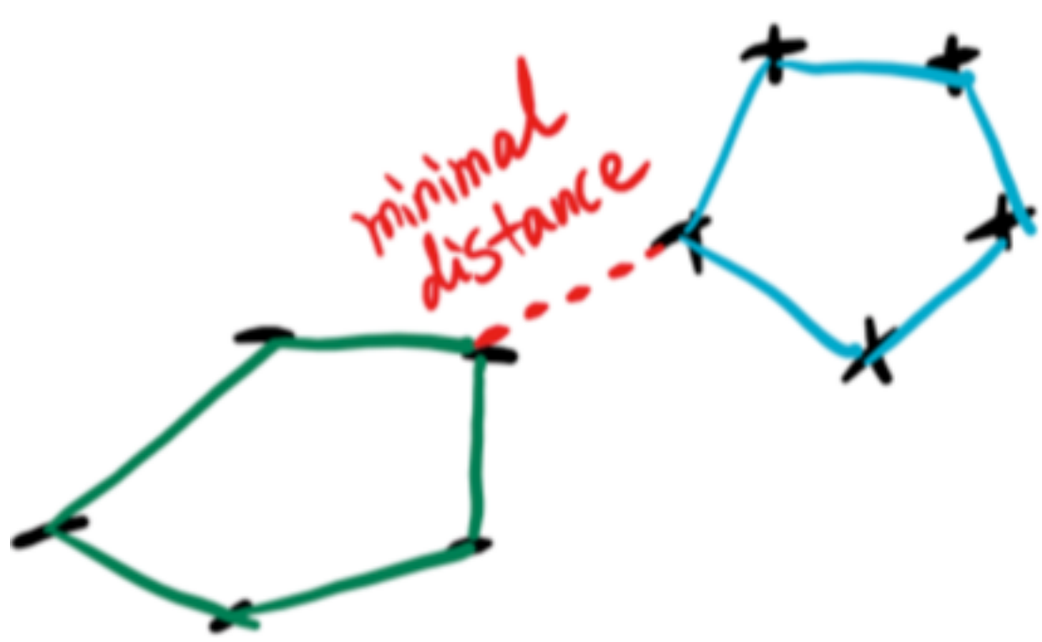


Convex Hull Interpretation

max. distance between convex hulls



how many support vectors in 2D?



weight vector is determined by the support vectors alone

c.f. perceptron: $\mathbf{w} = \sum_{(\mathbf{x},y)\in errors} y \cdot \mathbf{x}$

what about MIRA?

why don't use convex hulls for SVMs in practice??

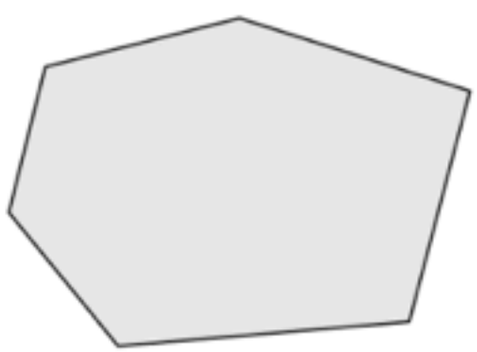
Convexity and Convex Hulls

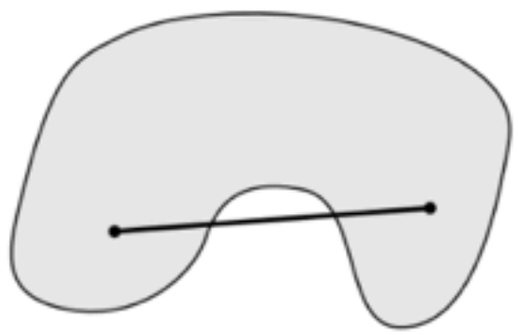
Convex set. A point set $C \in \mathbf{R}^d$ is convex if the line segment [x,y] connecting any two points x and y in C lies entirely in C.

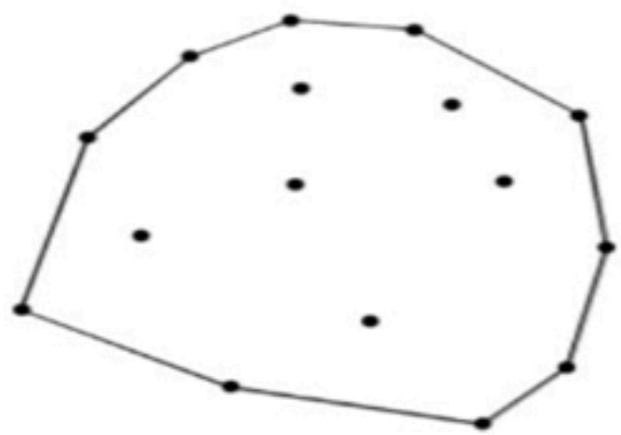
convex combination

Convex hull. Smallest convex set containing C.

$$\mathrm{ch}(C) := \left\{ \sum_i \alpha_i \mathbf{x}_i : \mathbf{x}_i \in C, \alpha_i \geq 0, \sum_i \alpha_i = 1 \right\}.$$





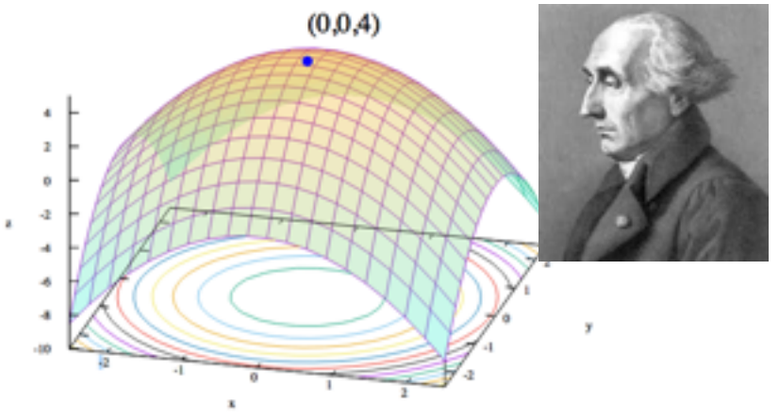


Optimization

- Primal optimization problem

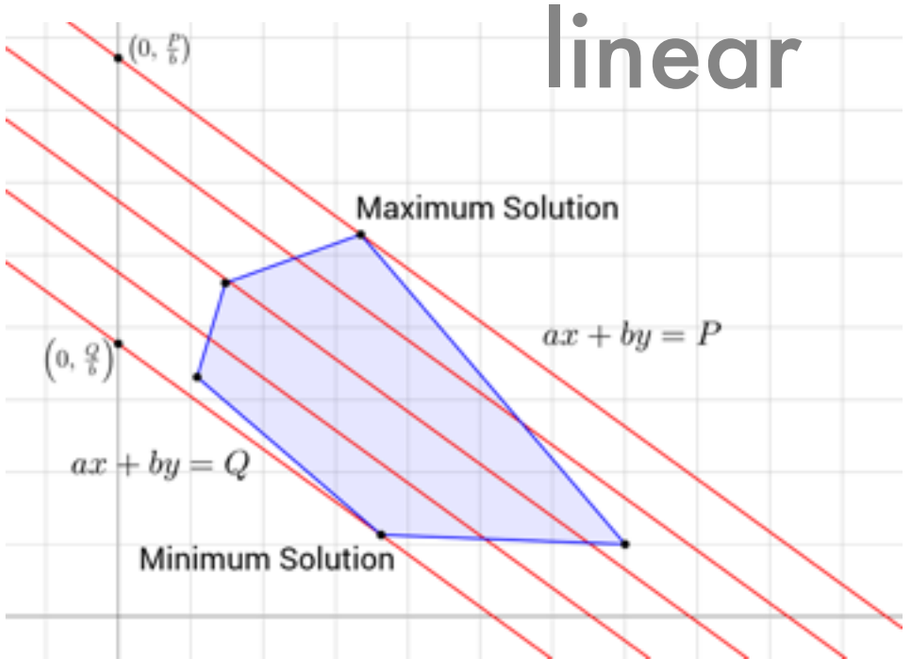
$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$

constraint

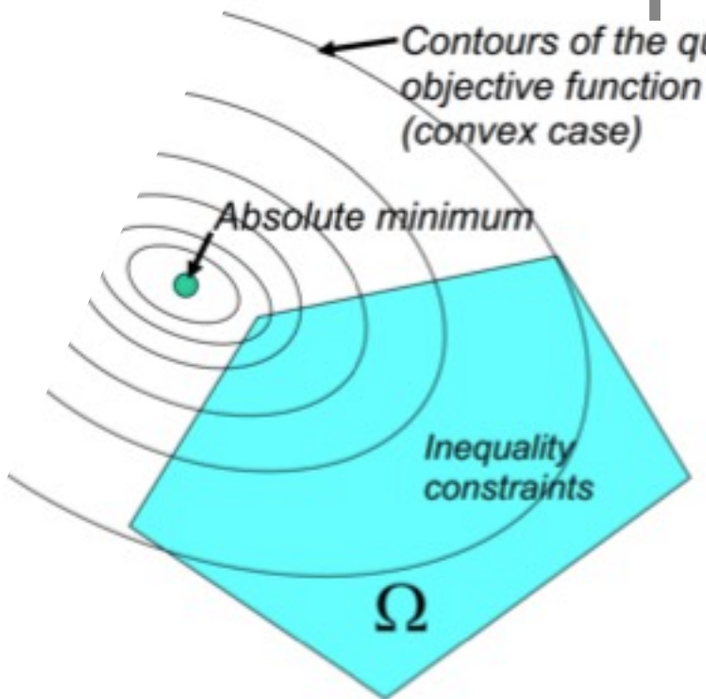


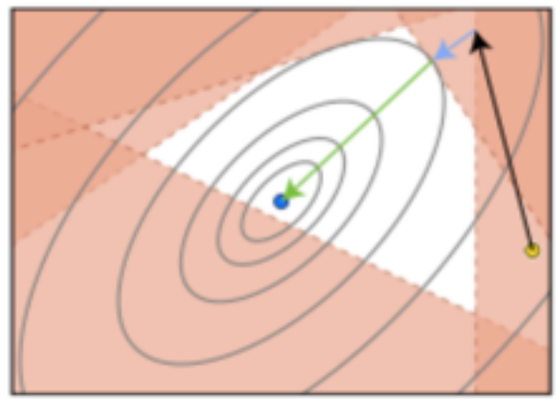
- Convex optimization: convex function over convex set!
- Quadratic prog.: quadratic function w/ linear constraints

linear



quadratic



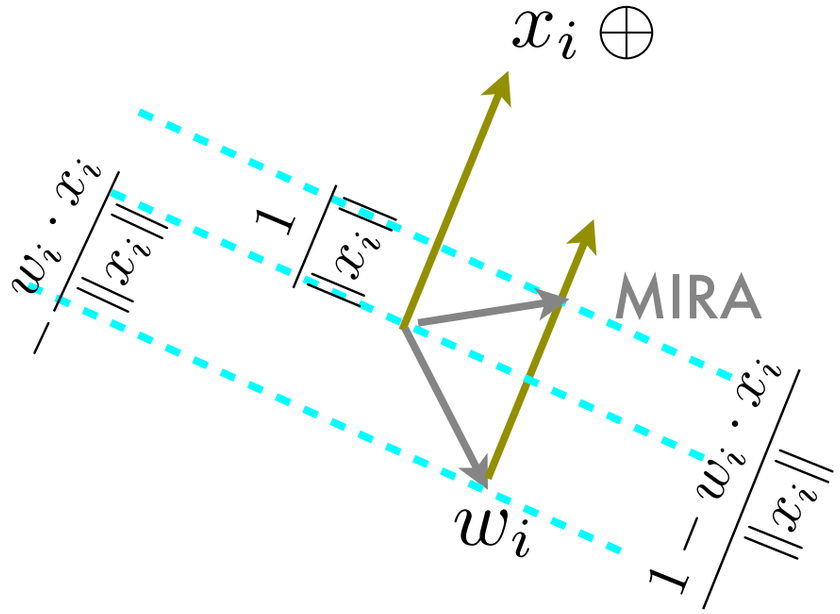


MIRA as QP

- MIRA is a trivial QP; can be solved geometrically
- what about multiple constraints (e.g. minibatch)?

$$\min_{\mathbf{w}'} \|\mathbf{w}' - \mathbf{w}\|^2$$

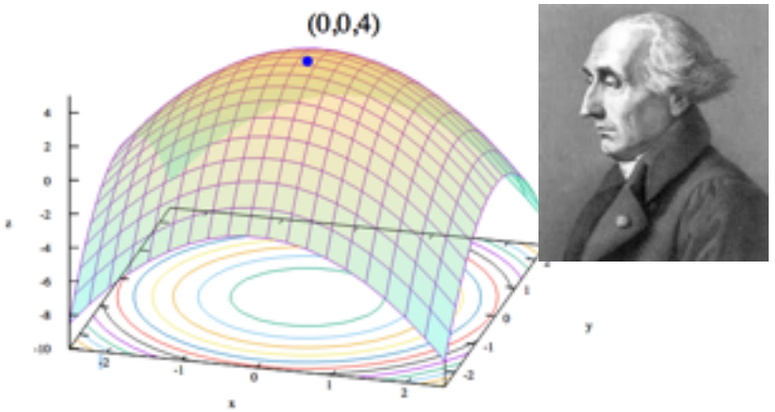
$$\text{s.t. } \mathbf{w}' \cdot \mathbf{x} \geq 1$$



Optimization

- Primal optimization problem

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } \forall (\mathbf{x}, y) \in D, y(\mathbf{w} \cdot \mathbf{x}) \geq 1$$



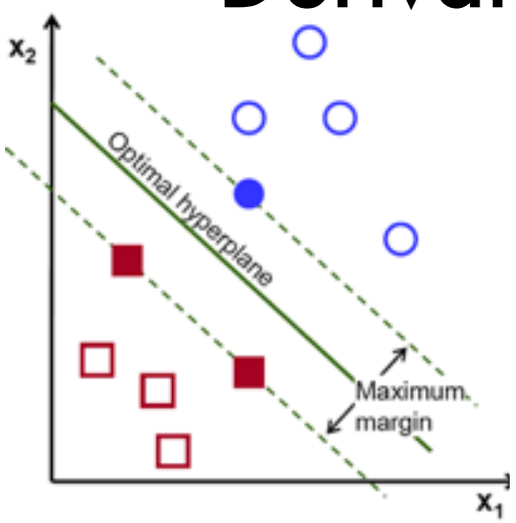
- Convex optimization: convex function over convex set!

- Lagrange function

$$L(w, \quad \alpha) = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i [y_i [\langle x_i, w \rangle + b] - 1]$$

constraint

Derivatives in **w** need to vanish



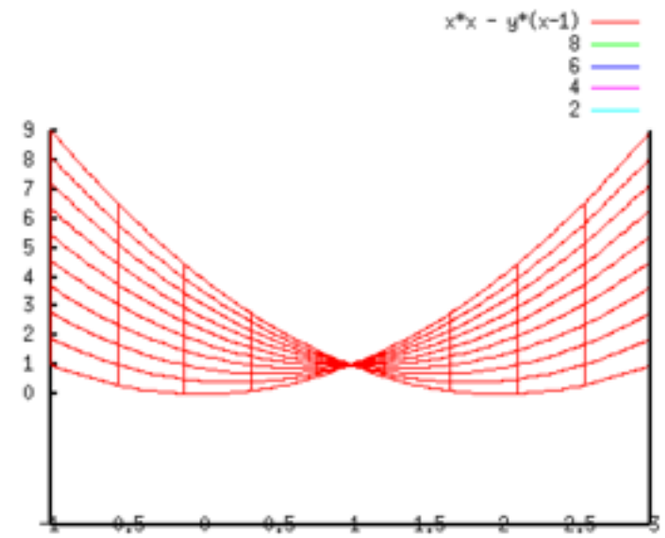
$$\partial_w L(w, \ \alpha \ ) = w - \sum_i \alpha_i y_i x_i = 0$$

$$w = \sum_i y_i \alpha_i x_i$$

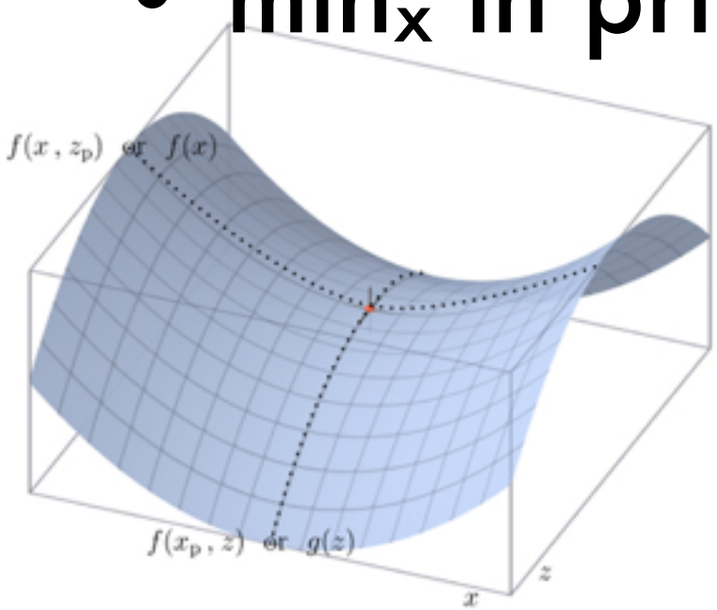
model is a linear combo of a *small subset* of input (the support vectors) i.e., those with $\alpha_i > 0$

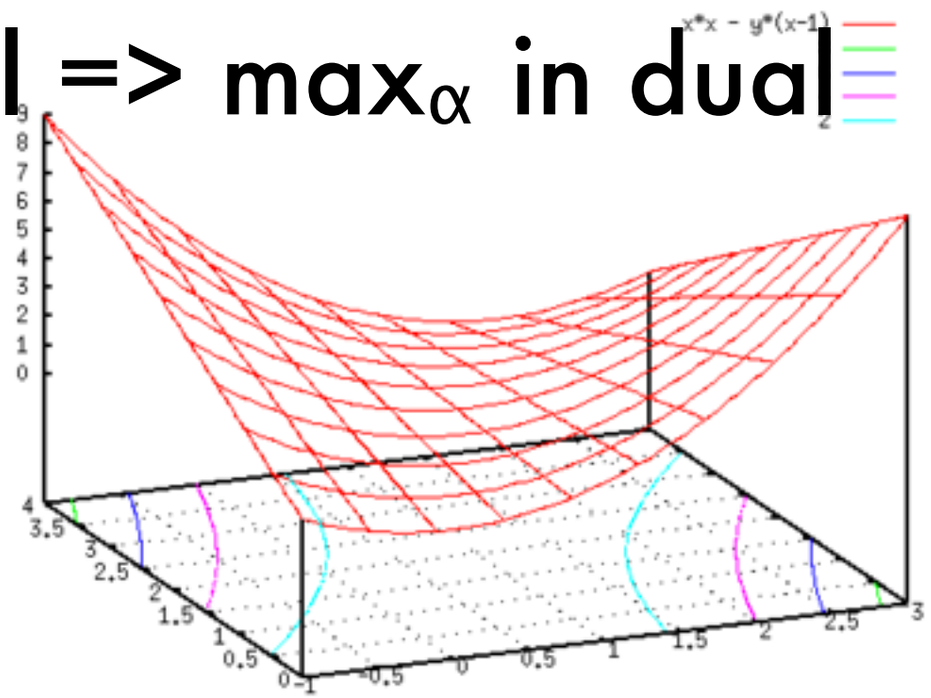
Lagrangian & Saddle Point

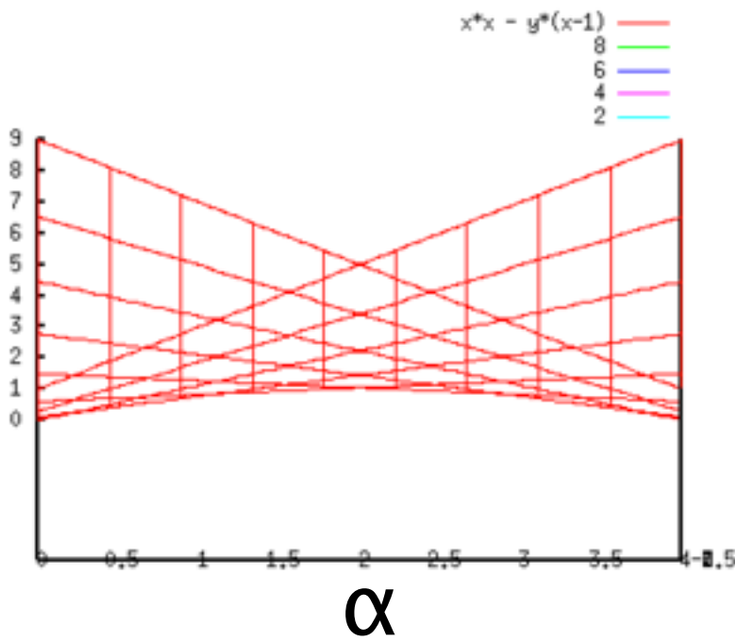
- equality: min x^2 s.t. $x = 1$
- inequality: min x^2 s.t. $x >= 1$
  - Lagrangian: $L(x, \alpha)=x^2 - \alpha(x-1)$
  - derivative in x need to vanish



- optimality is at saddle point with α
- $\min_x$ in primal => $\max_\alpha$ in dual







Constrained Optimization

constraint

$$\underset{w,b}{\text{minimize}}\ \frac{1}{2}\|w\|^2 \ \text{subject to}\ y_i\left[\langle x_i, w\rangle + b\right] \geq 1$$

$$w = \sum_i y_i \alpha_i x_i$$

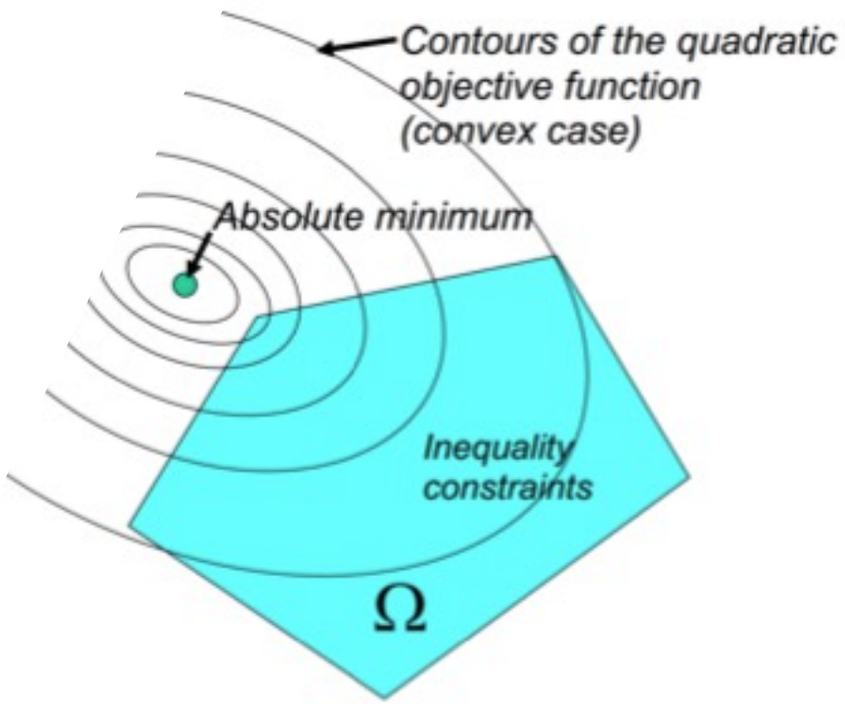
- Quadratic Programming
  - Quadratic Objective
  - Linear Constraints

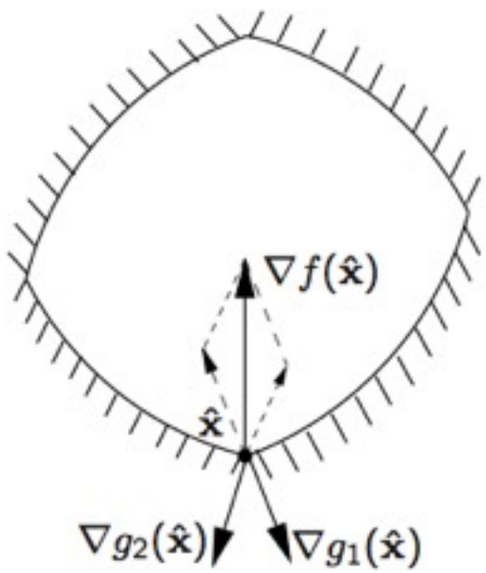
Karush–Kuhn–Tucker

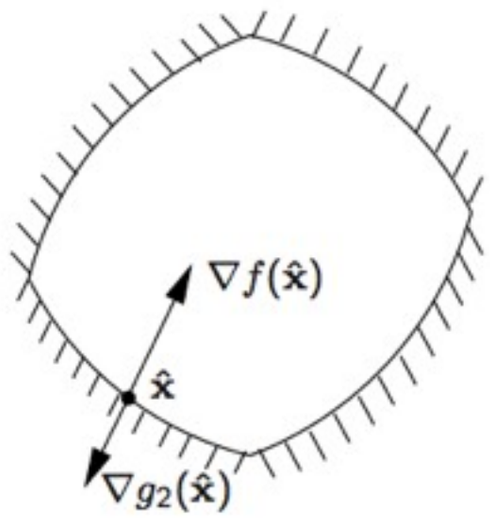
KKT condition (*complementary slackness*)

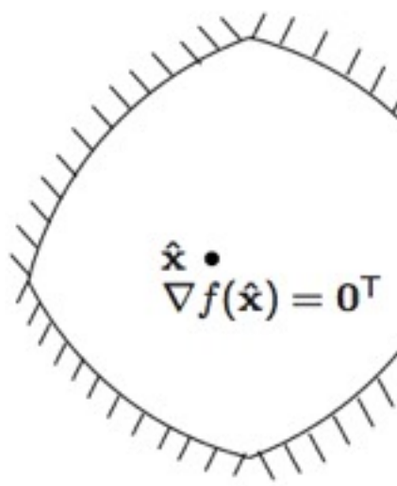
optimal point is achieved at *active constraints*
where $\alpha_i > 0$ ($\alpha_i = 0$ => inactive)

$$\alpha_i \left[y_i \left[\langle w, x_i\rangle + b\right] - 1\right] = 0$$





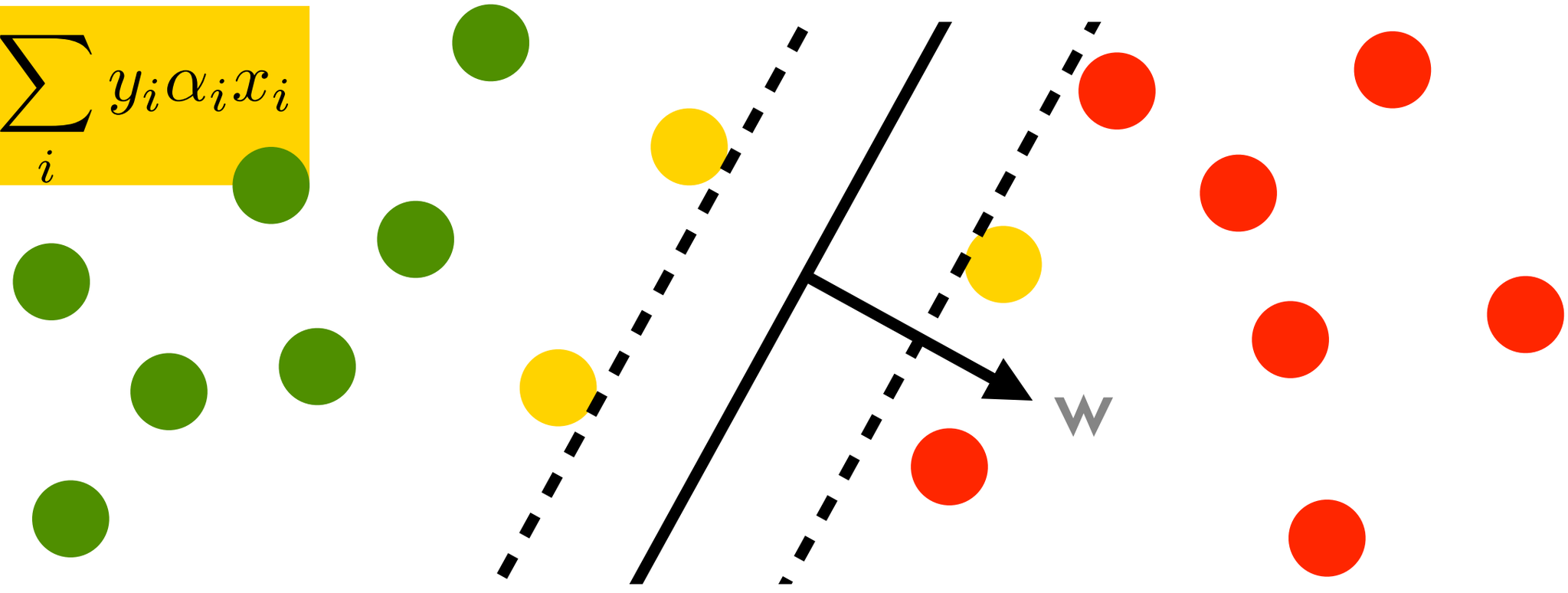




KKT => Support Vectors

$$\underset{w,b}{\text{minimize}}\ \frac{1}{2}\|w\|^2 \ \text{ subject to } y_i\left[\langle x_i, w\rangle + b\right] \geq 1$$

$$w = \sum_i y_i \alpha_i x_i$$



Karush Kuhn Tucker (KKT) Optimality Condition

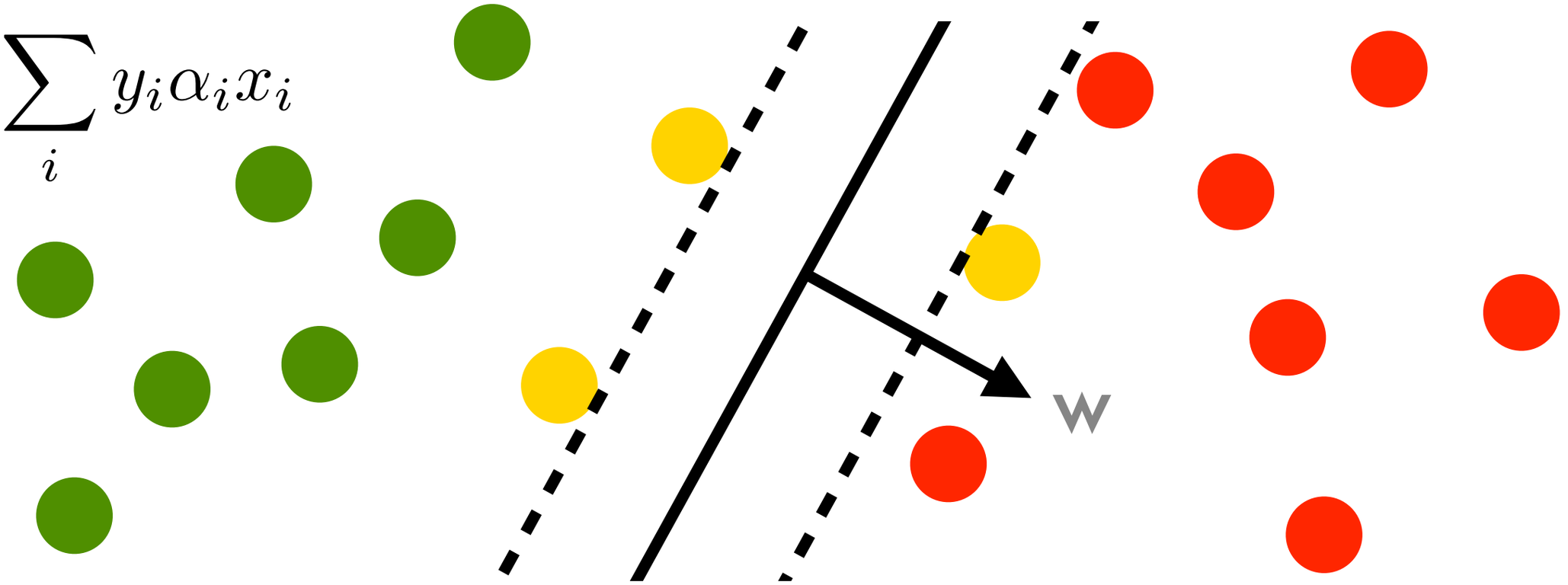
$$\alpha_i \left[y_i \left[\langle w, x_i\rangle + b\right] - 1\right] = 0$$

$$\alpha_i = 0$$

$$\alpha_i > 0 \Longrightarrow y_i \left[\langle w, x_i\rangle + b\right] = 1$$

Properties

$$w = \sum_i y_i \alpha_i x_i$$



- Weight vector w as weighted linear combination of instances
- Only points on margin matter (ignore the rest and get same solution)
- Only inner products matter
  - Quadratic program
  - We can replace the inner product by a kernel
- Keeps instances away from the margin

Alternative: Primal=>Dual

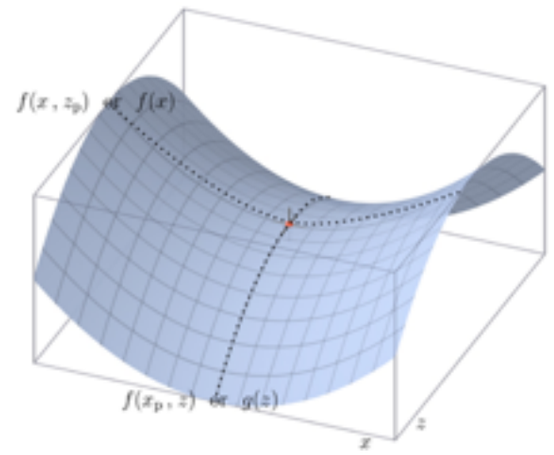
- Lagrange function

$$L(w, \quad \alpha) = \frac{1}{2} \|w\|^2 - \sum_i \alpha_i \left[y_i \left[\langle x_i, w \rangle \qquad - 1 \right] \right.$$

- Derivatives in **w** need to vanish

$$\partial_w L(w, \quad \alpha) = w - \sum_i \alpha_i y_i x_i = 0$$

$$w = \sum_i y_i \alpha_i x_i$$



- Plugging **w** back into *L* yields

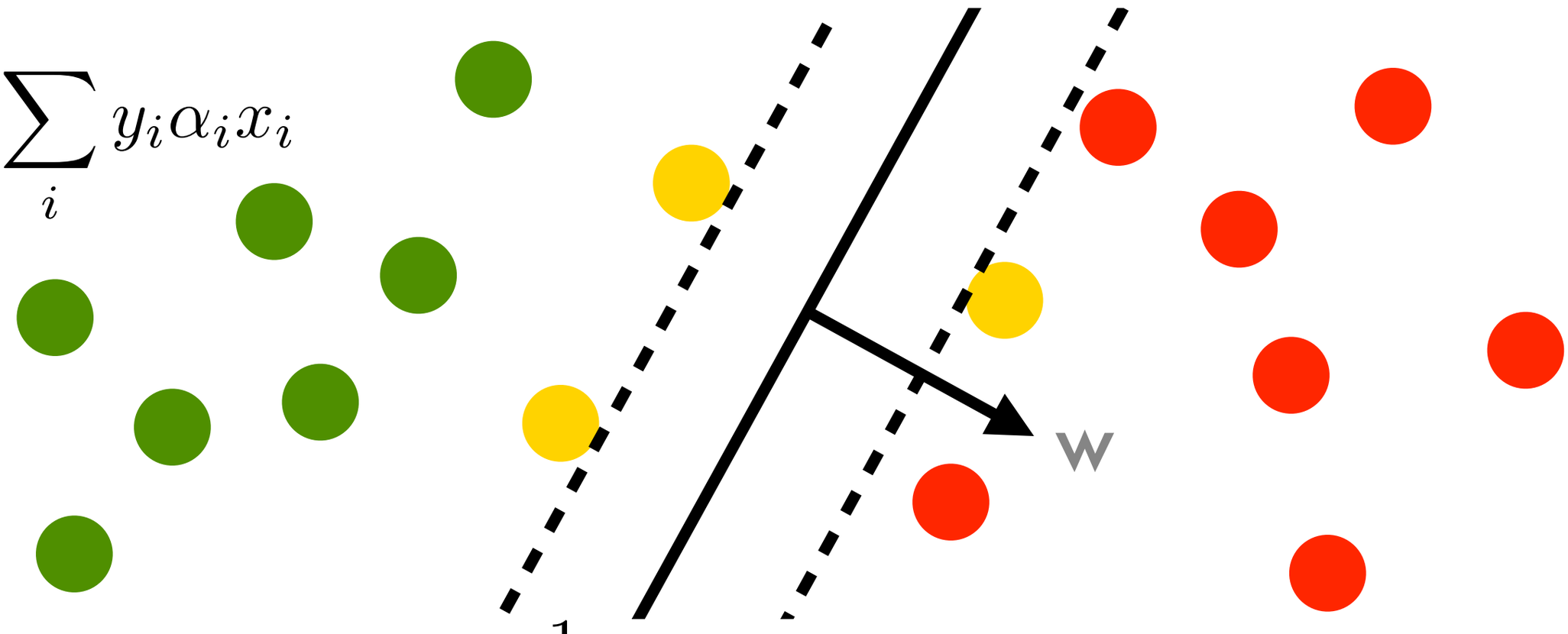
$$\underset{\alpha}{\text{maximize}} - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i$$

subject to **dual variables** $\alpha_i \geq 0$

Primal vs. Dual

Primal $$\underset{w,b}{\text{minimize}} \frac{1}{2}\|w\|^2 \text{ subject to } y_i\left[\langle x_i, w\rangle + b\right] \geq 1$$

$$w = \sum_i y_i \alpha_i x_i$$



Dual $$\underset{\alpha}{\text{maximize}} -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle + \sum_i \alpha_i$$

subject to **dual variables** $\alpha_i \geq 0$

Solving the optimization problem

- Dual problem

$$\underset{\alpha}{\text{maximize}} -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle + \sum_i \alpha_i$$

$$\text{subject to } \textbf{dual variables } \alpha_i \geq 0$$

- If problem is small enough (1000s of variables) we can use off-the-shelf solver (CVXOPT, CPLEX, OOQP, LOQO)
- For larger problem use fact that only SVs matter and solve in blocks (active set method).

Quadratic Program in Dual

- Dual problem

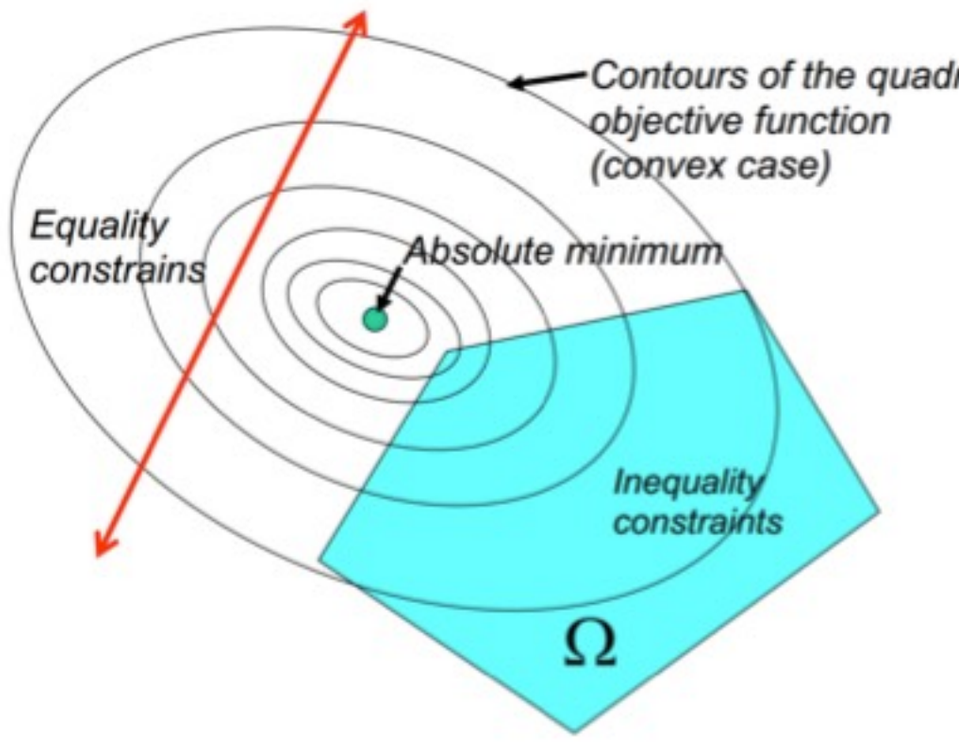
$$\underset{\alpha}{\text{maximize}} -\frac{1}{2}\alpha^T Q\alpha - \alpha^T b$$

$$\text{subject to } \alpha \geq 0$$

Q: what's the Q in SVM primal?
how about Q in SVM dual?

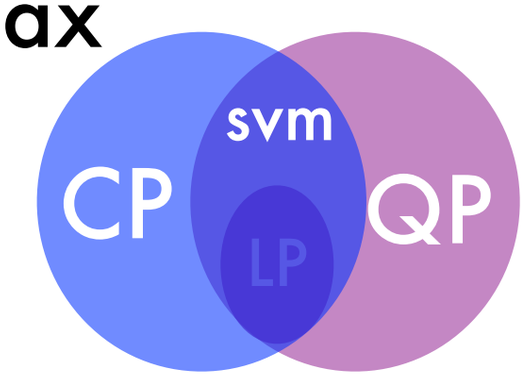
- Quadratic Programming
  - Objective: Quadratic function
    - Q is <u>positive semidefinite</u>
  - Constraints: Linear functions

- Methods
  - Gradient Descent
  - Coordinate Descent
    - aka., Hildreth Algorithm
  - Sequential Minimal Optimization (SMO)

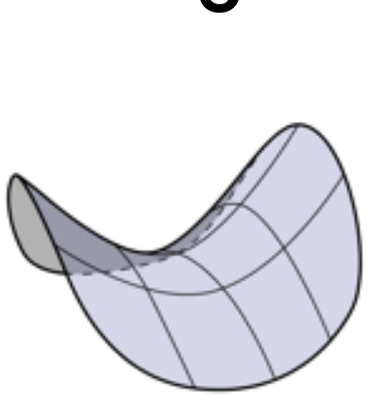


Convex QP

- if Q is positive (semi)definite, i.e., $x^TQx \geq 0$ for all x, then convex QP => local min/max is global min/max
- if Q = 0, it reduces to linear programming
- if Q is indefinite => saddlepoint

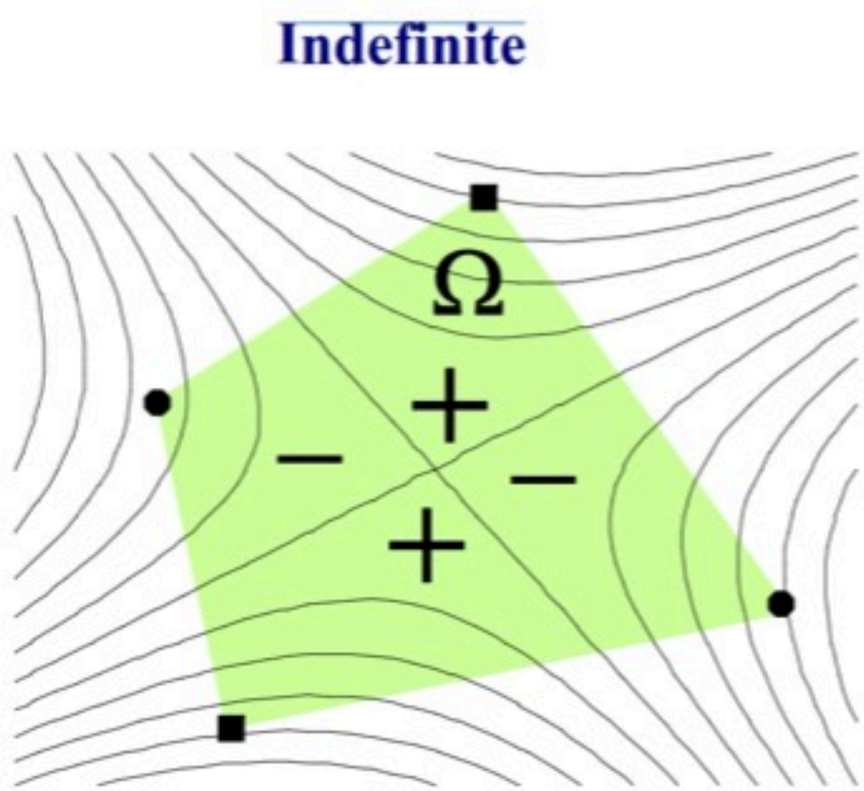


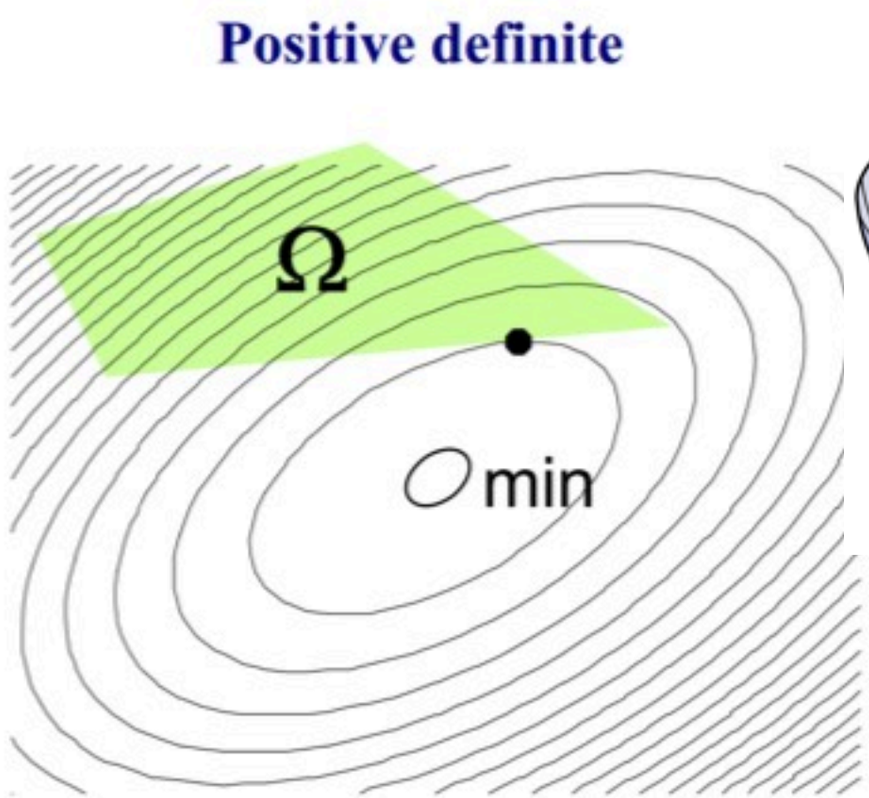
- general QP is NP-hard; convex QP is polynomial-time

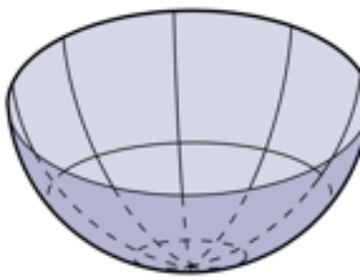


$x^2 - y^2$

(indefinite)







$x^2 + y^2$

(definite)

QP: Hildreth Algorithm

- idea 1:
  - update one coordinate while fixing all other coordinates
  - e.g., update coordinate i is to solve:

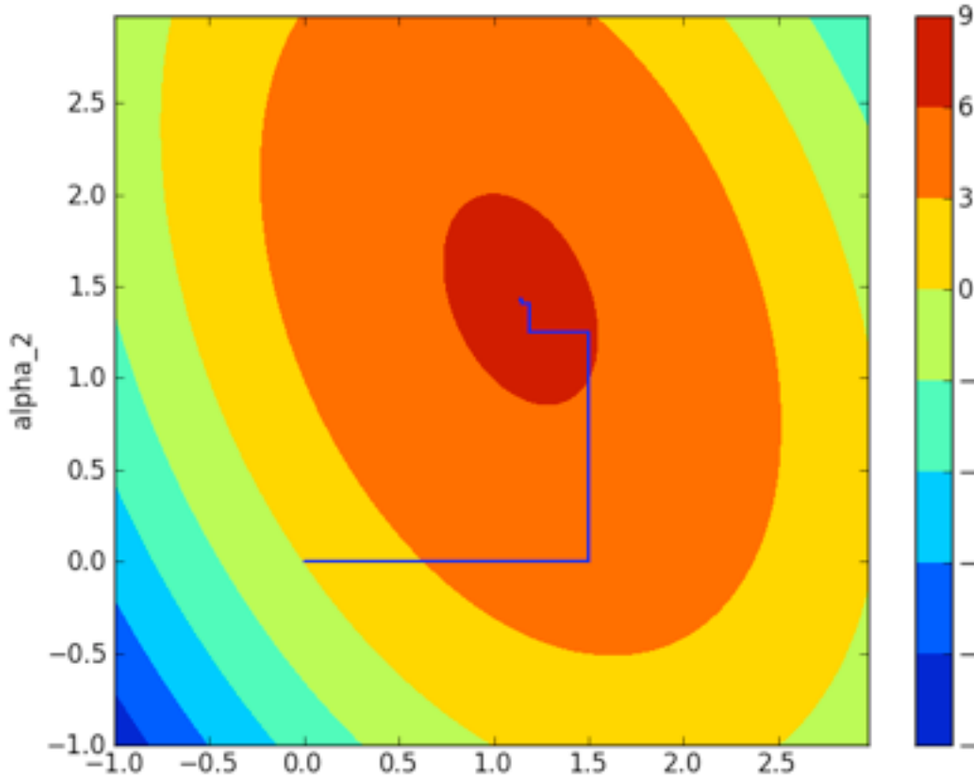
$$\underset{\alpha_i}{\text{argmax}} -\frac{1}{2}\alpha^T Q\alpha - \alpha^T b$$

$$\text{subject to } \alpha \geq 0$$

Quadratic function with only one variable
Maximum => first-order derivative is 0

QP: Hildreth Algorithm

- idea 2:
  - choose another coordinate and repeat until meet stopping criterion
    - reach maximum or
    - increase between 2 consecutive iterations is very small or
    - after some # of iterations
- how to choose coordinate: sweep patter
  - Sequential:
    - 1, 2, …, n, 1, 2, …, n, …
    - 1, 2, …, n, n-1, n-2, …,1, 2, …
  - Random: permutation of 1,2, …, n
  - Maximal Descent
    - choose i with maximal descent in objecti



QP: Hildreth Algorithm

initialize $\alpha_i = 0$ for all i

repeat

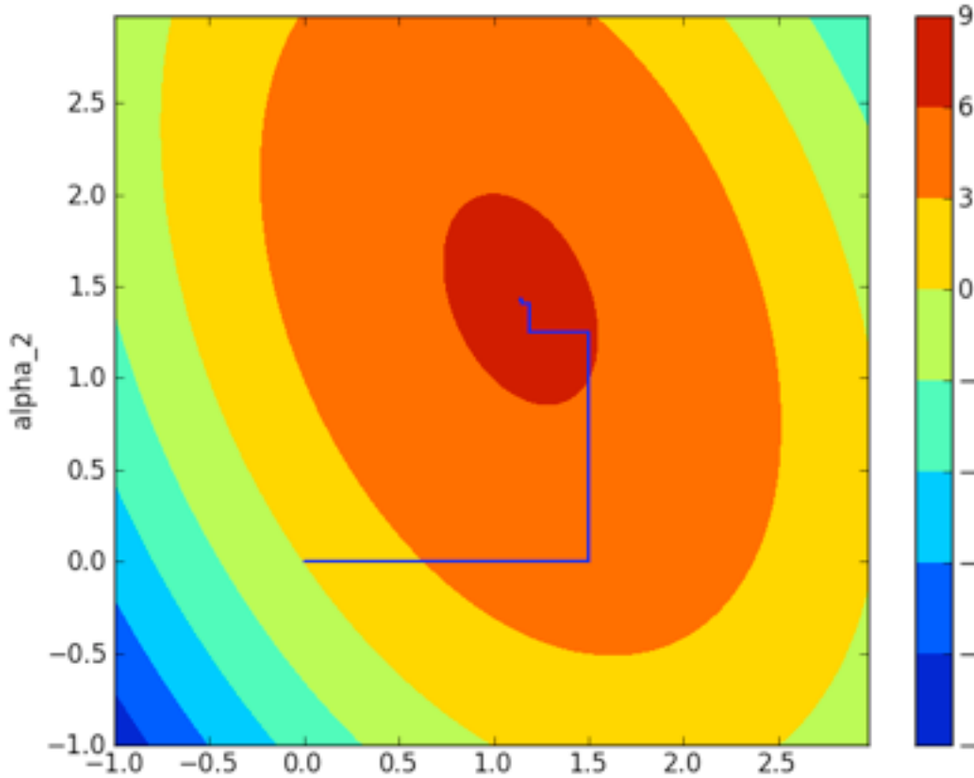
pick i following sweep pattern

solve

$$\alpha_i \leftarrow \underset{\alpha_i}{\text{argmax}} -\frac{1}{2}\alpha^T Q \alpha - \alpha^T b$$

$$\text{subject to } \alpha \geq 0$$

until meet stopping criterion

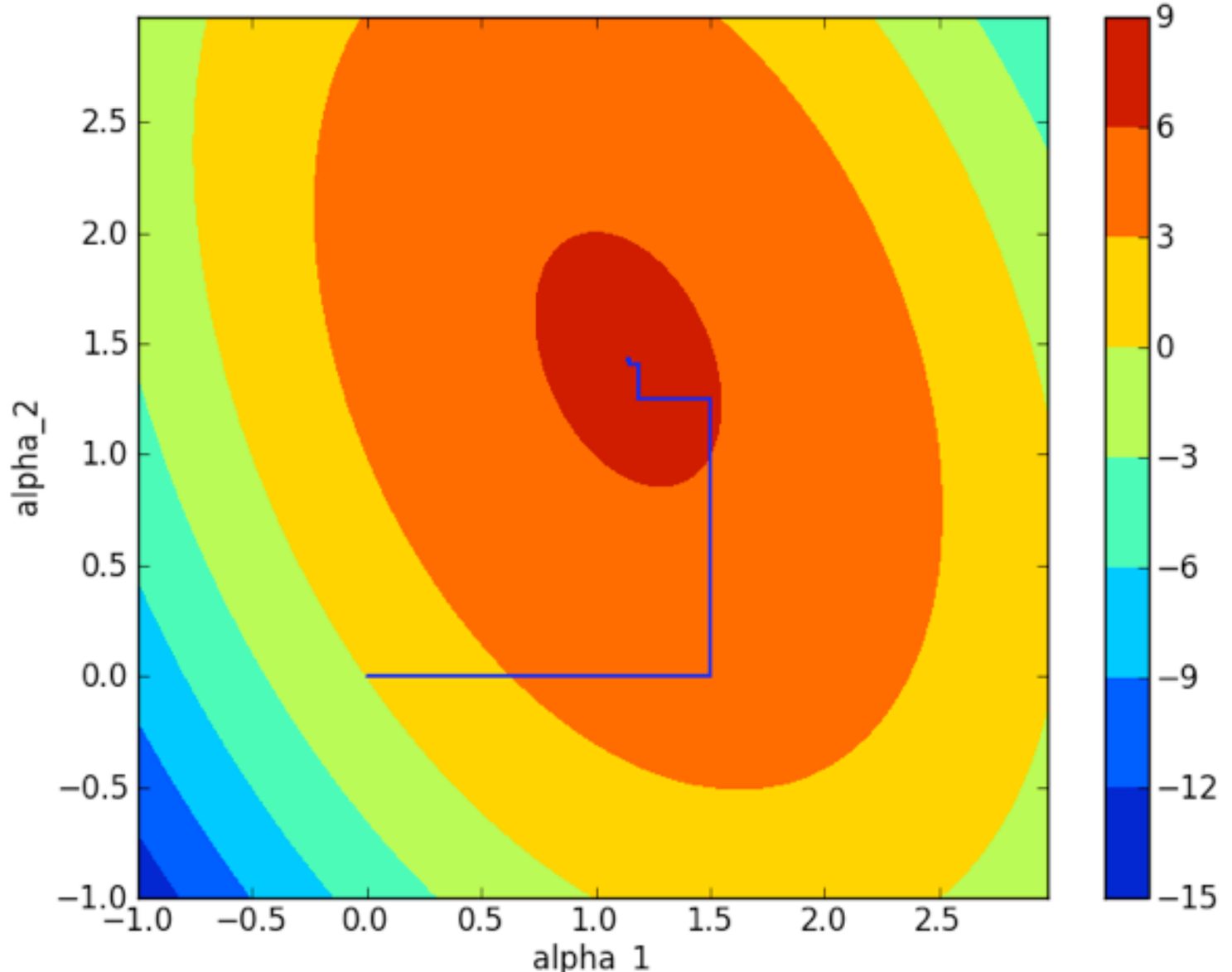


QP: Hildreth Algorithm

$$\underset{\alpha}{\text{maximize}} -\frac{1}{2}\alpha^T \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix} \alpha - \alpha^T \begin{pmatrix} -6 \\ -4 \end{pmatrix}$$

$$\text{subject to } \alpha \geq 0$$

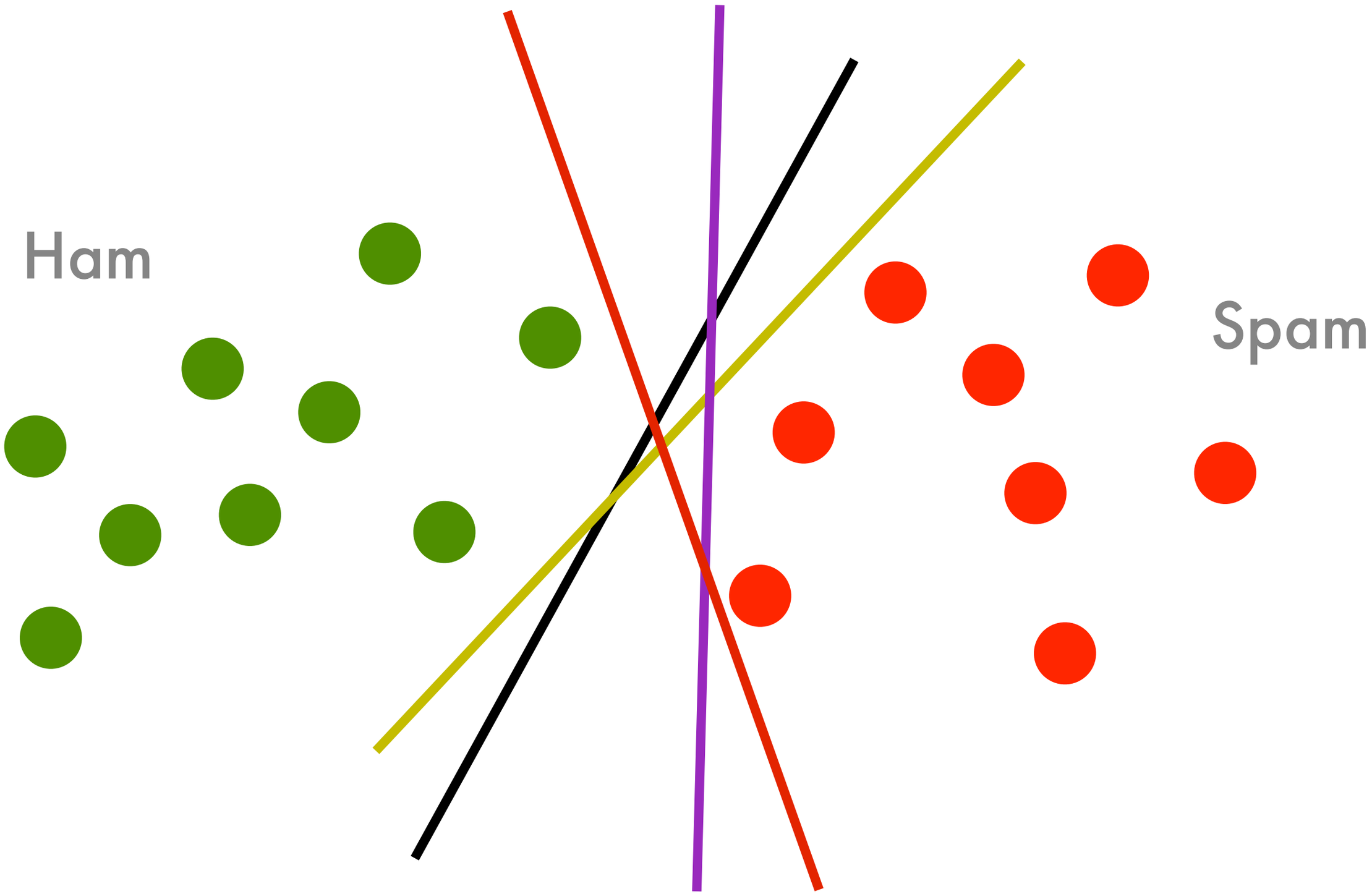
- choose coordinates
  - 1, 2, 1, 2, ...



QP: Hildreth Algorithm

- pros:
  - extremely simple
  - no gradient calculation
  - easy to implement
- cons:
  - converges slow, compared to other methods
  - can't deal with too many constraints
    - works for minibatch MIRA but not SVM

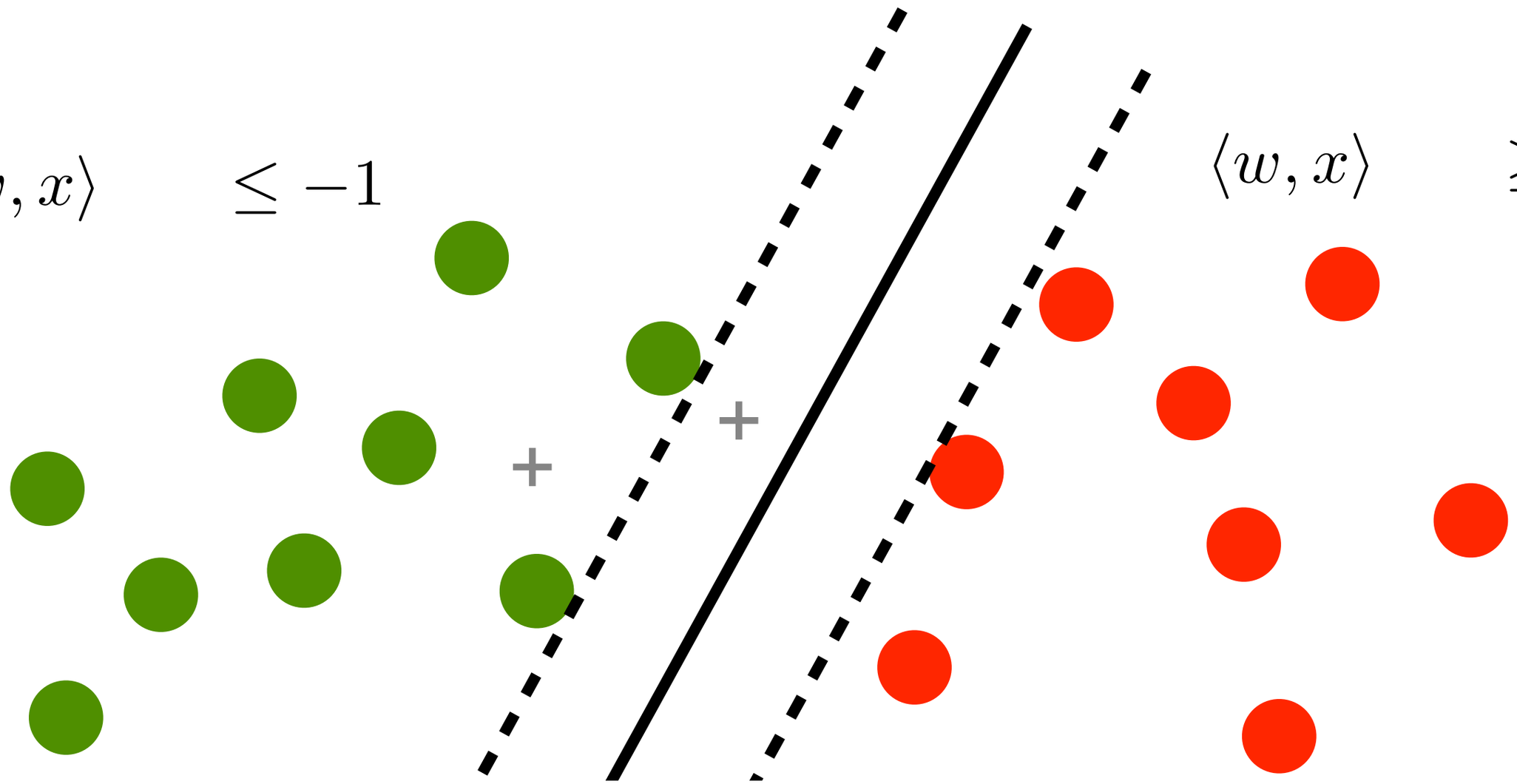
Linear Separator



Large Margin Classifier

$\langle w, x \rangle \quad \leq -1$

$\langle w, x \rangle \quad \geq 1$



linear function

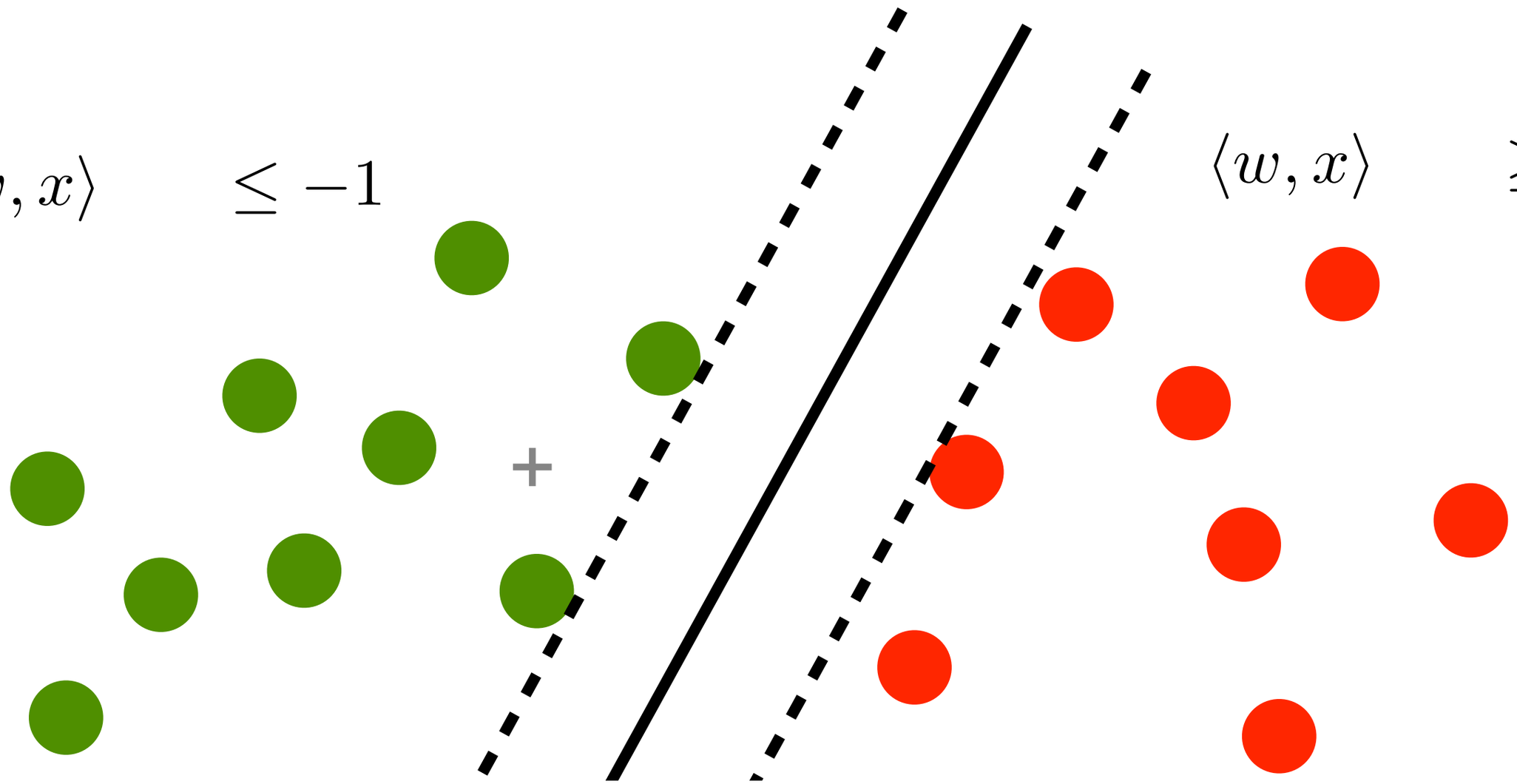
$$f(x) = \langle w, x \rangle + b$$

linear separator is impossible

Large Margin Classifier

$\langle w, x \rangle \quad \leq -1$

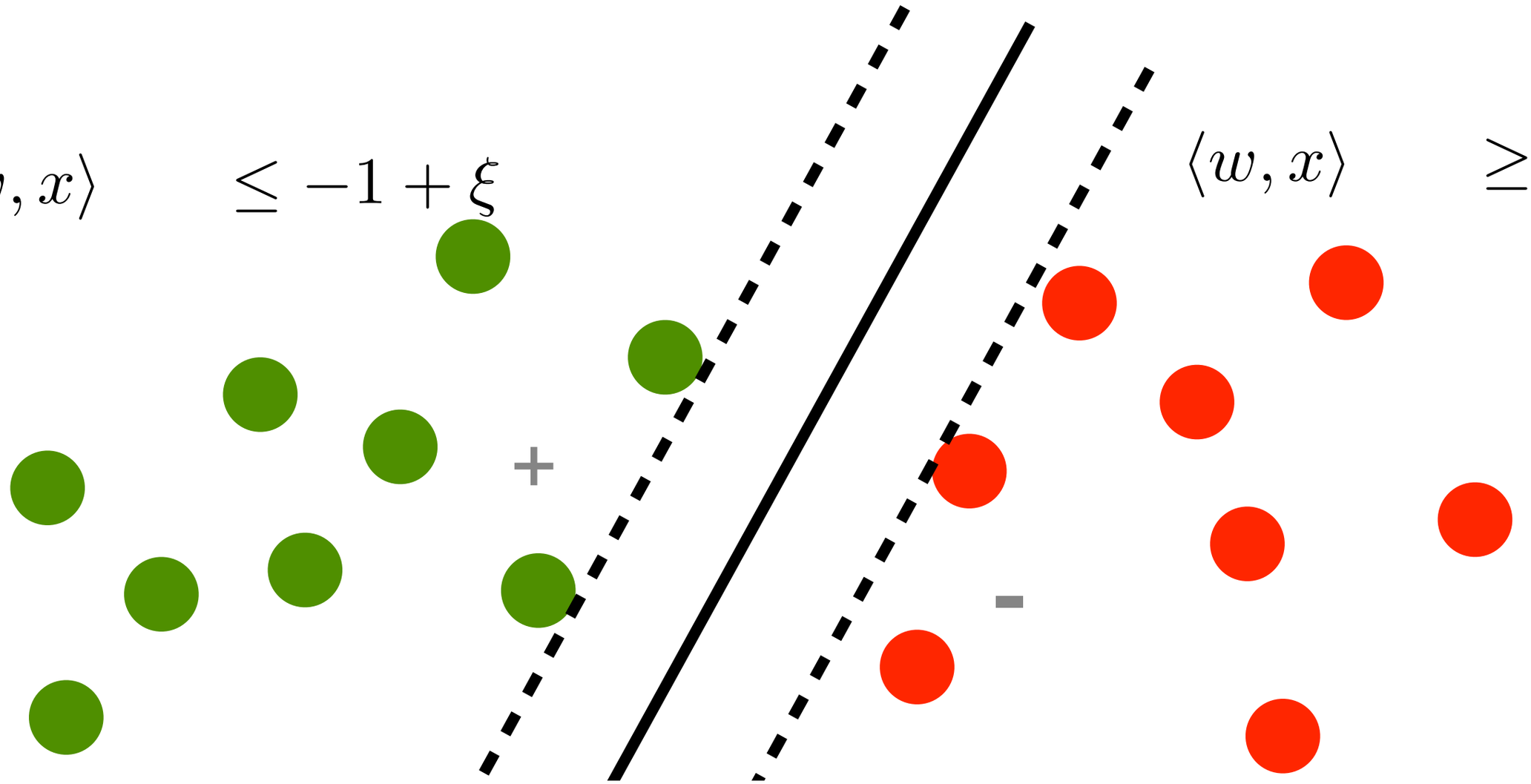
$\langle w, x \rangle \quad \geq 1$



minimum error separator is impossible

Theorem (Minsky & Papert)
Finding the minimum error separating hyperplane is NP hard

Adding slack variables



Convex optimization problem

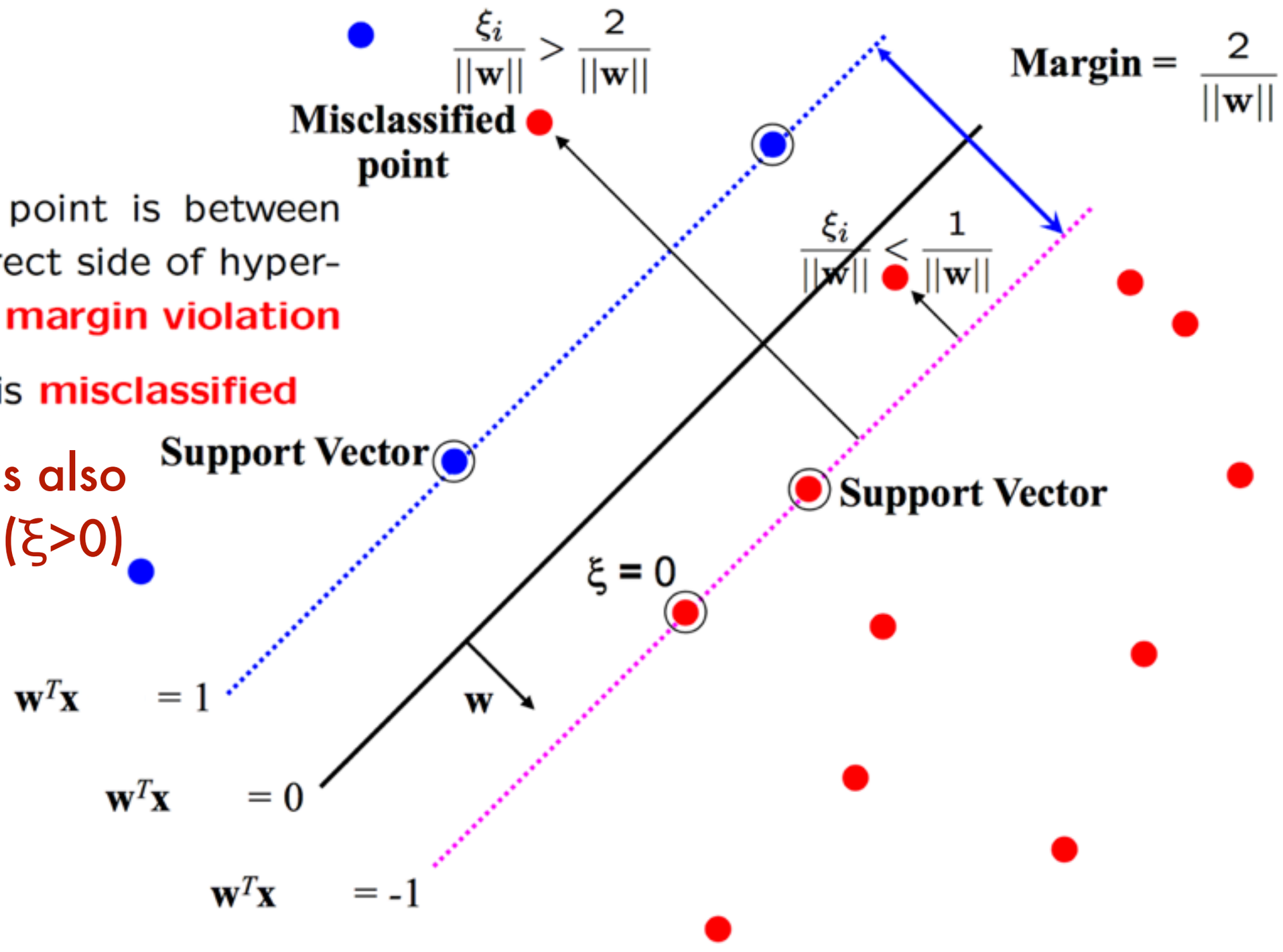
minimize amount of slack

margin violation vs. misclassification

$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**

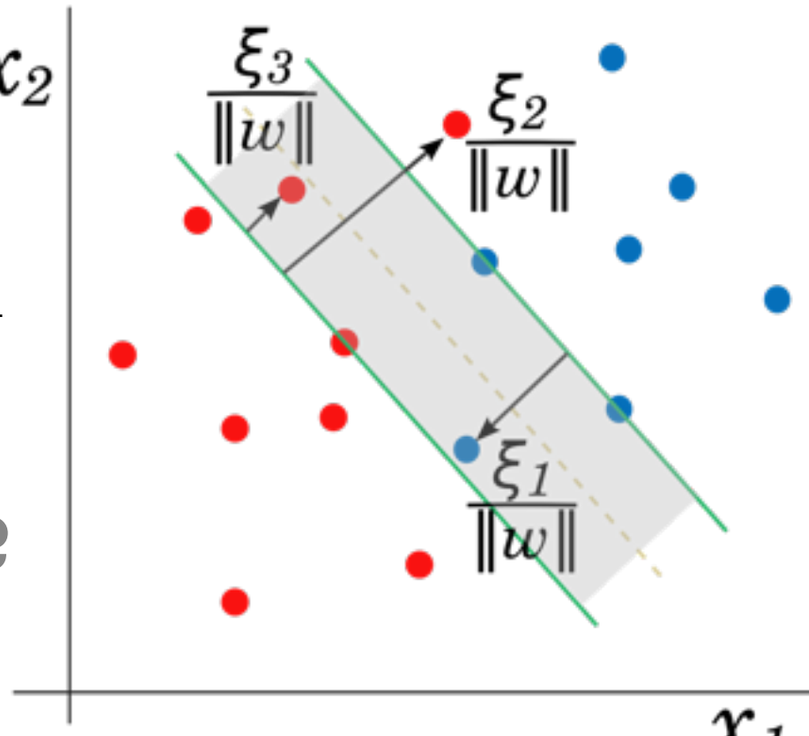
misclassification is also margin violation (ξ>0)



Adding slack variables

- Hard margin problem

$$\underset{w}{\text{minimize}}\ \frac{1}{2}\|w\|^2 \ \text{ subject to } y_i\left[\langle w, x_i\rangle\right. \qquad \geq 1$$



- With slack variables C=0? C=+∞?

$$\underset{w\ \ \xi}{\text{minimize}}\ \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

w determines **ξ**

$$\text{subject to } y_i\left[\langle w, x_i\rangle\right. \qquad \geq 1-\xi_i \text{ and } \xi_i \geq 0$$

Problem is always feasible. Proof:

$w = 0$ and $\xi_i = 1$ (also yields upper bound)

C=+∞ => not tolerant on violation => hard margin

Review: Convex Optimization

- Primal optimization problem

$$\underset{x}{\text{minimize}}\, f(x) \text{ subject to } c_i(x) \leq 0$$

- Lagrange function

$$L(x, \alpha) = f(x) + \sum_i \alpha_i c_i(x)$$

- First order optimality conditions in x

$$\partial_x L(x, \alpha) = \partial_x f(x) + \sum_i \alpha_i \partial_x c_i(x) = 0$$

- Solve for x and plug it back into L

$$\underset{\alpha}{\text{maximize}}\, L(x(\alpha), \alpha)$$

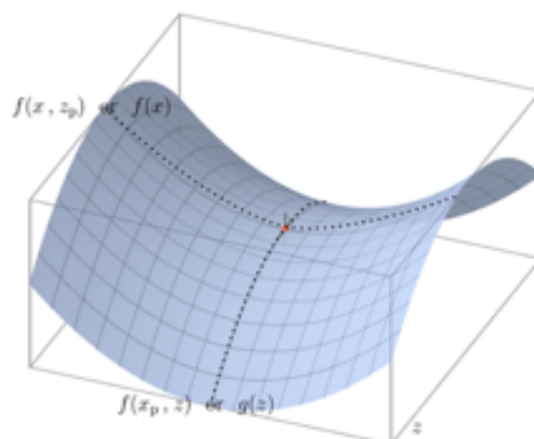
(keep explicit constraints)

Dual Problem

- Primal optimization problem

$$\underset{w,\,\xi}{\text{minimize}} \;\; \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$\text{subject to } y_i\left[\langle w, x_i\rangle + b\right] \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$



- Lagrange function

$$L(w, \xi, \alpha, \eta) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left[y_i\left[\langle x_i, w\rangle + b\right] + \xi_i - 1\right] - \sum_i \eta_i \xi_i$$

optimality in (w, ξ) is at saddle point with α,η

- Derivatives in (w, ξ) need to vanish

Dual Problem

- Lagrange function

$$L(w, \xi, \alpha, \eta) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left[y_i \left[\langle x_i, w\rangle + b\right] + \xi_i - 1\right] - \sum_i \eta_i \xi_i$$

- Derivatives in w need to vanish

$$\partial_w L(w, b, \xi, \alpha, \eta) = w - \sum_i \alpha_i y_i x_i = 0$$

$$\partial_{\xi_i} L(w, b, \xi, \alpha, \eta) = C - \alpha_i - \eta_i = 0$$

- Plugging terms back into L yields

$$\underset{\alpha}{\text{maximize}} -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle + \sum_i \alpha_i$$

subject to **dual variables** $\alpha_i \in [0, C]$

bound influence

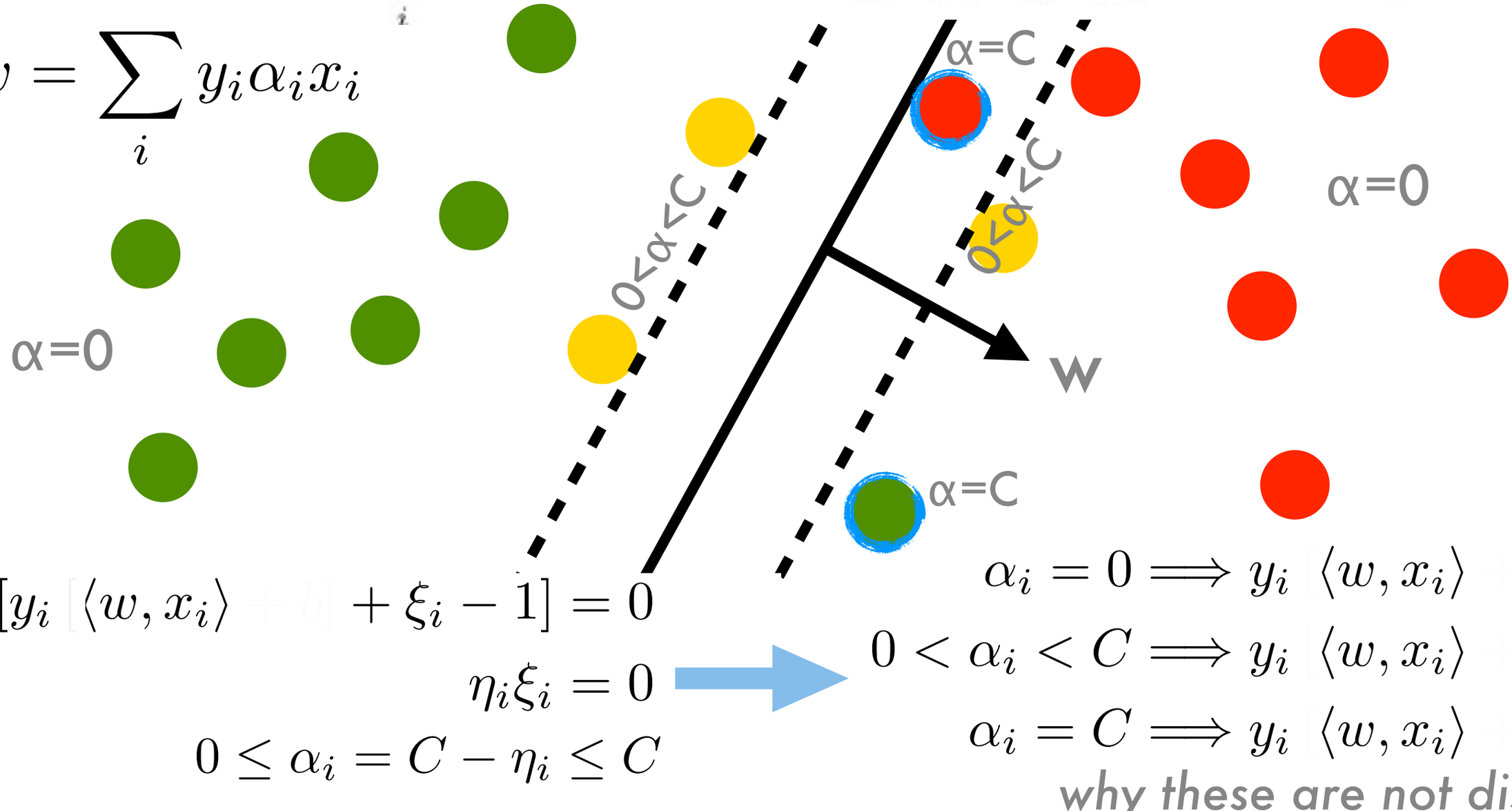
Karush Kuhn Tucker Conditions

$$L(w, \xi, \alpha, \eta) = \frac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \left[y_i \left[\langle x_i, w\rangle + b\right] + \xi_i - 1\right] - \sum_i \eta_i \xi_i$$

$$\partial_w L(w, b, \xi, \alpha, \eta) = w - \sum_i \alpha_i y_i x_i = 0$$

$$\partial_{\xi_i} L(w, b, \xi, \alpha, \eta) = C - \alpha_i - \eta_i = 0$$

$$w = \sum_i y_i \alpha_i x_i$$



$$\alpha_i \left[y_i \left[\langle w, x_i\rangle + b\right] + \xi_i - 1\right] = 0$$

$$\eta_i \xi_i = 0$$

$$0 \le \alpha_i = C - \eta_i \le C$$

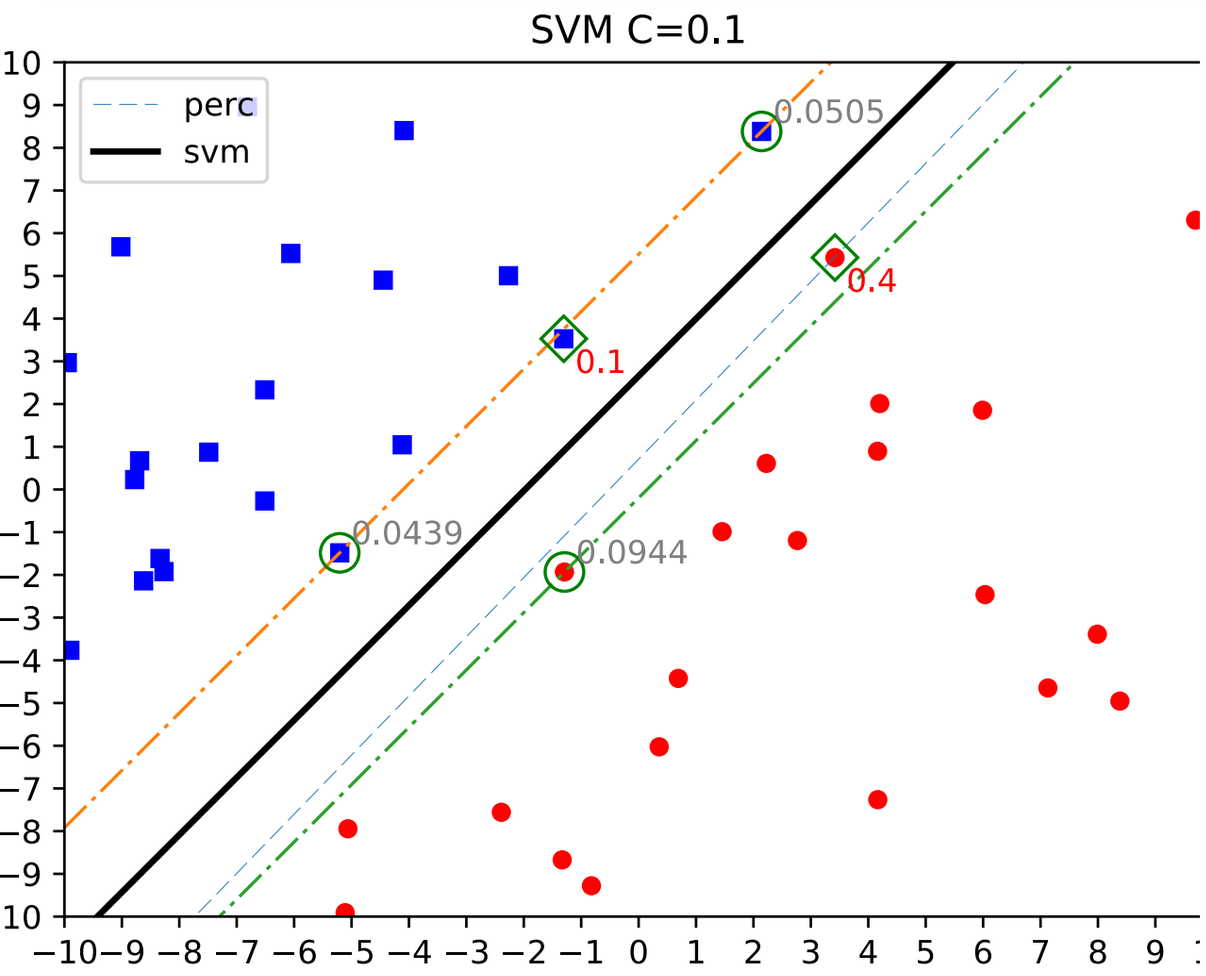
$$\alpha_i = 0 \Longrightarrow y_i \left[\langle w, x_i\rangle + b\right] \ge 1$$

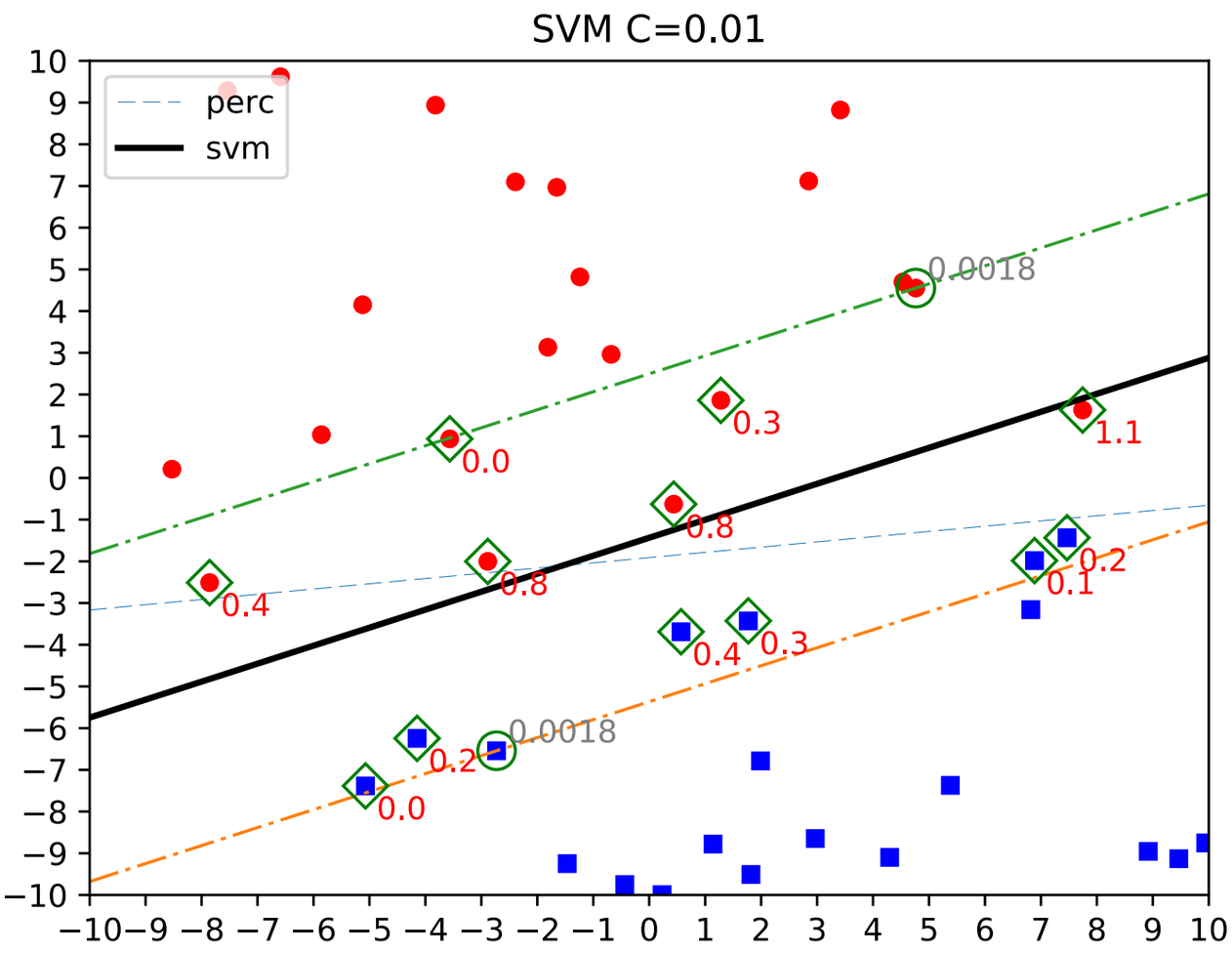
$$0 < \alpha_i < C \Longrightarrow y_i \left[\langle w, x_i\rangle + b\right] = 1$$

$$\alpha_i = C \Longrightarrow y_i \left[\langle w, x_i\rangle + b\right] \le 1$$

why these are not disjoint?

Support Vectors and Violations

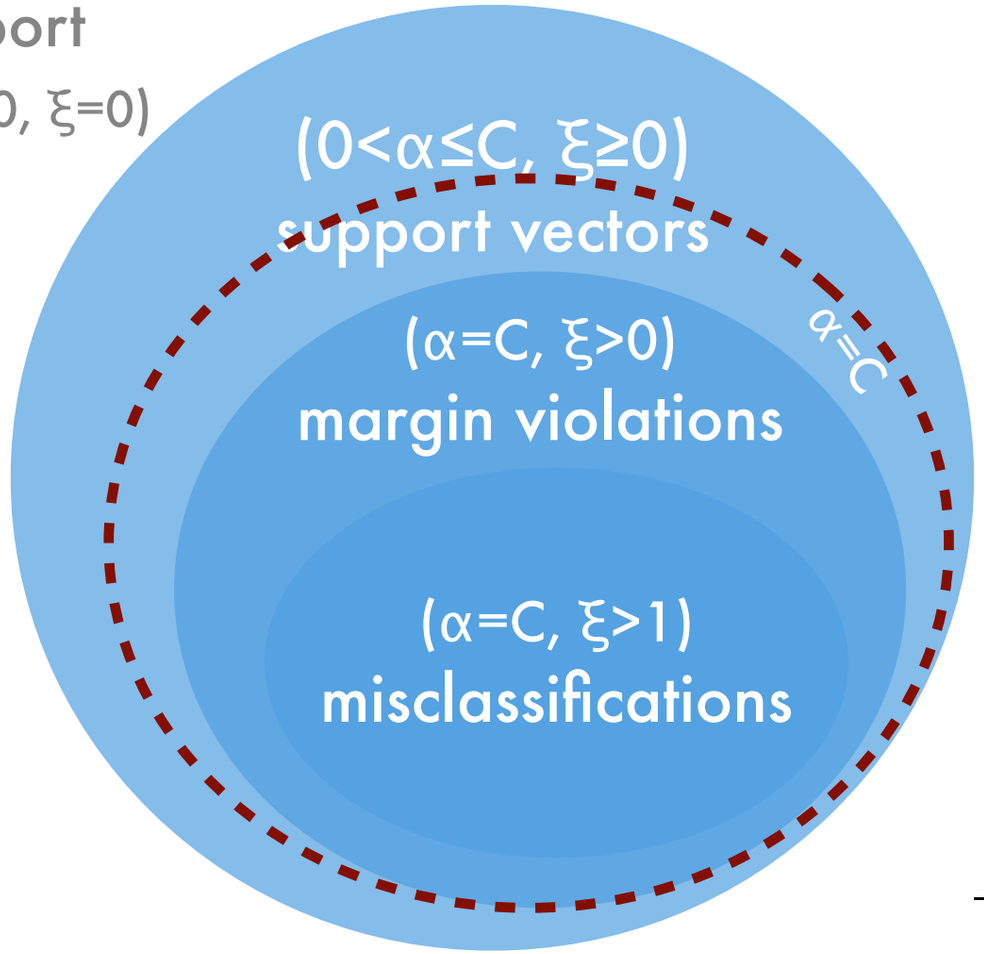


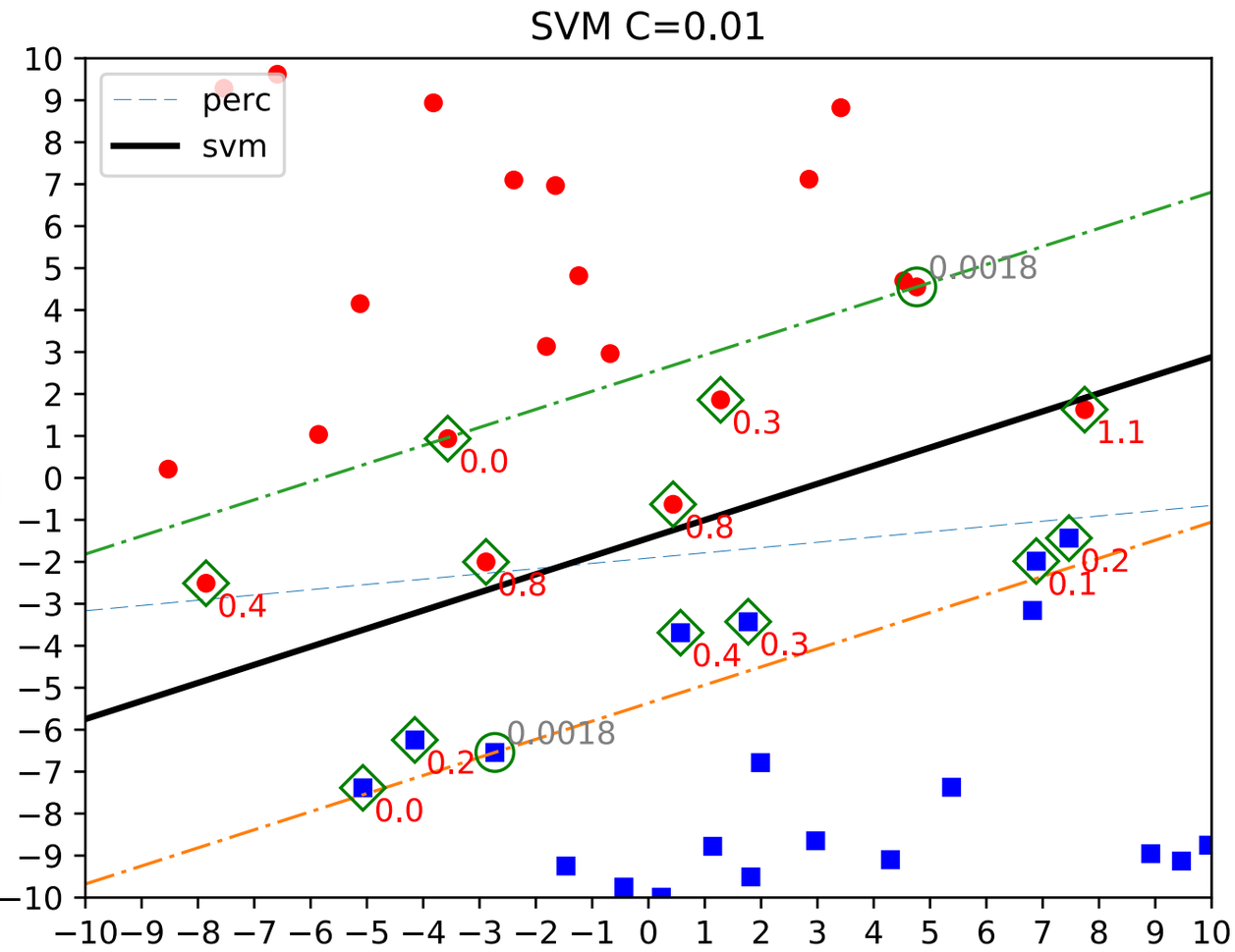


all circled and squared examples are support vectors ($\alpha>0$)
they include $\xi=0$ (hard-margin SVs), $\xi>0$ (margin violations),
and $\xi>1$ (misclassifications)

Support Vectors and Violations

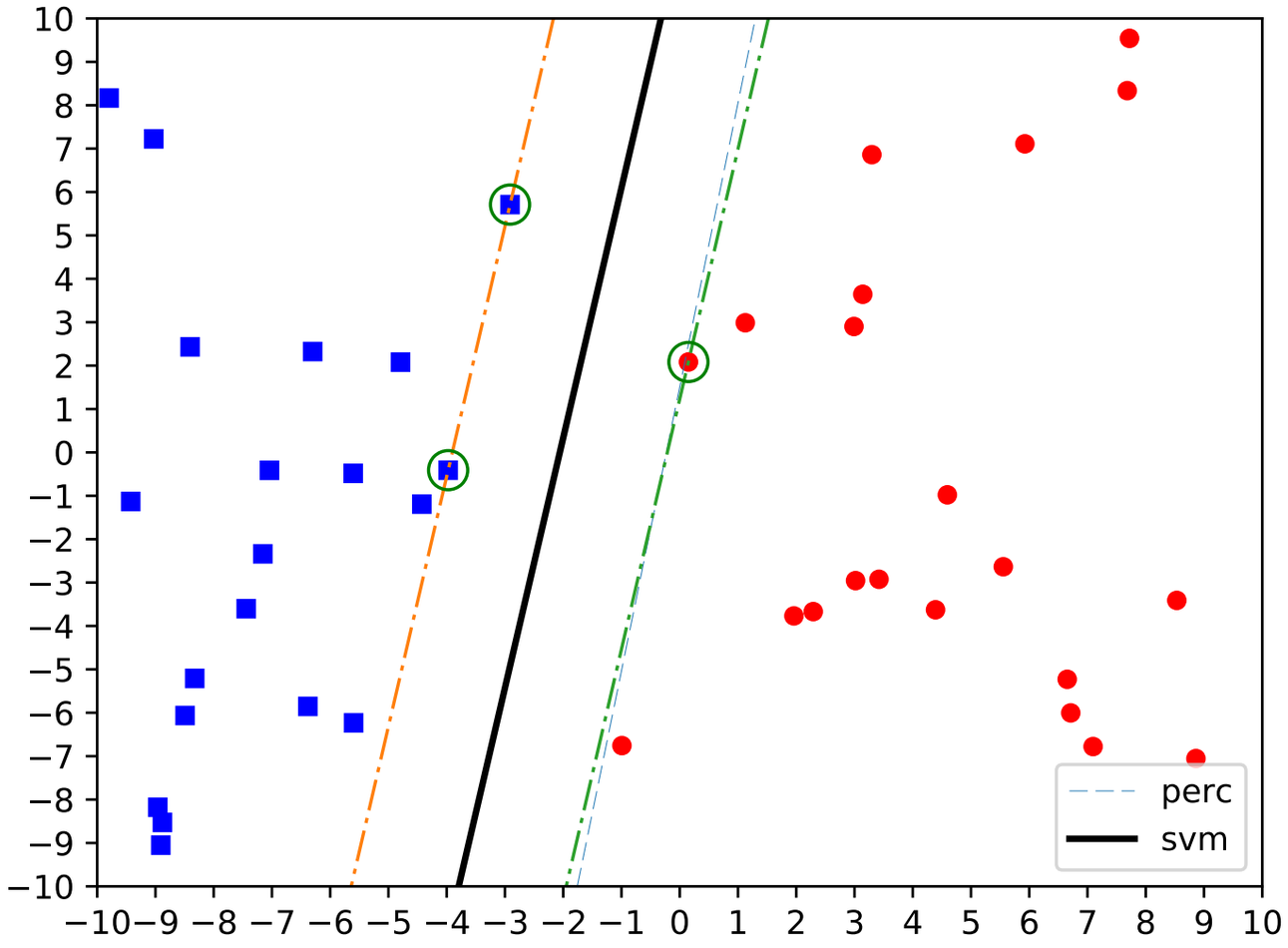
non-support vectors (α=0, ξ=0)



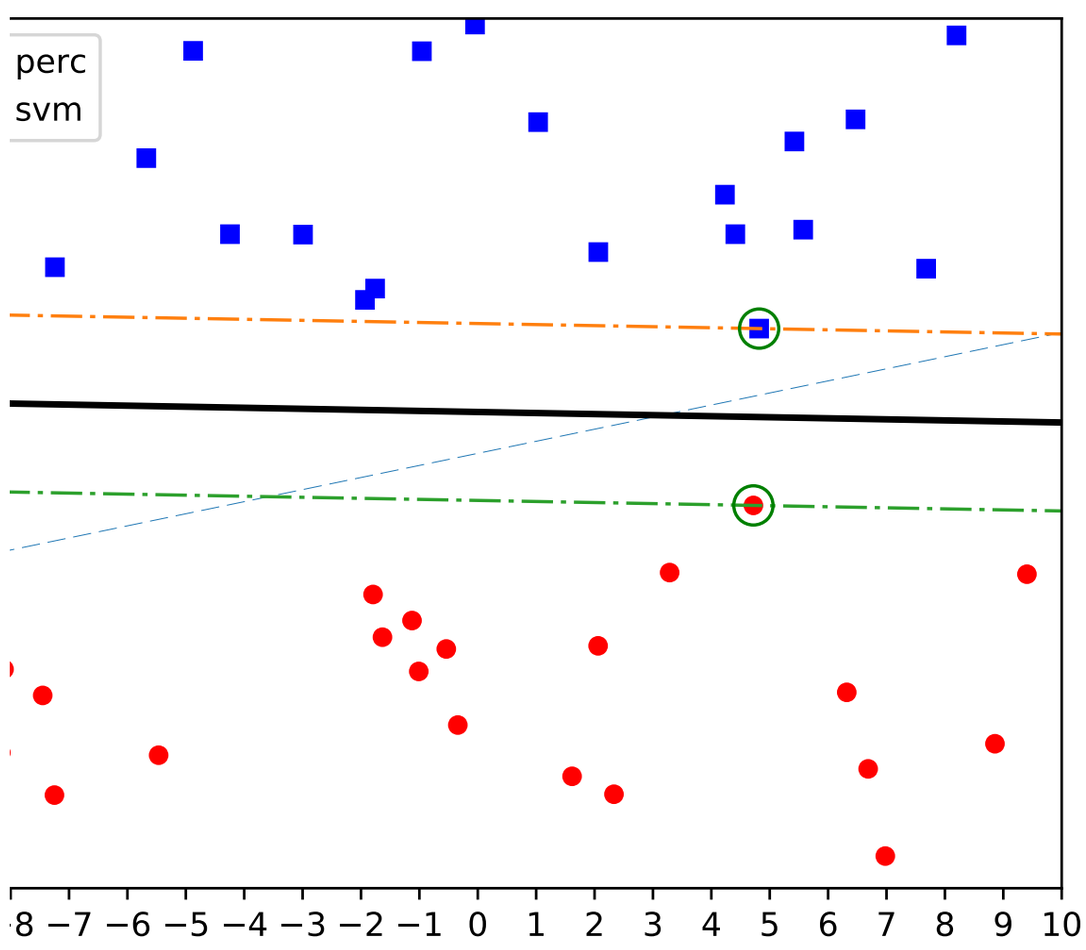


all circled and squared examples are support vectors (α>0)
they include ξ=0 (hard-margin SVs), ξ>0 (margin violations), and ξ>1 (misclassifications)

SVM with sklearn



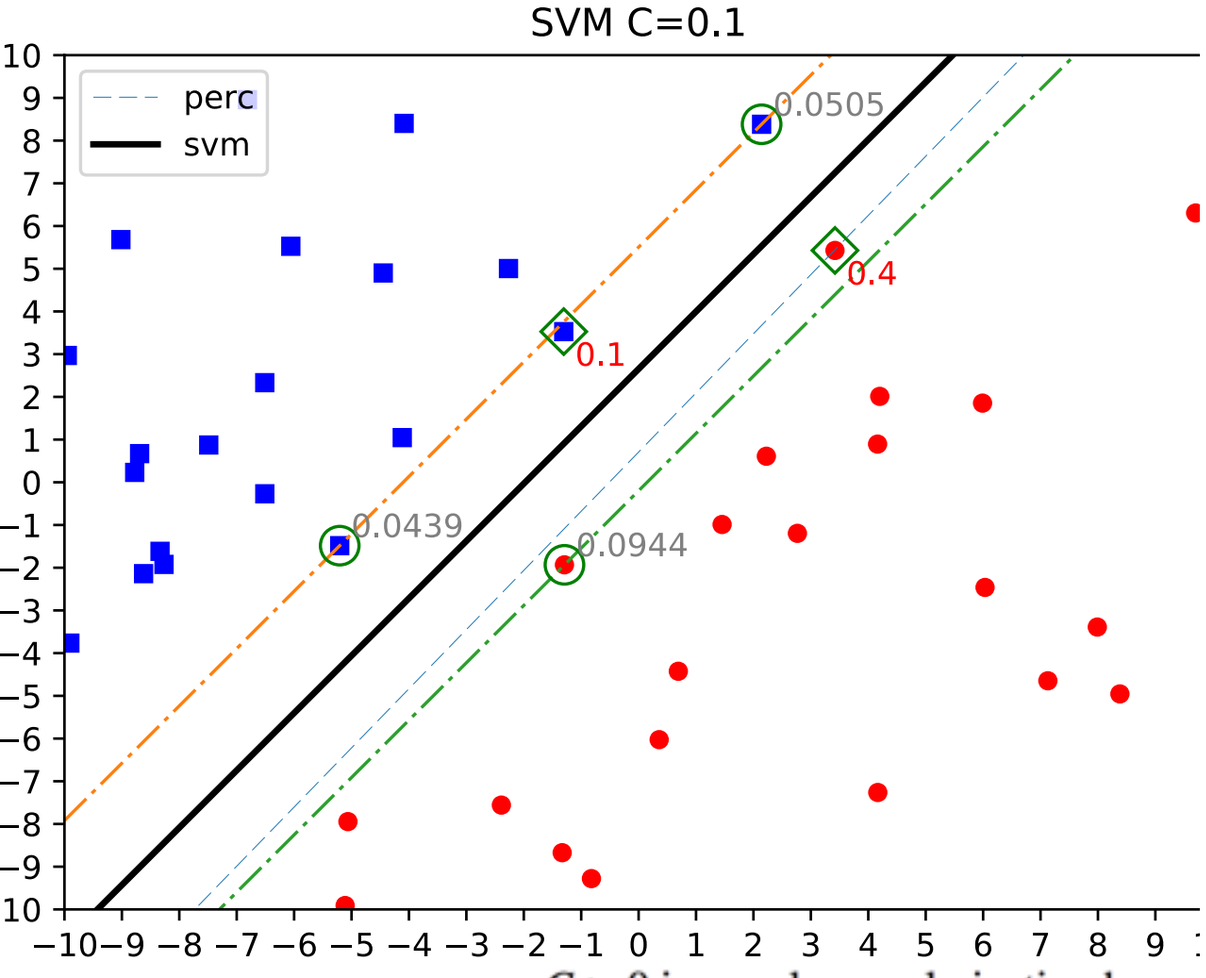
python demo.py 1e10



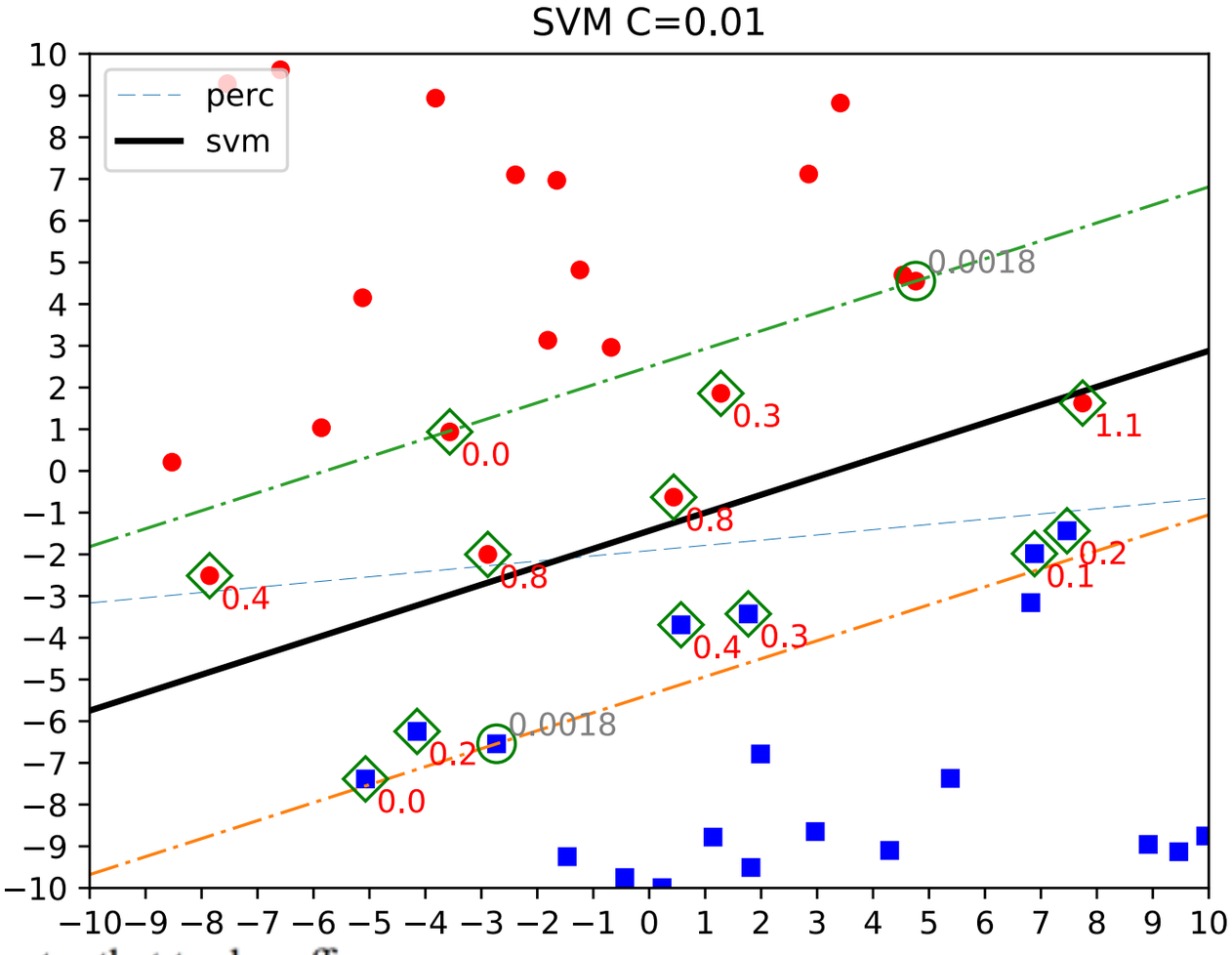
python demo.py 1e10

SVM with sklearn

`python demo.py 0.1`



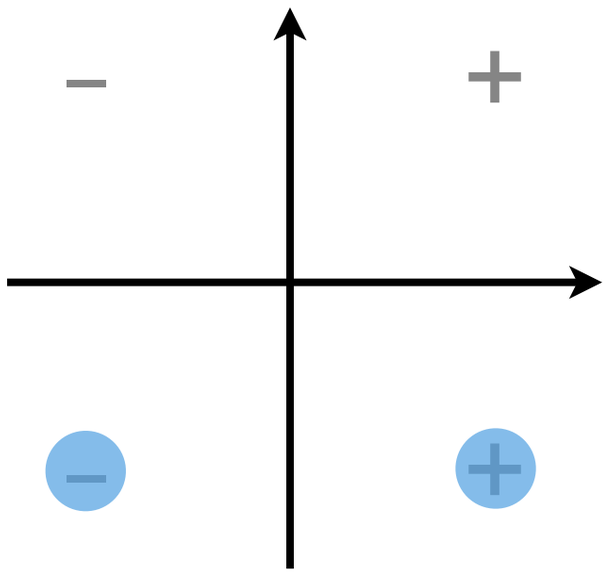
`python demo.py 0.01`



$C > 0$ is a scalar regularization hyperparameter that trades off:

	small C	big C
desire	maximize margin $1/\|w\|$	keep most slack variables zero or small
danger	underfitting (misclassifies much training data)	overfitting (awesome training, awful test)
outliers	less sensitive	very sensitive
boundary	more “flat”	more sinuous

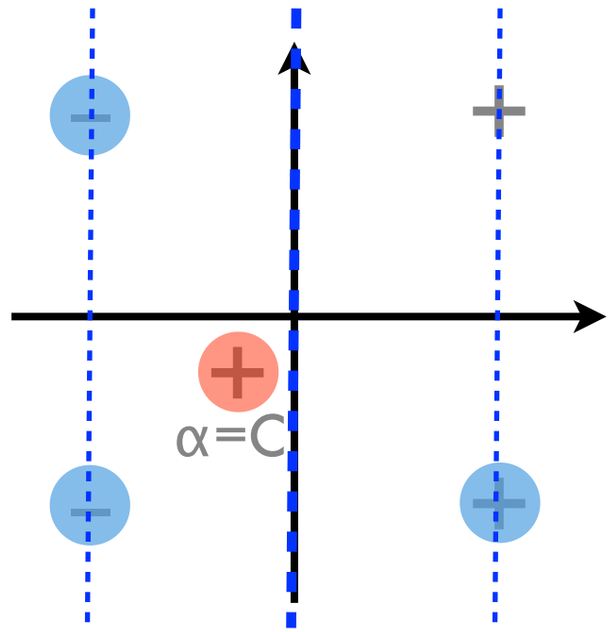
SVM with sklearn



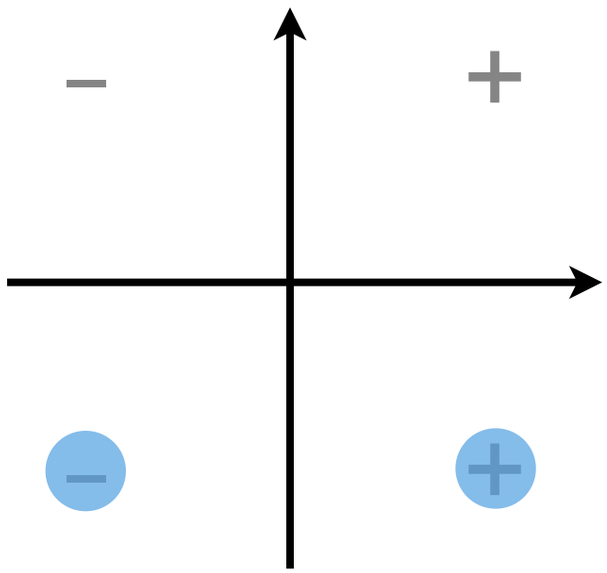

```
In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[-1., -1.], [ 1., -1.]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)
```

```
In [2]: X = [[1,1], [1,-1], [-1,1], [-1,-1], [-0.1,-0.1]]
In [3]: Y = [1,1,-1,-1, 1]
In [4]: clf = svm.SVC(kernel='linear', C=1)
In [5]: clf.fit(X, Y)
In [6]: clf.support_vectors_
Out[6]: array([[-1. ,  1. ],
               [-1. , -1. ],
               [ 1. , -1. ],
               [-0.1, -0.1]])
In [7]: clf.dual_coef_
Out[7]: array([[-0.45, -0.6 ,  0.05,  1.  ]])
In [8]: clf.coef_
Out[8]: array([[  1.00000000e+00,   1.49011611e-09]])
In [9]: clf.intercept_
Out[9]: array([-0.])
```

α=C

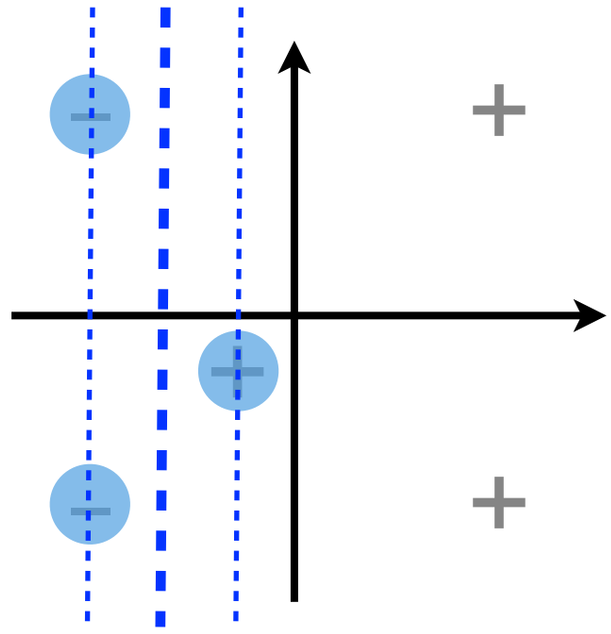


SVM with sklearn

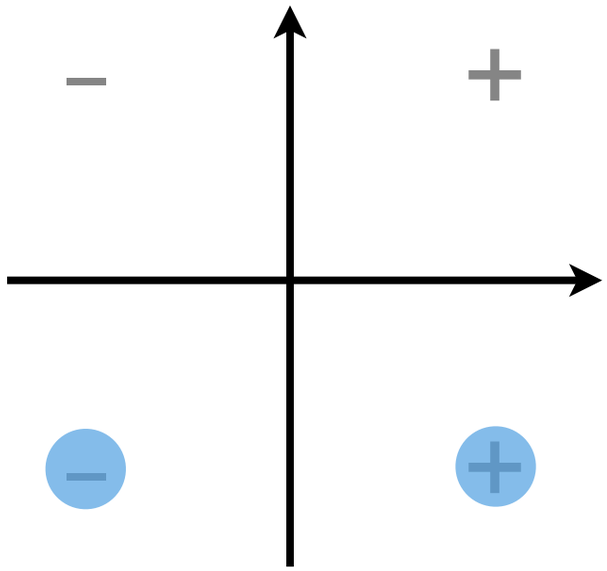



```
In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[-1., -1.], [ 1., -1.]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)
```

```
In [2]: X = [[1,1], [1,-1], [-1,1], [-1,-1], [-0.1,-0.1]]
In [3]: Y = [1,1,-1,-1, 1]
In [12]: clf = svm.SVC(kernel='linear', C=1e10)
In [13]: clf.fit(X, Y)
In [14]: clf.coef_
Out[14]: array([[  2.02010102e+00,   1.00999543e-04]])
In [15]: clf.intercept_
Out[15]: array([ 1.02013469])
In [16]: clf.support_vectors_
Out[16]: array([[-1.  ,  1.  ],
                [-1.  , -1.  ],
                [-0.01, -0.01]])
In [17]: clf.dual_coef_
Out[17]: array([[-1.01000001, -1.03050607,  2.04050608]])
```



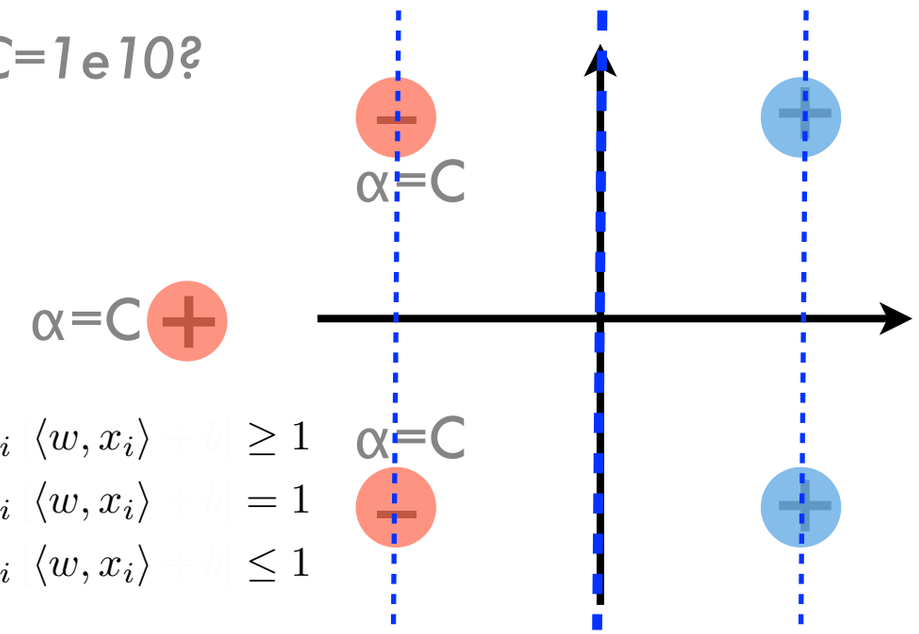
SVM with sklearn




```
In [2]: clf = svm.SVC(kernel='linear', C=1e10)
In [3]: X = [[1,1], [1,-1], [-1,1], [-1,-1]]
In [4]: Y = [1,1,-1,-1]
In [5]: clf.fit(X, Y)
In [6]: clf.support_
Out[6]: array([3, 1], dtype=int32)
In [7]: clf.dual_coef_
Out[7]: array([[-0.5,  0.5]])
In [8]: clf.coef_
Out[8]: array([[ 1.,  0.]])
In [9]: clf.intercept_
Out[9]: array([-0.])
In [10]: clf.support_vectors_
Out[10]: array([[-1., -1.], [ 1., -1.]])
In [11]: clf.n_support_
Out[11]: array([1, 1], dtype=int32)
```

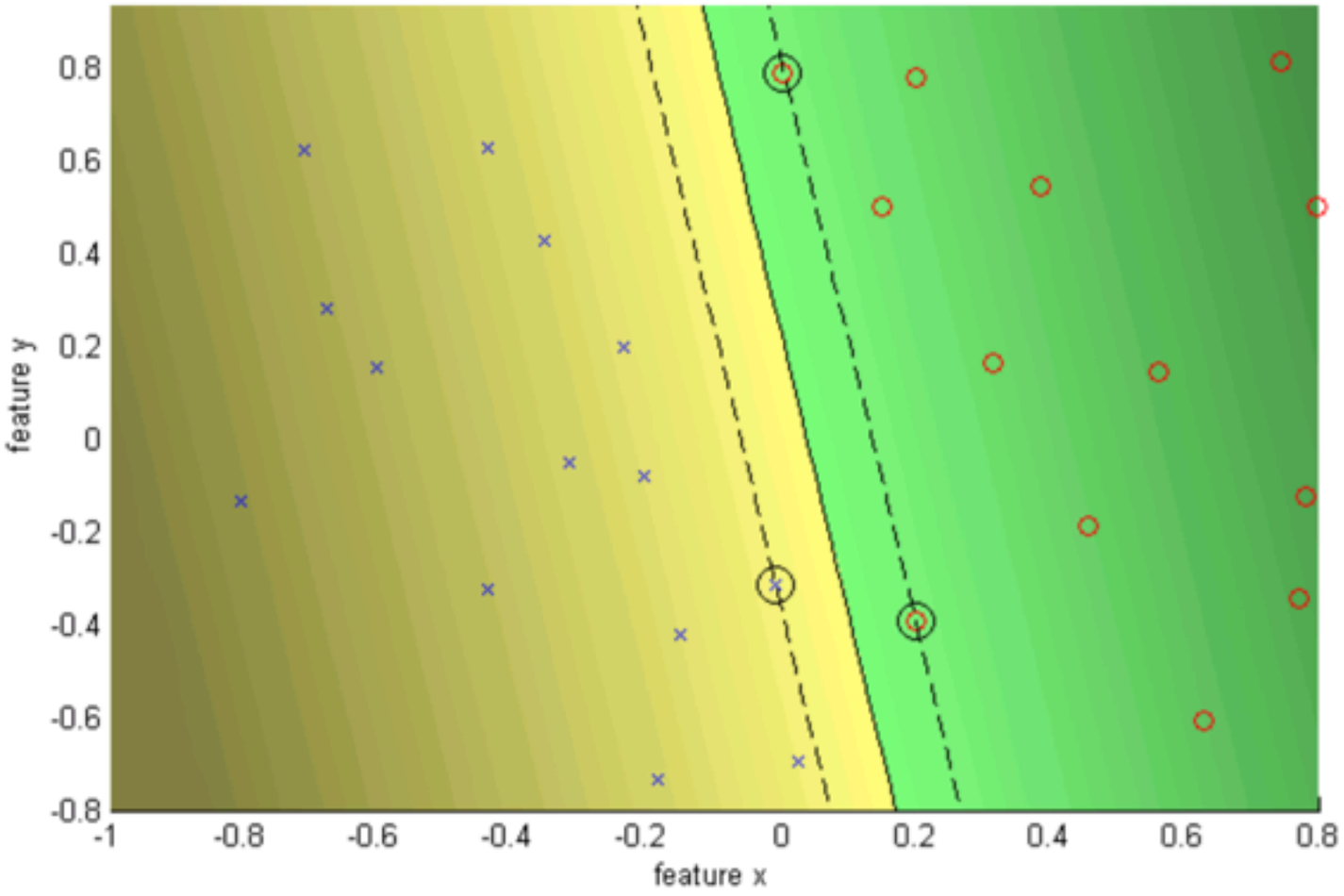
```
In [2]: X = [[1,1], [1,-1], [-1,1], [-1,-1], [-2,0]]
In [3]: Y = [1,1,-1,-1, 1]
In [4]: clf = svm.SVC(kernel='linear', C=1)
In [5]: clf.fit(X, Y)
In [6]: clf.coef_
Out[6]: array([[ 1.,  0.]])
In [7]: clf.support_vectors_
Out[7]: array([[-1.,  1.],
               [-1., -1.],
               [ 1.,  1.],
               [ 1., -1.],
               [-2.,  0.]])
In [8]: clf.dual_coef_
Out[8]: array([[-1. , -1. ,  0.5,  0.5,  1. ]])
In [9]: clf.intercept_
Out[9]: array([-0.])
```

what if C=1e10?



$$\alpha_i = 0 \Longrightarrow y_i \langle w, x_i \rangle + b \geq 1$$
$$0 < \alpha_i < C \Longrightarrow y_i \langle w, x_i \rangle + b = 1$$
$$\alpha_i = C \Longrightarrow y_i \langle w, x_i \rangle + b \leq 1$$

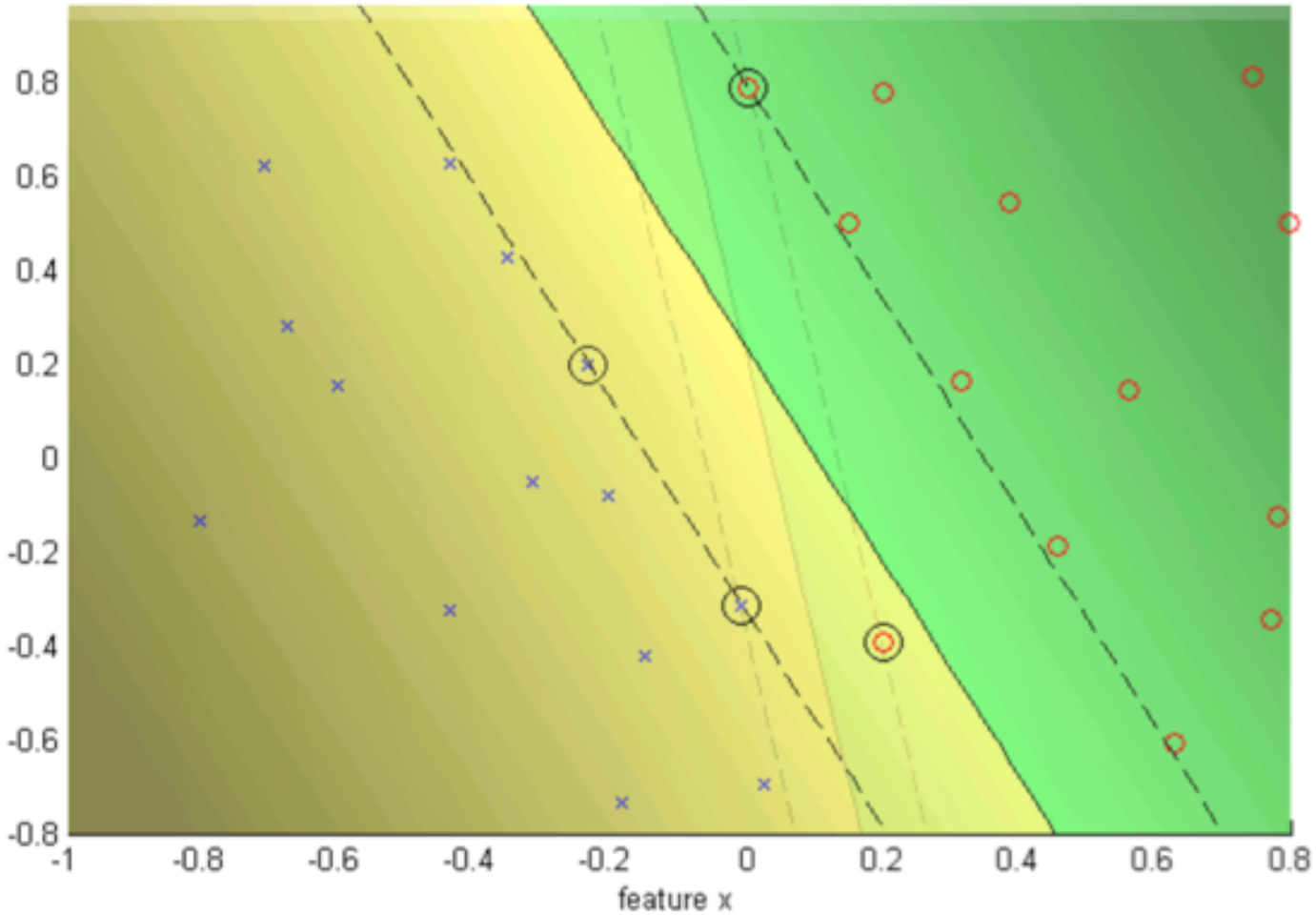
hard vs. soft margins



Comment Window

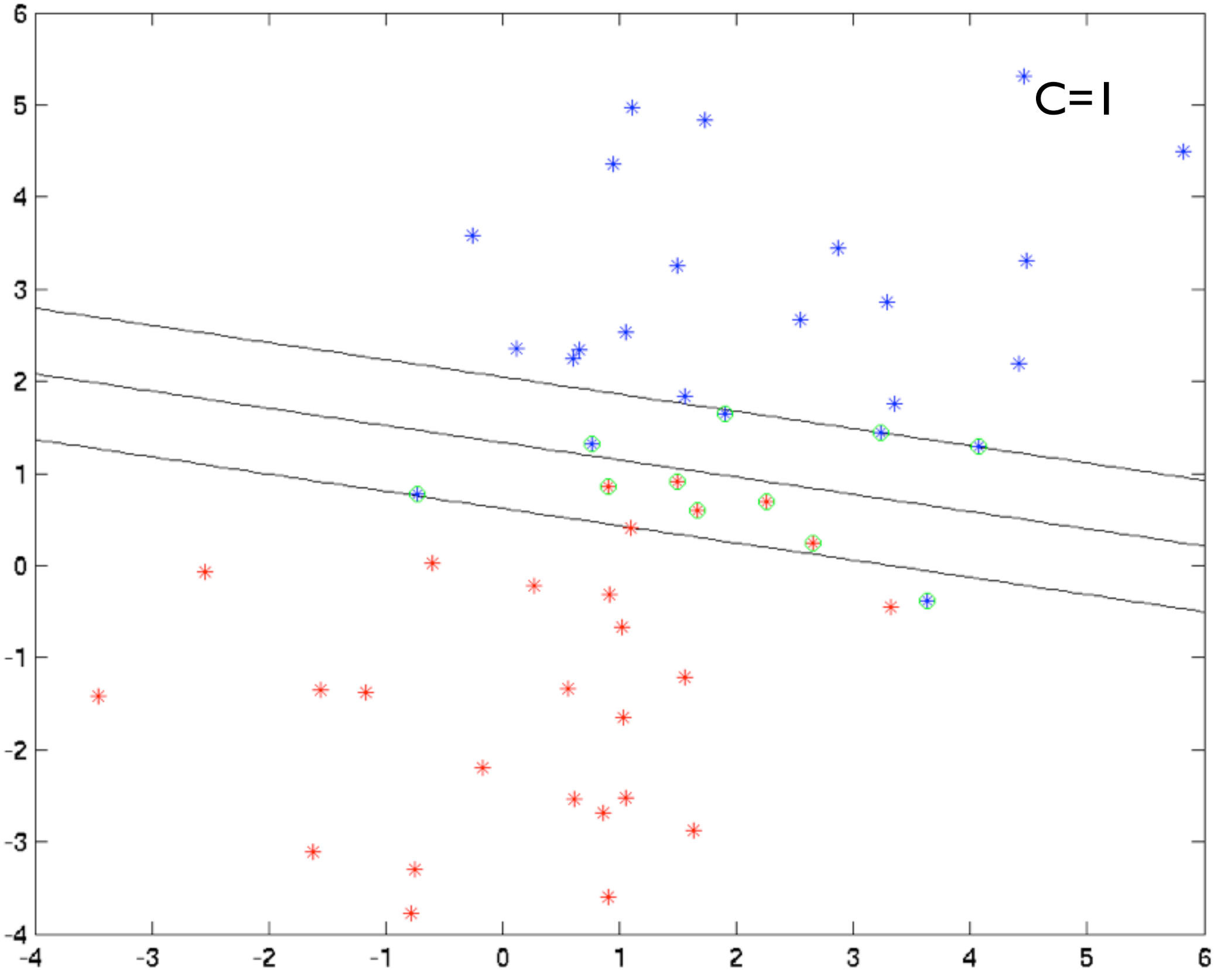
SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: Inf
Kernel evaluations: 971
Number of Support Vectors: 3
Margin: 0.0966
Training error: 0.00%

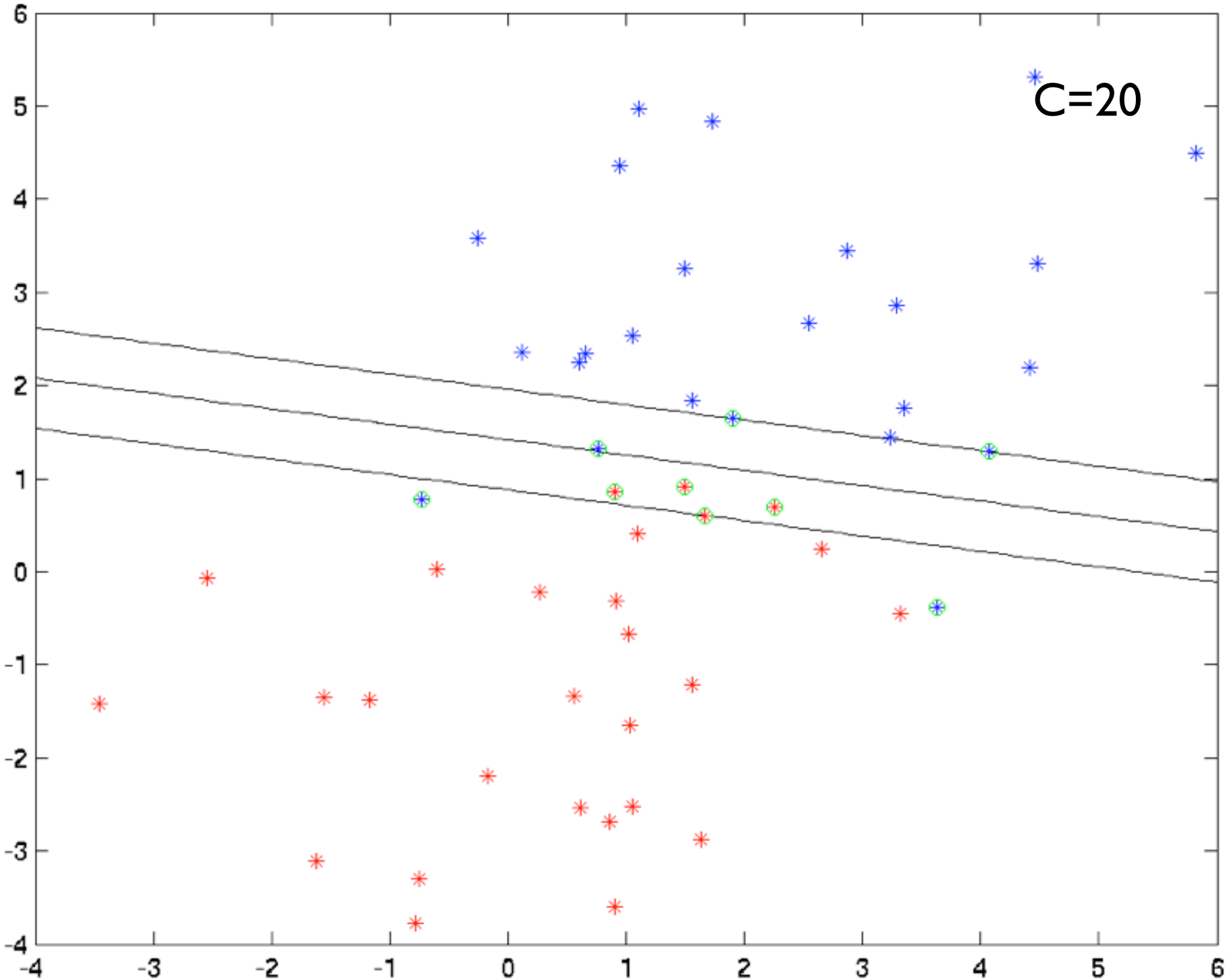
hard vs. soft margins



Comment Window

SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: 10.0000
Kernel evaluations: 2645
Number of Support Vectors: 4
Margin: 0.2265
Training error: 3.70%


C=1
6
5
4
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4
5
6


C=20
6
5
4
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3
4
5
6

From Constrained Optimization to Unconstrained Optimization (back to Primal)

Learning an SVM has been formulated as a constrained optimization problem over $\mathbf{w}$ and ξ

$$\min_{\mathbf{w}\in\mathbb{R}^d,\xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i \text{ subject to } y_i\left(\mathbf{w}^\top\mathbf{x}_i \quad\right) \geq 1-\xi_i \text{ for } i=1\ldots N$$

The constraint $y_i\left(\mathbf{w}^\top\mathbf{x}_i \quad\right) \geq 1-\xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1-\xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max(0, 1-y_i f(\mathbf{x}_i))$$

$\mathbf{w}$ determines $\boldsymbol{\xi}$

Hence the learning problem is equivalent to the unconstrained optimization problem over $\mathbf{w}$

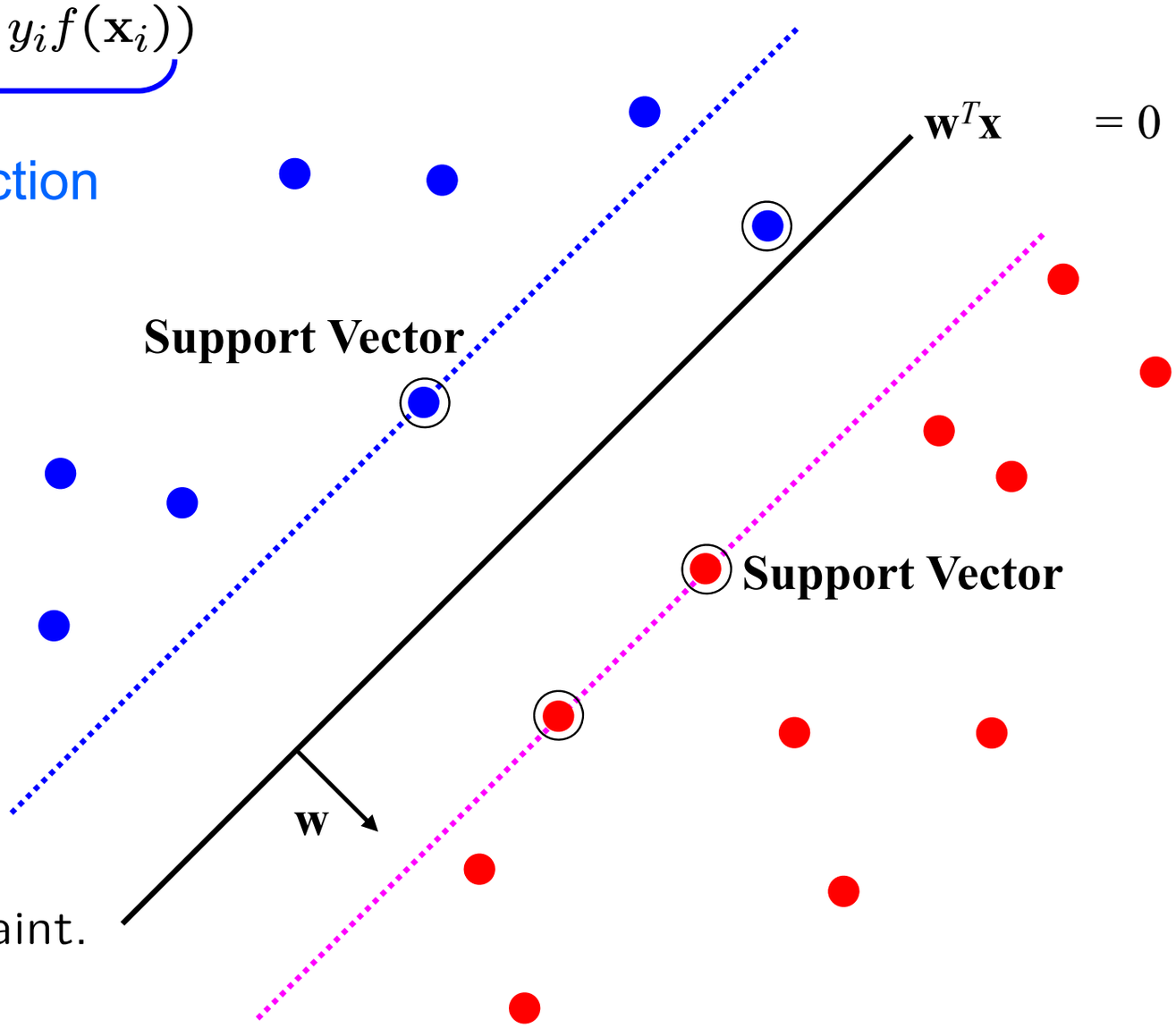
$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max(0, 1-y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Loss function

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_i^N \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

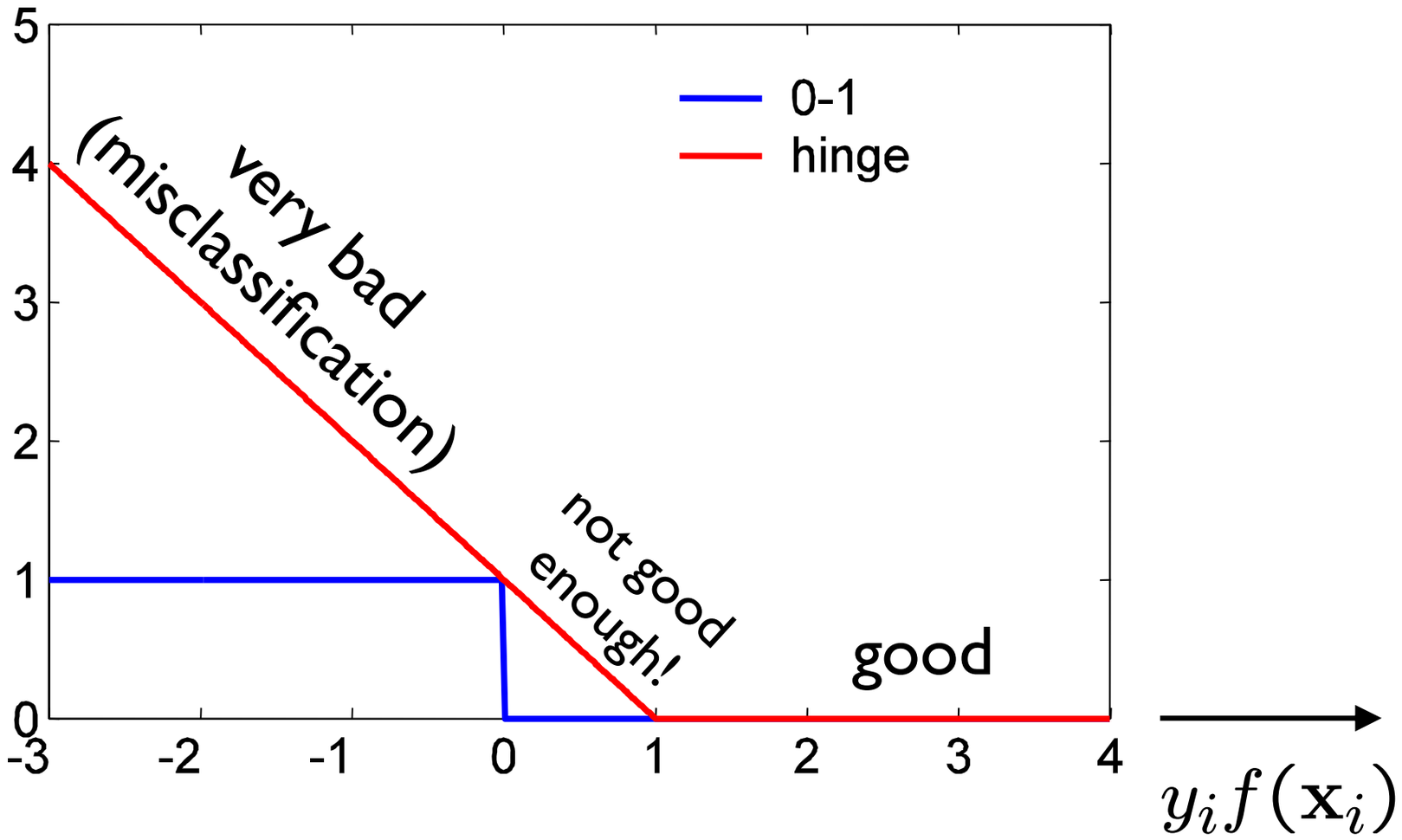
Points are in three categories:

1. $y_i f(x_i) > 1$
   Point is outside margin.
   No contribution to loss
2. $y_i f(x_i) = 1$
   Point is on margin.
   No contribution to loss.
   As in hard margin case.
3. $y_i f(x_i) < 1$
   Point violates margin constraint.
   Contributes to loss



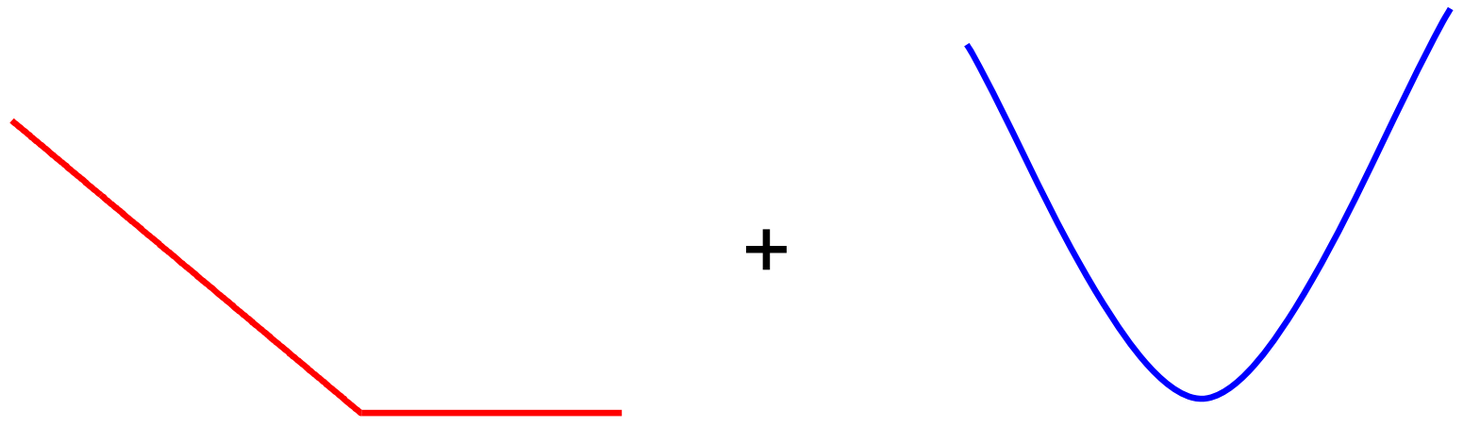
(margin violation ξ>0,
including misclassification ξ>1)

Loss functions



- SVM uses "hinge" loss $\max(0, 1 - y_i f(\mathbf{x}_i))$
- an approximation to the 0-1 loss

(perceptron uses a shifted hinge-loss touching the origin)



SVM

$$\min_{\mathbf{w}\in\mathbb{R}^d} C\sum_i^N \max\left(0, 1 - y_i f(\mathbf{x}_i)\right) + ||\mathbf{w}||^2$$ convex

convex + convex = convex!

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an average

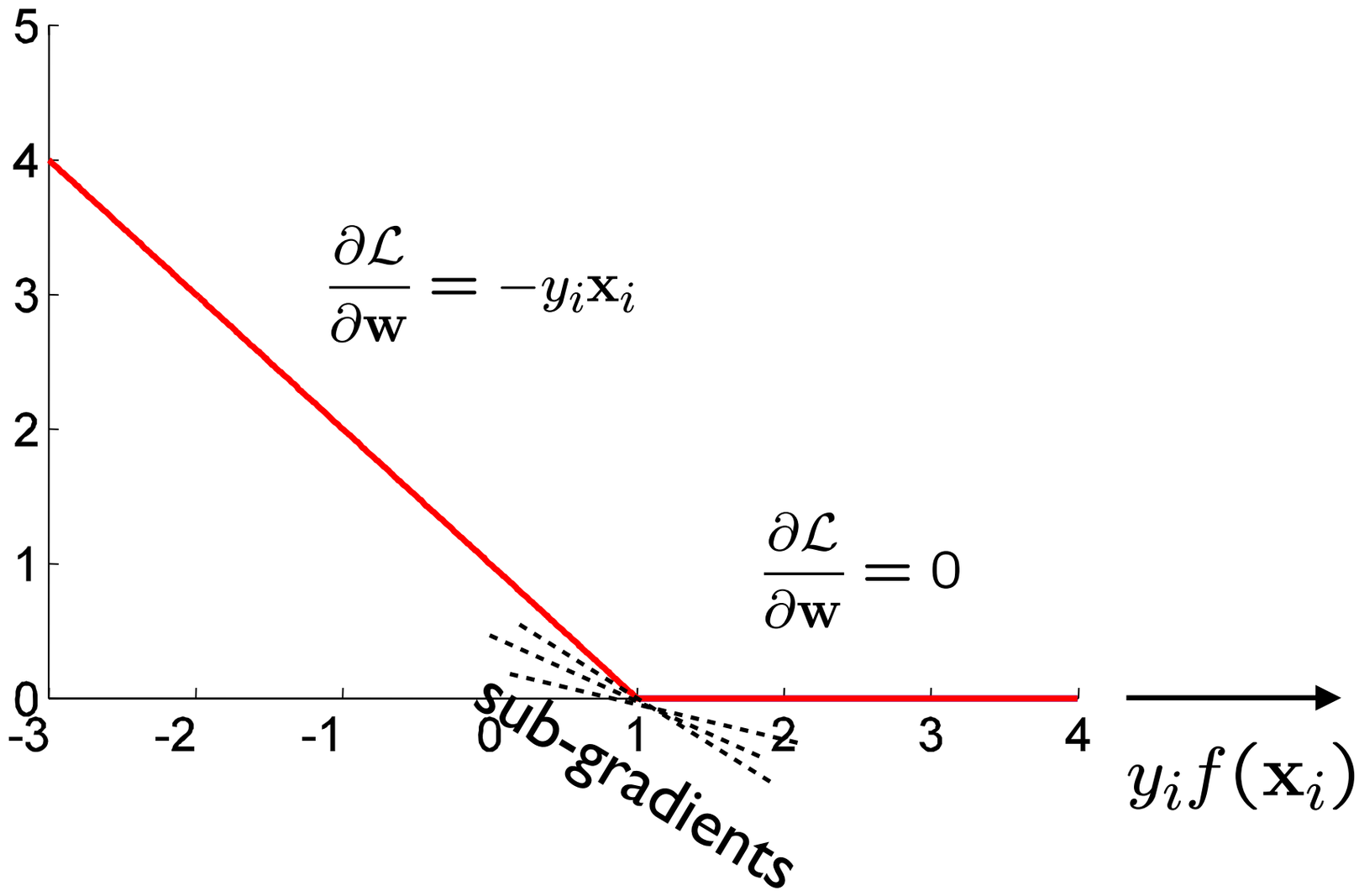
$$\begin{aligned} \min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{N}\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N}\sum_i^N \left(\frac{\lambda}{2}||\mathbf{w}||^2 + \max(0, 1 - y_i f(\mathbf{x}_i))\right) \end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$

Because the hinge loss is not differentiable, a sub-gradient is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \qquad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i$$



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N}\sum_{i}^{N}\left(\frac{\lambda}{2}||\mathbf{w}||^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w})\right)$$

The iterative update is

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta\nabla_{\mathbf{w}_t}\mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta\frac{1}{N}\sum_{i}^{N}(\lambda\mathbf{w}_t + \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t))\end{aligned}$$ **batch gradient**

where η is the learning rate.

Then each iteration t involves cycling through the training data with the updates:

just like perceptron! **perc: ≤0**

$$\begin{aligned}\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda\mathbf{w}_t - y_i\mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta\lambda\mathbf{w}_t && \text{otherwise}\end{aligned}$$ **online gradient**

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$ $\lambda = 2/(NC)$

INPUT: S, λ, T

INITIALIZE: Set $\mathbf{w}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.

Set $\eta_t = \frac{1}{\lambda t}$

If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$

Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$

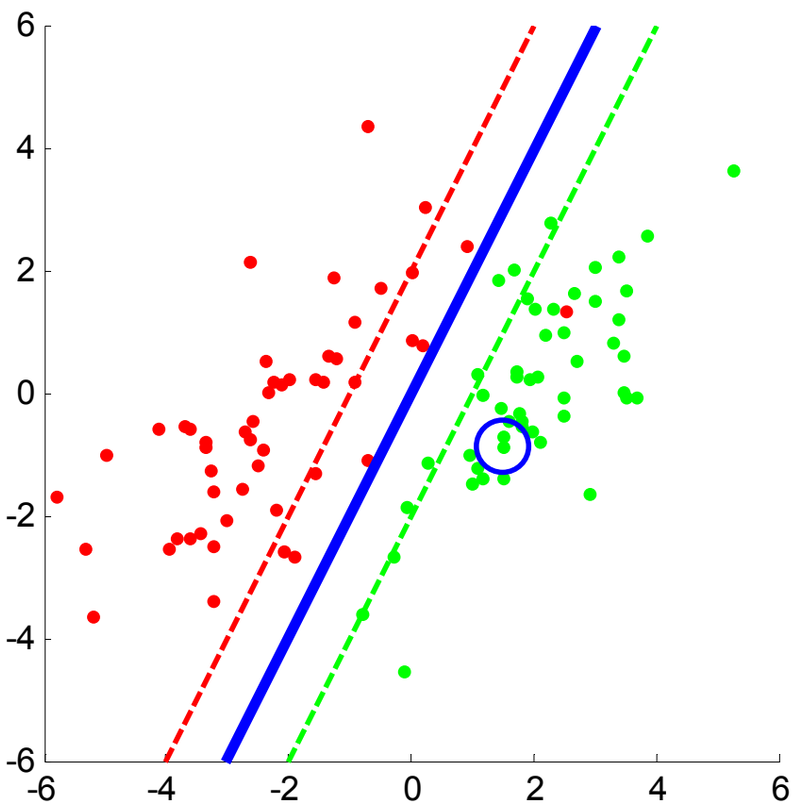
[Optional: $\mathbf{w}_{t+1} \leftarrow \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}_{t+1}\|}\right\} \mathbf{w}_{t+1}$]

OUTPUT: $\mathbf{w}_{T+1}$

$\lambda = 2/(NC)$

Pegasos – Stochastic Gradient Descent Algorithm (SGD)

Randomly sample from the training data



SGD is online update: gradient on one example (unbiasedly) approximates the gradient on the whole training data

slides 49-56 from Andrew Zisserman (Oxford/DeepMind); with annotations