Kernel Methods in Computer Vision

Christoph Lampert
Max Planck Institute for
Biological Cybernetics, Tübingen

Matthew Blaschko
MPI Tübingen and
University of Oxford

June 20, 2009

Overview...

14:00 – 15:00 Introduction to Kernel Classifiers

15:20 – 15:50 Selecting and Combining Kernels

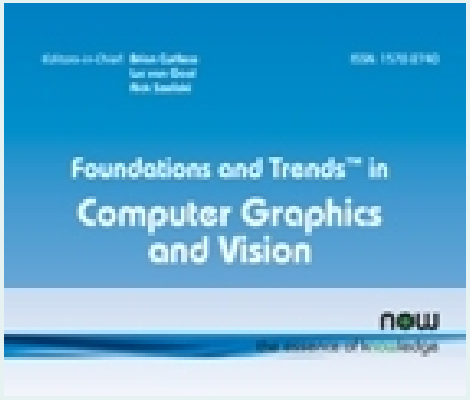
15:50 – 16:20 Other Kernel Methods

16:40 – 17:40 Learning with Structured Outputs

Slides and Additional Material (soon)

`http://www.christoph-lampert.de`

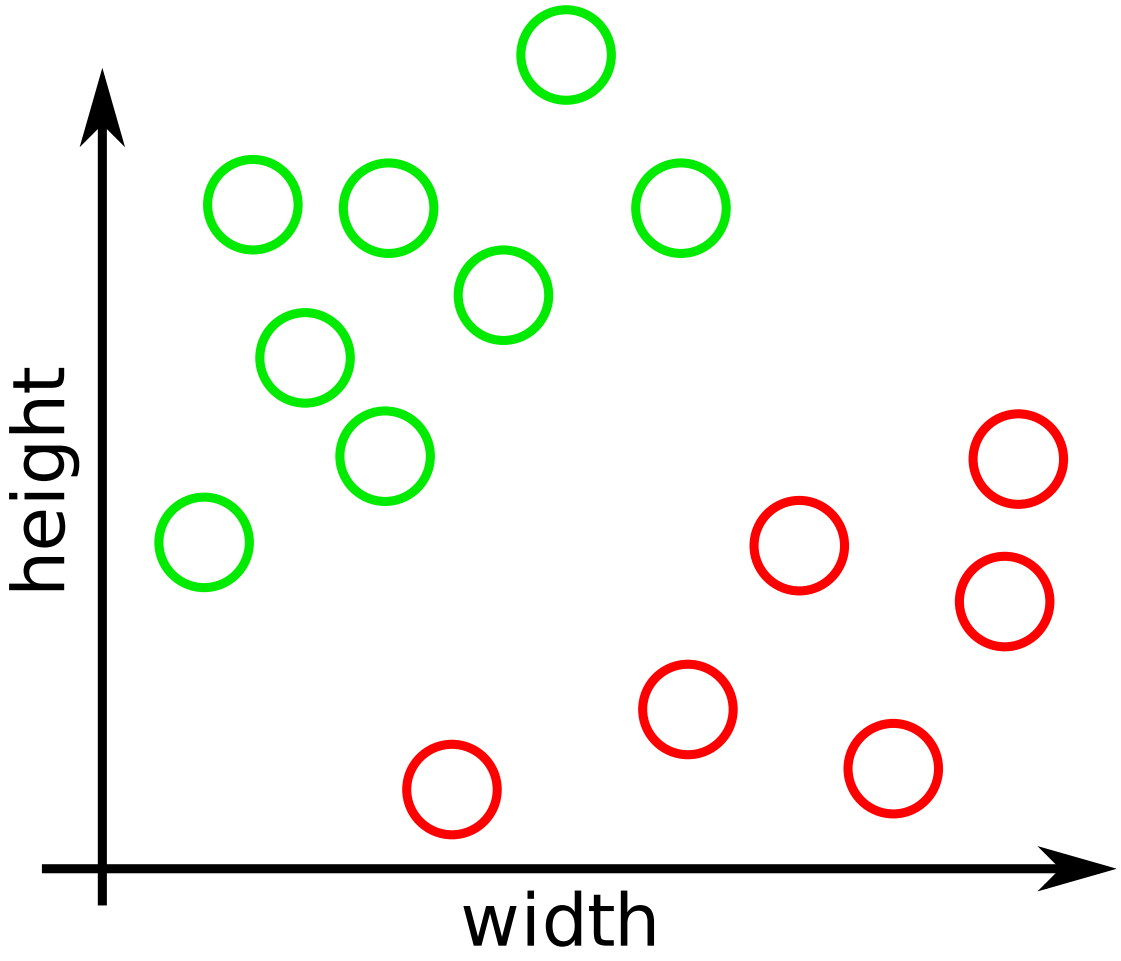
also watch out for



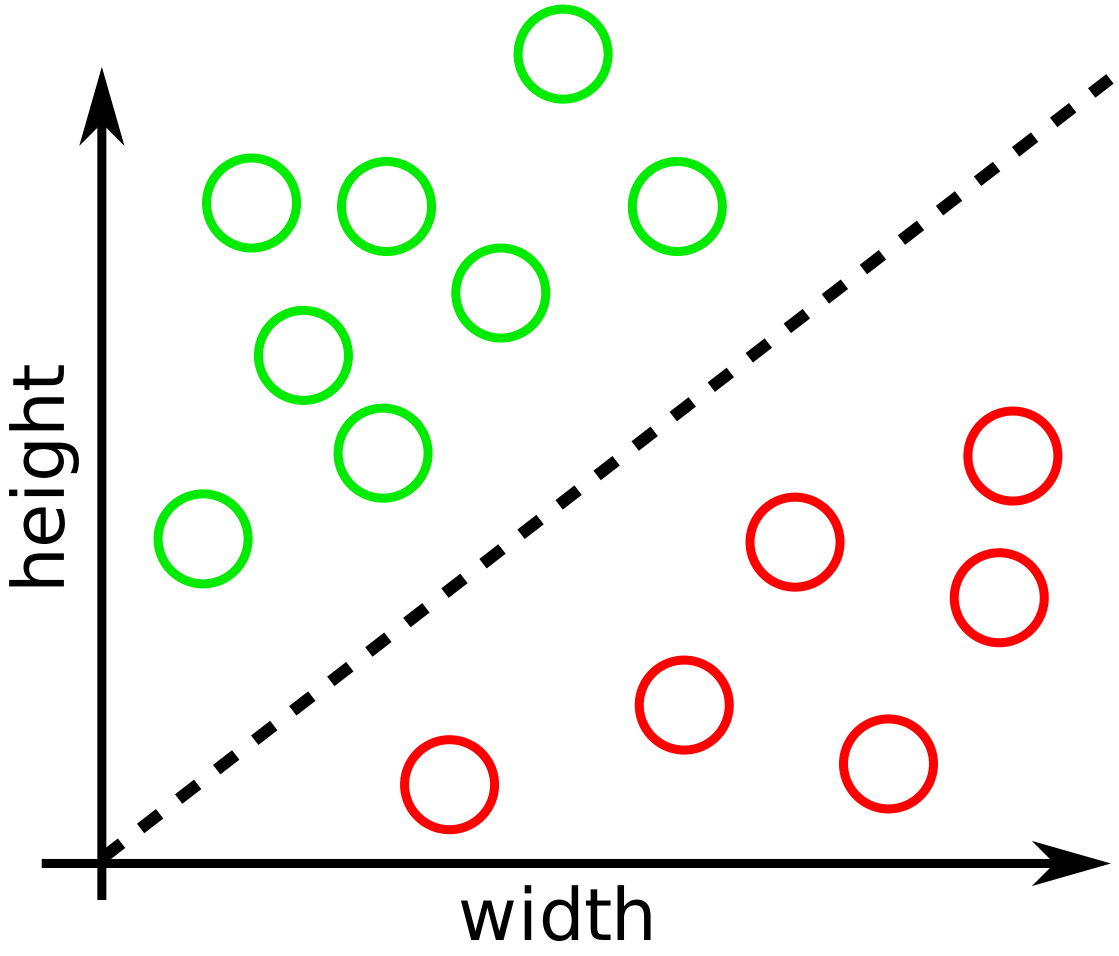
Introduction to Kernel Classifiers

Linear Classification

Separates these two sample sets.



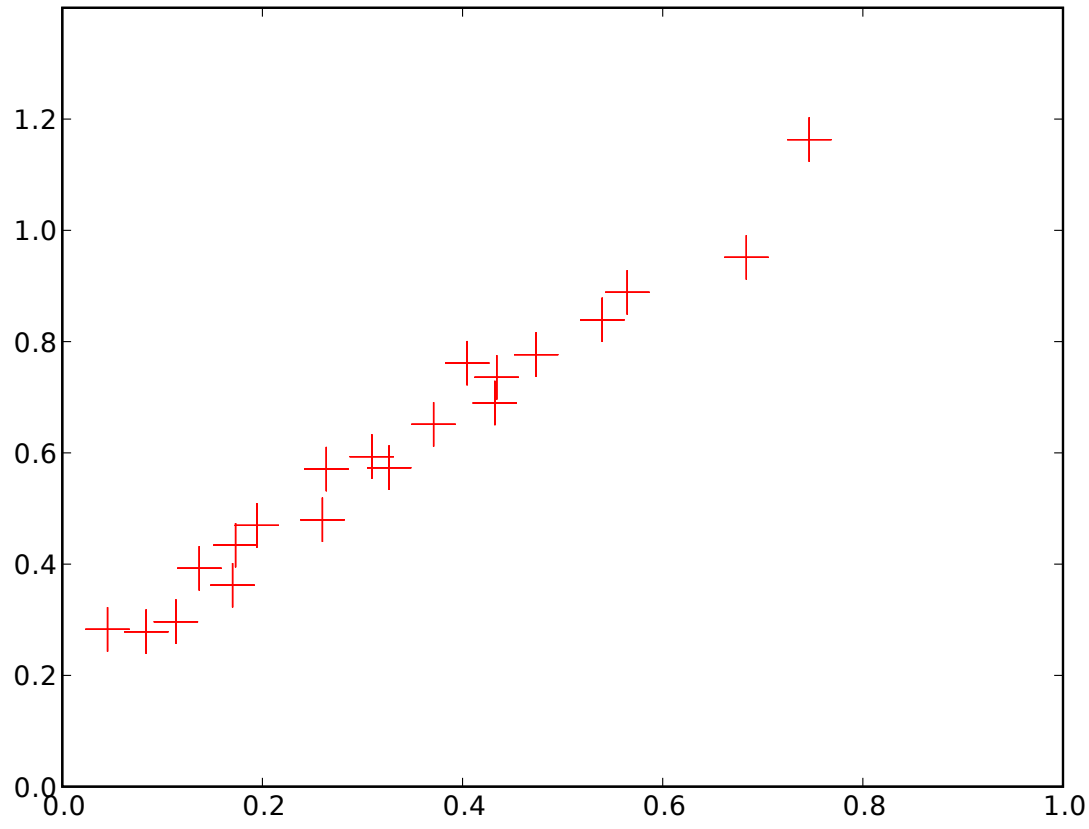
Separates these two sample sets.



Linear Classification

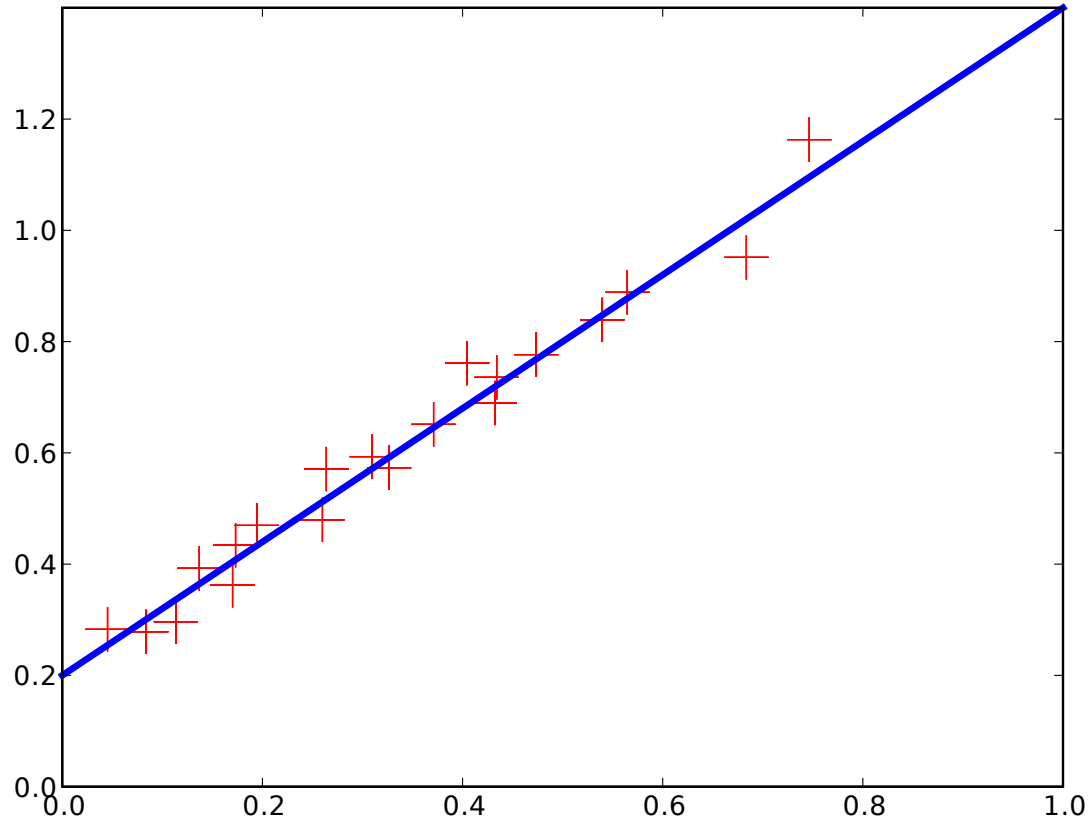
Linear Regression

Find a function that interpolates data points.



Linear Regression

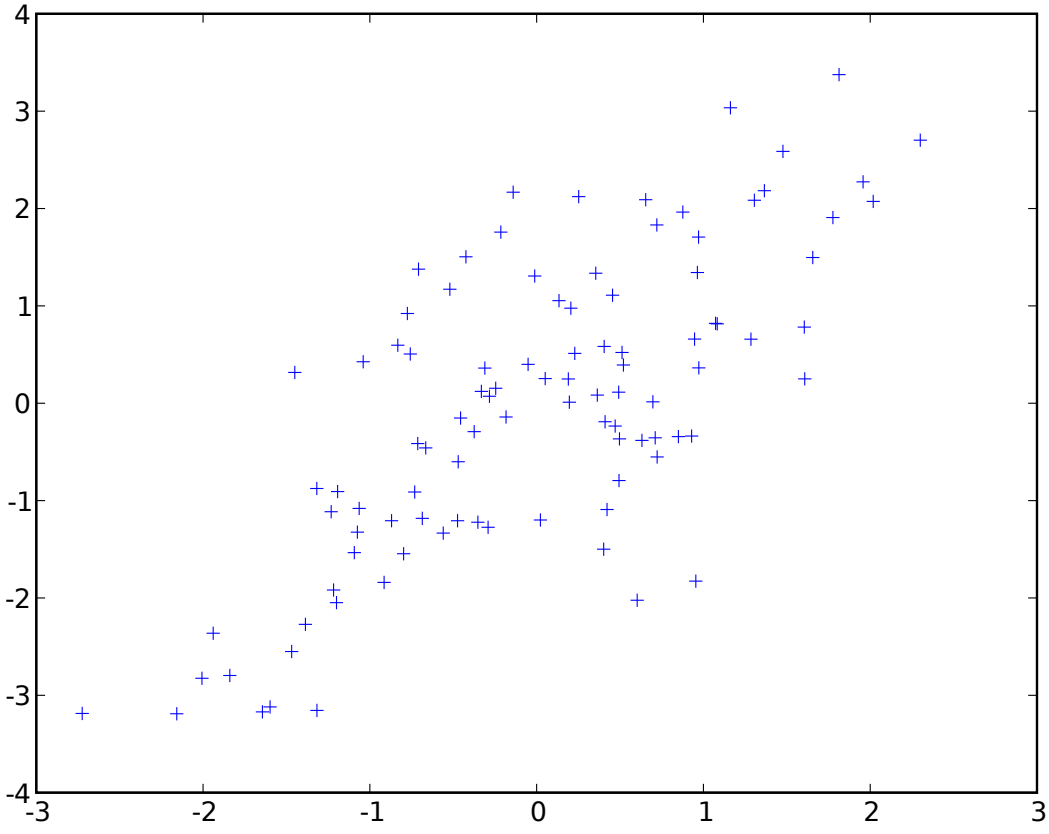
Find a function that interpolates data points.



Least Squares Regression

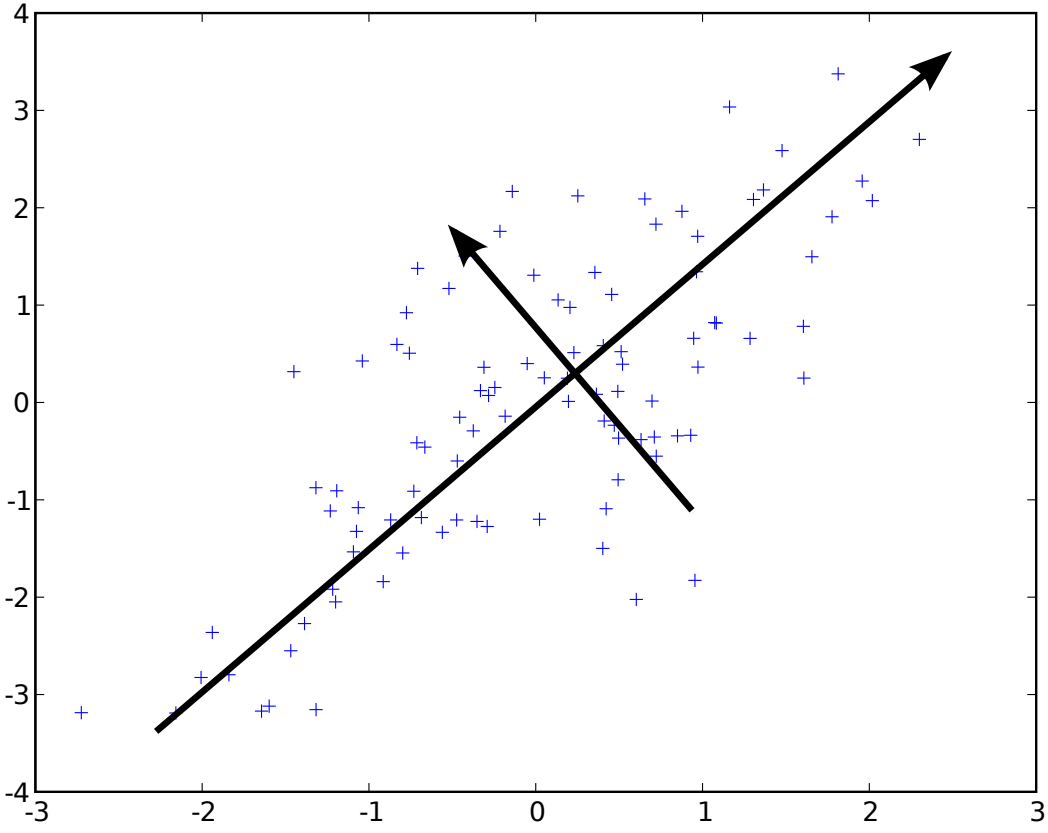
Linear Dimensionality Reduction

Reduce the dimensionality of a dataset while preserving its structure.



Linear Dimensionality Reduction

Reduce the dimensionality of a dataset while preserving its structure.



Principal Component Analysis

Linear Techniques

Three different elementary tasks:

- classification,
- regression,
- dimensionality reduction.

In each case, linear techniques are very successful.

Linear Techniques

Linear techniques...

- often work well,
  - most natural functions are smooth,
  - smooth function can be approximated, at least locally, by linear functions.
- are fast and easy to solve
  - elementary maths, even closed form solutions
  - typically involve only matrix operation
- are intuitive
  - solution can be visualized geometrically,
  - solution corresponds to common sense.

Example: Maximum Margin Classification

Notation:

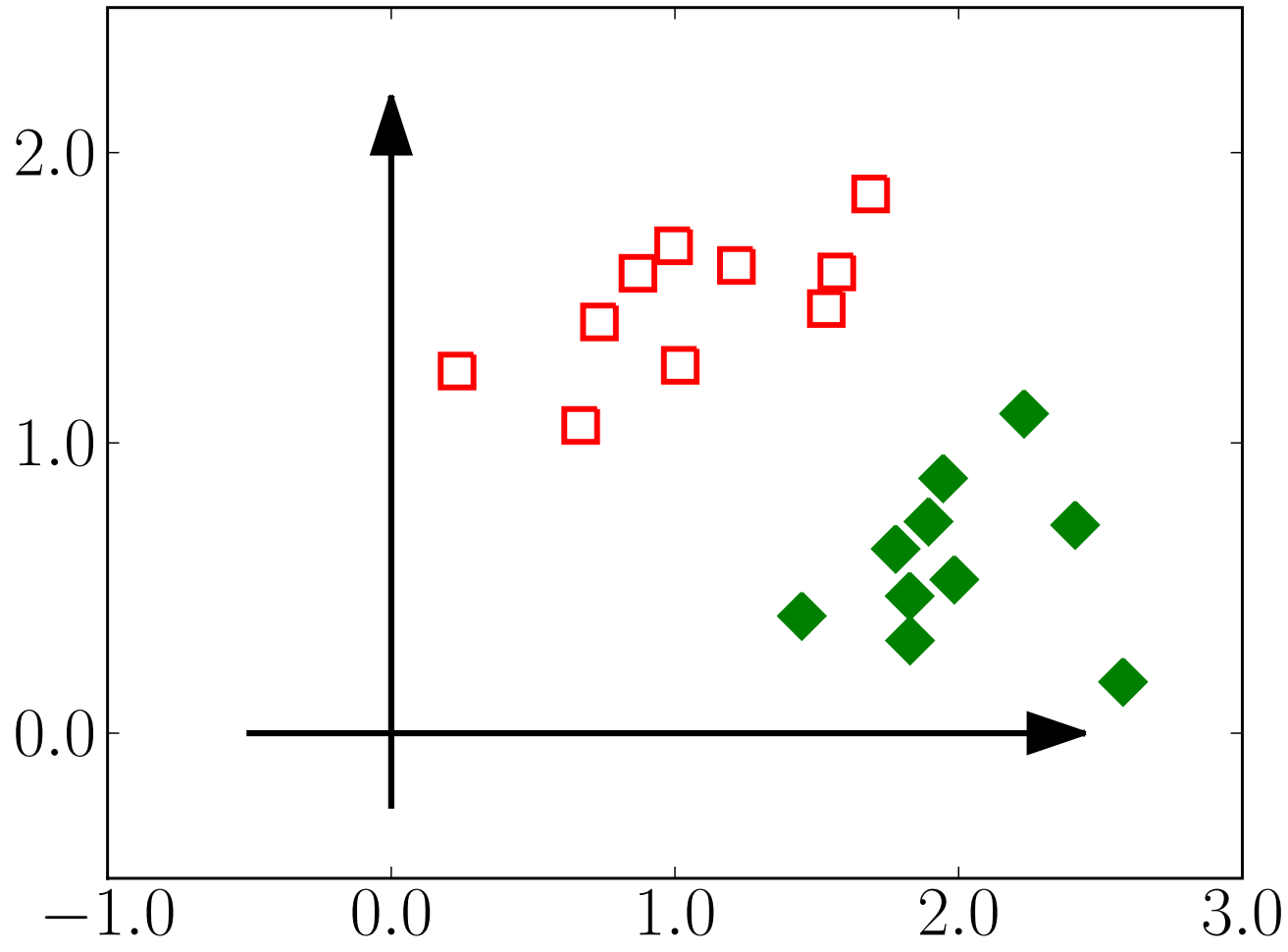
- data points $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$,
- class labels $Y = \{y_1, \ldots, y_n\}$, $y_i \in \{+1, -1\}$.
- linear (decision) function $f : \mathbb{R}^d \to \mathbb{R}$,
- decide classes based on $\operatorname{sign} f : \mathbb{R}^d \to \{-1, 1\}$.
- parameterize

$$\begin{aligned} f(x) &= a^1 x^1 + a^2 x^2 + \ldots a^n x^n + a^0 \\ &\equiv \langle w, x \rangle + b \qquad \text{with} \quad w = (a^1, \ldots, a^n),\ b = a^0. \end{aligned}$$

  $\langle .\,, . \rangle$ is the scalar product is $\mathbb{R}^d$.
- f is uniquely determined by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, but we usually ignore b and only study w
  - b can be absorbed into w. Set $w' = (w, b)$, $x' = (x, 1)$.

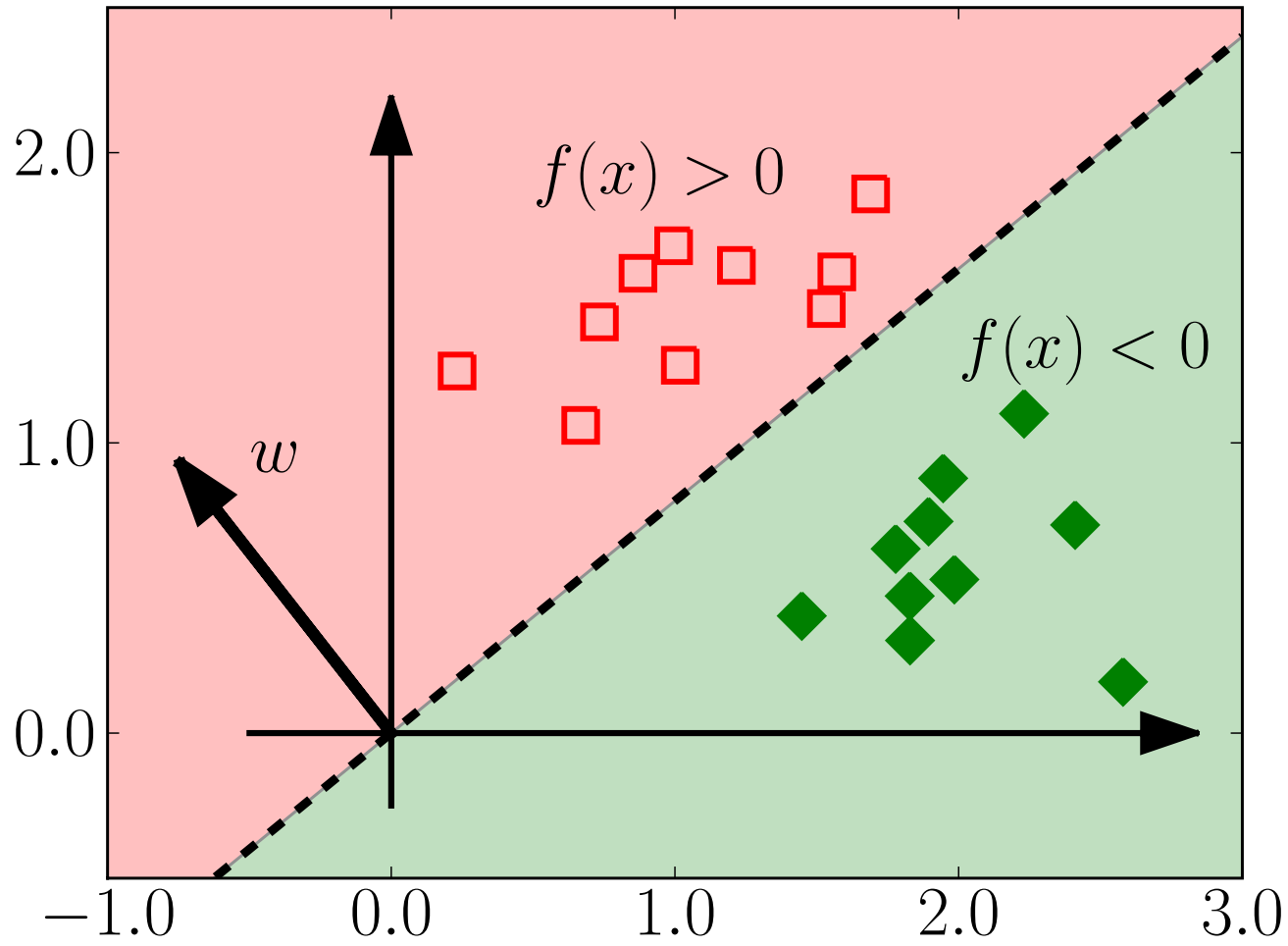
Example: Maximum Margin Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$.



Example: Maximum Margin Classification

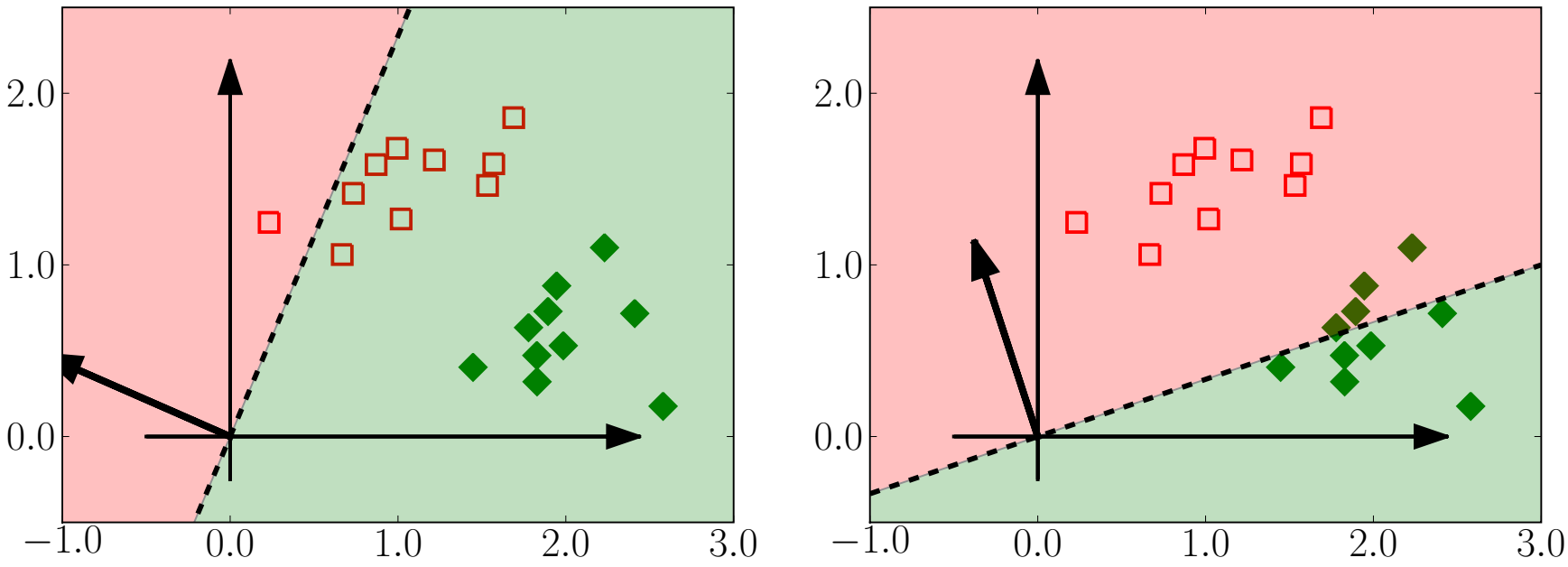
Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Any w partitions the data space into two half-spaces, i.e. defines a classifier.



"What's the best w?"

Example: Maximum Margin Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. What's the best w?

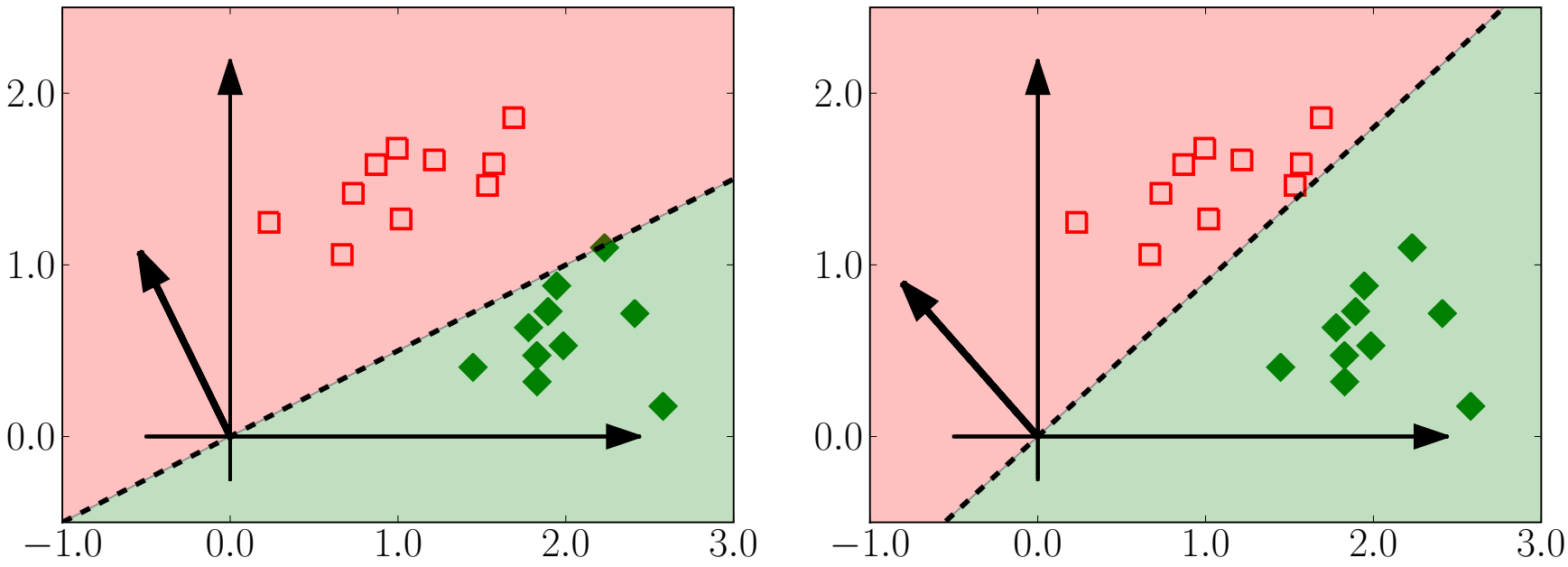


Not these, since they misclassify many examples.

Criterion 1: Enforce $\operatorname{sign}\langle w, x_i \rangle = y_i$ for $i = 1, \ldots, n$.

Example: Maximum Margin Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. What's the best w?

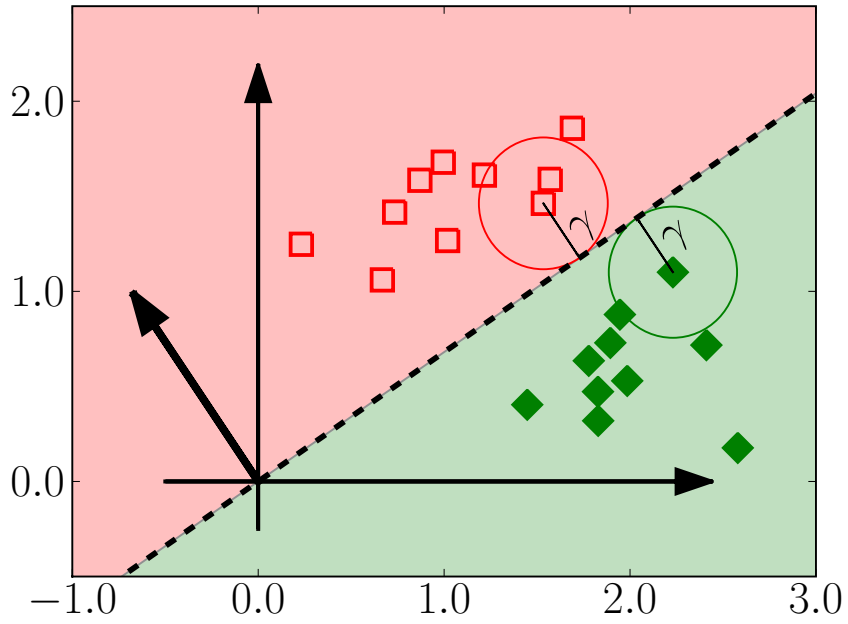


Better not these, since they would be "risky" for future samples.

Criterion 2: Try to ensure $\operatorname{sign}\langle w, x\rangle = y$ for future (x, y) as well.

Example: Maximum Margin Classification

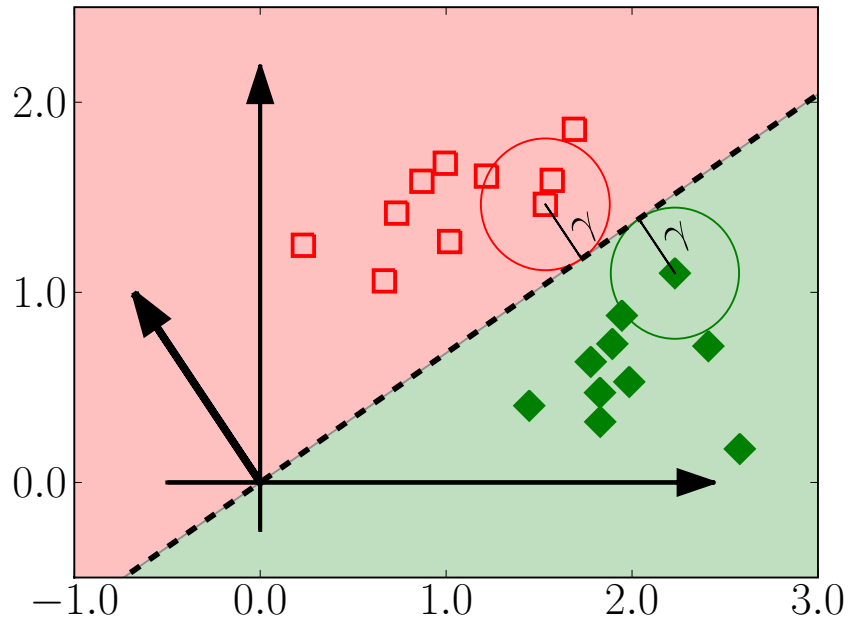
Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Assume that future samples are *similar* to current ones. What's the best w?

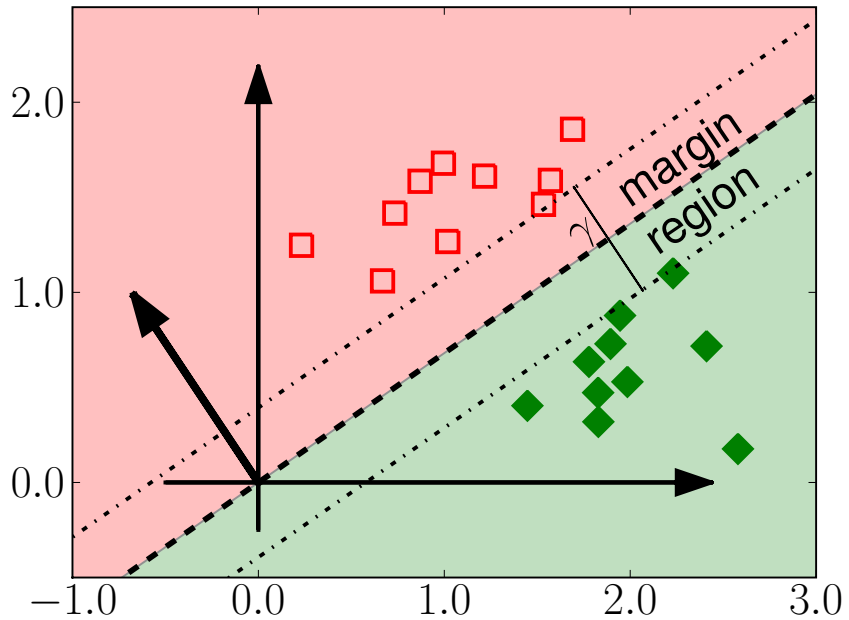


Maximize "stability": use w such that we can maximally perturb the input samples without introducing misclassifications.

Example: Maximum Margin Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Assume that future samples are *similar* to current ones. What's the best w?





Maximize "stability": use w such that we can maximally perturb the input samples without introducing misclassifications.

Central quantity:

$$\textit{margin}(x) = \textit{distance of } x \textit{ to decision hyperplane} = \left\langle \frac{w}{\|w\|}, x \right\rangle$$

Example: Maximum Margin Classification

Maximum-margin solution is determined by a *maximization problem*:

$$\max_{w \in \mathbb{R}^d, \gamma \in \mathbb{R}^+} \gamma$$

subject to

$$\operatorname{sign}\langle w, x_i \rangle = y_i \qquad \text{for } i = 1, \dots n.$$

$$\left| \langle \frac{w}{\|w\|}, x_i \rangle \right| \geq \gamma \qquad \text{for } i = 1, \dots n.$$

Classify new samples using $f(x) = \langle w, x \rangle$.

Example: Maximum Margin Classification

Maximum-margin solution is determined by a *maximization problem*:

$$\max_{\substack{w \in \mathbb{R}^d, \|w\|=1 \\ \gamma \in \mathbb{R}}} \gamma$$

subject to

$$y_i \langle w, x_i \rangle \geq \gamma \qquad \text{for } i = 1, \ldots n.$$

Classify new samples using $f(x) = \langle w, x \rangle$.

Example: Maximum Margin Classification

We can rewrite this as a *minimization problem*:

$$\min_{w \in \mathbb{R}^d} \quad \|w\|^2$$

subject to

$$y_i \langle w, x_i \rangle \geq 1 \qquad \text{for } i = 1, \dots n.$$

Classify new samples using $f(x) = \langle w, x \rangle$.

Example: Maximum Margin Classification

From the view of optimization theory

$$\min_{w \in \mathbb{R}^d} \quad \|w\|^2$$

subject to

$$y_i \langle w, x_i \rangle \geq 1 \qquad \text{for } i = 1, \ldots n$$

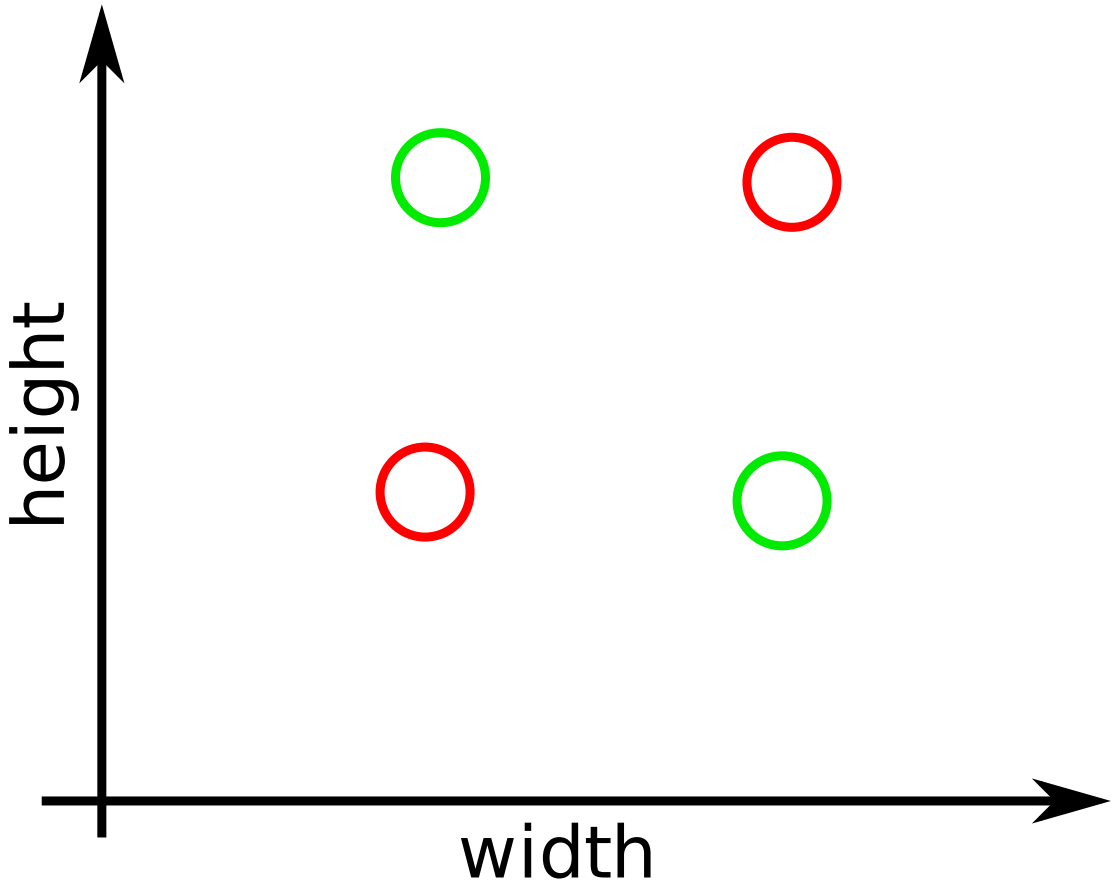
is rather easy:

- The objective function is differentiable and *convex*.
- The constraints are all linear.

We can find the *globally* optimal w in $O(n^3)$ (or faster).

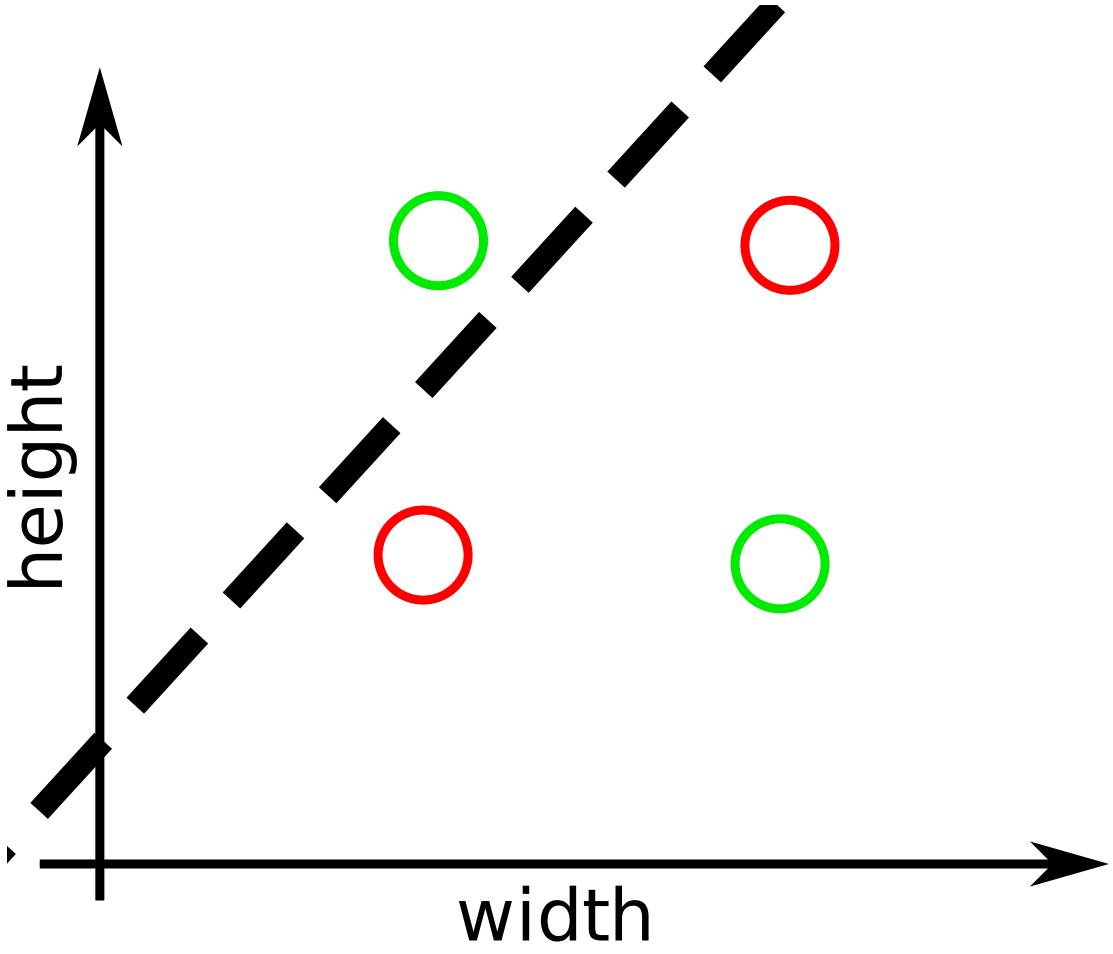
Linear Separability

What is the best w for this dataset?



Linear Separability

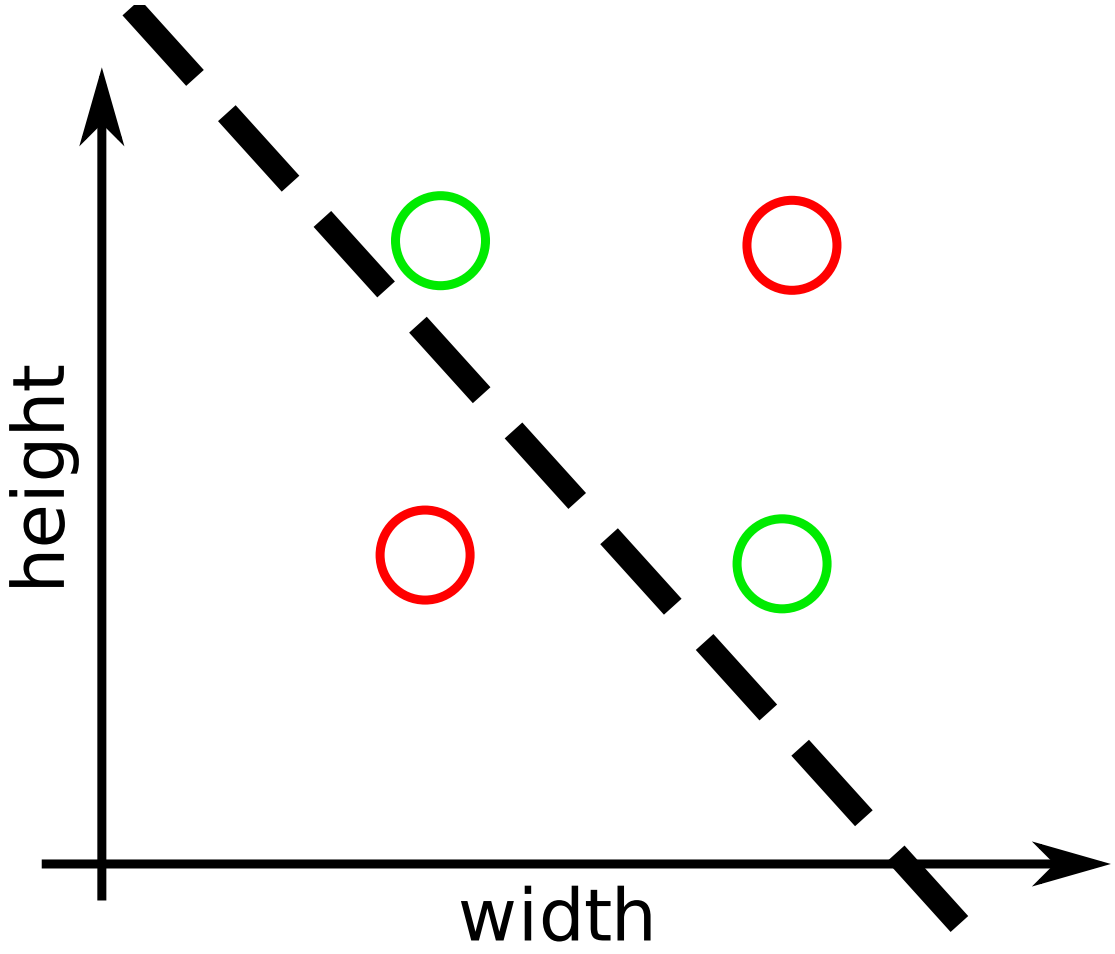
What is the best w for this dataset?



Not this.

Linear Separability

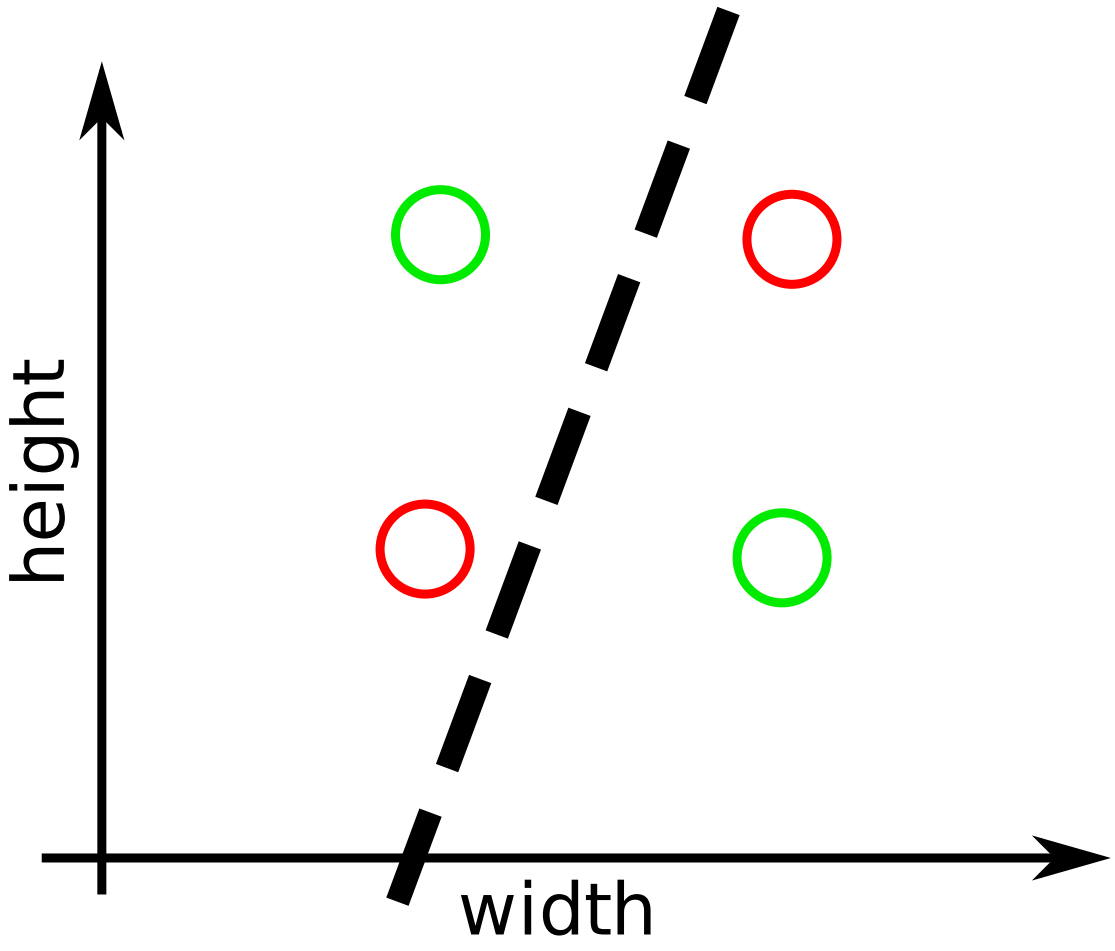
What is the best w for this dataset?



Not this.

Linear Separability

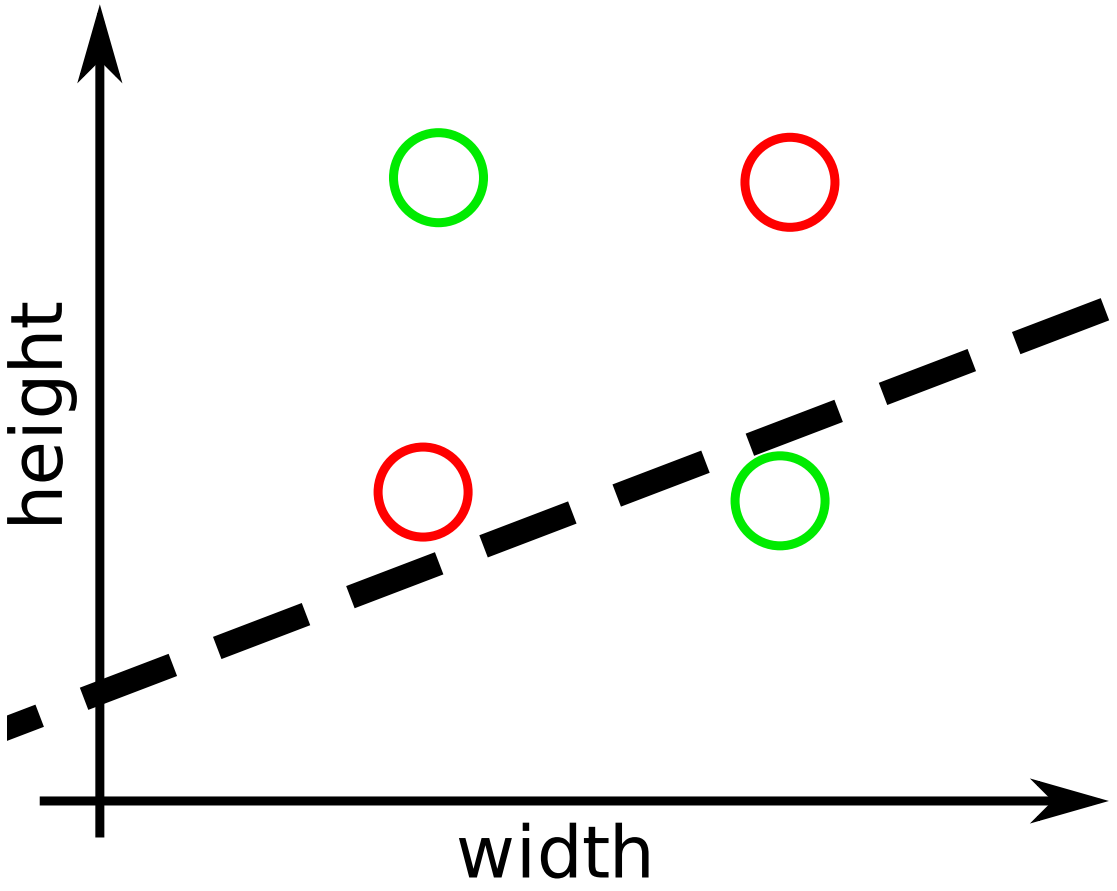
What is the best w for this dataset?



Not this.

Linear Separability

What is the best w for this dataset?



Not this.

Linear Separability

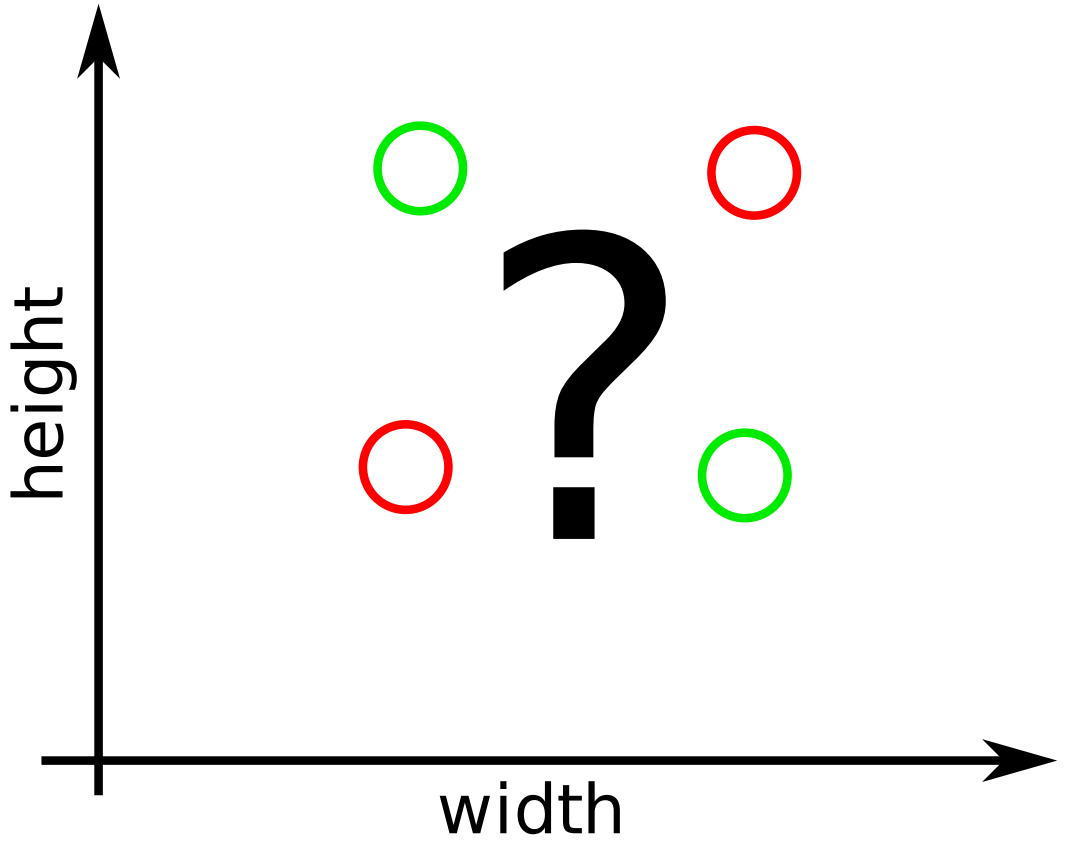
The problem

$$\min_{w \in \mathbb{R}^d} \|w\|^2$$

subject to

$$y_i \langle w, x_i \rangle \geq 1$$

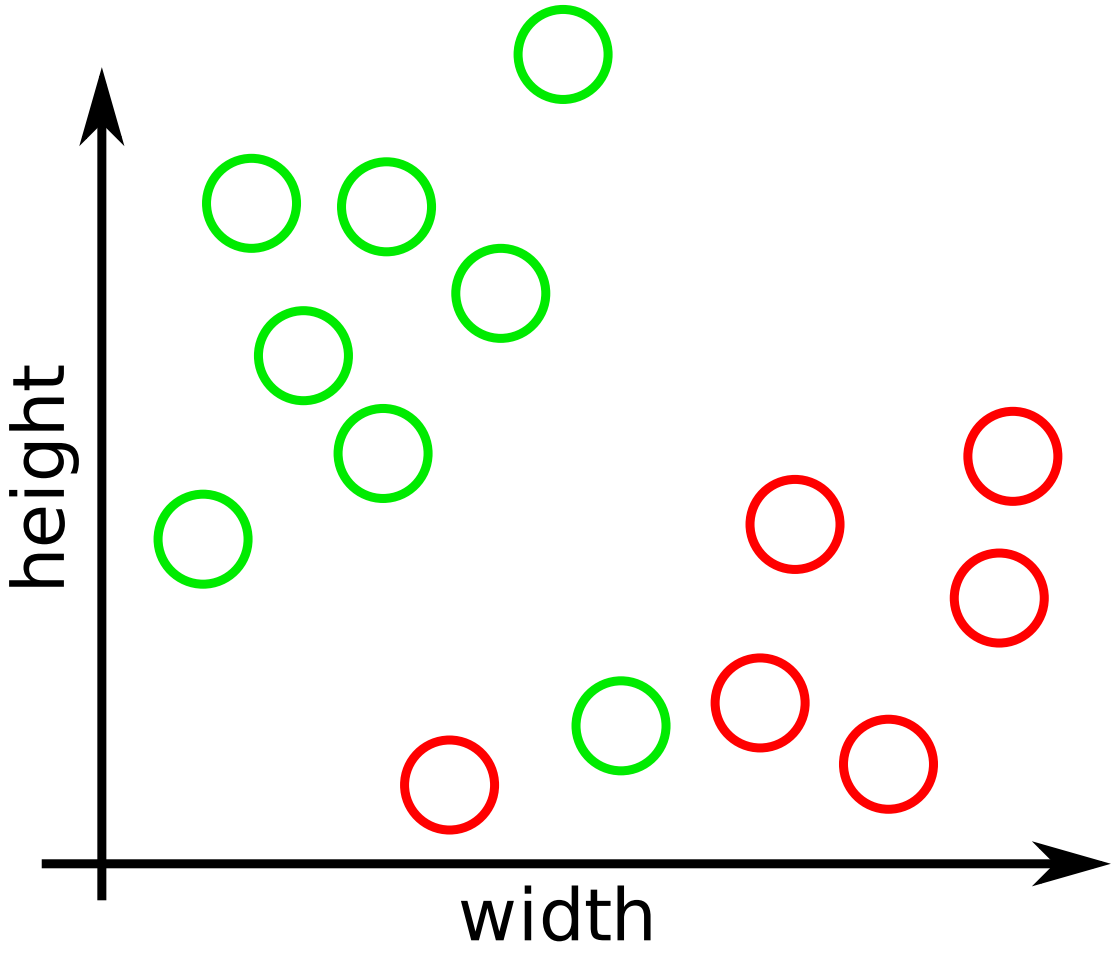
has **no solution**.
The constraints contradict each other!



We cannot find a maximum-margin hyperplane here, because there is none. To fix this, we must allow hyperplanes that *make mistakes*.

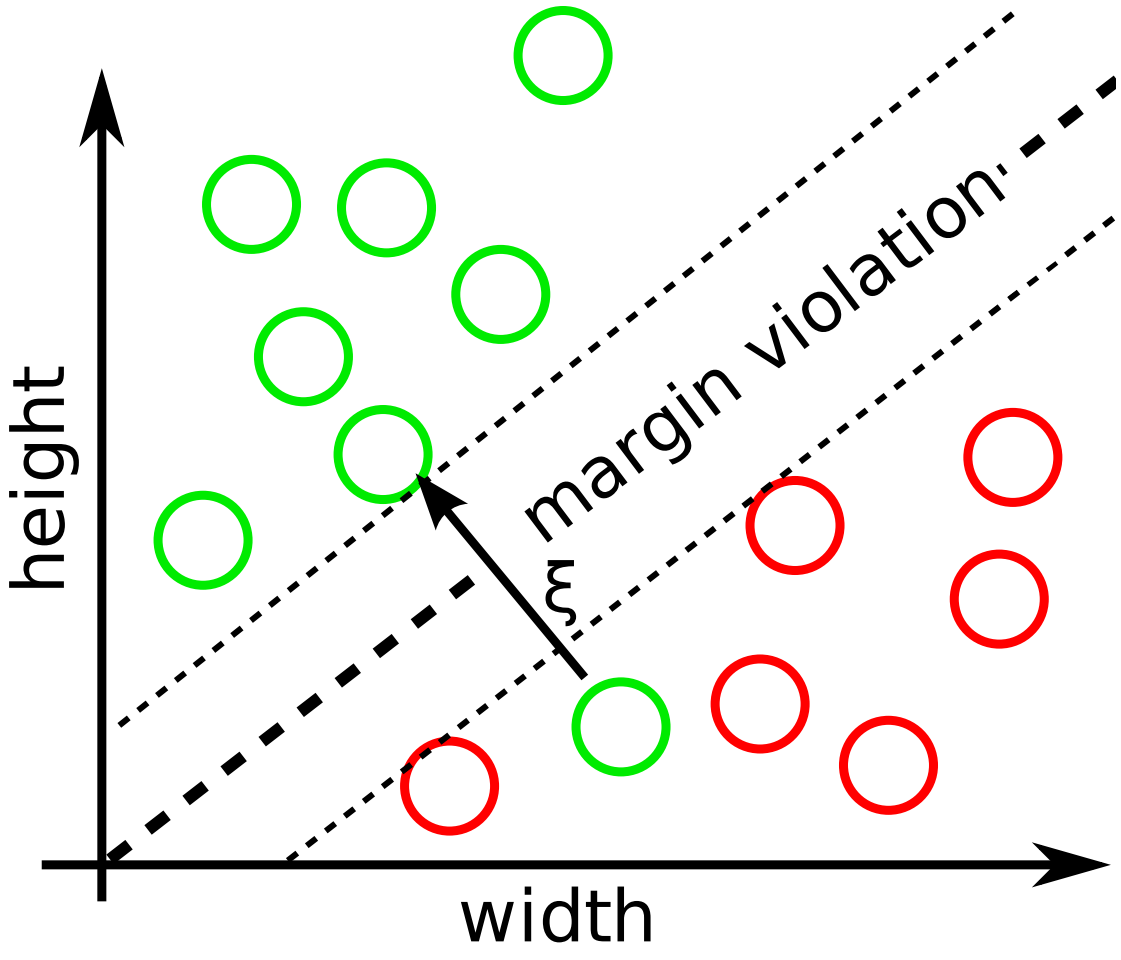
Linear Separability

What is the best w for this dataset?



Linear Separability

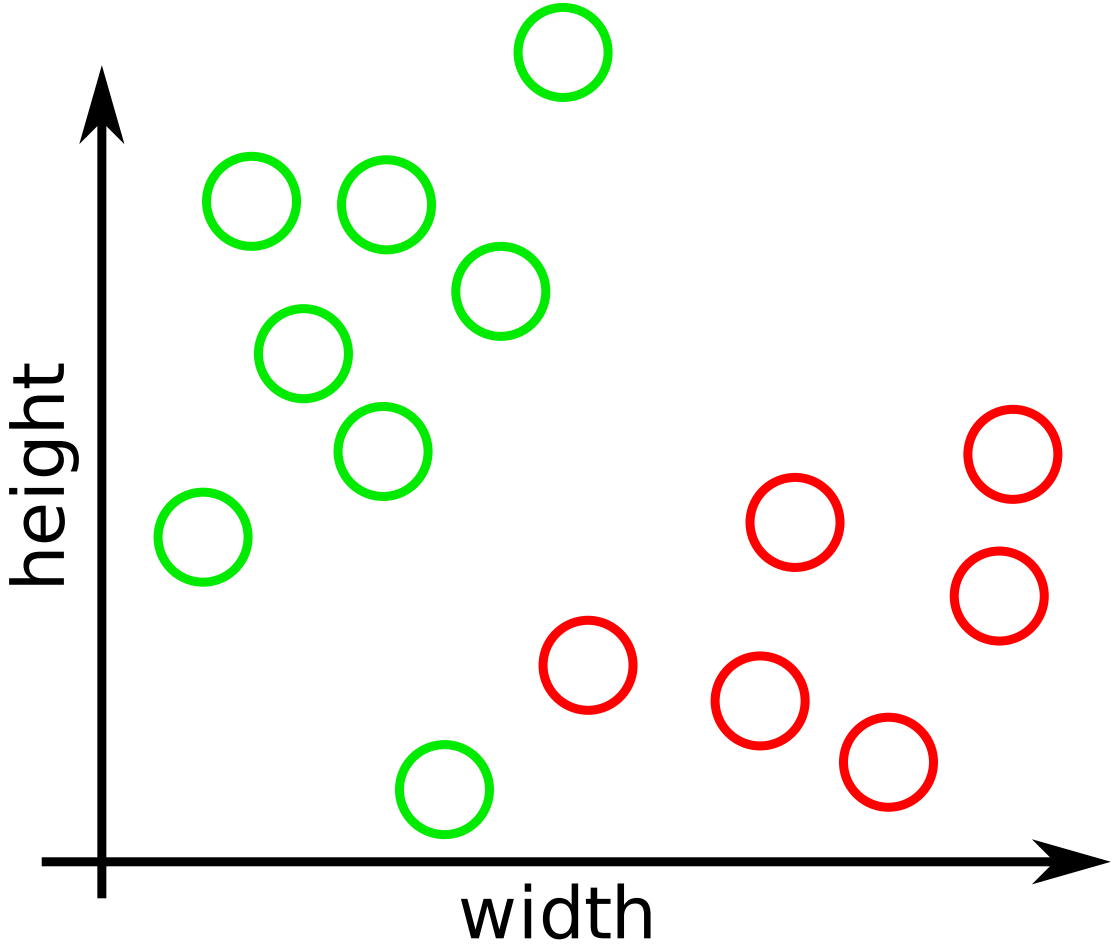
What is the best w for this dataset?



Possibly this one, even though one sample is misclassified.

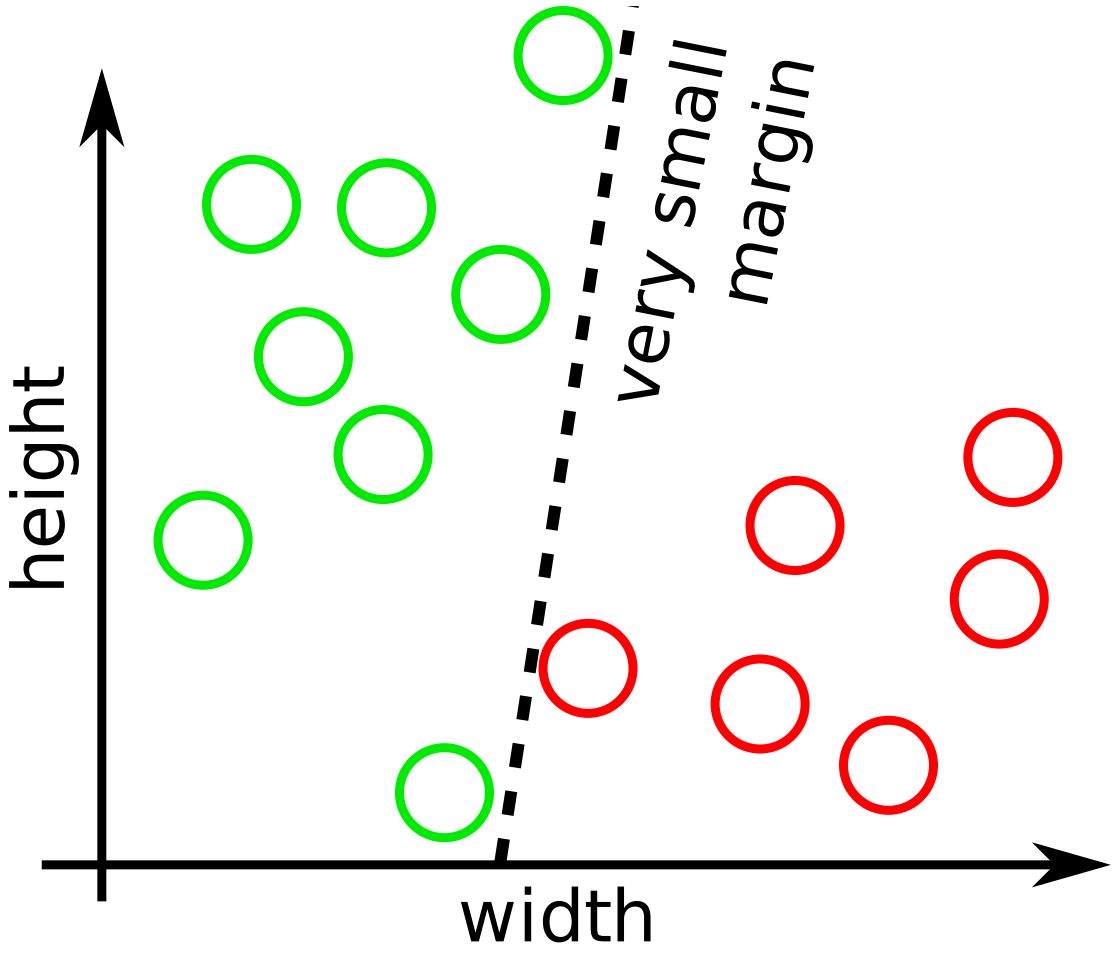
Linear Separability

What is the best w for this dataset?



Linear Separability

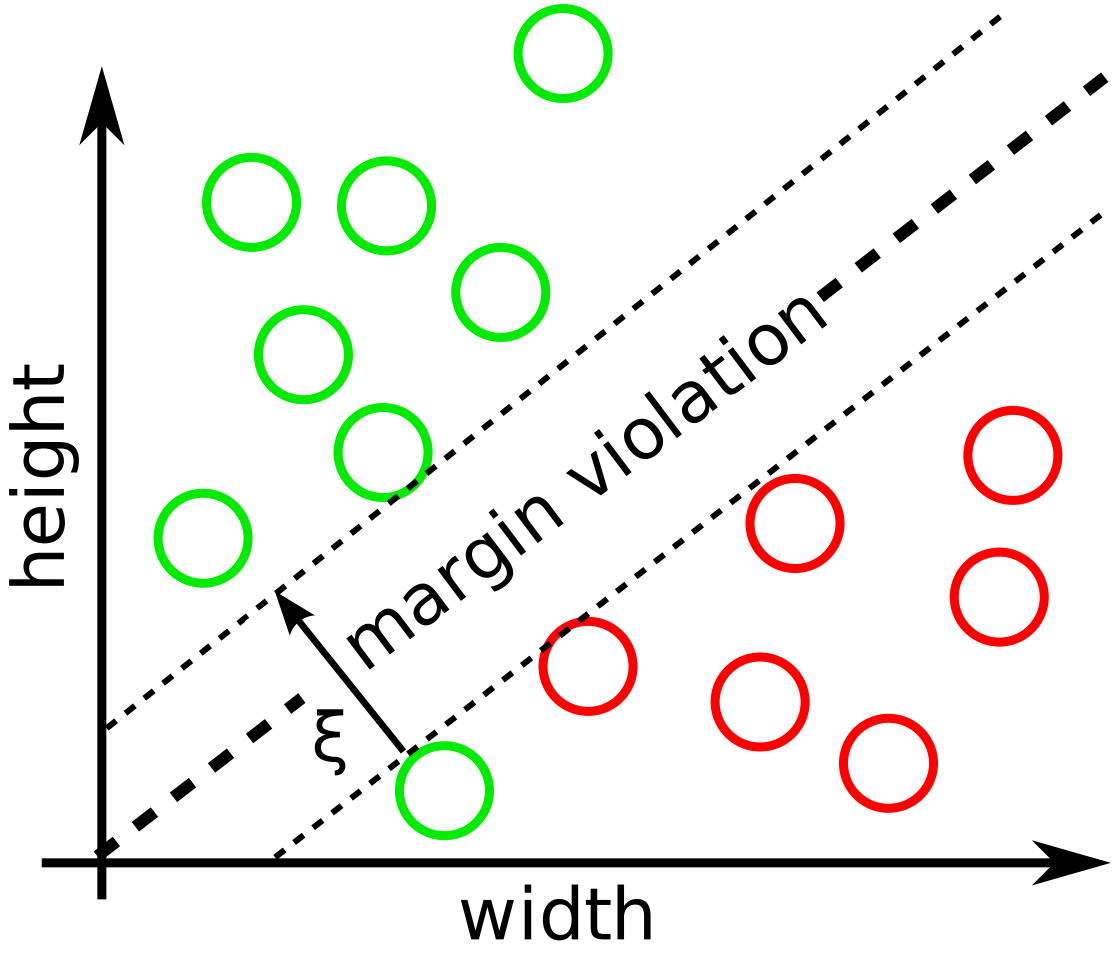
What is the best w for this dataset?



Maybe not this one, even though all points are classified correctly.

Linear Separability

What is the best w for this dataset?



Trade-off: *large margin* vs. *few mistakes* on training set

Solving for Soft-Margin Solution

Mathematically, we formulate the trade-off by *slack*-variables ξ_i:

$$\min_{w \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

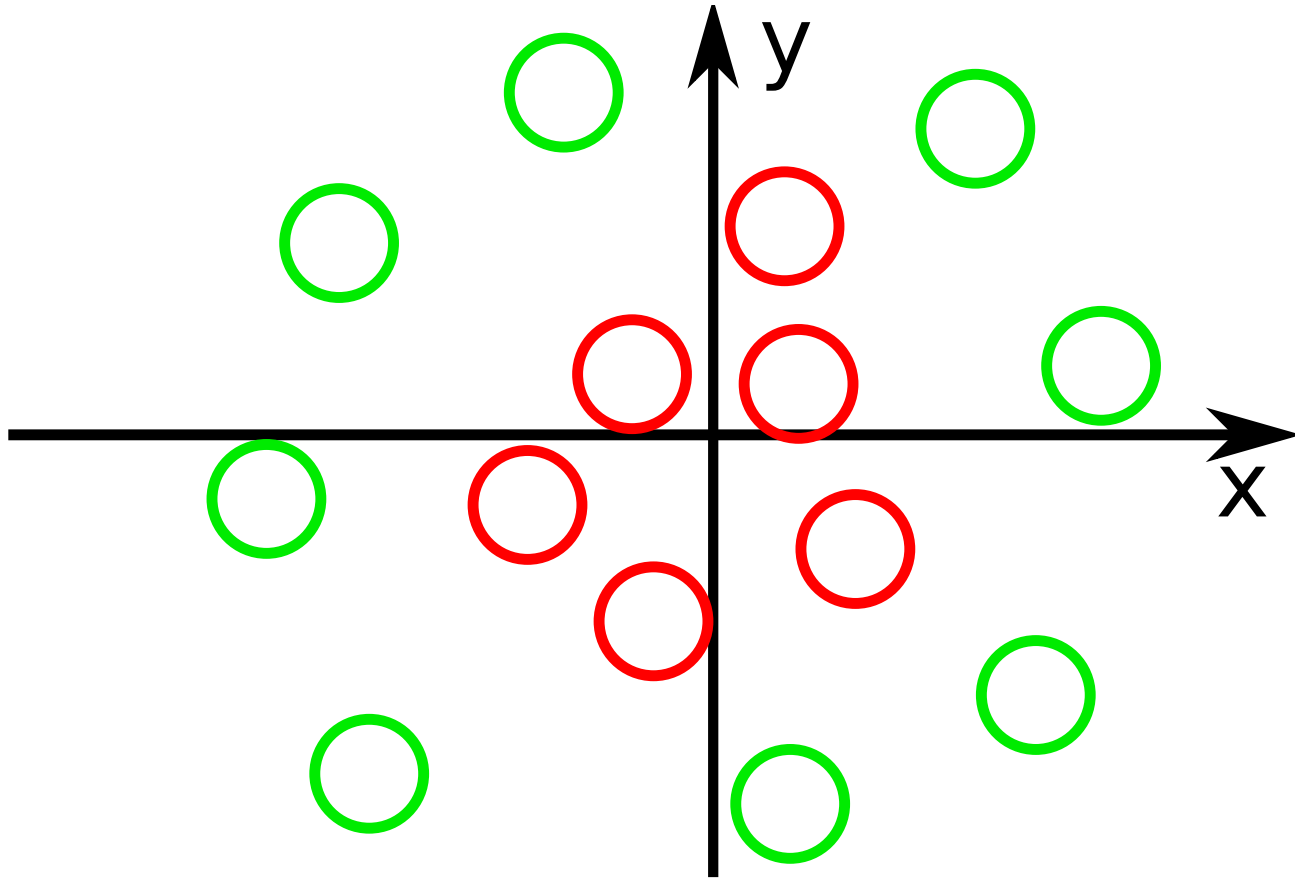
subject to

$$y_i \langle w, x_i \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

- We can fulfill *every* constraint by choosing ξ_i large enough.
- The larger ξ_i, the larger the objective (that we try to minimize).
- C is a *regularization*/trade-off parameter:
  - small $C \rightarrow$ constraints are easily ignored
  - large $C \rightarrow$ constraints are hard to ignore
  - $C = \infty \rightarrow$ hard margin case $\rightarrow$ no training error
- Note: The problem is still convex and efficiently solvable.

Linear Separability

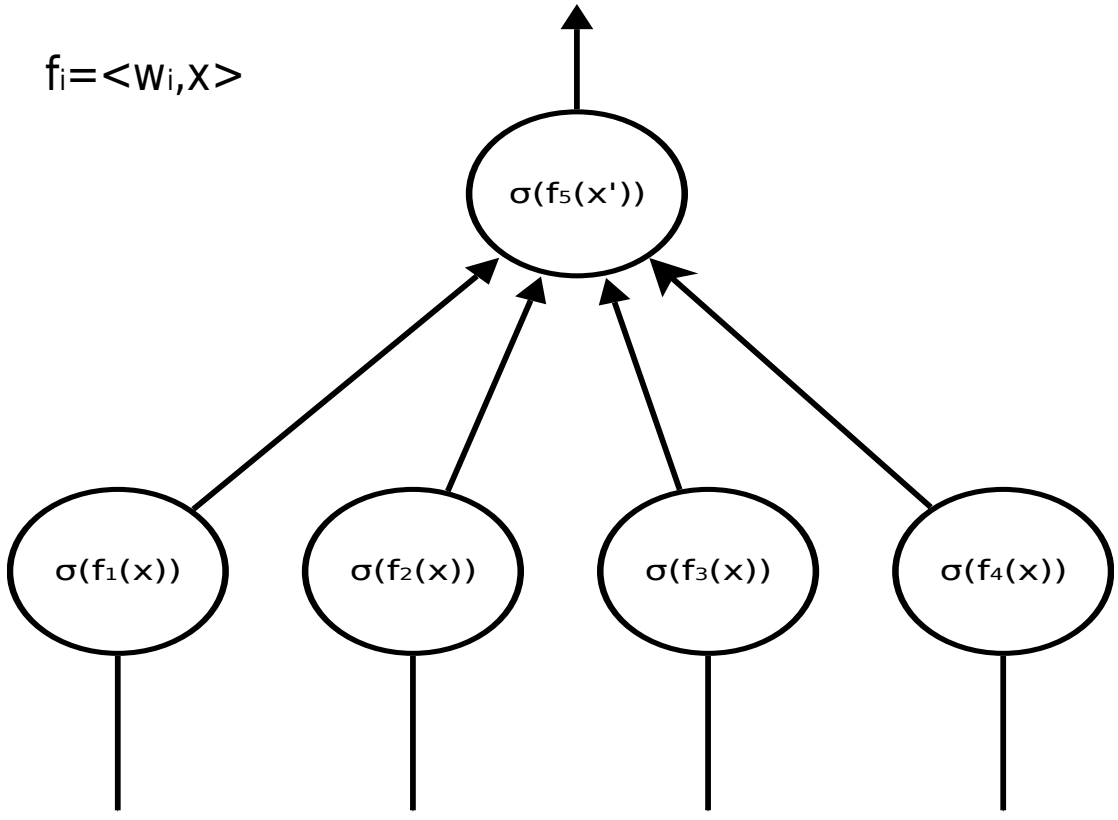
So, what is the best soft-margin w for this dataset?



None. We need something non-linear!

Non-Linear Classification: Stacking

Idea 1) Use classifier output as input to other (linear) classifiers:

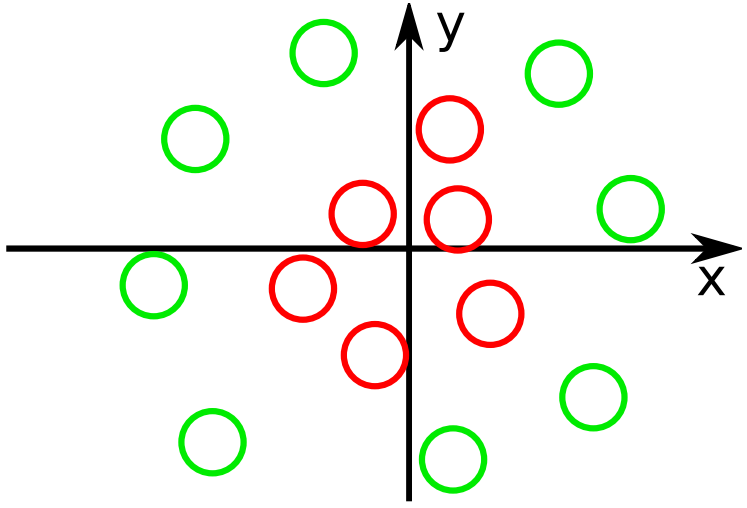


Multilayer Perceptron (Artificial Neural Network) or *Boosting*
$\Rightarrow$ decisions depend non-linearly on x and w_j.

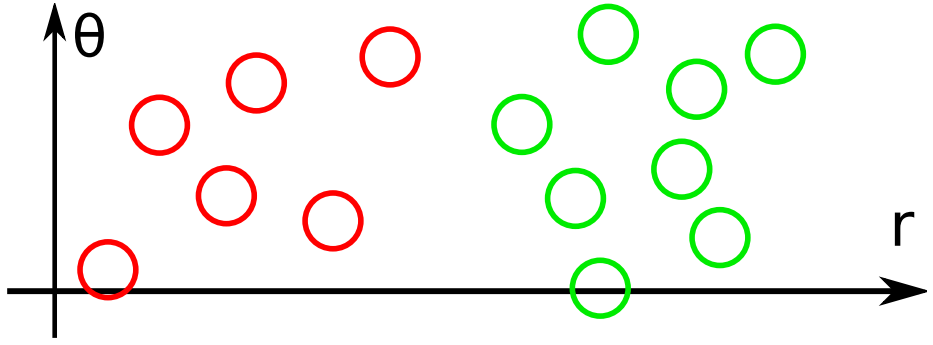
Non-linearity: Data Preprocessing

Idea 2) Preprocess the data:

This dataset is not
(well) *linearly separable*:



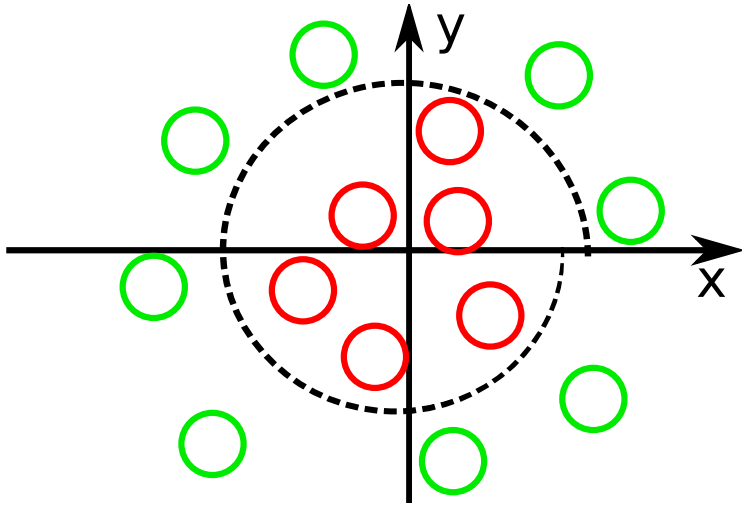
This one is:



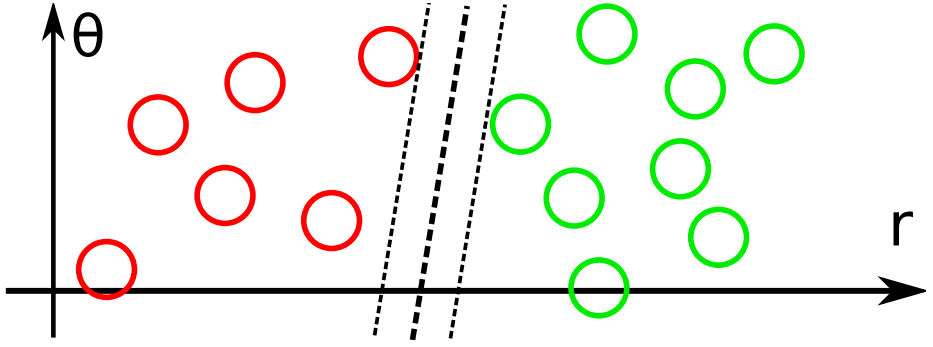
In fact, both are *the same dataset*!
Top: Cartesian coordinates. Bottom: polar coordinates

Non-linearity: Data Preprocessing

Non-linear separation



Linear separation



Linear classifier in polar space; acts non-linearly in Cartesian space.

Generalized Linear Classifier

- Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$.
- Given any (non-linear) feature map $\varphi : \mathbb{R}^k \to \mathbb{R}^m$.
- Solve the minimization for $\varphi(x_1), \ldots, \varphi(x_n)$ instead of $x_1, \ldots, x_n$:

$$\min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

- The weight vector w now comes from the target space $\mathbb{R}^m$.
- Distances/angles are measure by the scalar product $\langle ., . \rangle$ in $\mathbb{R}^m$.
- Classifier $f(x) = \langle w, \varphi(x) \rangle$ is *linear* in w, but *non-linear* in x.

Example Feature Mappings

- Polar coordinates:

$$\varphi : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} \sqrt{x^2 + y^2} \\ \angle(x, y) \end{pmatrix}$$

- d-th degree polynomials:

$$\varphi : \Big(x_1, \ldots, x_n\Big) \mapsto \Big(1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, \ldots, x_1^d, \ldots, x_n^d\Big)$$

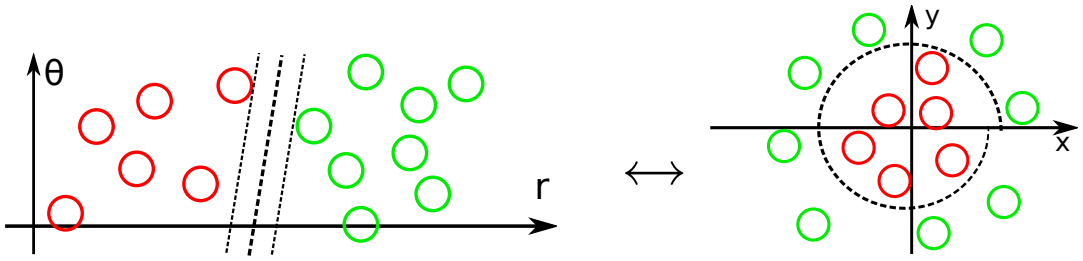
- Distance map:

$$\varphi : \vec{x} \mapsto \Big(\|\vec{x} - \vec{p}_i\|, \ldots, \|\vec{x} - \vec{p}_N\|\Big)$$

for a set of N prototype vectors $\vec{p}_i$, $i = 1, \ldots, N$.

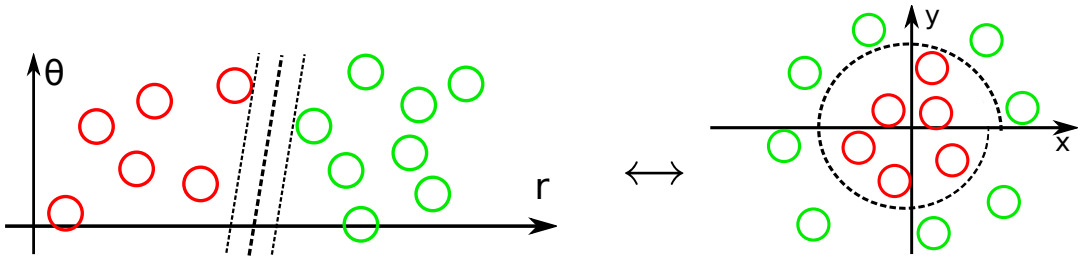
Is this enough?

In this example, changing the coordinates did help.
Does this trick *always* work?



Is this enough?

In this example, changing the coordinates did help.
Does this trick *always* work?



Answer: In a way, yes!

Lemma

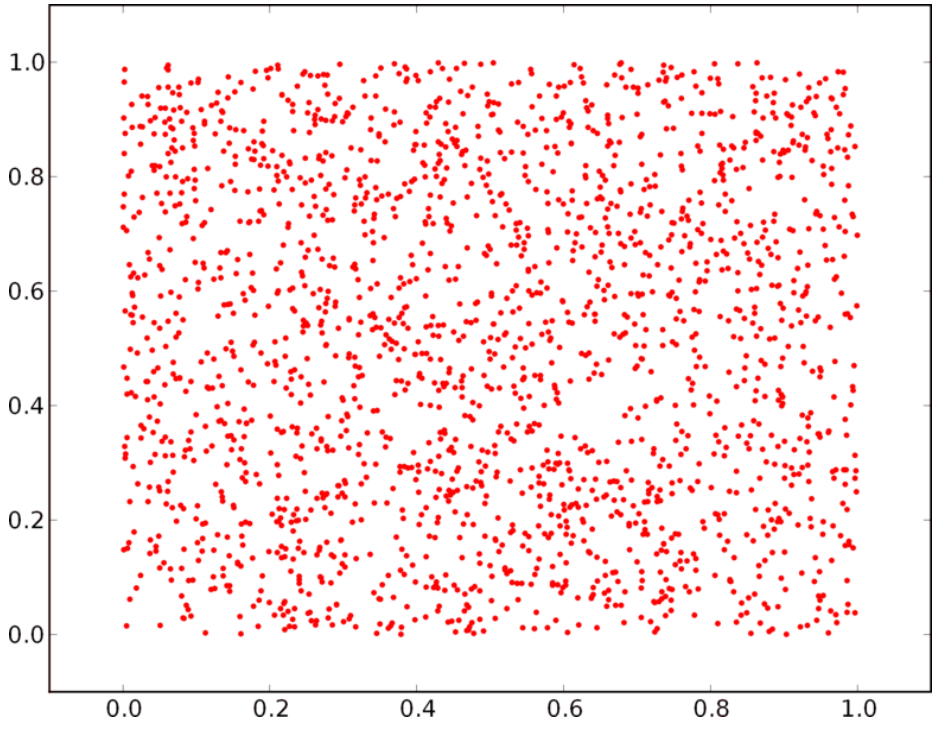
Let $(x_i)_{i=1,\dots,n}$ *with* $x_i \neq x_j$ *for* $i \neq j$. *Let* $\varphi : \mathbb{R}^k \to \mathbb{R}^m$ *be a feature map. If the set* $\varphi(x_i)_{i=1,\dots,n}$ *is linearly independent, then the points* $\varphi(x_i)_{i=1,\dots,n}$ *are linearly separable.*

Lemma

If we choose $m > n$ *large enough, we can always find a map* φ.

Is this enough?

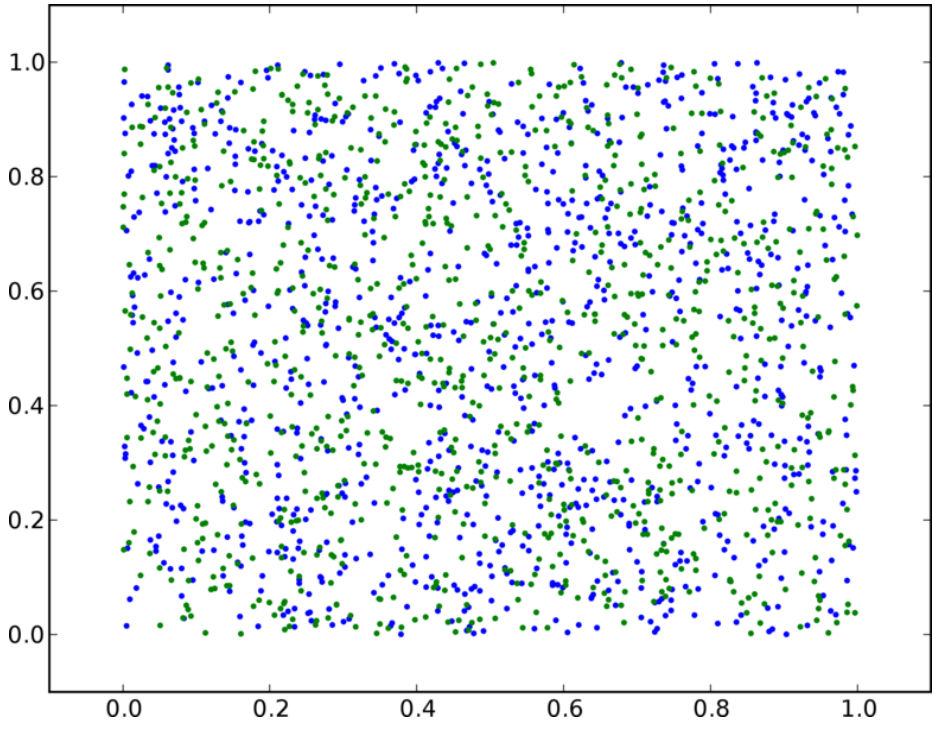
Caveat: We can separate *any* set, not just one with "reasonable" y_i:



There is a fixed feature map $\varphi : \mathbb{R}^2 \longrightarrow \mathbb{R}^{20001}$ such that – no matter how we label them – there is always a hyperplane classifier that has zero training error.

Is this enough?

Caveat: We can separate *any* set, not just one with "reasonable" y_i:



There is a fixed feature map $\varphi : \mathbb{R}^2 \longrightarrow \mathbb{R}^{20001}$ such that – no matter how we label them – there is always a hyperplane classifier that has 0 training error.

Representer Theorem

Solve the soft-margin minimization for $\varphi(x_1), \ldots, \varphi(x_n) \in \mathbb{R}^m$:

$$\min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \tag{1}$$

subject to

$$y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i \quad \text{for } i = 1, \ldots n.$$

For large m, won't solving for $w \in \mathbb{R}^m$ become impossible?

Representer Theorem

Solve the soft-margin minimization for $\varphi(x_1), \ldots, \varphi(x_n) \in \mathbb{R}^m$:

$$\min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \tag{1}$$

subject to

$$y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i \quad \text{for } i = 1, \ldots n.$$

For large m, won't solving for $w \in \mathbb{R}^m$ become impossible? *No!*

Theorem (Representer Theorem)

The minimizing solution w to problem (1) can always be written as

$$w = \sum_{j=1}^{n} \alpha_j \varphi(x_j) \qquad \text{for coefficients } \alpha_1, \ldots, \alpha_n \in \mathbb{R}.$$

Kernel Trick

The representer theorem allows us to rewrite the optimization:

$$\min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

Insert $w = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$:

Kernel Trick

We can minimize over α_i instead of w:

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \|\sum_{j=1}^{n} \alpha_j \varphi(x_j)\|^2 + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \langle \sum_{j=1}^{n} \alpha_j \varphi(x_j), \varphi(x_i) \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

Kernel Trick

Use $\|w\|^2 = \langle w, w \rangle$:

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \sum_{j,k=1}^{n} \alpha_j \alpha_k \langle \varphi(x_j), \varphi(x_k) \rangle + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j \langle \varphi(x_j), \varphi(x_i) \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

Note: φ only occurs in $\langle \varphi(.), \varphi(.) \rangle$ pairs.

Kernel Trick

Set $\langle \varphi(x), \varphi(x') \rangle =: k(x, x')$, called **kernel function**.

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \sum_{j,k=1}^{n} \alpha_j \alpha_k k(x_j, x_k) + C \sum_{i=1}^{n} \xi_i$$

subject to

$$y_i \sum_{j=1}^{n} \alpha_j k(x_j, x_i) \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n.$$

The maximum-margin classifier in this form with a kernel function is often called **Support-Vector Machine** (SVM).

Why use $k(x, x')$ instead of $\langle \varphi(x), \varphi(x') \rangle$?

1) Speed:

- We might find an expression for $k(x_i, x_j)$ that is faster to calculate than forming $\varphi(x_i)$ and then $\langle \varphi(x_i), \varphi(x_j) \rangle$.

  Example: $2nd$-order polynomial kernel (here for $x \in \mathbb{R}^1$):

$$\varphi : x \mapsto (1, \sqrt{2}x, x^2) \in \mathbb{R}^3$$

$$\begin{aligned} \langle \varphi(x_i), \varphi(x_j) \rangle &= \langle\ (1, \sqrt{2}x_i, x_i^2), (1, \sqrt{2}x_j, x_j^2)\ \rangle \\ &= 1 + 2x_i x_j + x_i^2 x_j^2 \end{aligned}$$

  But equivalently (and faster) we can calculate without φ:

$$\begin{aligned} k(x_i, x_j) &:= (1 + x_i x_j)^2 \\ &\left[\ = 1 + 2x_i x_j + x_i^2 x_j^2\ \right] \end{aligned}$$

Why use $k(x, x')$ instead of $\langle \varphi(x), \varphi(x') \rangle$?

2) Flexibility:

- There are kernel functions $k(x_i, x_j)$, for which we *know* that a feature transformation φ *exists*, but we don't know what φ is.

Why use $k(x, x')$ instead of $\langle \varphi(x), \varphi(x') \rangle$?

2) Flexibility:

- There are kernel functions $k(x_i, x_j)$, for which we *know* that a feature transformation φ *exists*, but we don't know what φ is.
- How that???

Theorem

*Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a **positive definite kernel function**. Then there exists a **Hilbert Space** $\mathcal{H}$ and a mapping $\varphi : \mathcal{X} \to \mathcal{H}$ such that*

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

where $\langle ., . \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$.

Positive Definite Kernel Function

Definition (Positive Definite Kernel Function)

Let $\mathcal{X}$ be a non-empty set. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called **positive definite kernel function**, iff

- k is symmetric, i.e. $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$.
- For any set of points $x_1, \ldots, x_n \in \mathcal{X}$, the matrix

$$K_{ij} = \left(k(x_i, x_j)\right)_{i,j}$$

is positive (semi-)definite, i.e. for all vectors $t \in \mathbb{R}^n$:

$$\sum_{i,j=1}^{n} t_i K_{ij} t_j \geq 0.$$

Note: Instead of "*positive definite kernel function*", we will often just say "*kernel*".

Hilbert Spaces

Definition (Hilbert Space)

A **Hilbert Space** $\mathcal{H}$ is a vector space H with an *inner product* $\langle .\,, . \rangle_{\mathcal{H}}$, e.g. a mapping

$$\langle .\,, . \rangle_{\mathcal{H}} : H \times H \to \mathbb{R}$$

which is

- symmetric: $\langle v, v' \rangle_{\mathcal{H}} = \langle v', v \rangle_{\mathcal{H}}$ for all $v, v' \in H$,
- positive definite: $\langle v, v \rangle_{\mathcal{H}} \geq 0$ for all $v \in H$,
  where $\langle v, v \rangle_{\mathcal{H}} = 0$ only for $v = \vec{0} \in H$.
- bilinear: $\langle av, v' \rangle_{\mathcal{H}} = a \langle v, v' \rangle_{\mathcal{H}}$ for $v \in H, a \in \mathbb{R}$
  $\langle v + v', v'' \rangle_{\mathcal{H}} = \langle v, v'' \rangle_{\mathcal{H}} + \langle v', v'' \rangle_{\mathcal{H}}$

We can treat a Hilbert space like some $\mathbb{R}^n$, if we only use concepts like *vectors*, *angles*, *distances*. Note: $\dim \mathcal{H} = \infty$ is possible!

Kernels for Arbitrary Sets

Theorem

*Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a **positive definite kernel function**. Then there exists a **Hilbert Space** $\mathcal{H}$ and a mapping $\varphi : \mathcal{X} \to \mathcal{H}$ such that*

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

where $\langle ., . \rangle_{\mathcal{H}}$ is the inner product in $\mathcal{H}$.

Translation

Take *any* set $\mathcal{X}$ and *any* function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.
If k is a positive definite kernel, then we can use k to learn a (soft) maximum-margin classifier for the elements in $\mathcal{X}$!

Note: $\mathcal{X}$ can be any set, e.g. $\mathcal{X} = \{$ *all images* $\}$.

How to Check if a Function is a Kernel

Problem:

- Checking if a given $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ fulfills the conditions for a kernel is *difficult*:
- We need to prove or disprove

$$\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0.$$

for *any set* $x_1, \ldots, x_n \in \mathcal{X}$ *and any* $t \in \mathbb{R}^n$ *for any* $n \in \mathbb{N}$.

Workaround:

- It is easy to *construct* functions k that are positive definite kernels.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

Constructing Kernels

1) We can *construct kernels from scratch*:

- For any $\varphi : \mathcal{X} \to \mathbb{R}^m$, $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathbb{R}^m}$ is a kernel.
- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *distance function*, i.e.
  - $d(x, x') \geq 0$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') = 0$ only for $x = x'$,
  - $d(x, x') = d(x', x)$ for all $x, x' \in \mathcal{X}$,
  - $d(x, x') \leq d(x, x'') + d(x'', x')$ for all $x, x', x'' \in \mathcal{X}$,

  then $k(x, x') := \exp(-d(x, x'))$ is a kernel.

2) We can *construct kernels from other kernels*:

- if k is a kernel and $\alpha > 0$, then αk and $k + \alpha$ are kernels.
- if k_1, k_2 are kernels, then $k_1 + k_2$ and $k_1 \cdot k_2$ are kernels.

Constructing Kernels

Examples for kernels for $\mathcal{X} = \mathbb{R}^d$:

- any linear combination $\sum_j \alpha_j k_j$ with $\alpha_j \geq 0$,
- *polynomial kernels* $k(x, x') = (1 + \langle x, x' \rangle)^m$, $m > 0$
- *Gaussian* or RBF $k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$,

Constructing Kernels

Examples for kernels for $\mathcal{X} = \mathbb{R}^d$:

- any linear combination $\sum_j \alpha_j k_j$ with $\alpha_j \geq 0$,
- *polynomial kernels* $k(x, x') = (1 + \langle x, x' \rangle)^m$, $m > 0$
- *Gaussian* or RBF $k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$,

Examples for kernels for other $\mathcal{X}$:

- $k(h, h') = \sum_{i=1}^n \min(h_i, h'_i)$ for n-bin histograms h, h'.
- $k(p, p') = \exp(-KL(p, p'))$ with KL the symmetrized *KL-divergence* between positive probability distributions.
- $k(s, s') = \exp(-D(s, s'))$ for *strings* s, s' and $D =$ *edit distance*

Constructing Kernels

Examples for kernels for $\mathcal{X} = \mathbb{R}^d$:

- any linear combination $\sum_j \alpha_j k_j$ with $\alpha_j \geq 0$,
- *polynomial kernels* $k(x, x') = (1 + \langle x, x' \rangle)^m$, $m > 0$
- *Gaussian* or RBF $k(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$,

Examples for kernels for other $\mathcal{X}$:

- $k(h, h') = \sum_{i=1}^n \min(h_i, h'_i)$ for n-bin histograms h, h'.
- $k(p, p') = \exp(-KL(p, p'))$ with KL the symmetrized *KL-divergence* between positive probability distributions.
- $k(s, s') = \exp(-D(s, s'))$ for *strings* s, s' and $D =$ *edit distance*

Examples for functions $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that are not kernels:

- $\tanh(\kappa\langle x, x' \rangle + \theta)$ (matrix K_{ij} can have negative eigenvalues)

Kernels in Computer Vision

$\mathcal{X} = \{$ *images* $\}$, treat feature extraction as part of kernel definition

- OCR/handwriting recognition
  - resize image, normalize brightness/contrast/rotation/skew
  - *polynomial kernel* $k(x, x') = (1 + \langle x, x' \rangle)^d$, $d > 0$
    [DeCoste, Schölkopf. ML2002]
- Pedestrian detection
  - resize image, calculate local intensity gradient directions
  - local thresholding + *linear kernel* [Dalal, Triggs. CVPR 2005]
    or
    local L^1-normalization + *histogram intersection kernel*
    [Maji, Berg, Malik. CVPR 2008]

Kernels in Computer Vision

$\mathcal{X} = \{$ *images* $\}$, treat feature extraction as part of kernel definition

- object category recognition
  - extract local image descriptors, e.g. SIFT
  - calculate multi-level pyramid histograms $h^{l,k}(x)$
  - *pyramid match kernel* [Grauman, Darrell. ICCV 2005]

$$k_{PMK}(x, x') = \sum_{l=1}^{L} 2^l \sum_{k=1}^{2^{l-1}} \min\left(h^{l,k}(x), h^{l,k}(x')\right)$$

- scene/object category recognition
  - extract local image descriptors, e.g. SIFT
  - quantize descriptors into bag-of-words histograms
  - χ^2-kernel [Puzicha, Buhmann, Rubner, Tomasi. ICCV1999]

$$k_{\chi^2}(h, h') = \exp\left(-\gamma\chi^2(h, h')\right) \quad \text{for } \gamma > 0$$

$$\chi^2(h, h') = \sum_{k=1}^{K} \frac{(h_k - h'_k)^2}{h_k + h'_k}$$

Summary

Linear methods are popular and well understood

- *classification*, regression, dimensionality reduction, ...

Kernels are at the same time...

1) Similarity measure between (arbitrary) objects,
2) Scalar products in a (hidden) vector space.

Kernelization can make linear techniques more powerful

- implicit preprocessing, *non-linear* in the original data.
- still linear in *some* feature space $\Rightarrow$ still intuitive/interpretable

Kernels can be defined over arbitrary inputs, e.g. images

- unified framework for all preprocessing steps
- different features, normalization, etc., becomes kernel choices

What did we not see?

We have skipped the largest part of theory on kernel methods:

- Optimization
  - Dualization
  - Algorithms to train SVMs
- Kernel Design
  - Systematic methods to construct data-dependent kernels.
- Statistical Interpretations
  - What do we assume about samples?
  - What performance can we expect?
- Generalization Bounds
  - The *test* error of a (kernelized) linear classifier can be controlled using its *modelling* error and its *training* error.
- "Support Vectors"

This and much more in standard references.

Selecting and Combining Kernels

Selecting From Multiple Kernels

Typically, one has many different kernels to choose from:

- different functional forms
  - linear, polynomial, RBF, ...
- different parameters
  - polynomial degree, Gaussian bandwidth, ...

Selecting From Multiple Kernels

Typically, one has many different kernels to choose from:

- different functional forms
  - linear, polynomial, RBF, ...
- different parameters
  - polynomial degree, Gaussian bandwidth, ...

Different *image features* give rise to different *kernels*

- Color histograms,
- SIFT bag-of-words,
- HOG,
- Pyramid match,
- Spatial pyramids, ...

Selecting From Multiple Kernels

Typically, one has many different kernels to choose from:

- different functional forms
  - linear, polynomial, RBF, ...
- different parameters
  - polynomial degree, Gaussian bandwidth, ...

Different *image features* give rise to different *kernels*

- Color histograms,
- SIFT bag-of-words,
- HOG,
- Pyramid match,
- Spatial pyramids, ...

How to choose?

- Ideally, based on the kernels' *performance* on task at hand:
  - estimate by cross-validation or validation set error
- Classically part of "Model Selection".

Kernel Parameter Selection

Note: Model Selection makes a difference!

- Action Classification, KTH dataset

Method	**Accuracy**
Dollár et al. VS-PETS 2005: *"SVM classifier"*	80.66
Nowozin et al., ICCV 2007: *"baseline RBF"*	85.19

- identical features, same kernel function
- difference: Nowozin used cross-validation for model selection (bandwidth and C)

Note: there is *no overfitting* involved here. Model selection is fully automatic and uses only training data.

Kernel Parameter Selection

Rule of thumb for kernel parameters

- For kernels based on the exponential function

$$k(x, x') = \exp(-\frac{1}{\gamma} X(x, x'))$$

with any X, set

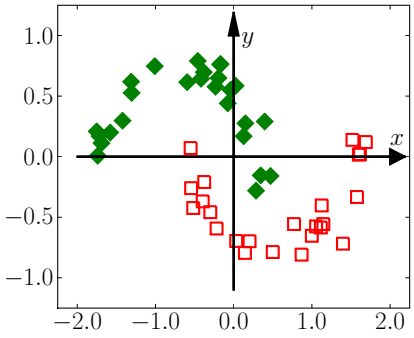
$$\gamma \approx \text{mean}_{i,j=1,\ldots,n}\, X(x_i, x_j).$$

Sometimes better: use only $X(x_i, x_j)$ with $y_i \neq y_j$.

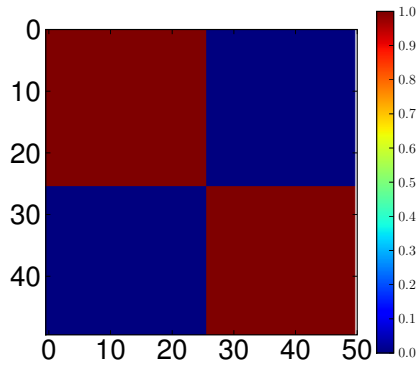
- In general, if there are several classes, then the *kernel matrix*:

$$K_{ij} = k(x_i, x_j)$$

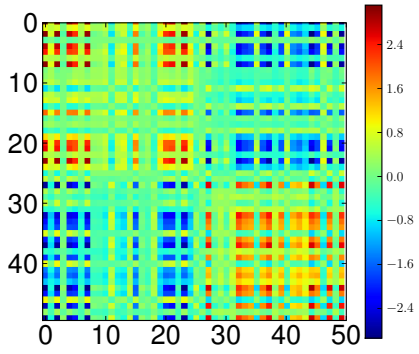
should have a *block structure* w.r.t. the classes.



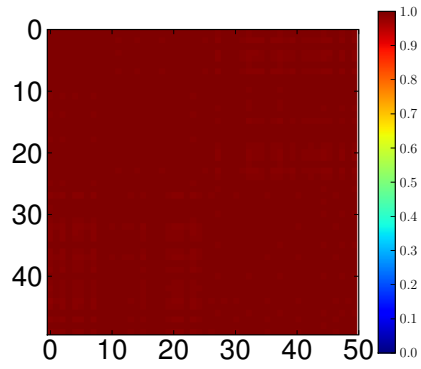
two moons



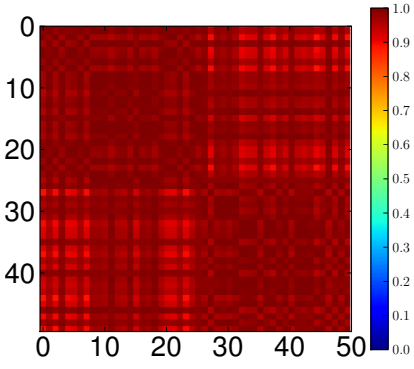
label “kernel”



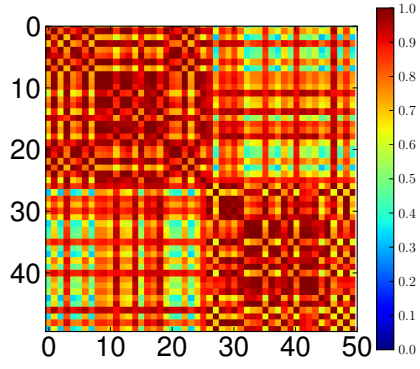
linear



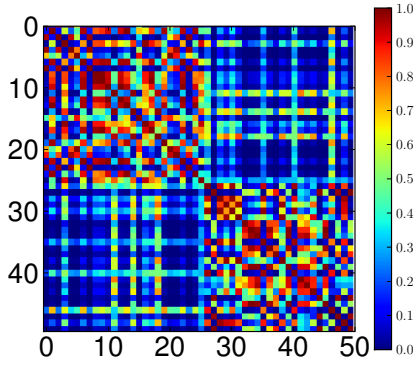
RBF: $\gamma = 0.001$



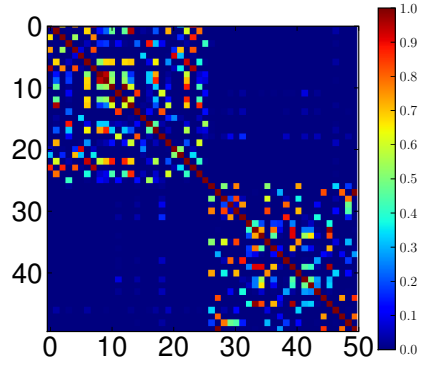
$\gamma = 0.01$



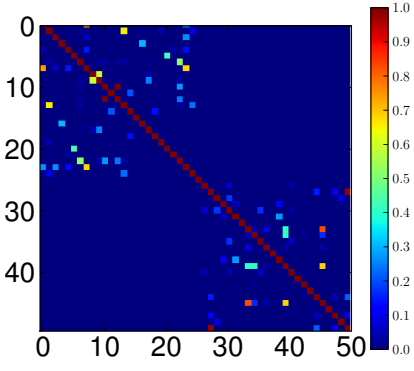
$\gamma = 0.1$



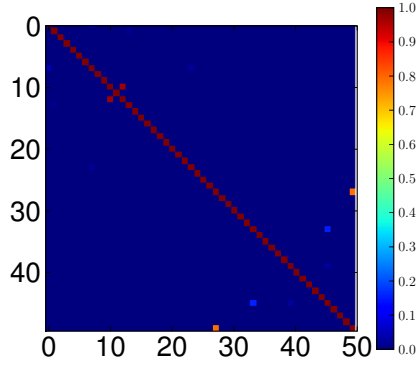
$\gamma = 1$



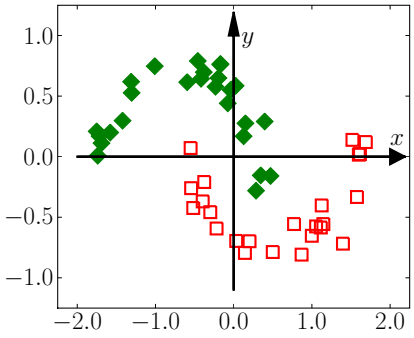
$\gamma = 10$



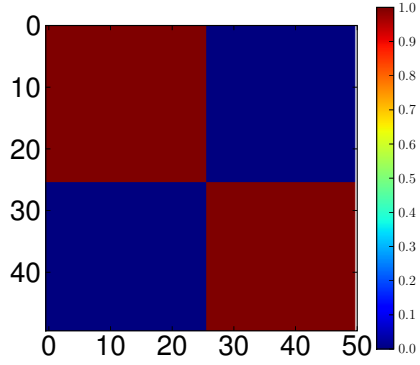
$\gamma = 100$



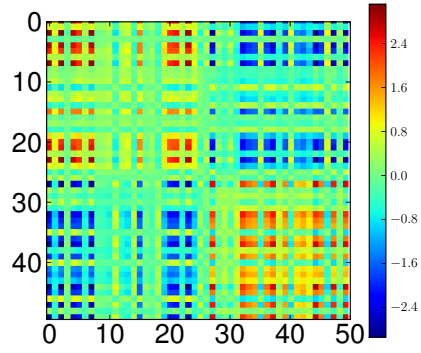
$\gamma = 1000$



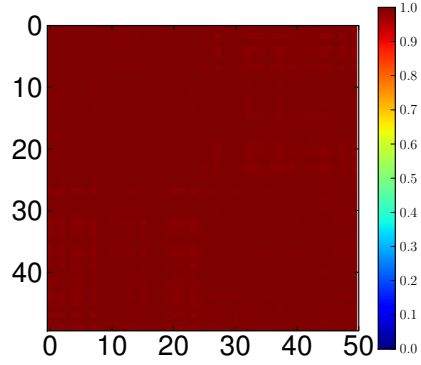
two moons



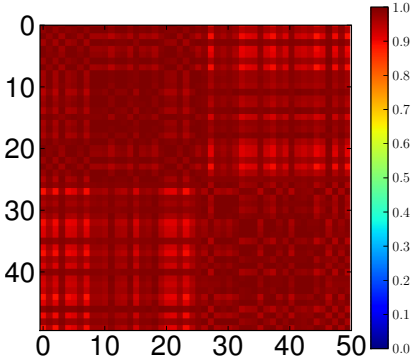
label “kernel”



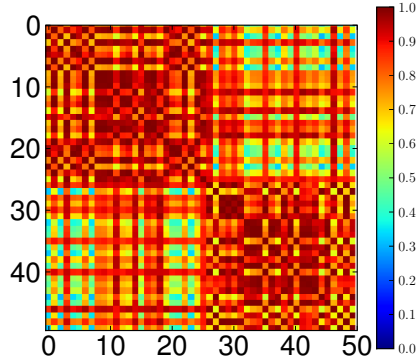
linear



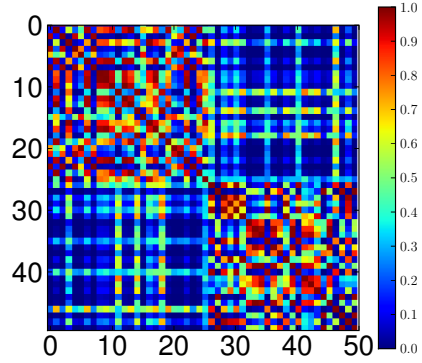
RBF: $\gamma = 0.001$



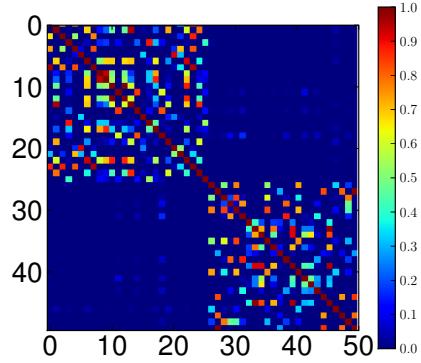
$\gamma = 0.01$



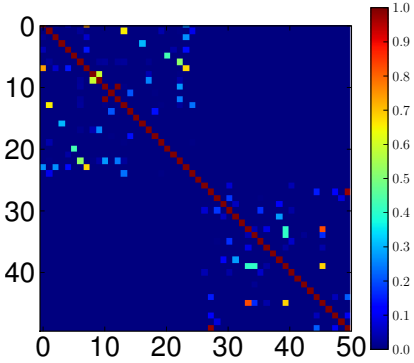
$\gamma = 0.1$



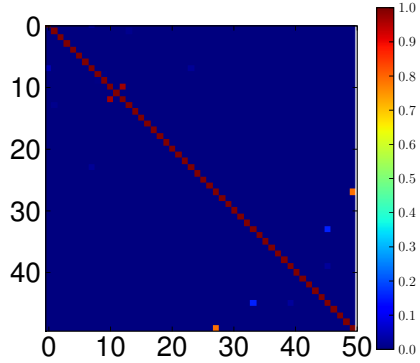
$\gamma = 1$



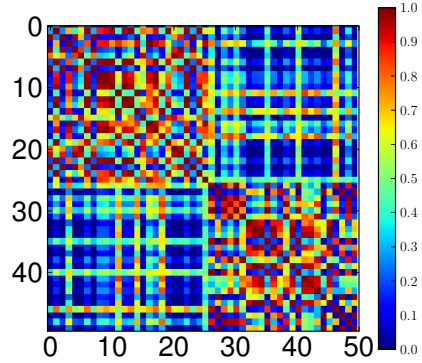
$\gamma = 10$



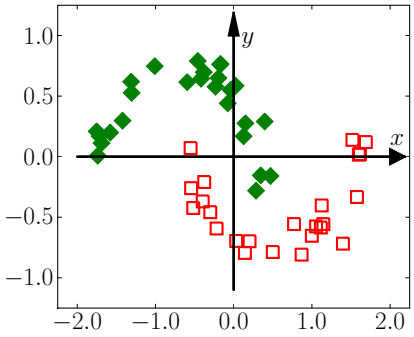
$\gamma = 100$



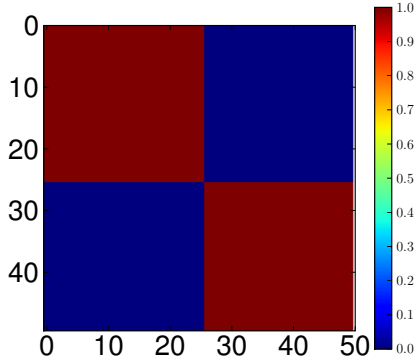
$\gamma = 1000$



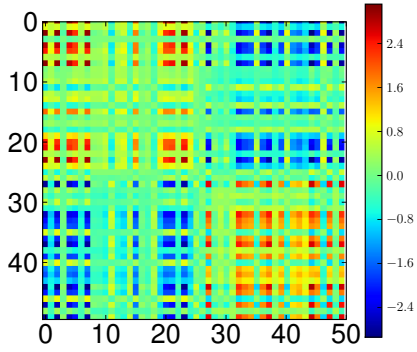
$\gamma = 0.6$
rule of thumb



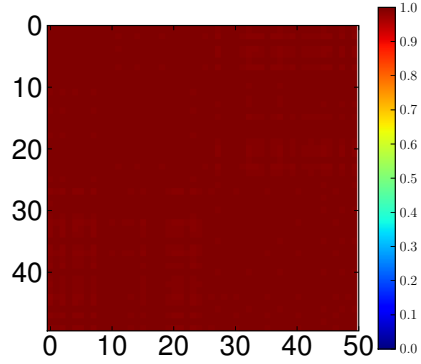
two moons



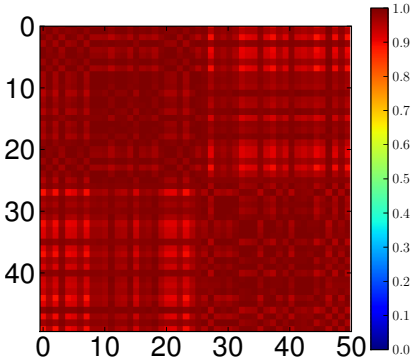
label "kernel"



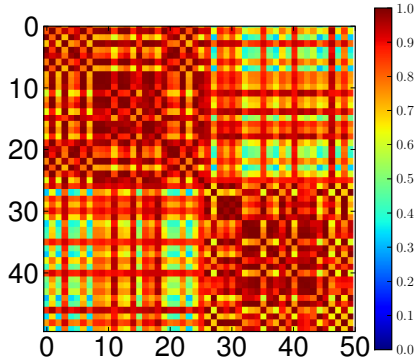
linear



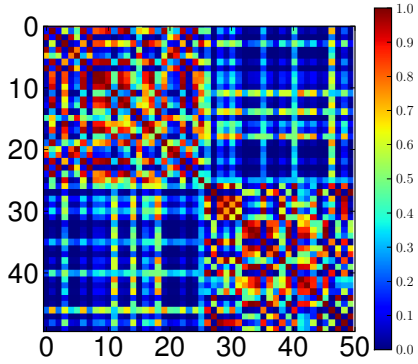
RBF: $\gamma = 0.001$



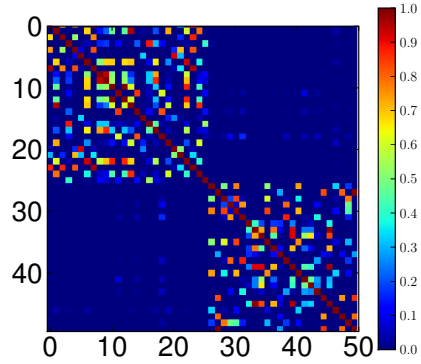
$\gamma = 0.01$



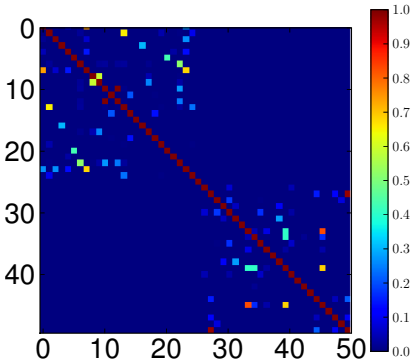
$\gamma = 0.1$



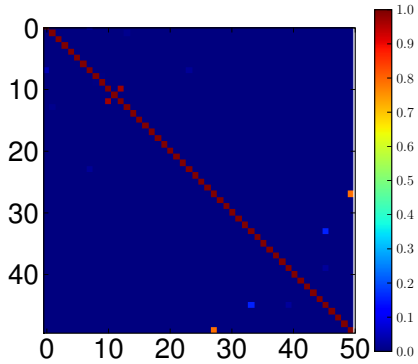
$\gamma = 1$



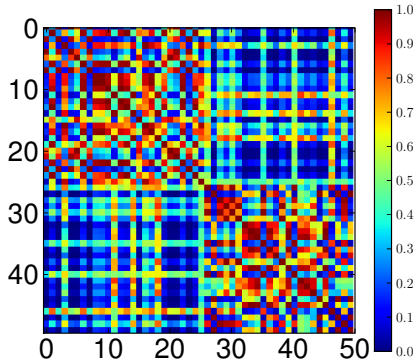
$\gamma = 10$



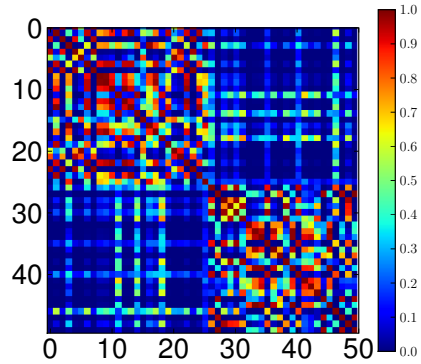
$\gamma = 100$



$\gamma = 1000$



$\gamma = 0.6$
rule of thumb



$\gamma = 1.6$
5-fold CV

Kernel Selection $\leftrightarrow$ Kernel Combination

Is there a single *best* kernel at all?

- Kernels are typcally designed to capture *one aspect* of the data
  - texture, color, edges, ...
- *Choosing* one kernel means to *select* exactly one such aspect.

Kernel Selection $\leftrightarrow$ Kernel Combination

Is there a single *best* kernel at all?

- Kernels are typcally designed to capture *one aspect* of the data
  - texture, color, edges, ...
- *Choosing* one kernel means to *select* exactly one such aspect.
- *Combining* aspects if often better than *Selecting*.

Method	Accuracy
Colour	60.9 ± 2.1
Shape	70.2 ± 1.3
Texture	63.7 ± 2.7
HOG	58.5 ± 4.5
HSV	61.3 ± 0.7
siftint	70.6 ± 1.6
siftbdy	59.4 ± 3.3
combination	$\mathbf{85.2 \pm 1.5}$

Mean accuracy on Oxford Flowers dataset [Gehler, Nowozin: ICCV2009]

Combining Two Kernels

For two kernels k_1, k_2:

- product $k = k_1 k_2$ is again a kernel
  - Problem: very small kernel values suppress large ones
- average $k = \frac{1}{2}(k_1 + k_2)$ is again a kernel
  - Problem: k_1, k_2 on different scales. Re-scale first?
  - convex combination $k_\beta = (1 - \beta)k_1 + \beta k_2$ with $\beta \in [0, 1]$
- Model selection: cross-validate over $\beta \in \{0, 0.1, \ldots, 1\}$.

Combining Many Kernels

Multiple kernels: $k_1, \ldots, k_K$

- all convex combinations are kernels:

$$k = \sum_{j=1}^{K} \beta_j k_j \qquad \text{with } \beta_j \geq 0, \quad \sum_{j=1}^{K} \beta = 1.$$

- Kernels can be "deactivated" by $\beta_j = 0$.
- Combinatorial explosion forbids cross-validation over all combinations of β_j

Proxy: instead of CV, maximize SVM-objective.

- Each combined kernel induces a feature space.
- In which of the feature spaces can we best
  - *explain the training data*, and
  - achieve a *large margin* between the classes?

Feature Space View of Kernel Combination

Each kernel k_j induces

- a Hilbert Space $\mathcal{H}_j$ and a mapping $\varphi_j : \mathcal{X} \to \mathcal{H}_j$.

The weighted kernel $k_j^{\beta_j} := \beta_j k_j$ induces

- the *same* Hilbert Space $\mathcal{H}_j$, but
- a *rescaled* feature mapping $\varphi_j^{\beta_j}(x) := \sqrt{\beta_j}\,\varphi_j(x)$.

$$k^{\beta_j}(x, x') \equiv \langle \varphi_j^{\beta_j}(x), \varphi_j^{\beta_j}(x') \rangle_{\mathcal{H}} = \langle \sqrt{\beta_j}\varphi_j(x), \sqrt{\beta_j}\varphi_j(x') \rangle_{\mathcal{H}}$$
$$= \beta_j \langle \varphi_j(x), \varphi_j(x') \rangle_{\mathcal{H}} = \beta_j k(x, x').$$

The linear combination $\hat{k} := \sum_{j=1}^{K} \beta_j k_j$ induces

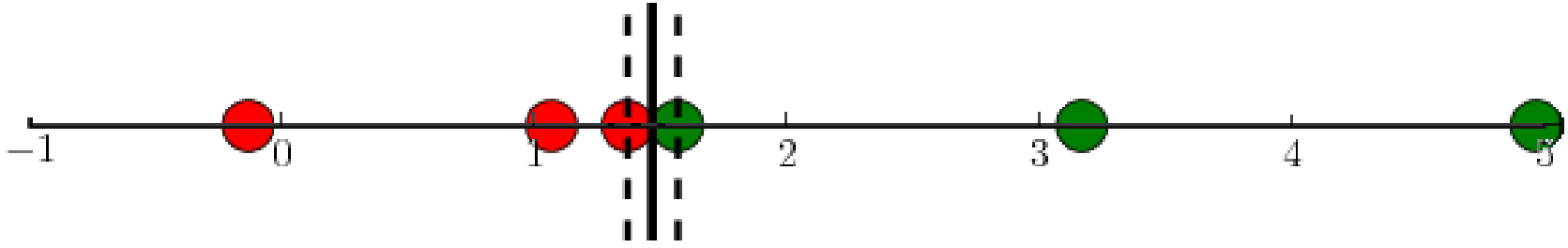
- the product space $\widehat{\mathcal{H}} := \oplus_{j=1}^{K} \mathcal{H}_j$, and
- the product mapping $\hat{\varphi}(x) := (\varphi_1^{\beta_1}(x), \ldots, \varphi_n^{\beta_n}(x))^t$

$$\hat{k}(x, x') \equiv \langle \hat{\varphi}(x), \hat{\varphi}(x') \rangle_{\widehat{\mathcal{H}}} = \sum_{j=1}^{K} \langle \varphi_j^{\beta_j}(x), \varphi_j^{\beta_j}(x') \rangle_{\mathcal{H}} = \sum_{j=1}^{K} \beta_j k(x, x')$$

Feature Space View of Kernel Combination

Implicit representation of a dataset using two kernels:

Kernel k_1, feature representation $\varphi_1(x_1), \ldots, \varphi_1(x_n) \in \mathcal{H}_1$

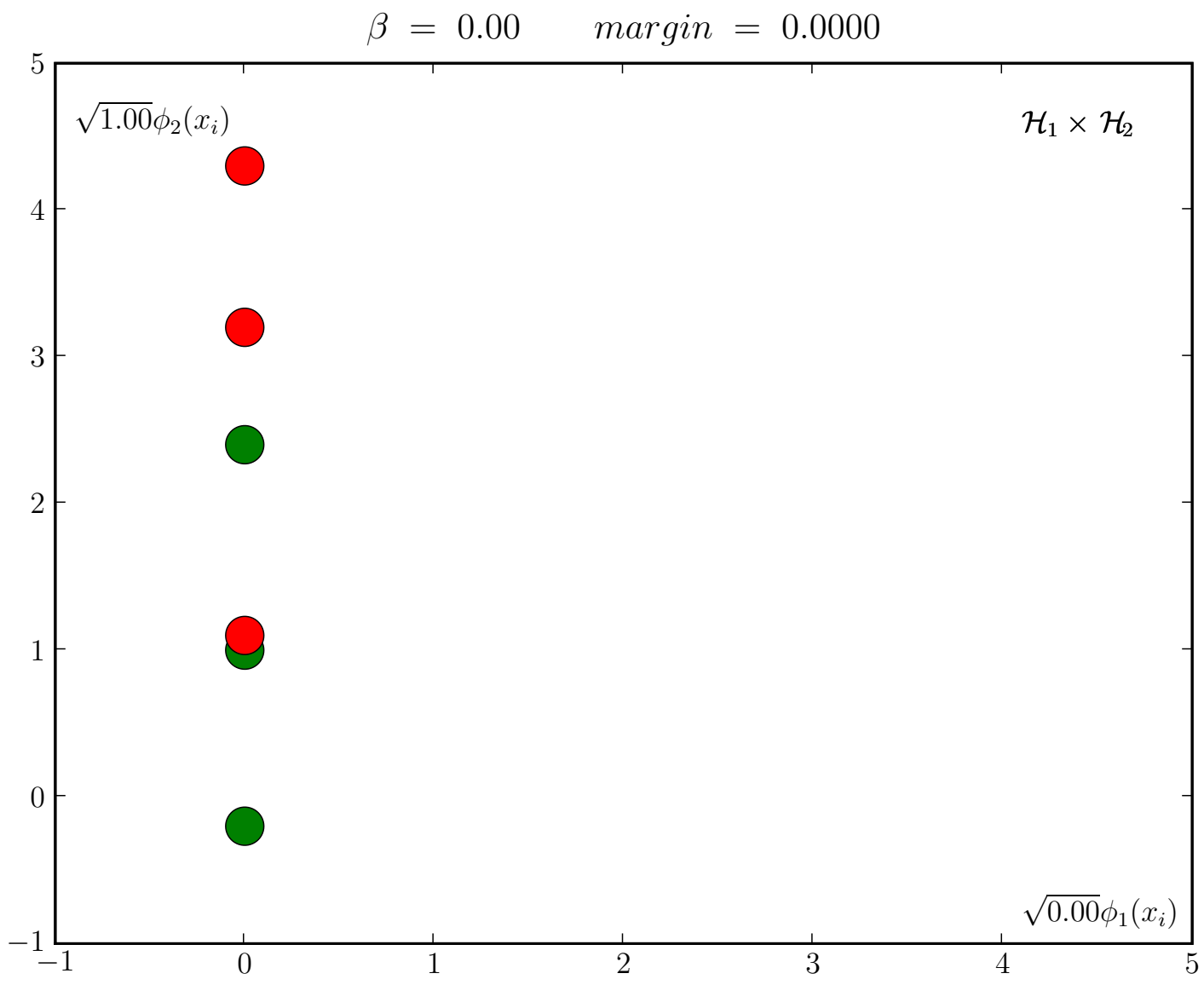


Kernel k_2, feature representation $\varphi_2(x_1), \ldots, \varphi_2(x_n) \in \mathcal{H}_2$

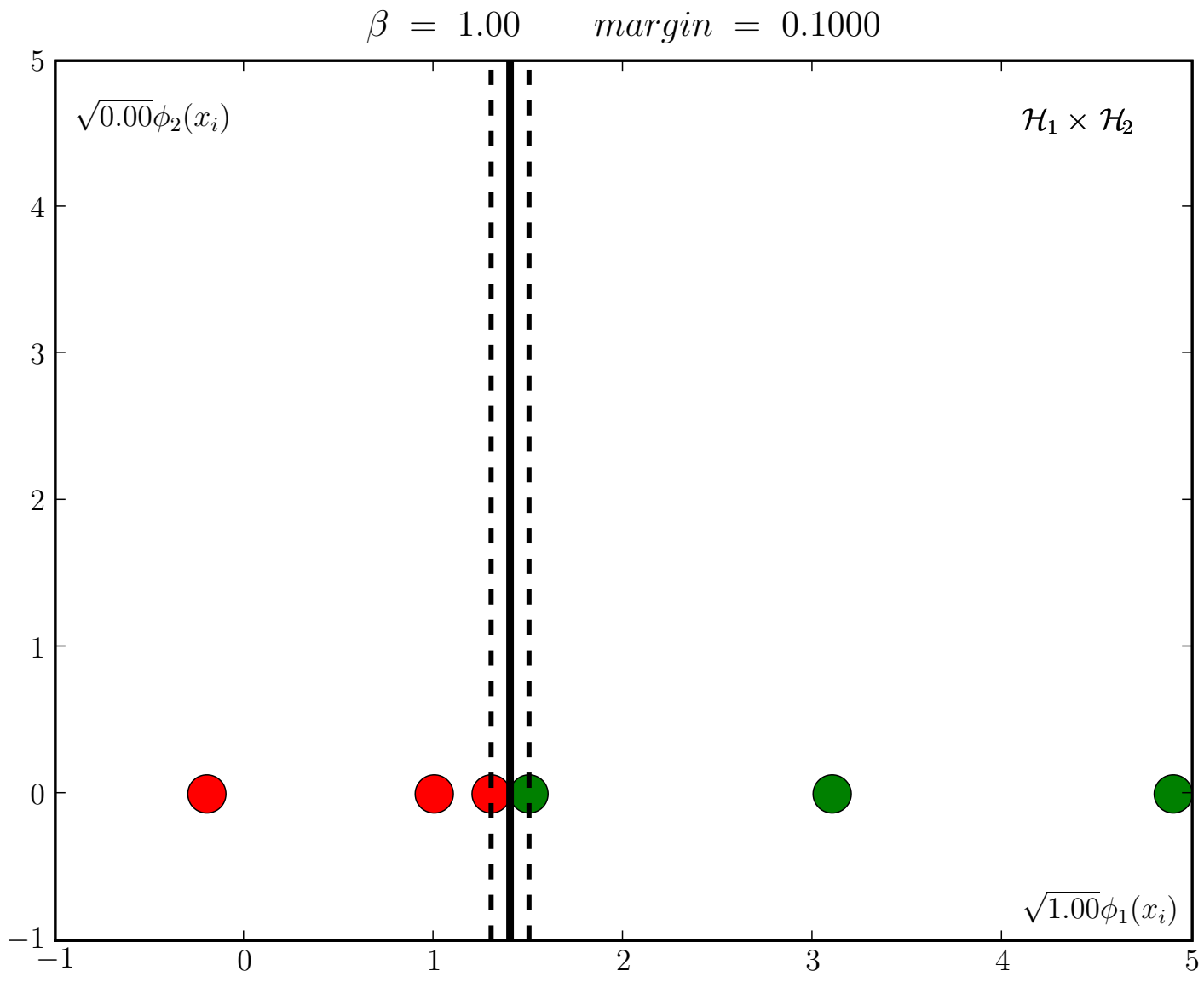
Kernel *Selection* would most likely pick k_2.

For $k = (1 - \beta)k_1 + \beta k_2$, top is $\beta = 0$, bottom is $\beta = 1$.

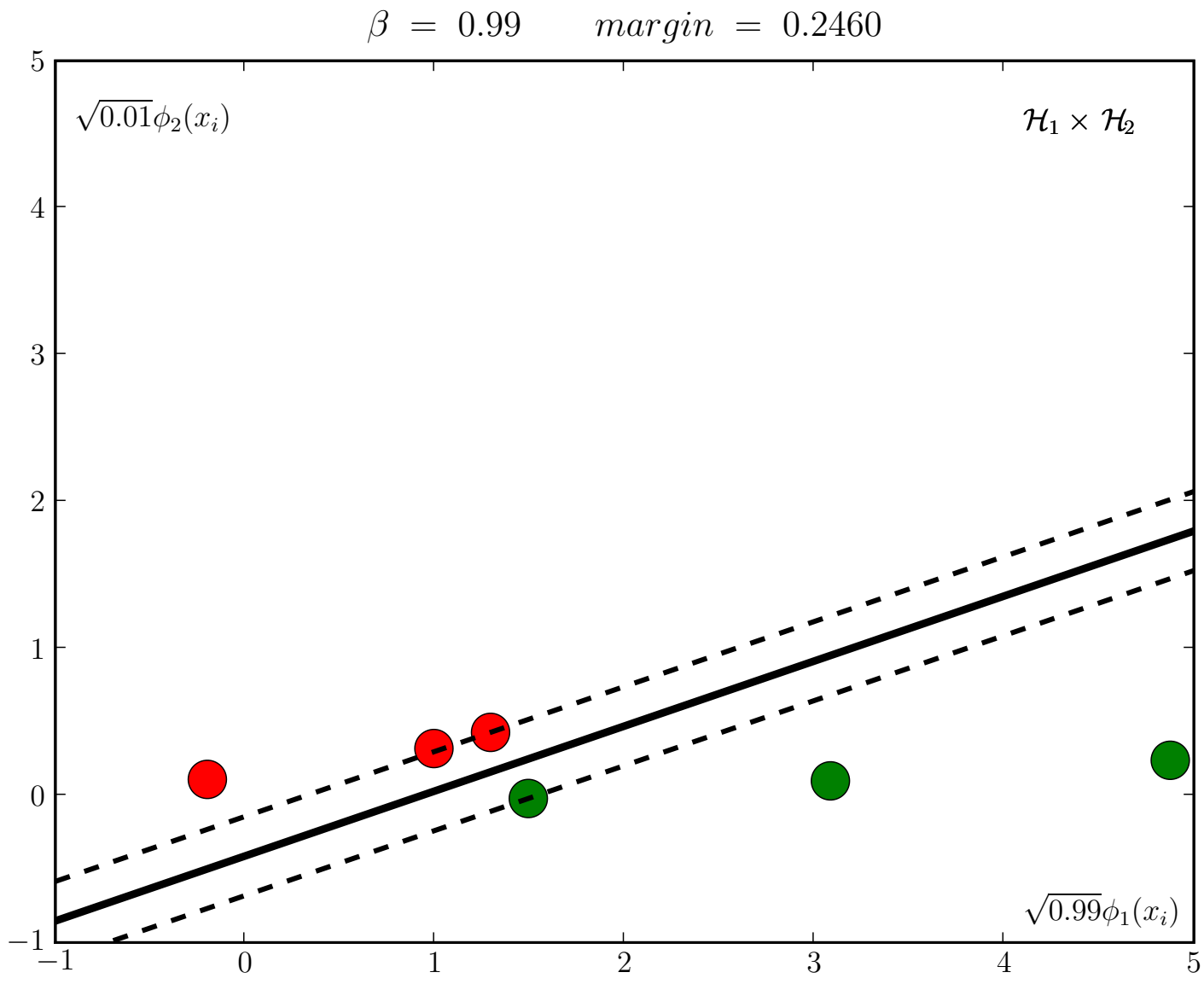
Feature Space View of Kernel Combination



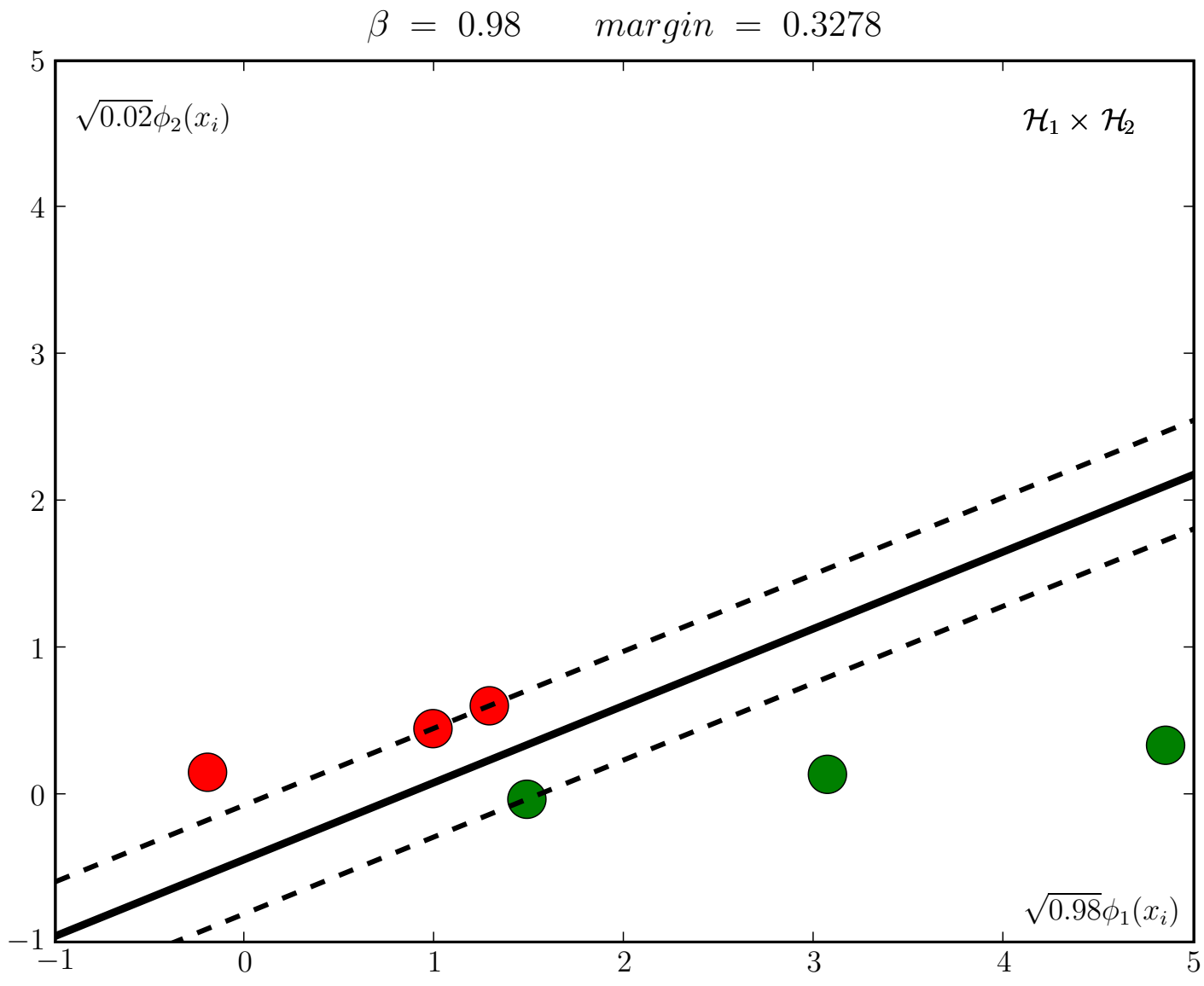
Feature Space View of Kernel Combination



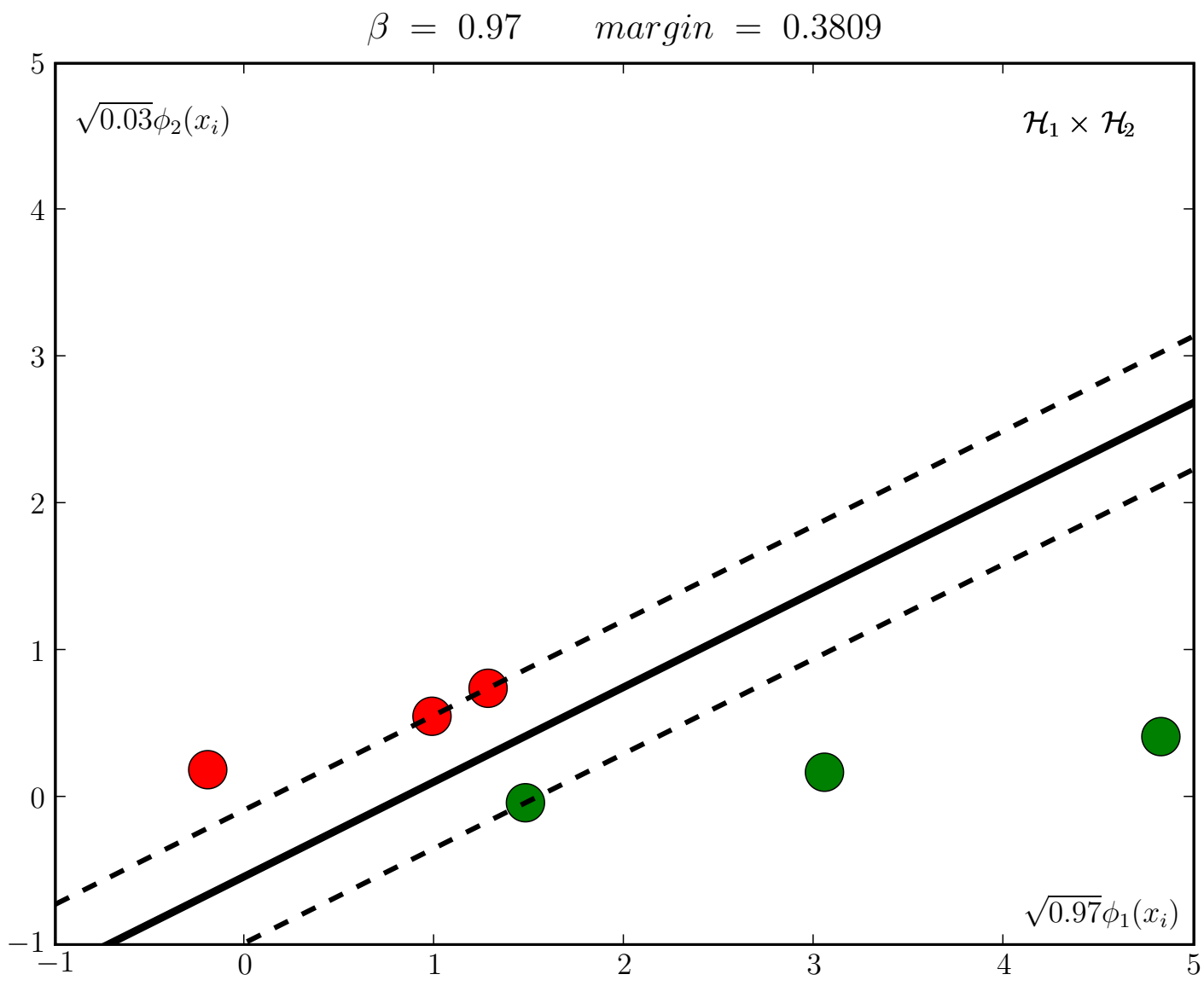
Feature Space View of Kernel Combination



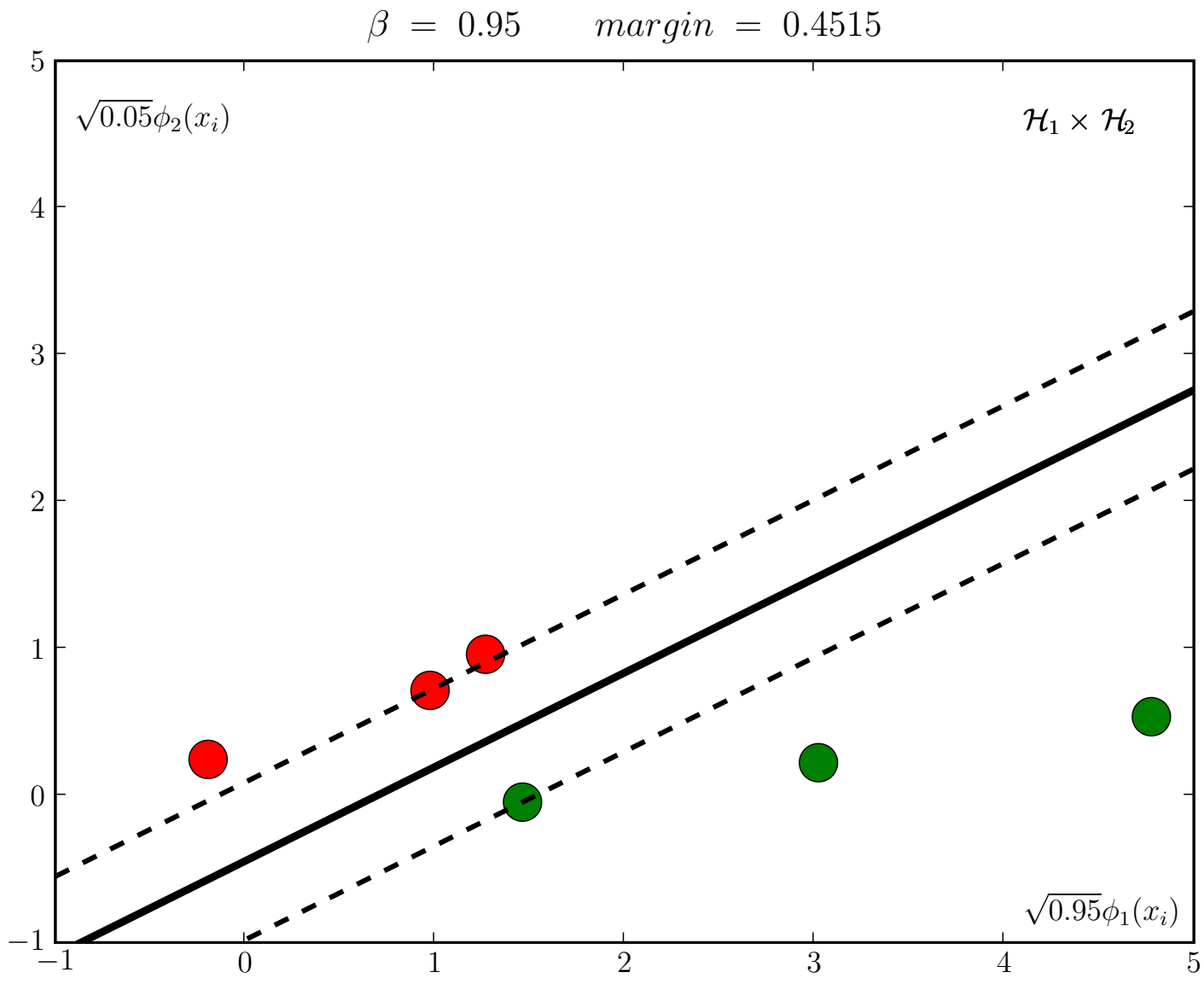
Feature Space View of Kernel Combination



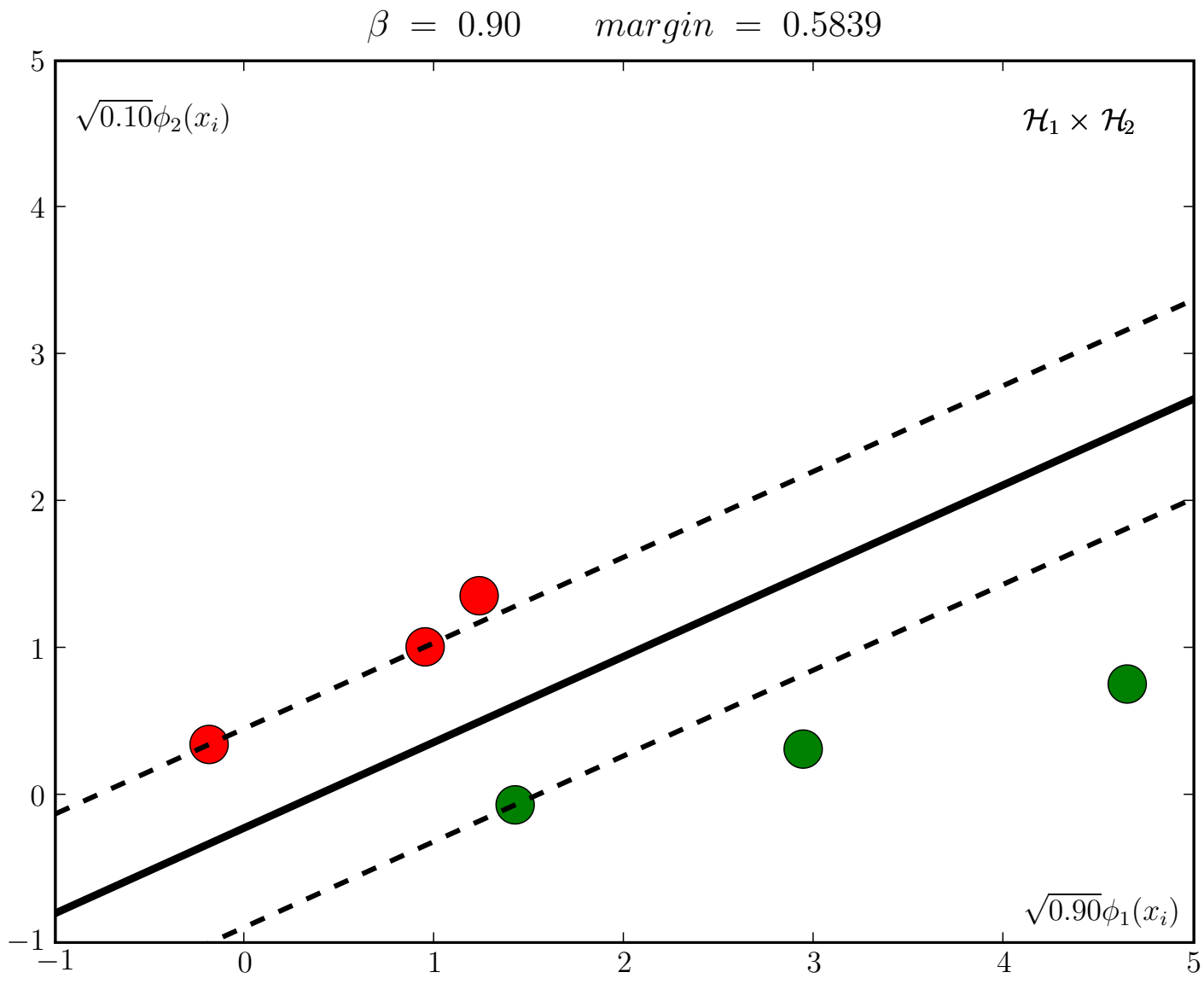
Feature Space View of Kernel Combination



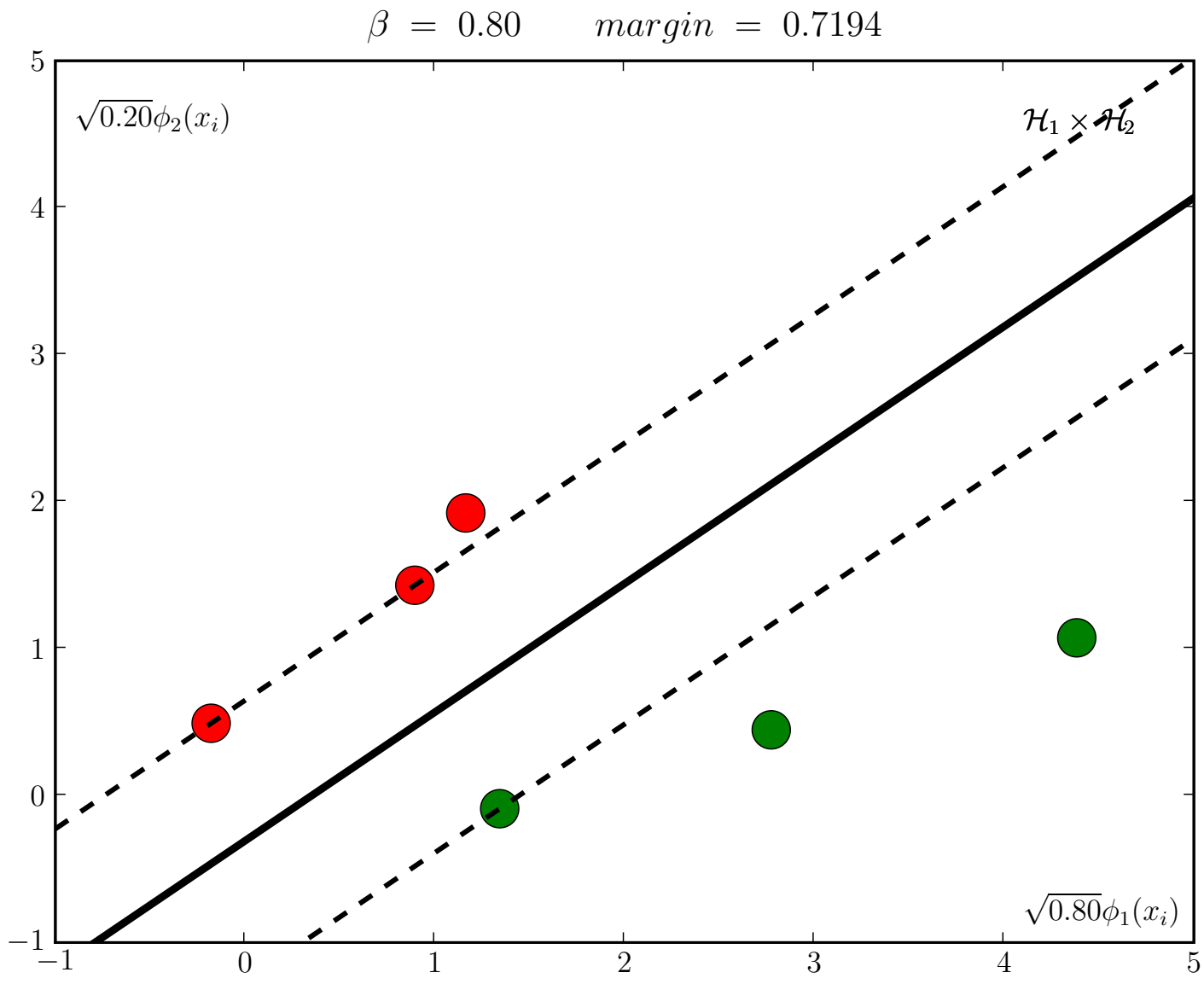
Feature Space View of Kernel Combination



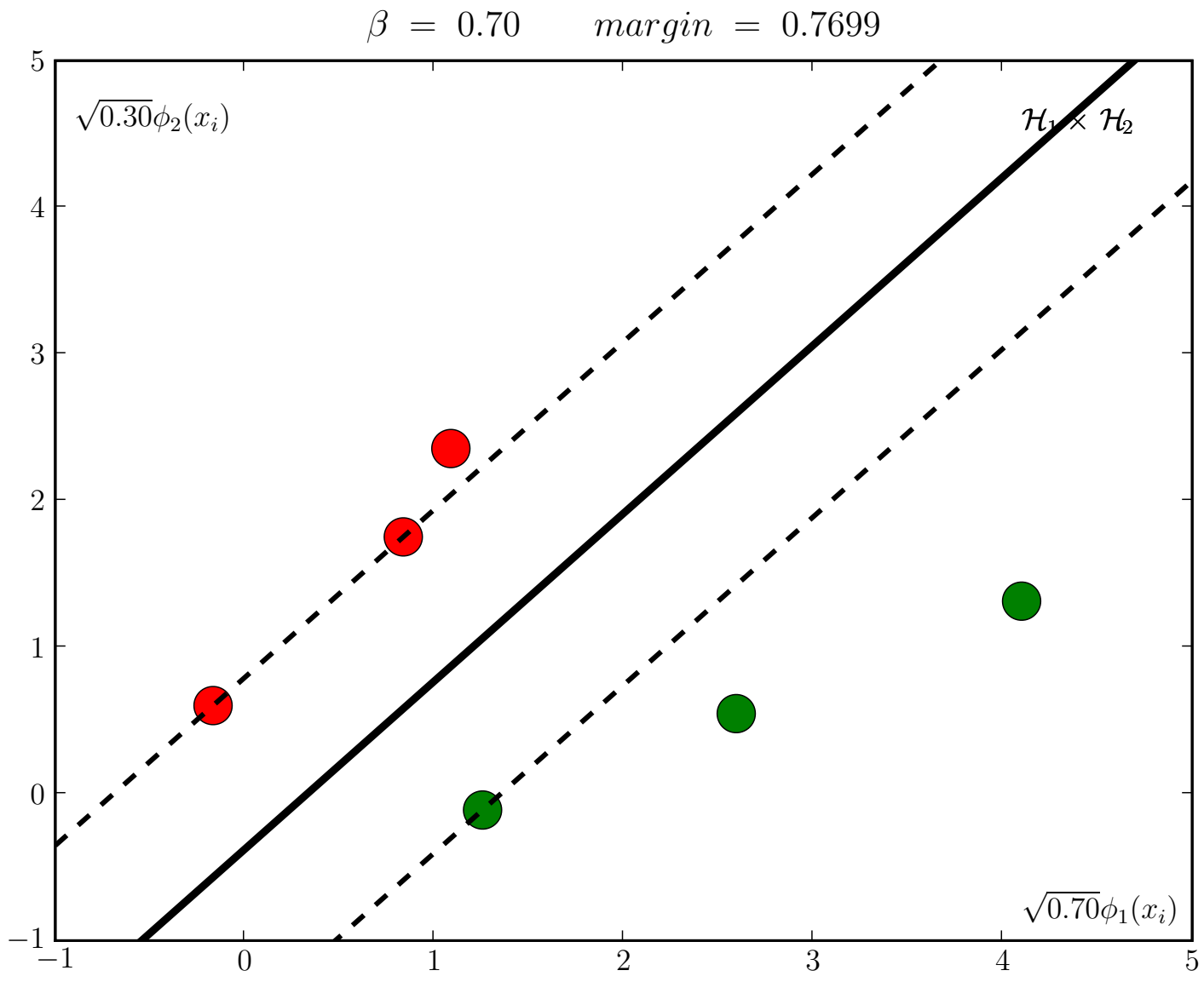
Feature Space View of Kernel Combination



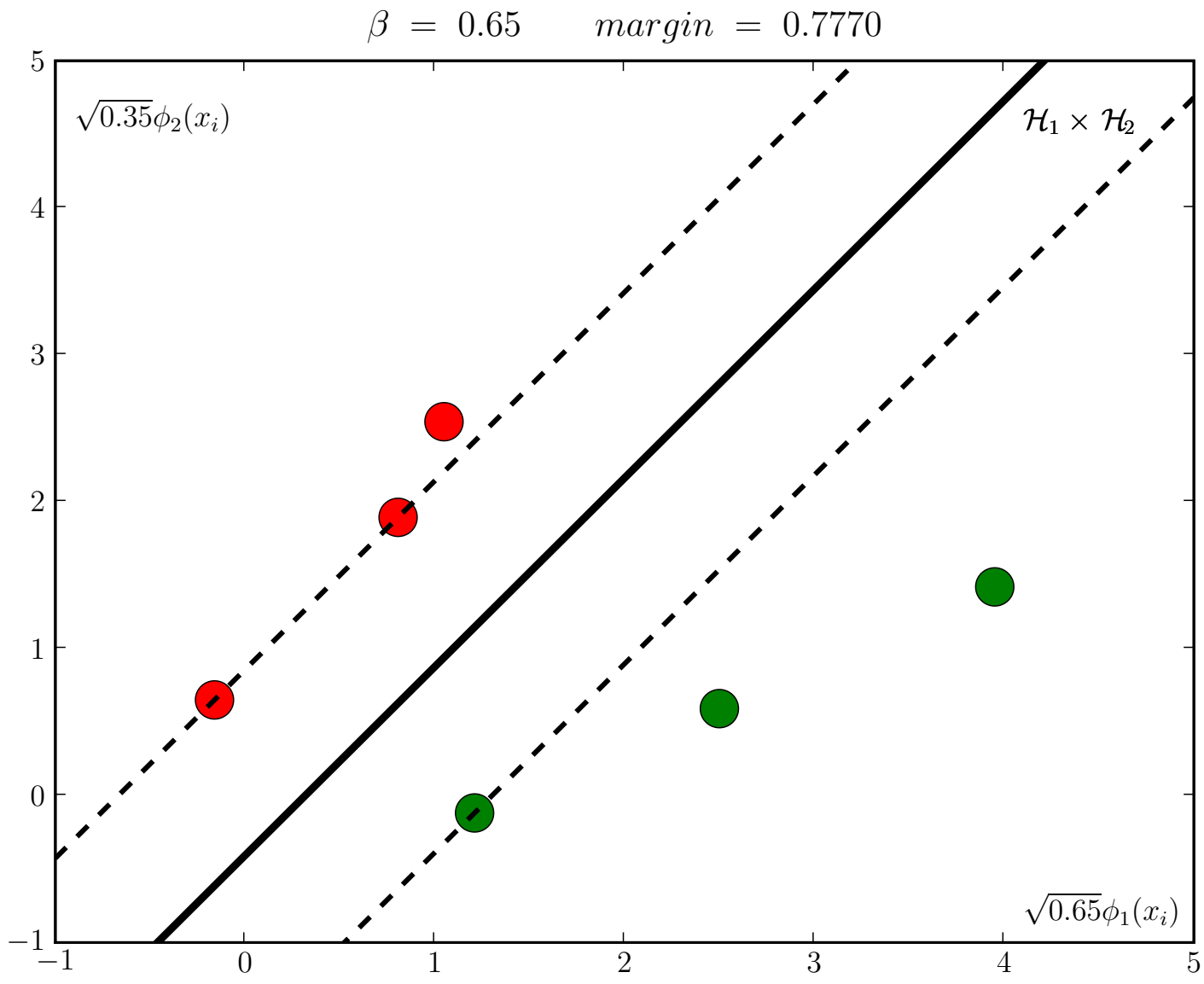
Feature Space View of Kernel Combination



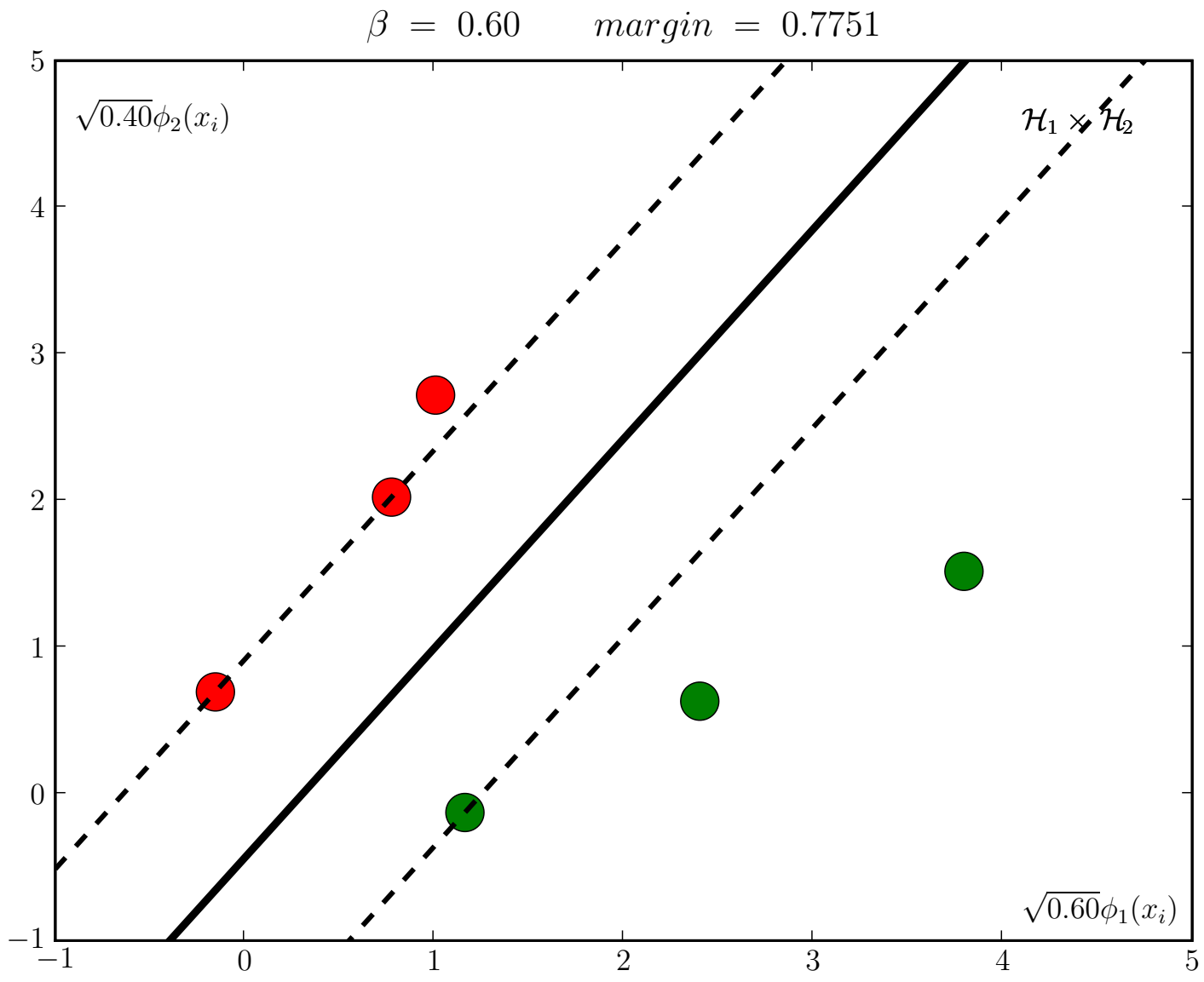
Feature Space View of Kernel Combination



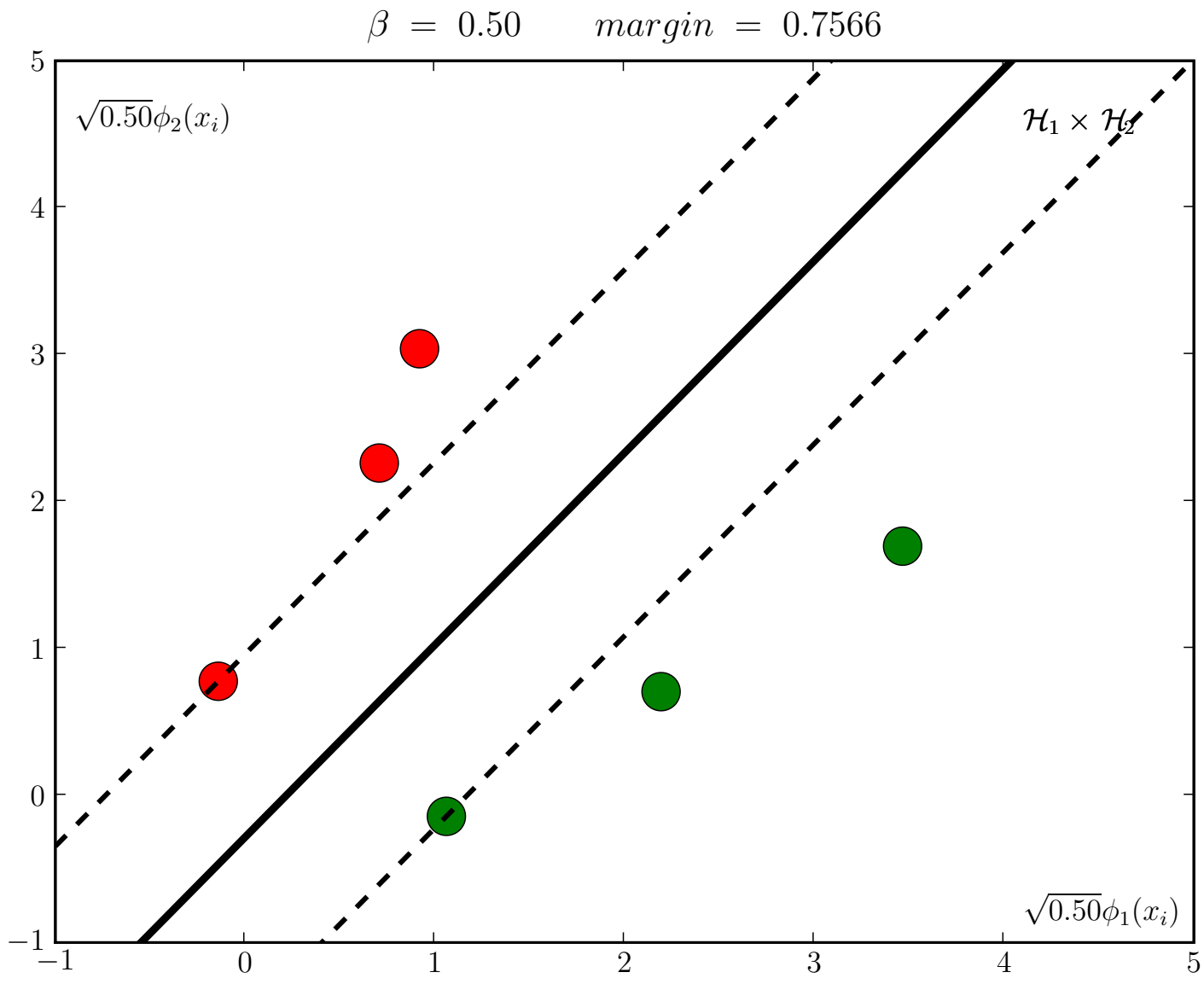
Feature Space View of Kernel Combination



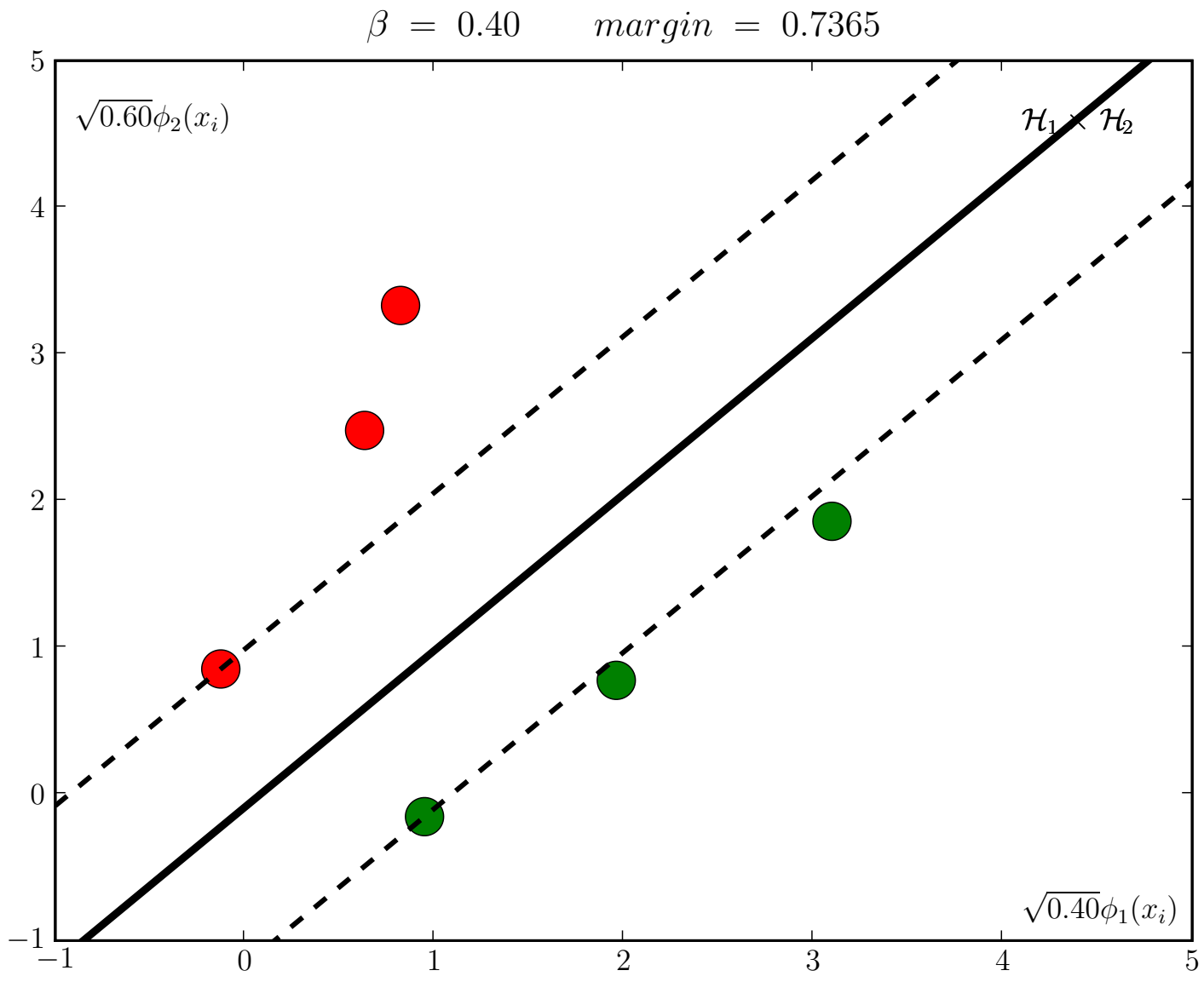
Feature Space View of Kernel Combination



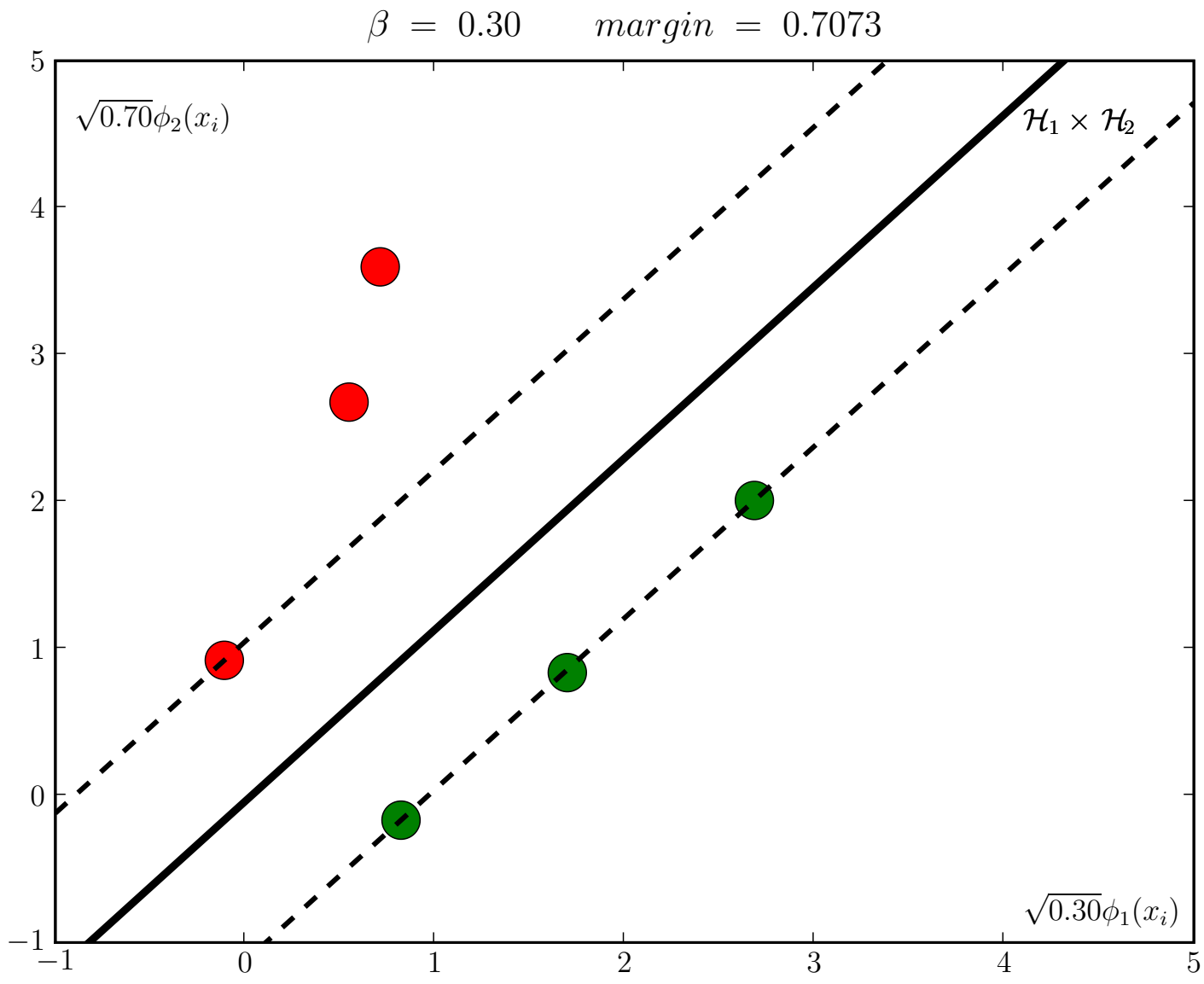
Feature Space View of Kernel Combination



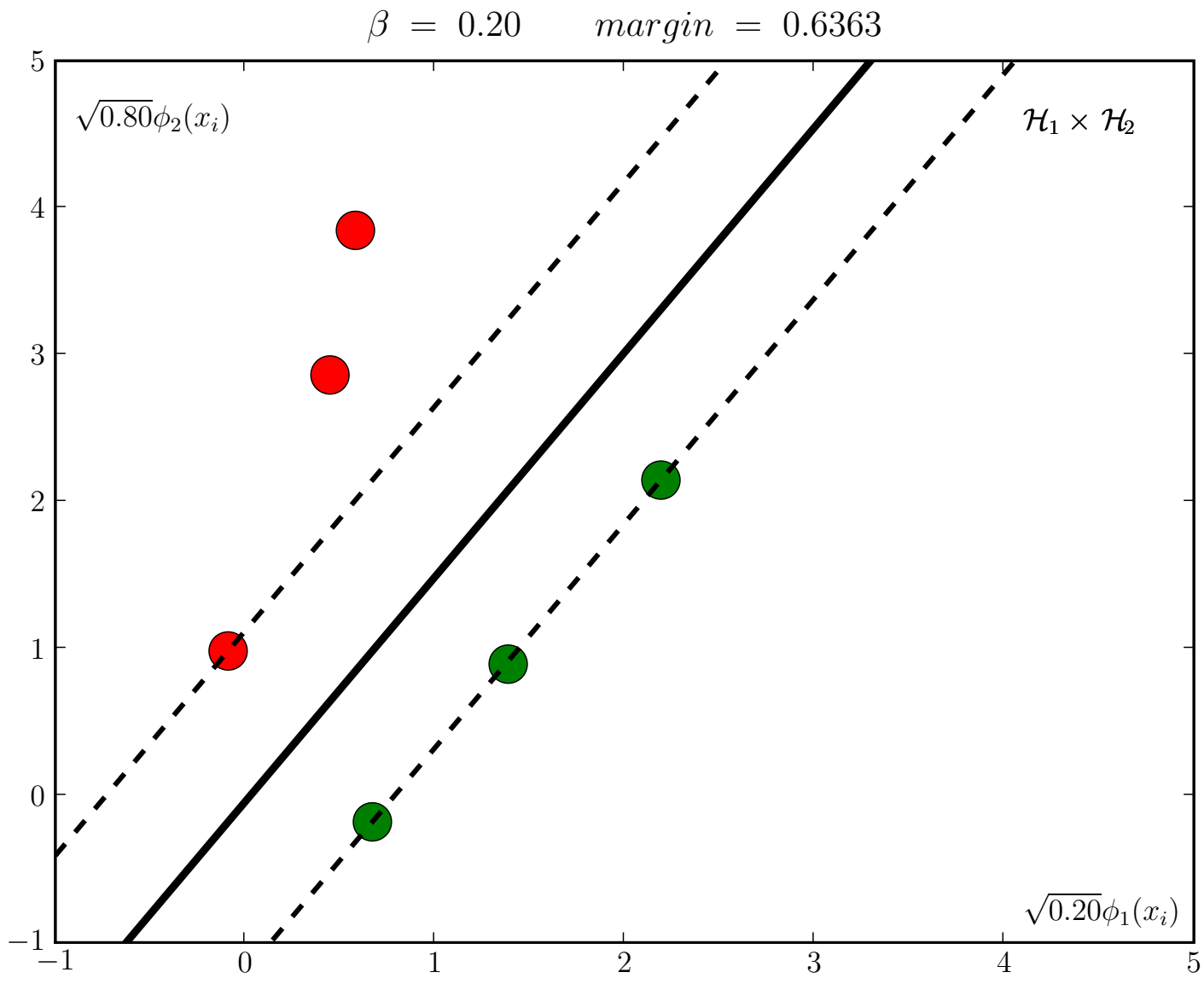
Feature Space View of Kernel Combination



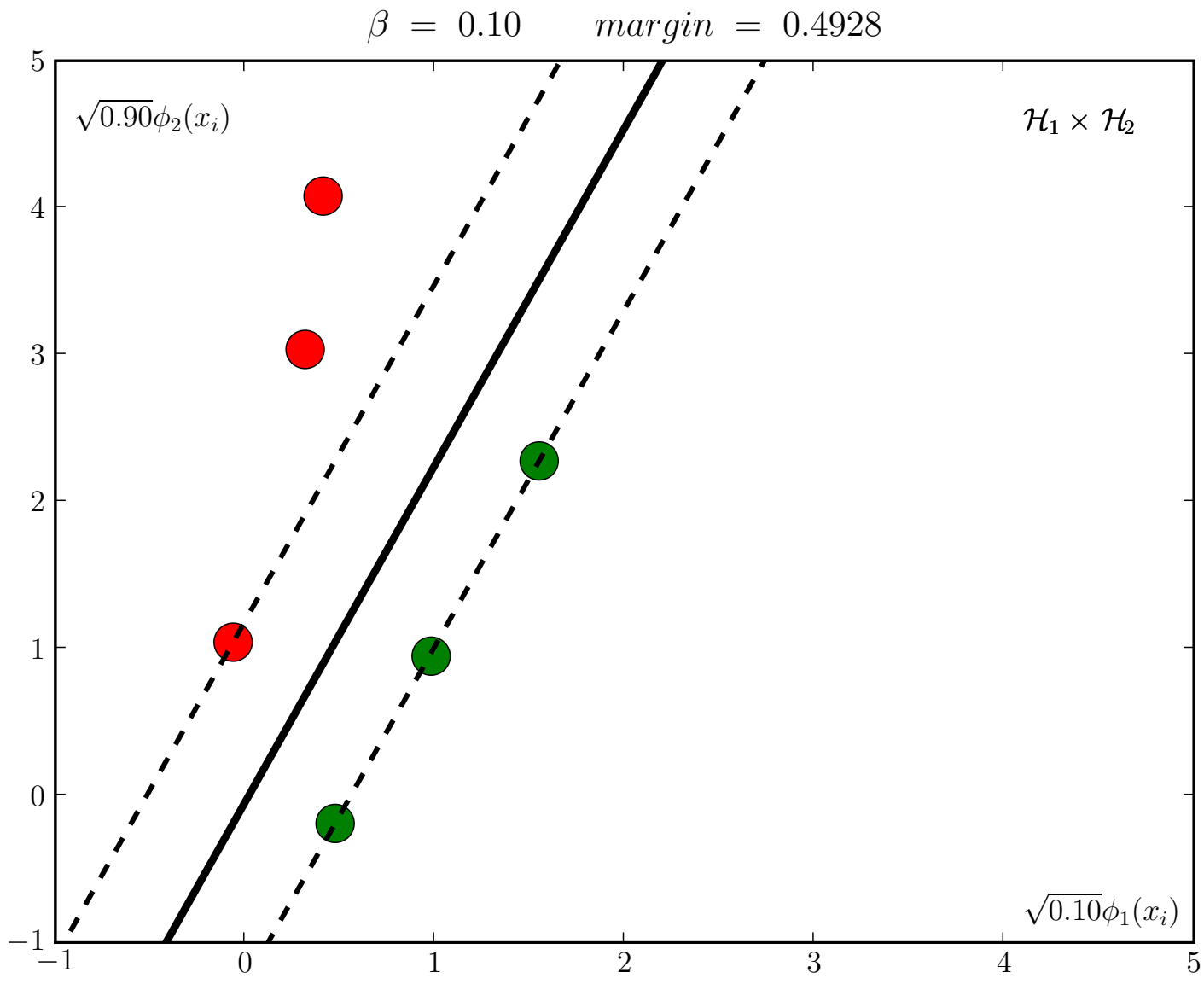
Feature Space View of Kernel Combination



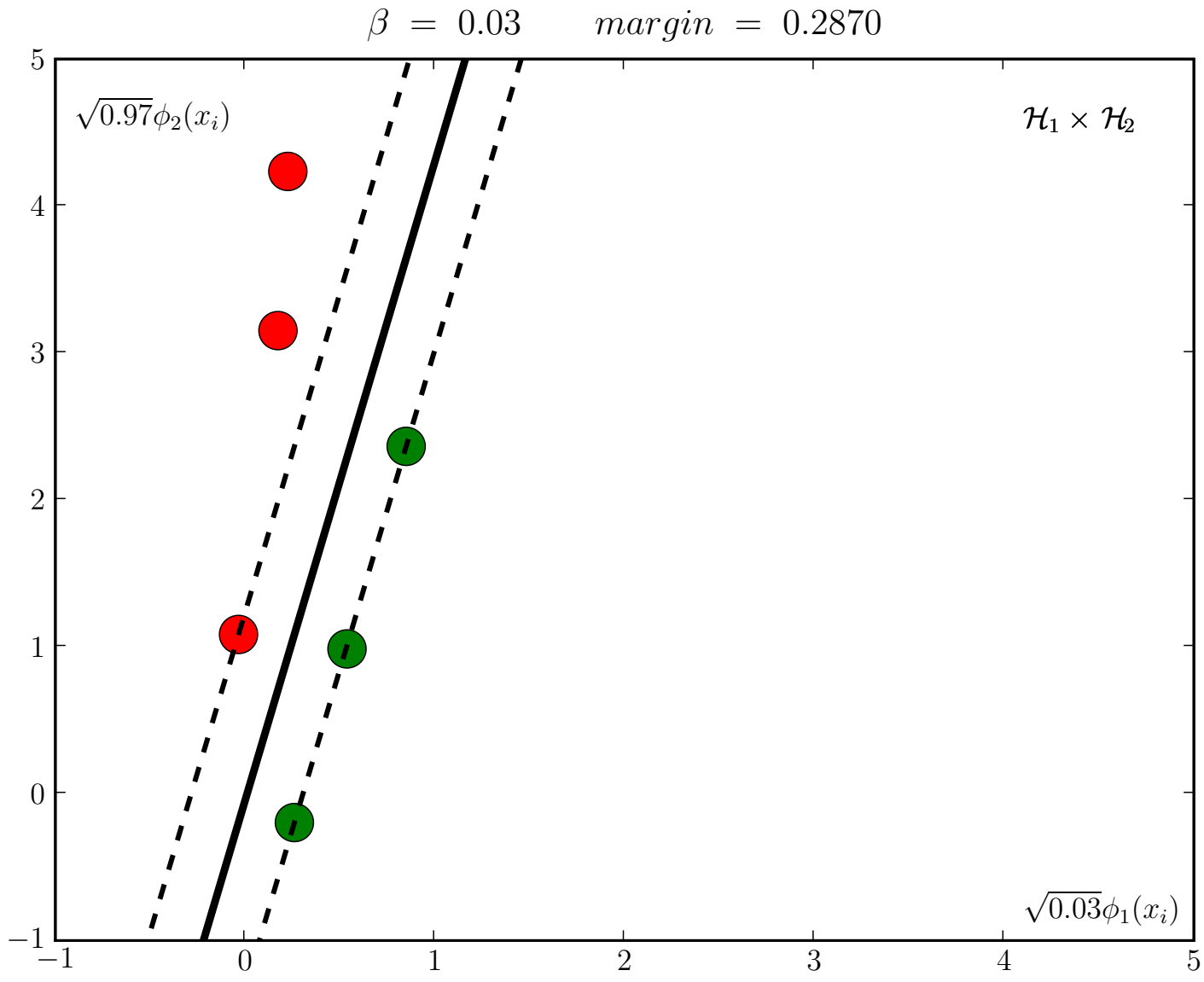
Feature Space View of Kernel Combination



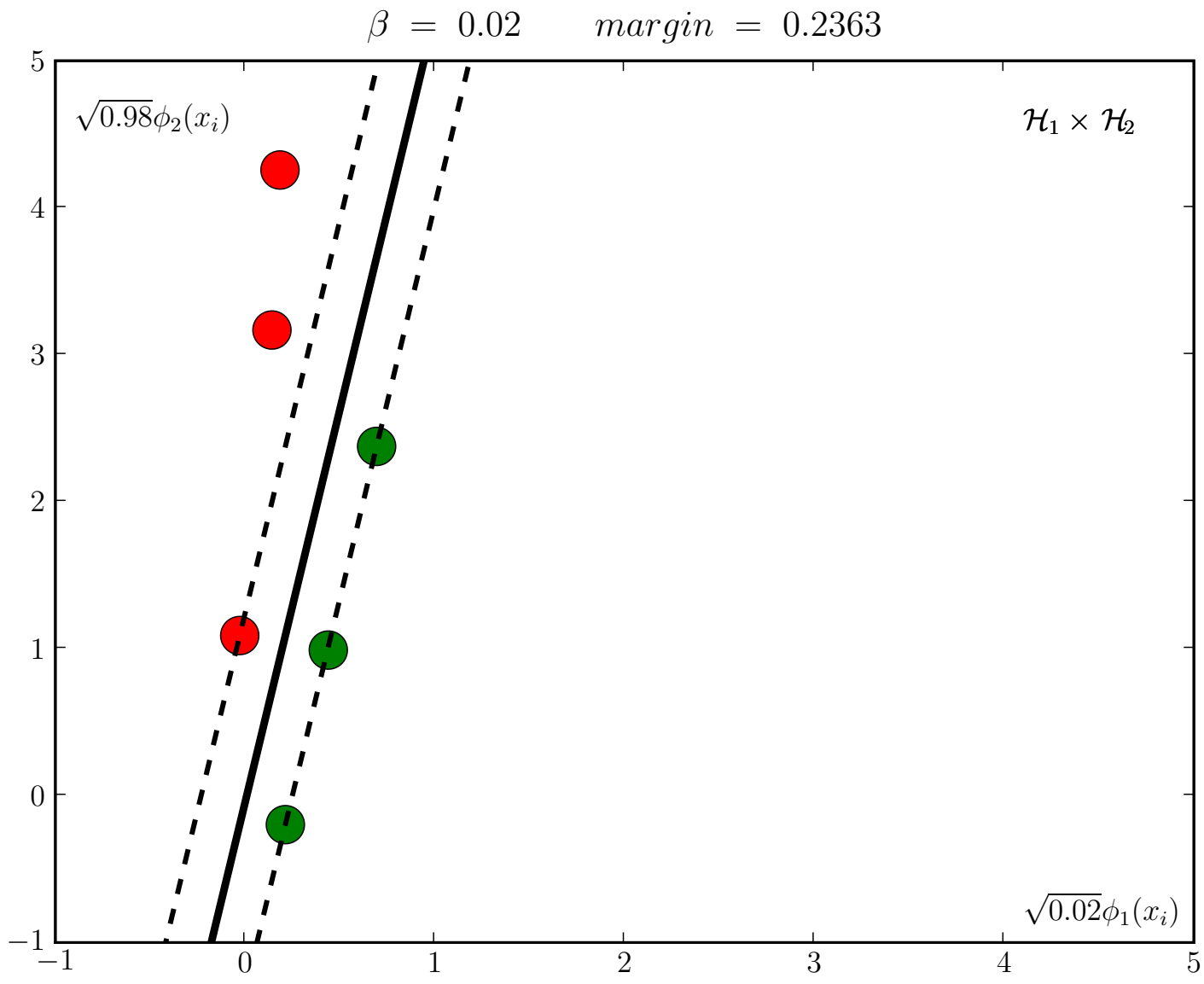
Feature Space View of Kernel Combination



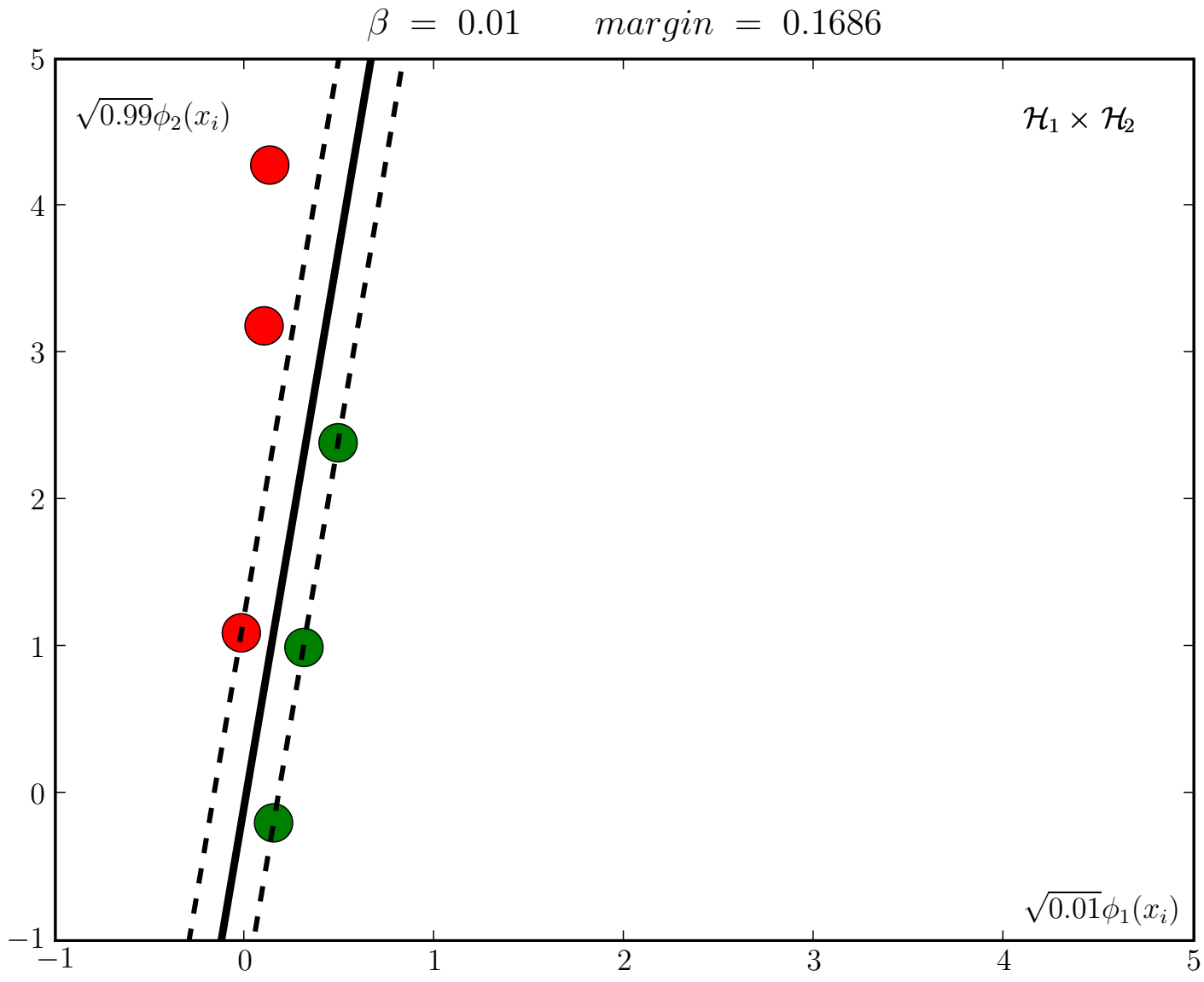
Feature Space View of Kernel Combination



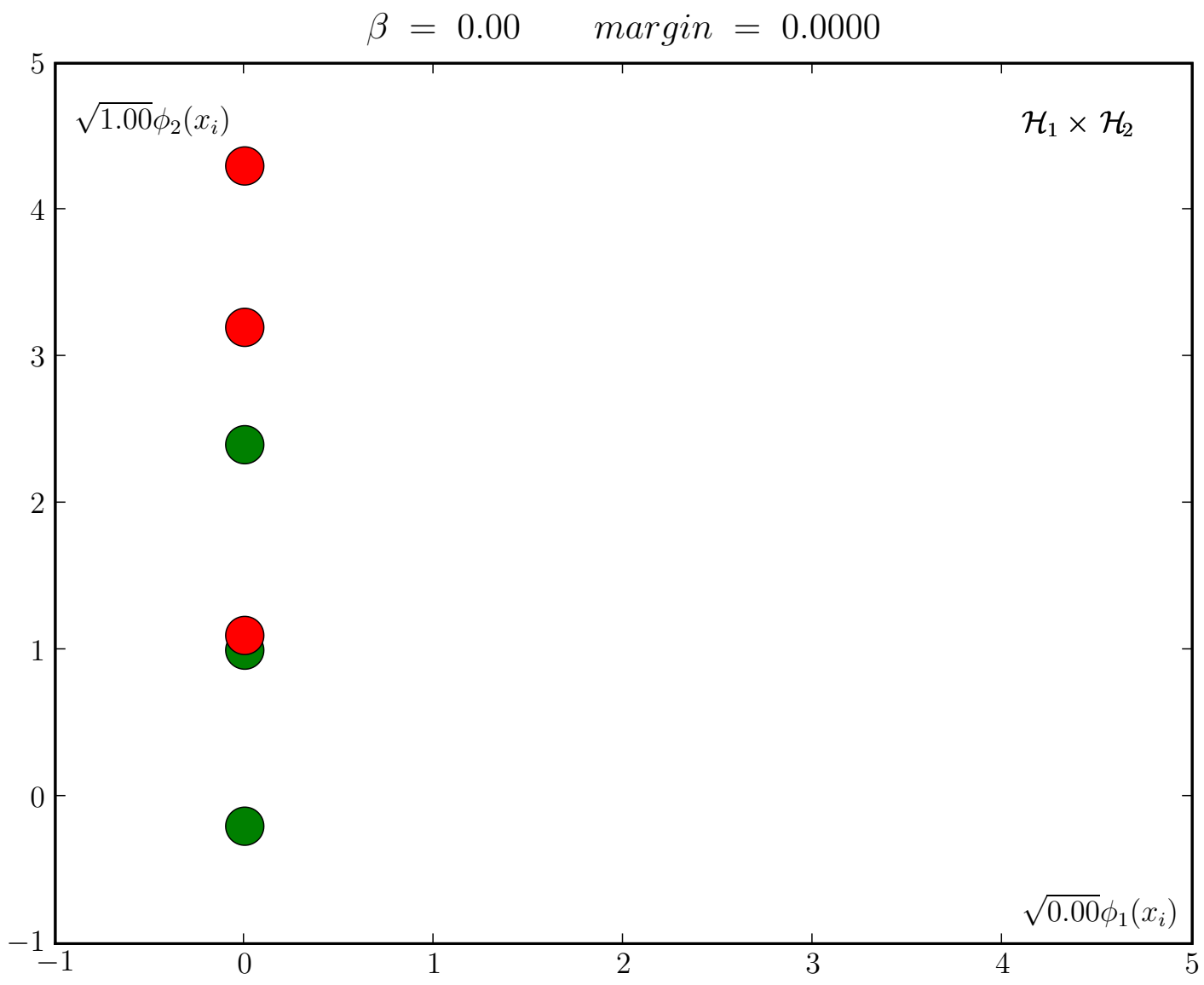
Feature Space View of Kernel Combination



Feature Space View of Kernel Combination



Feature Space View of Kernel Combination



Multiple Kernel Learning

Can we *calculate* coefficients β_j that realize the *largest margin*?

- Analyze: how does the margin depend on β_j?
- Remember standard SVM (here without slack variables):

$$\min_{w \in \mathcal{H}} \| w \|_{\mathcal{H}}^2$$

subject to

$$y_i \langle w, x_i \rangle_{\mathcal{H}} \geq 1 \qquad \text{for } i = 1, \ldots n.$$

- $\mathcal{H}$ and φ were induced by kernel k.
- New samples are classified by $f(x) = \langle w, x \rangle_{\mathcal{H}}$.

Multiple Kernel Learning

- Insert

$$k(x, x') = \sum_{j=1}^{K} \beta_j k_j(x, x') \tag{2}$$

with

- Hilbert space $\mathcal{H} = \oplus_j \mathcal{H}_j$,
- feature map $\varphi(x) = (\sqrt{\beta_1}\varphi_1(x), \ldots, \sqrt{\beta_K}\varphi_K(x))^t$,
- weight vector $w = (w_1, \ldots, w_K)^t$.

such that

$$\|w\|_{\mathcal{H}}^2 = \sum_j \|w_j\|_{\mathcal{H}_j}^2 \tag{3}$$

$$\langle w, \varphi(x_i)\rangle_{\mathcal{H}} = \sum_j \sqrt{\beta_j}\langle w_j, \varphi_j(x_i)\rangle_{\mathcal{H}_j} \tag{4}$$

Multiple Kernel Learning

- For fixed β_j, the largest margin hyperplane is given by

$$\min_{w_j \in \mathcal{H}_j} \sum_j \|w_j\|^2_{\mathcal{H}_j}$$

subject to

$$y_i \sum_j \sqrt{\beta_j} \langle w_j, \varphi_j(x_i) \rangle_{\mathcal{H}_j} \geq 1 \qquad \text{for } i = 1, \dots n.$$

- Renaming $v_j = \sqrt{\beta_j} w_j$ (and defining $\frac{0}{0} = 0$):

$$\min_{v_j \in \mathcal{H}_j} \sum_j \frac{1}{\beta_j} \|v_j\|^2_{\mathcal{H}_j}$$

subject to

$$y_i \sum_j \langle v_j, \varphi_j(x_i) \rangle_{\mathcal{H}_j} \geq 1 \qquad \text{for } i = 1, \dots n.$$

Multiple Kernel Learning

- Therefore, best hyperplane for variable β_j is given by:

$$\min_{\substack{v_j \in \mathcal{H}_j \\ \sum_j \beta_j = 1 \\ \beta_j \geq 0}} \sum_j \frac{1}{\beta_j} \|v_j\|^2_{\mathcal{H}_j} \tag{5}$$

  subject to

$$y_i \sum_j \langle v_j, \varphi_j(x_i) \rangle_{\mathcal{H}_j} \geq 1 \qquad \text{for } i = 1, \ldots n. \tag{6}$$

- This optimization problem is *jointly-convex* in v_j and β_j.
- There is a unique global minimum, and we can find it efficiently!

Multiple Kernel Learning

- Same for *soft-margin* with slack-variables:

$$\min_{\substack{v_j \in \mathcal{H}_j \\ \sum_j \beta_j = 1 \\ \beta_j \geq 0 \\ \xi_i \in \mathbb{R}^+}} \sum_j \frac{1}{\beta_j} \| v_j \|^2_{\mathcal{H}_j} + C \sum_i \xi_i \tag{7}$$

subject to

$$y_i \sum_j \langle v_j, \varphi_j(x_i) \rangle_{\mathcal{H}_j} \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n. \tag{8}$$

- This optimization problem is *jointly-convex* in v_j and β_j.
- There is a unique global minimum, and we can find it efficiently!

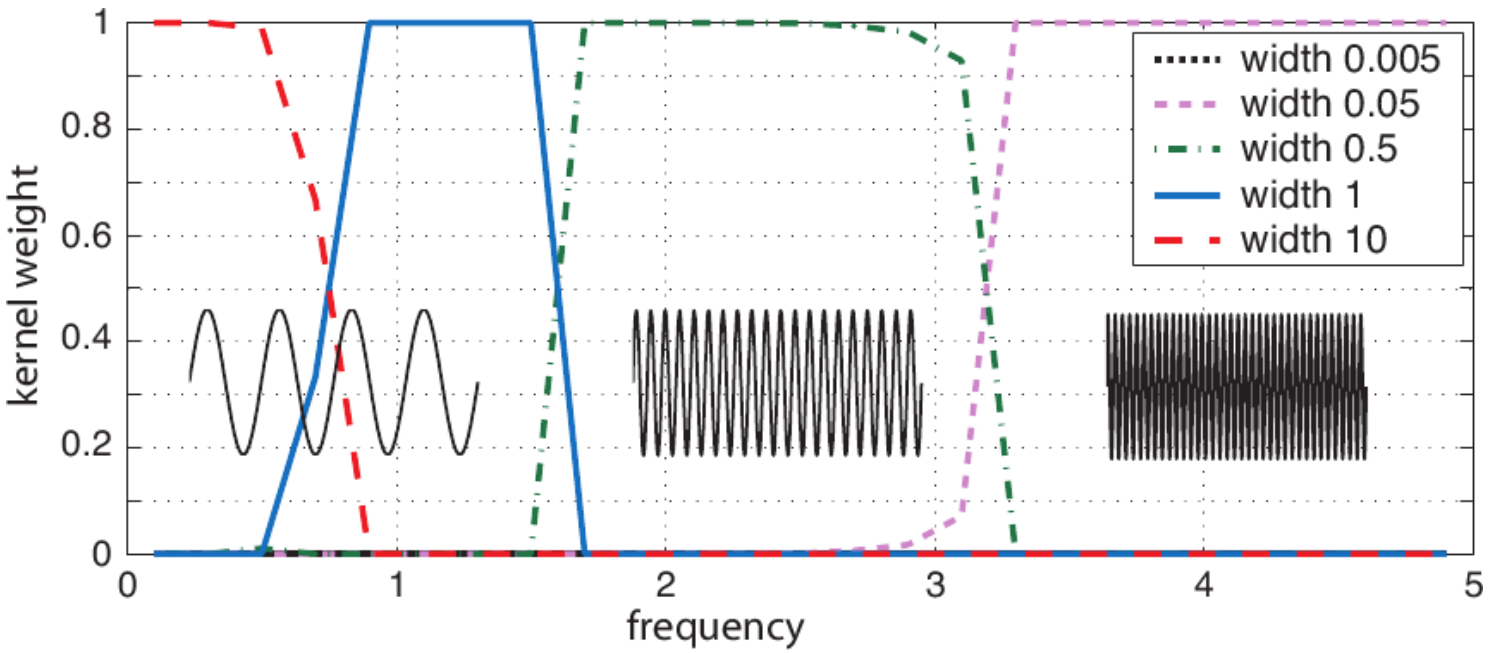
Software for Multiple Kernel Learning

- Existing toolboxes allow Multiple-Kernel SVM training:
  - Shogun (C++ with bindings to Matlab, Python etc.)
    `http://www.fml.tuebingen.mpg.de/raetsch/projects/shogun`
  - MPI IKL (Matlab with libSVM, CoinIPopt)
    `http://www.kyb.mpg.de/bs/people/pgehler/ikl-webpage/index.html`
  - SimpleMKL (Matlab)
    `http://asi.insa-rouen.fr/enseignants/~arakotom/code/mklindex.html`
  - SKMsmo (Matlab)
    `http://www.di.ens.fr/~fbach/` (older and slower than the others)
- Typically, one only has to specify the set of candidate kernels and the regularization parameter C.

MKL Toy Example

Support-vector regression to learn samples of $f(t) = \sin(\omega t)$

$$k_j(x, x') = \exp\left(\frac{\|x - x'\|^2}{2\sigma_j^2}\right) \text{ with } 2\sigma_j^2 \in \{0.005, 0.05, 0.5, 1, 10\}.$$



- Multiple-Kernel Learning correctly identifies the right bandwidth.

Combining Good Kernels

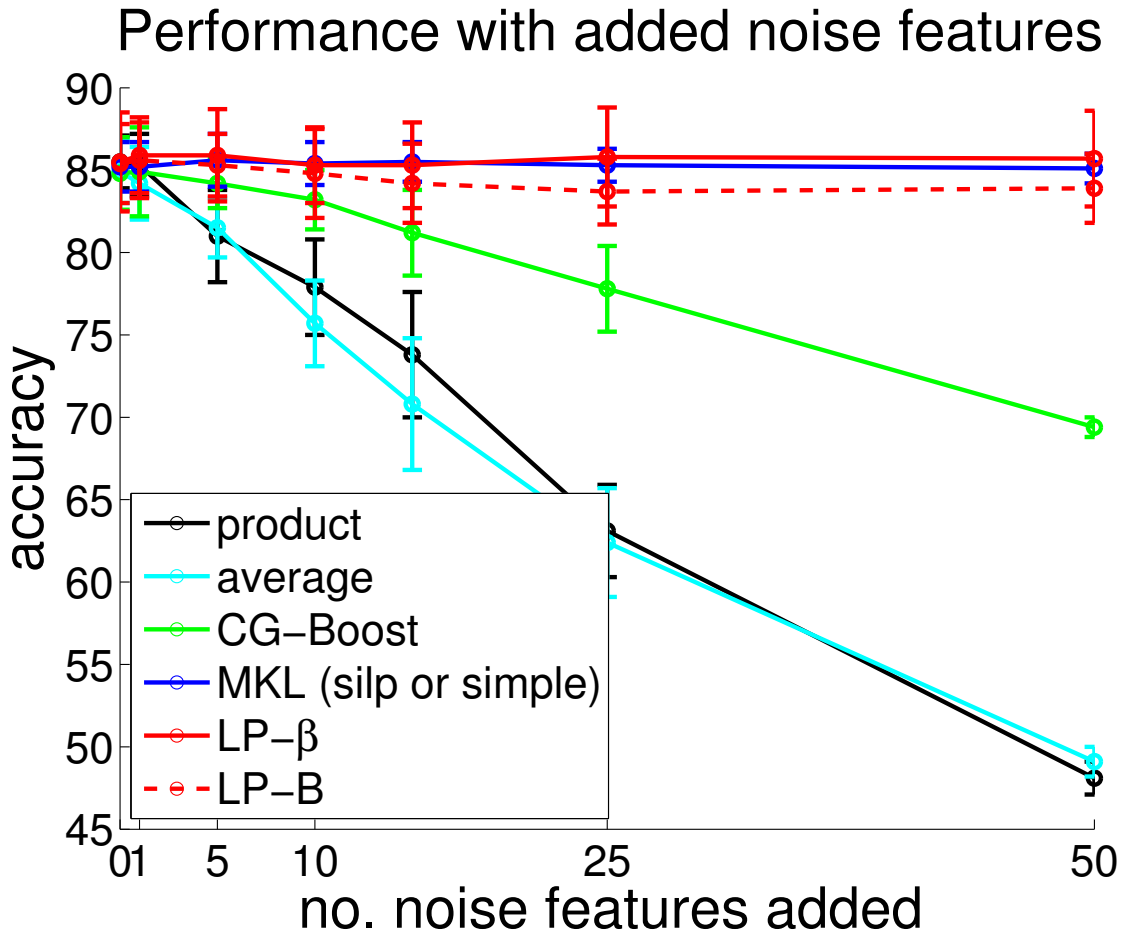
Observation: if all kernels are reasonable, simple combination methods work as well as difficult ones (and are much faster):

Single features			Combination methods		
Method	Accuracy	Time	Method	Accuracy	Time
Colour	60.9 ± 2.1	3	product	85.5 ± 1.2	2
Shape	70.2 ± 1.3	4	averaging	84.9 ± 1.9	10
Texture	63.7 ± 2.7	3	CG-Boost	84.8 ± 2.2	1225
HOG	58.5 ± 4.5	4	MKL (SILP)	85.2 ± 1.5	97
HSV	61.3 ± 0.7	3	MKL (Simple)	85.2 ± 1.5	152
siftint	70.6 ± 1.6	4	LP-β	85.5 ± 3.0	80
siftbdy	59.4 ± 3.3	5	LP-B	85.4 ± 2.4	98

Mean accuracy and total runtime (model selection, training, testing) on Oxford Flowers dataset [Gehler, Nowozin: ICCV2009]

Combining Good and Bad kernels

Observation: if some kernels are helpful, but others are not, smart techniques are better.

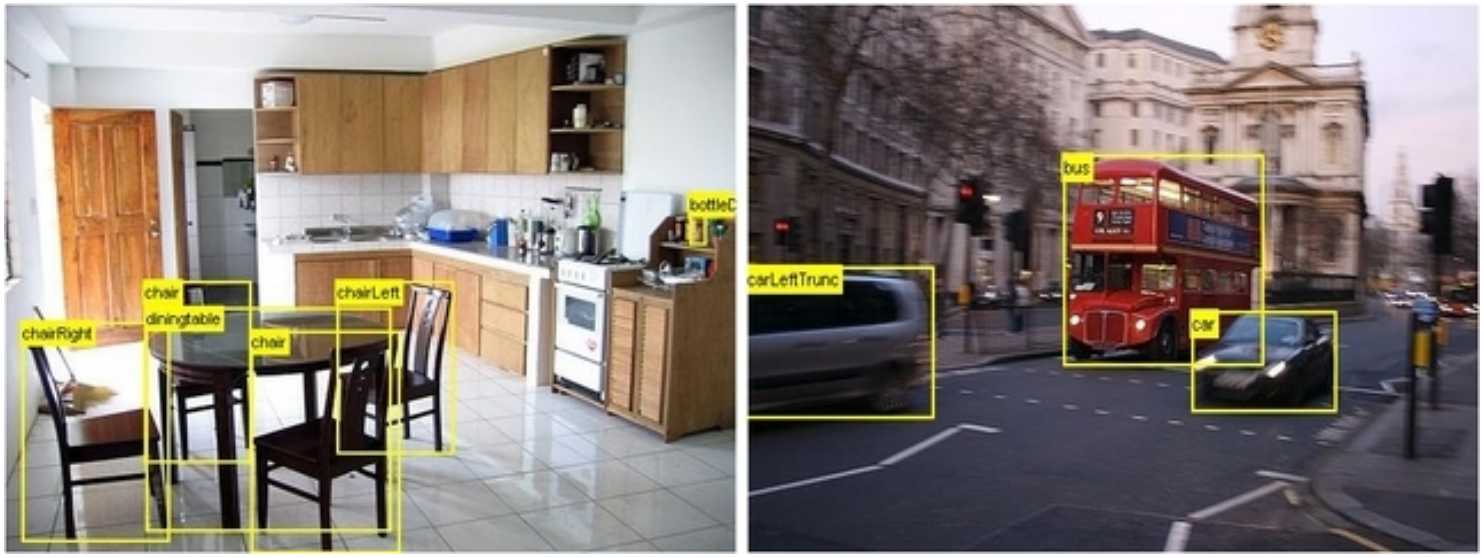


Mean accuracy on Oxford Flowers dataset [Gehler, Nowozin: ICCV2009]

Example: Multi-Class Object Localization

MKL for joint prediction of different object classes.

- Objects in images do not occur independently of each other.

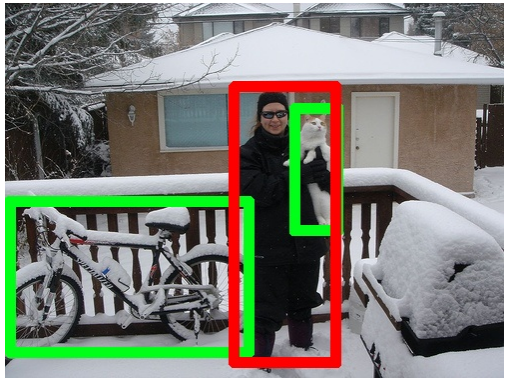


- *Chairs* and tables often occur together in indoor scenes.
- *Busses* often occur together with *cars* in street scenes.
- *Chairs* rarely occur together with *cars*.

One can make use of these dependencies to improve prediction.

Example: Multi-Class Object Localization

- Predict candidate regions for all object classes.
- Train a decision function for each class (red), taking into account candidate regions *for all classes* (red and green).
- Decide per-class which other object categories are worth using



$$k(I, I') = \beta_0 k_\chi^2(h, h') + \sum_{j=1}^{20} \beta_j k_\chi^2(h_j, h'_j)$$

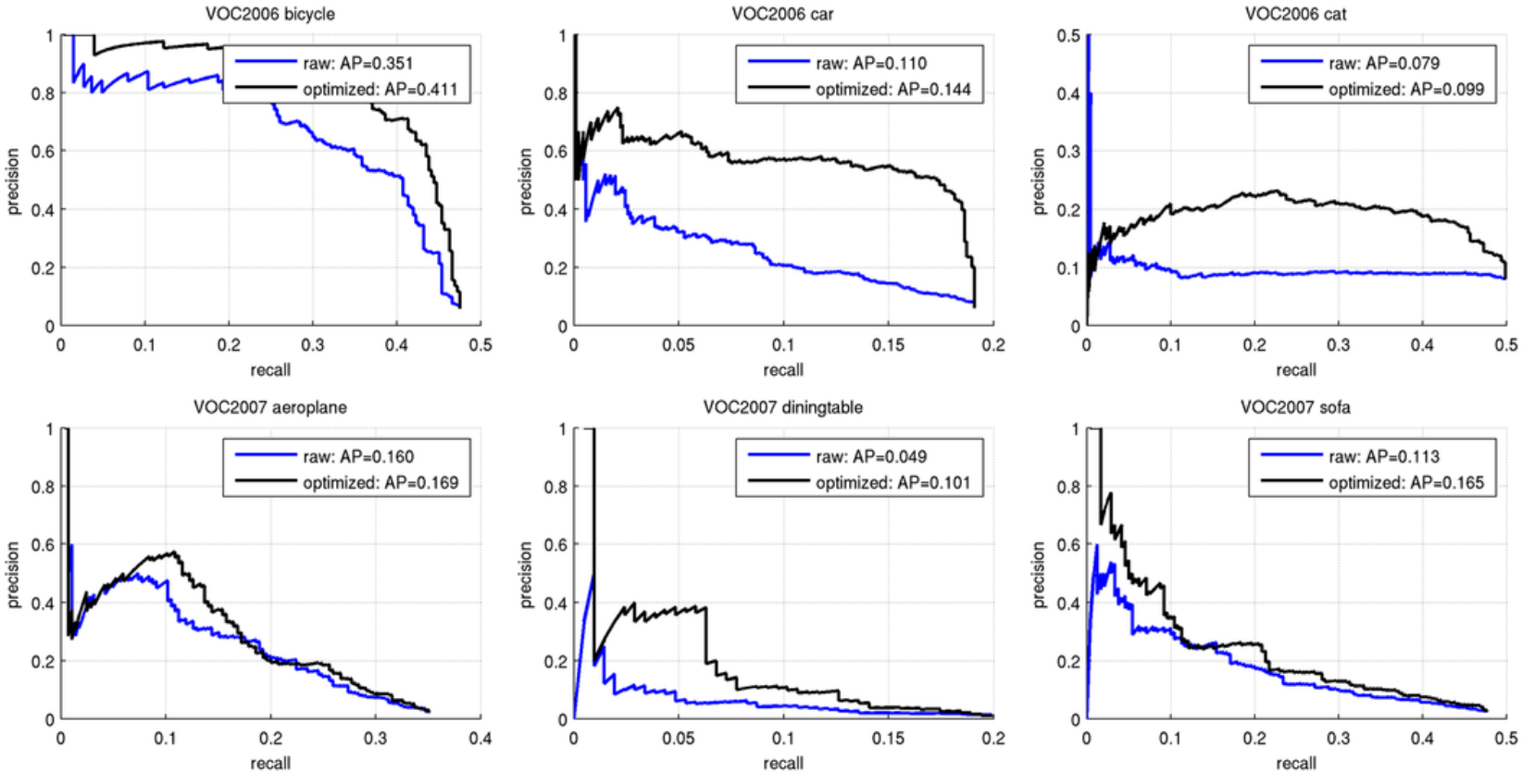
  - h: feature histogram for the full image x
  - h_j: histogram for the region predicted for object class j in x

- Use MKL to learn weights β_j, $j = 0, \ldots, 20$.

[Lampert and Blaschko, DAGM 2008]

Example: Multi-Class Object Localization

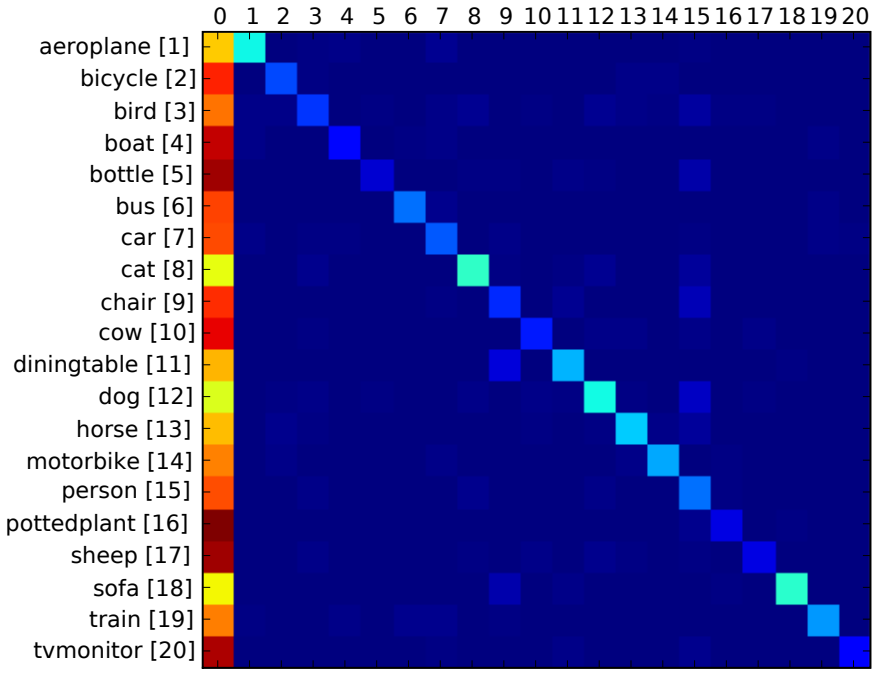
- Benchmark on PASCAL VOC 2006 and VOC 2007.
- Combination improves detection accuracy (black vs. blue).



Example: Multi-Class Object Localization

Interpretation of Weights (VOC 2007):

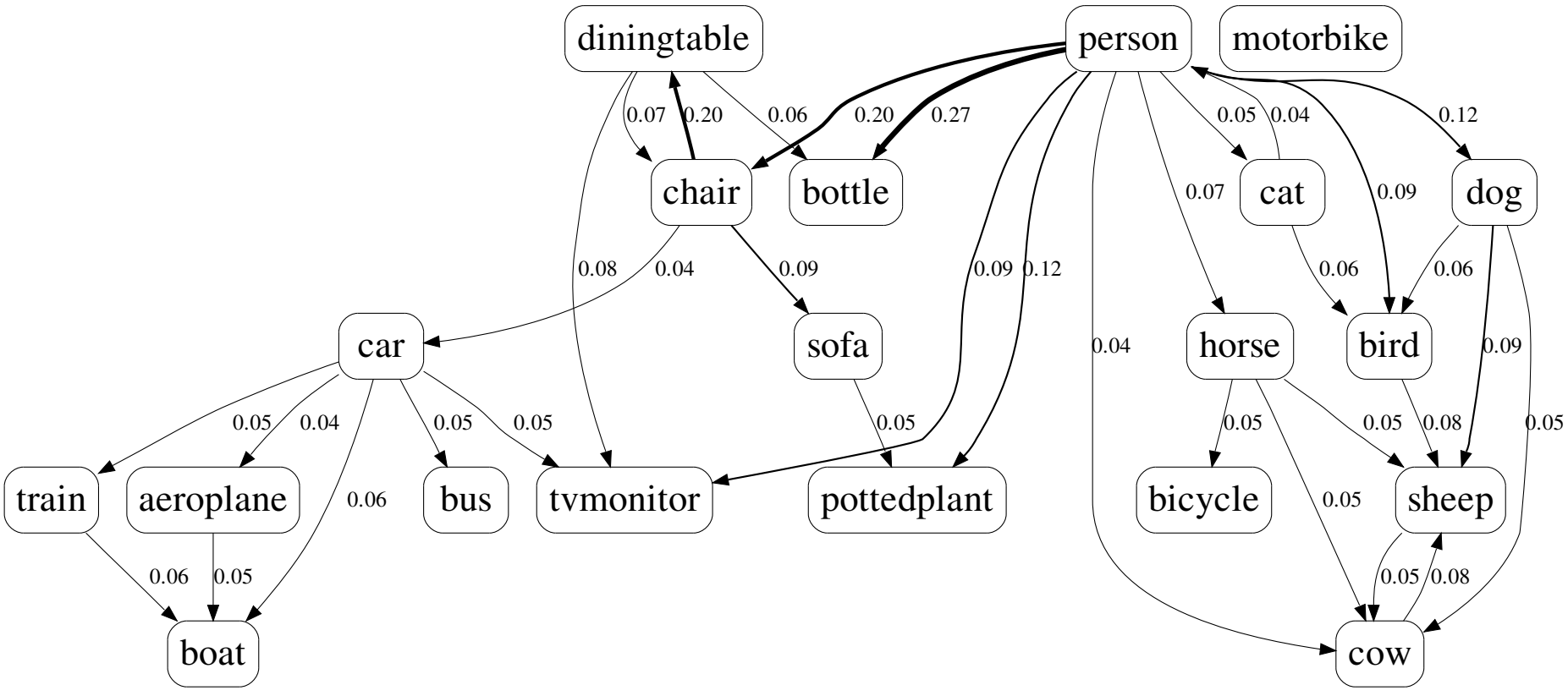
- Every class decision depends on the *full image* and on the *object box*.
- High *image* weights:
  $\longrightarrow$ *scene* classification?
- Intuitive connections:
  chair $\longrightarrow$ *diningtable*,
  person $\longrightarrow$ *bottle*,
  person $\longrightarrow$ *dog*.
- Many classes depend on the *person* class.



rows: class to be detected
columns: class candidate boxes

Example: Multi-Class Object Localization

We can turn the non-zero *weights* into a *dependency graph*:



- Threshold relative weights (without image component) at 0.04
- $i \longrightarrow j$ means *"Class i is used to predict class j."*
- Interpretable clusters: *vehicles*, *indoor*, *animals*.

Summary

Kernel Selection and Combination

- Model selection is important to achive highest accuracy
- Combining several kernels is often superior to selecting one

Multiple-Kernel Learning

- Learn weights for the "best" linear kernel combination:
  - unified approach to feature selection/combination.
    visit [Gehler, Nowozin. CVPR 2009] on Wednesday afternoon
- Beware: MKL is **no silver bullet**.
  - Other and even simpler techniques might be superior!
  - Always compare against *single best*, *averaging*, *product*.

Warning: Caltech101/256

- Be careful when reading kernel combination results
  - Many results reported rely on "broken" Bosch kernel matrices