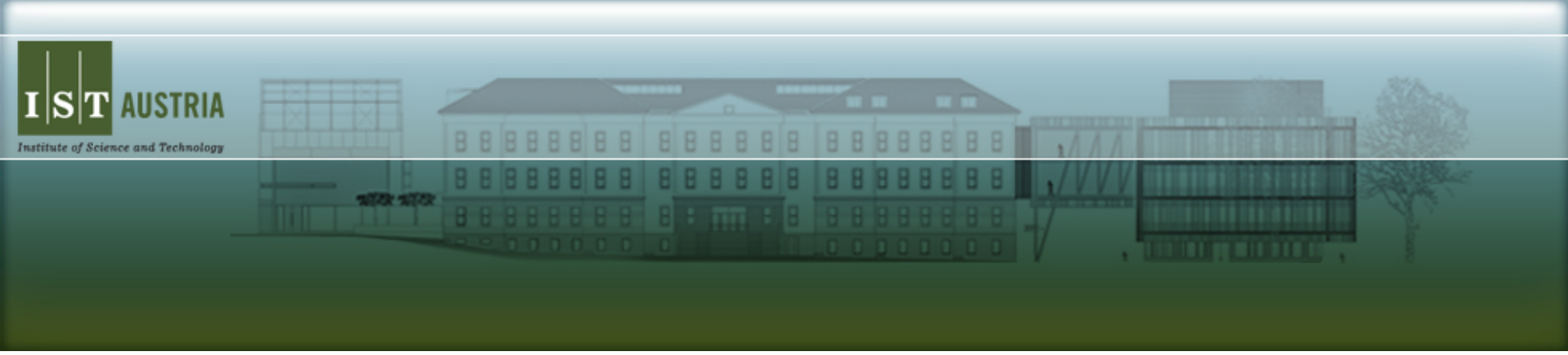


Kernel Methods in Computer Vision

Christoph Lampert
Institute of Science and Technology Austria
Vienna (Klosterneuburg), Austria

Computer Vision and Sports Summer School, Prague, 2012

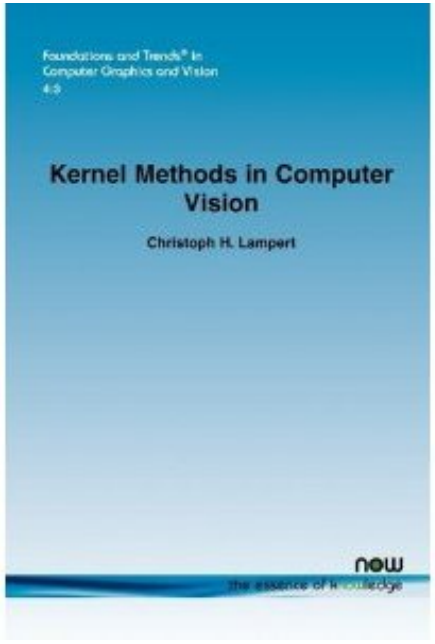
Kernel Methods in Computer Vision

Overview...

- 9:00 – 10:00 Introduction to Kernel Classifiers
- 10:00 – 10:10 — break —
- 10:10 – 11:10 Kernels in Computer Visions
- 11:10 – 11:20 — break —
- 11:20 – 12:00 More Kernel Methods
- 12:00 – 13:15 — Lunch —
- 13:15 – 14:15 Attribute-Based Classification

Slides available on my home page:
`http://www.ist.ac.at/~chl`

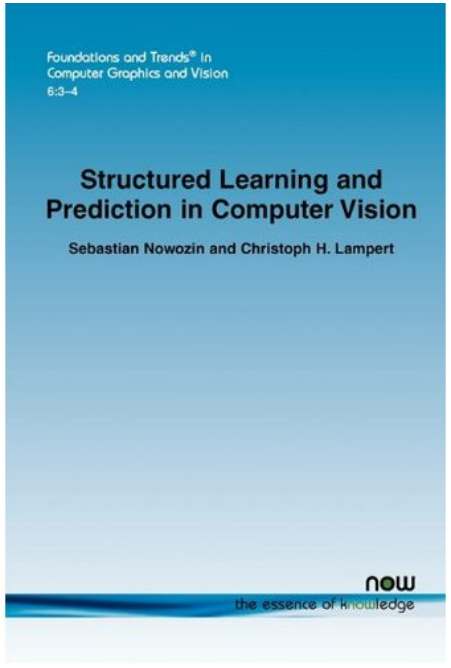
Extended versions of the lectures in book form



Foundations and Trends in Computer Graphics and Vision,

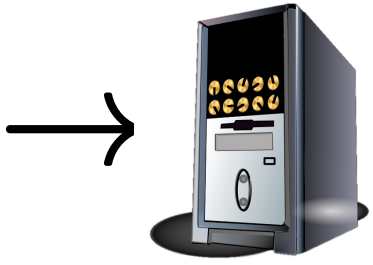
now publisher
`www.nowpublishers.com/`

Available as PDFs on my homepage



Computer Vision: Long term goal

Automatic systems that analyzes and interprets visual data

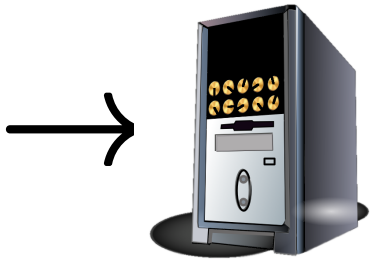


"Three men sit at a table in a pub, drinking beer. One of them talks while the other listen."

Image Understanding

Computer Vision: Short/medium term goal

Automatic systems that analyzes some aspects of visual data

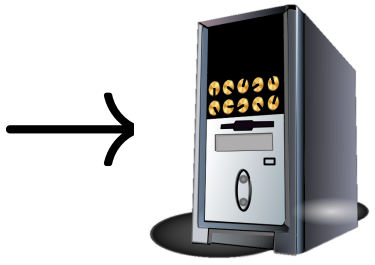


- indoors
- in a pub

Scene Classification

Computer Vision: Short/medium term goal

Automatic systems that analyzes some aspects of visual data

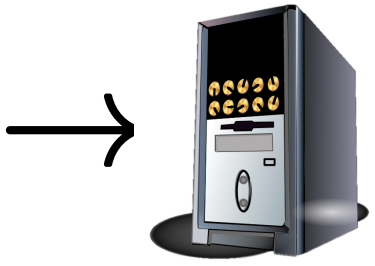


- drinking

Action Classification

Computer Vision: Short/medium term goal

Automatic systems that analyzes some aspects of visual data

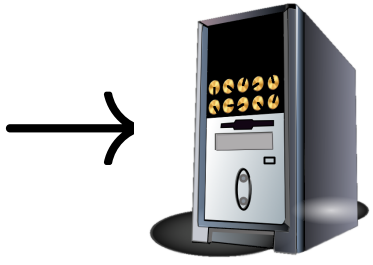


- three people
- one table
- three glasses

Object Recognition

Computer Vision: Short/medium term goal

Automatic systems that analyzes some aspects of visual data



Joint positions/
angles: $\theta_1, \ldots, \theta_K$

Pose Estimation

A Machine Learning View on Computer Vision Problems

Classification
- **Scene Classification**
- **Action Classification**
- **Object Recognition**
- Face Detection
- Sign Language Recognition

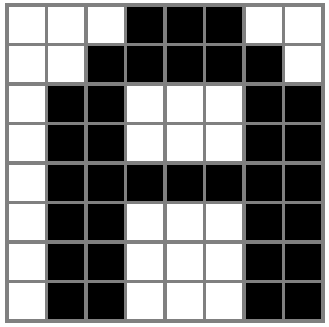
Regression
- **Pose Estimation**
- Stereo Reconstruction
- Image Denoising

Outlier Detection
- Anomaly Detection in Videos
- Video Summarization

Clustering
- Image Segmentation
- Image Duplicate Detection

A Machine Learning View on Computer Vision Problems

Classification
- ...
- Optical Character Recognition
- ...

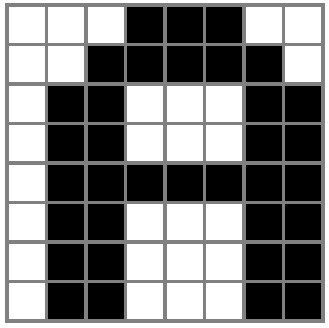


- It's difficult to *program* a solution to this.

```
if (I[0,5]<128) & (I[0,6] > 192) & (I[0,7] < 128):
  return 'A'
elif (I[7,7]<50) & (I[6,3]) != 0:
  return 'Q'
else:
  print "I don't know this letter."
```

A Machine Learning View on Computer Vision Problems

Classification
- ...
- Optical Character Recognition
- ...



- It's difficult to *program* a solution to this.

```
if (I[0,5]<128) & (I[0,6] > 192) & (I[0,7] < 128):
  return 'A'
elif (I[7,7]<50) & (I[6,3]) != 0:
  return 'Q'
else:
  print "I don't know this letter."
```

- With Machine Learning, we can avoid this:
  - We don't program a solution to the specific problem.
  - We program a generic *classification* program.
  - We solve the problem by *training* the classifier with examples.

A Machine Learning View on Computer Vision Problems

Classification
- ...
- Object Category Recognition
- ...

- It's ~~difficult~~ impossible to *program* a solution to this.

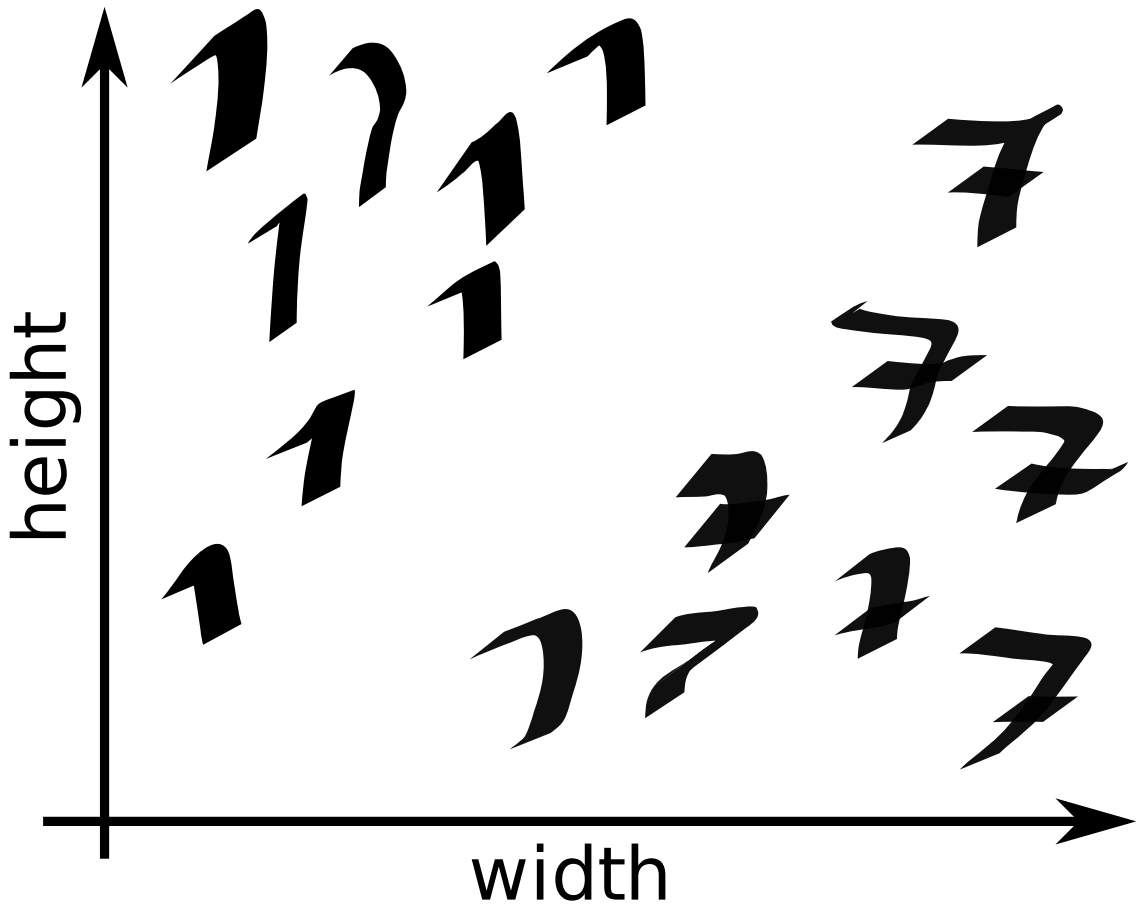
```
if ???
```

- With Machine Learning, we can avoid this:
  - We don't program a solution to the specific problem.
  - We re-use our previous classifier.
  - We solve the problem by training the classifier with examples.

Introduction to Kernel Classifiers

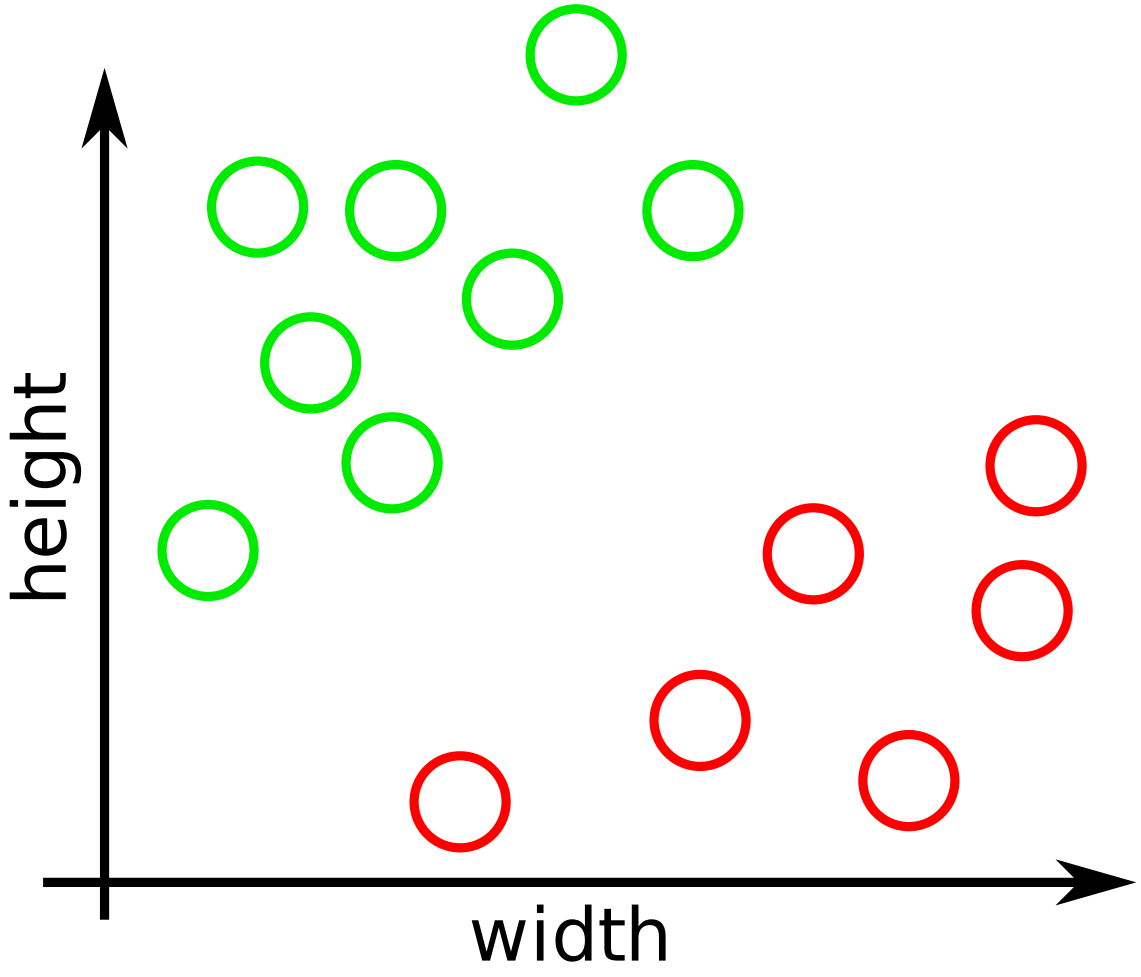
Linear Classification

Toy Problem: distinguish between **1**s and **7**s.



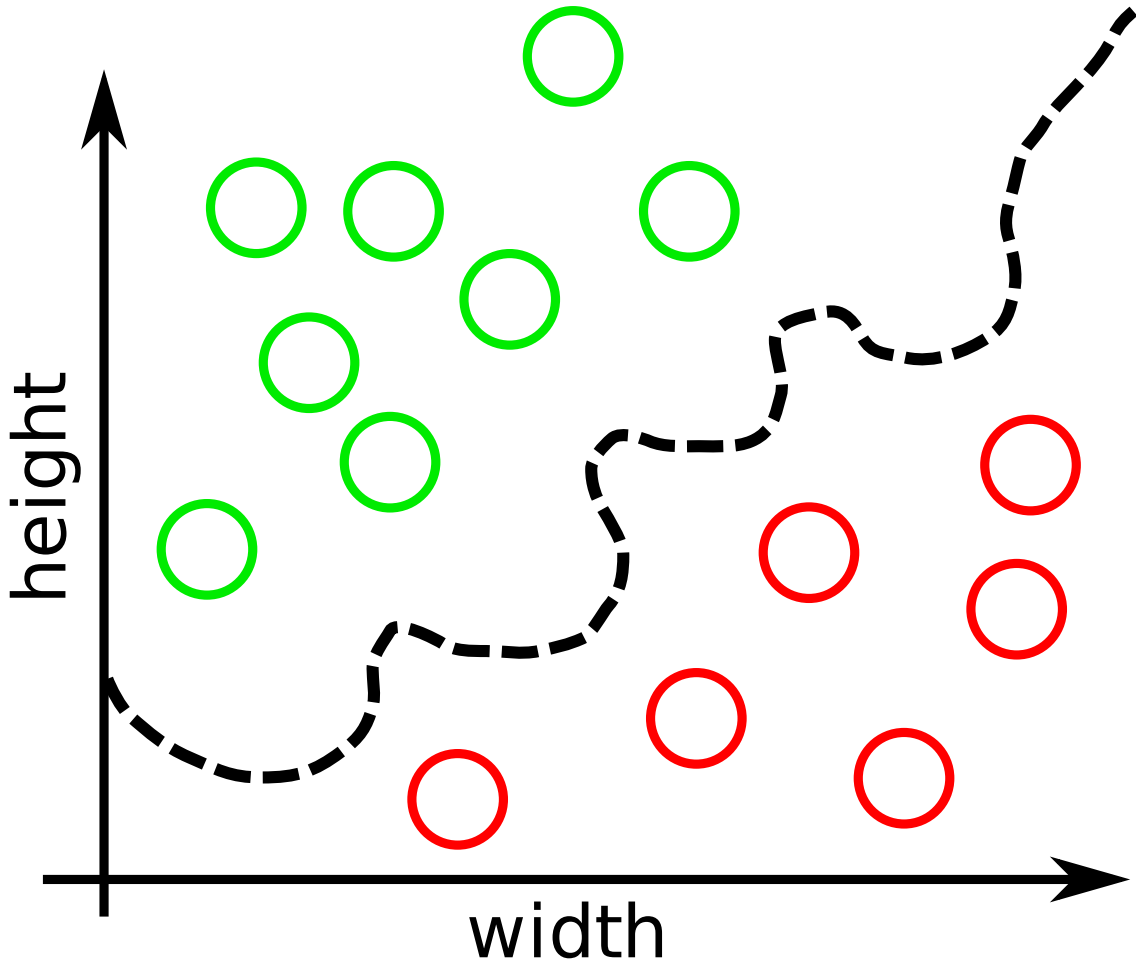
Linear Classification

Formalization: separate these two sample sets.

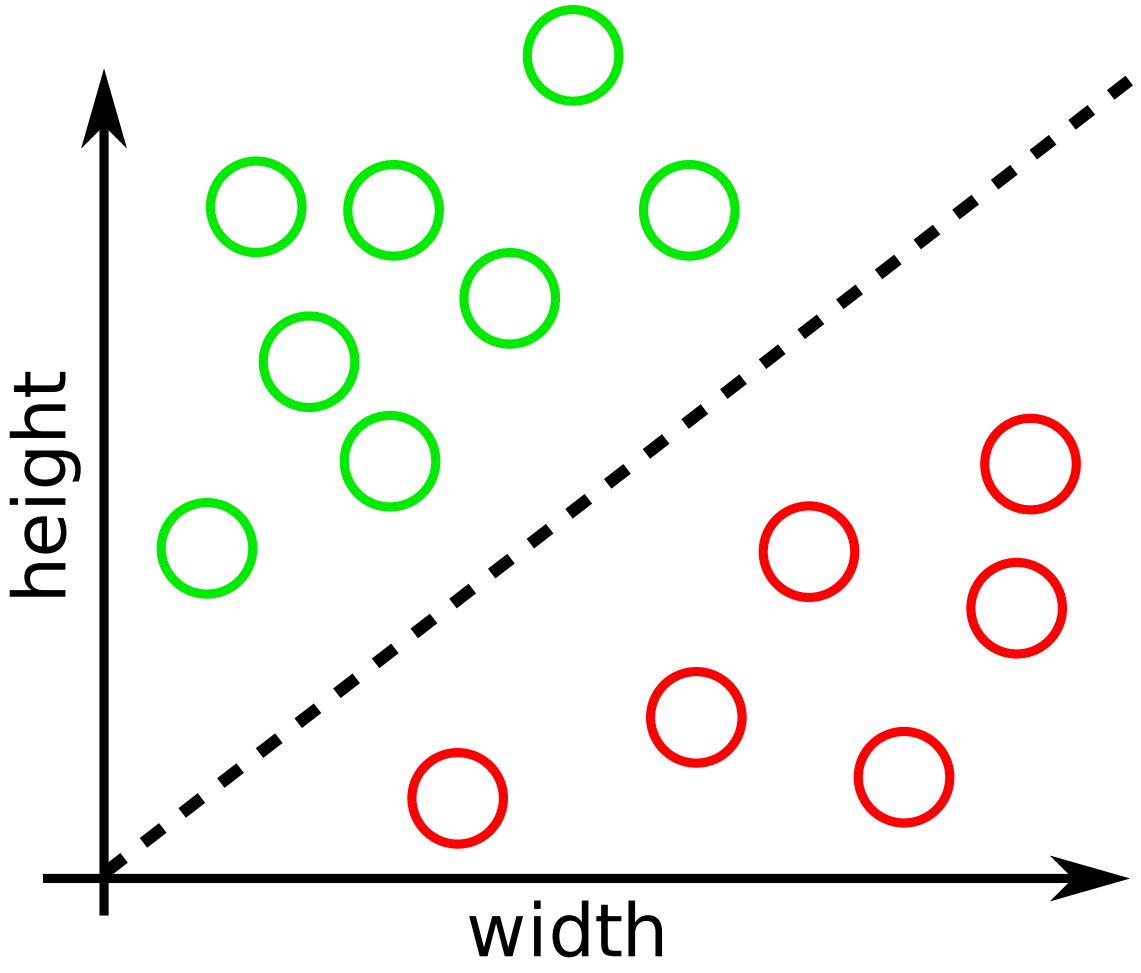


Linear Classification

Formalization: separate these two sample sets.



Formalization: separate these two sample sets.



Linear Classification

Why linear?

Linear techniques have been studied for centuries.

- they are fast and easy to solve
  - elementary maths, even closed form solutions
  - typically involve only matrix operation
- they are intuitive
  - simple is good,
  - solution can be derived geometrically,
  - solution corresponds to common sense.
- they often work very well,
  - many physical processes are linear,
  - almost all natural functions are smooth,
  - smooth function can be approximated, at least locally, by linear functions.

Why linear?

Linear techniques have been studied for centuries.

- they are fast and easy to solve
  - elementary maths, even closed form solutions
  - typically involve only matrix operation
- they are intuitive
  - simple is good,
  - solution can be derived geometrically,
  - solution corresponds to common sense.
- they often work very well,
  - many physical processes are linear,
  - almost all natural functions are smooth,
  - smooth function can be approximated, at least locally, by linear functions.

The whole lecture will be about linear methods (with a twist).

Notation...

- data points $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$, (images)
- class labels $Y = \{y_1, \ldots, y_n\}$, $y_i \in \{+1, -1\}$, (**cat** or **no cat**)
- goal: classification rule $g : \mathbb{R}^d \to \{-1, +1\}$.

Notation...

- data points $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$, (images)
- class labels $Y = \{y_1, \ldots, y_n\}$, $y_i \in \{+1, -1\}$, (**cat** or **no cat**)
- goal: classification rule $g : \mathbb{R}^d \to \{-1, +1\}$.

- parameterize $\boldsymbol{g(x) = \operatorname{sign} f(x)}$ with $f : \mathbb{R}^d \to \mathbb{R}$:

$$f(x) = a^1 x^1 + a^2 x^2 + \ldots a^n x^n + a^0$$

simplify notation: $\hat{x} = (1, x)$, $\hat{w} = (a^0, \ldots, a^n)$:

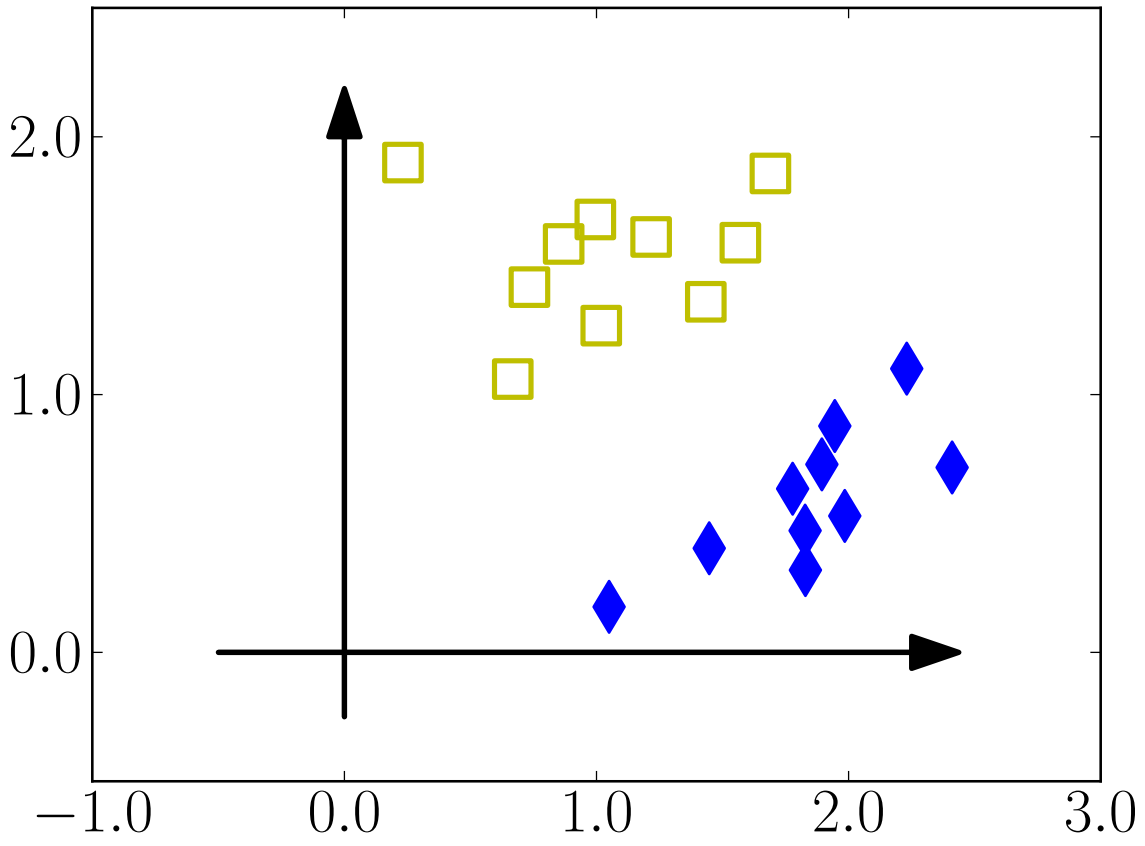
$$f(x) = \langle \hat{w}, \hat{x} \rangle$$ (inner/scalar product in $\mathbb{R}^{d+1}$)

(also: $\hat{w} \cdot \hat{x}$ or $\hat{w}^t \hat{x}$)

- out of lazyness, we just write $\boldsymbol{f(x) = \langle w, x \rangle}$ with $x, w \in \mathbb{R}^d$.

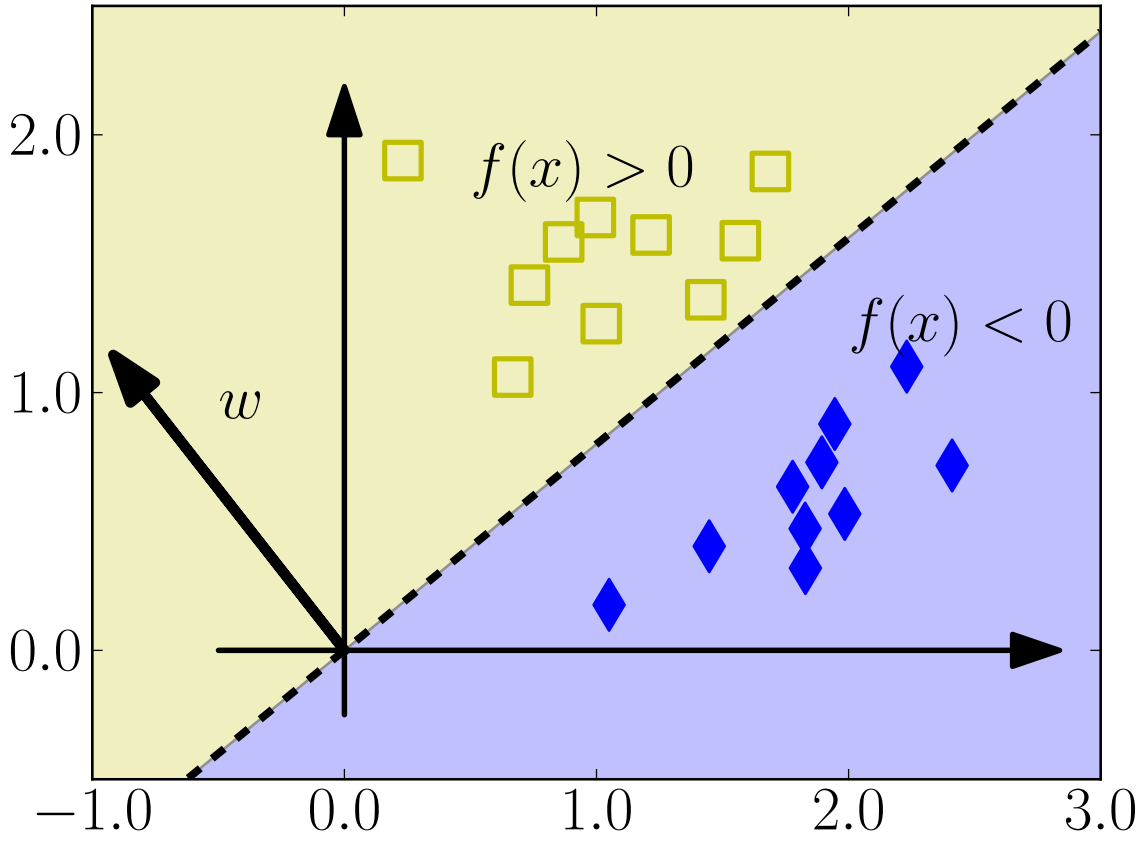
Example: Linear Classification

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



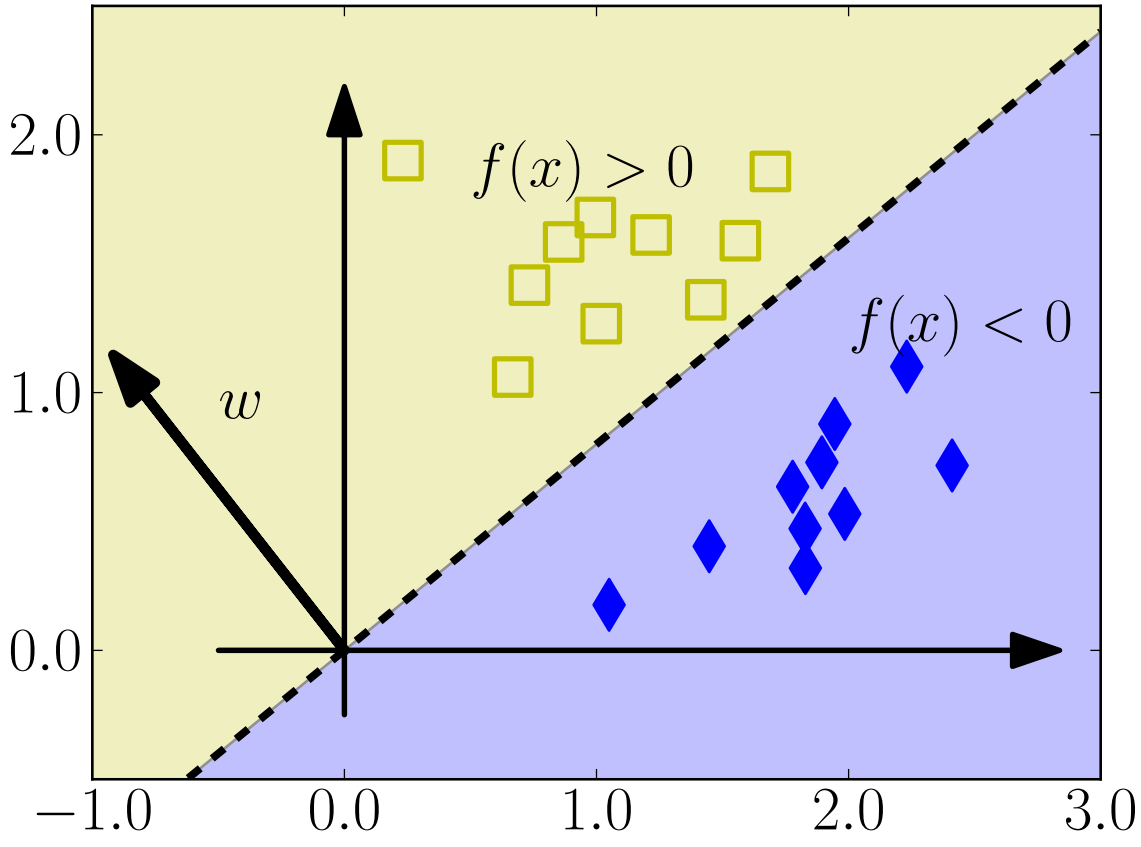
Example: Linear Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Any w partitions the data space into two half-spaces by means of $f(x) = \langle w, x \rangle$.



Example: Linear Classification

Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Any w partitions the data space into two half-spaces by means of $f(x) = \langle w, x \rangle$.



"How to find w?"

Example: Ad Hoc Linear Classification

Because everything is linear, we only need Linear Algebra!

- $\boldsymbol{X} = \Big(x_1, \ldots, x_n\Big) \in \mathbb{R}^{d \times n}$, $\boldsymbol{y} = \Big(y_1, \ldots, y_n\Big) \in \mathbb{R}^n$,
- remember $\langle w, x \rangle = w^t x$

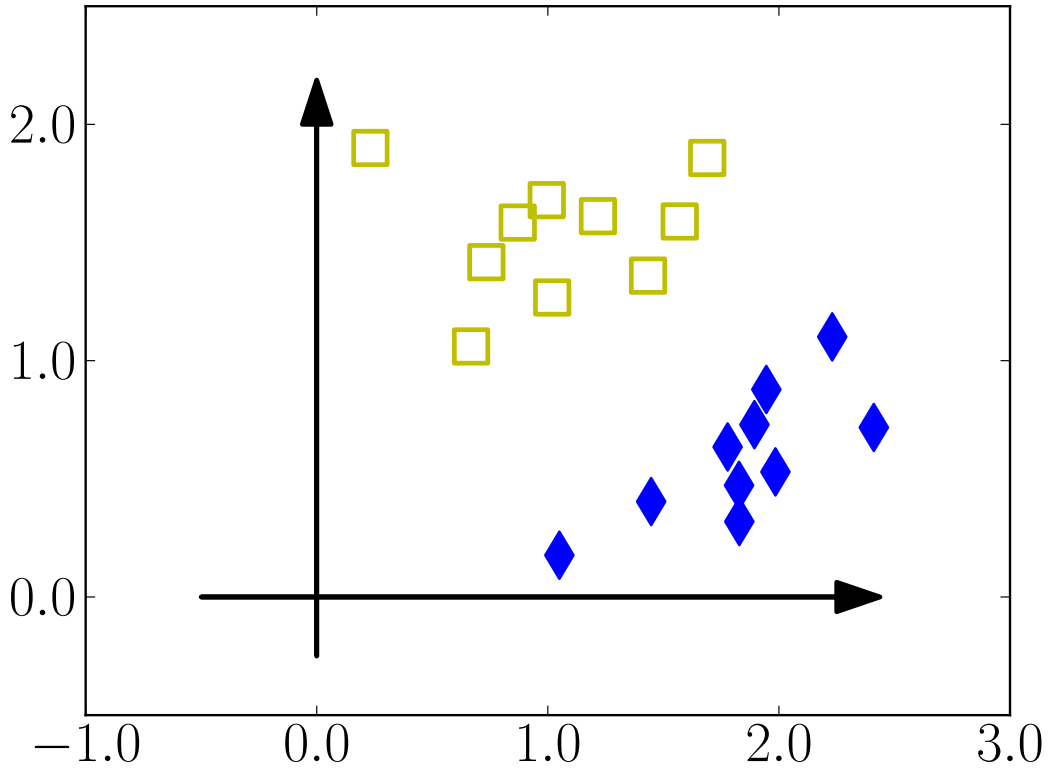
$$\begin{aligned}\Big(f(x_1), \ldots, f(x_n)\Big) &= \Big(\langle w, x_1\rangle, \ldots, \langle w, x_n\rangle\Big) \\ &= \Big(w^t x_1, \ldots, w^t x_n\Big) = w^t \Big(x_1, \ldots, x_n\Big) = w^t \boldsymbol{X}\end{aligned}$$

- We want $f(x_i) > 0$ for $y_i = +1$ and $f(x_i) < 0$ for $y_i = -1$. Let's just try $f(x_i) = y_i$ and solve

$$\begin{aligned} & w^t \boldsymbol{X} && = \boldsymbol{y} \\ \Rightarrow \quad & w^t \boldsymbol{X}\boldsymbol{X}^t && = \boldsymbol{y}\boldsymbol{X}^t \\ \Rightarrow \quad & w^t && = \underbrace{\boldsymbol{y}\boldsymbol{X}^t}_{1\times d}\underbrace{(\boldsymbol{X}\boldsymbol{X}^t)^{-1}}_{d\times d}\end{aligned}$$

Does it work?

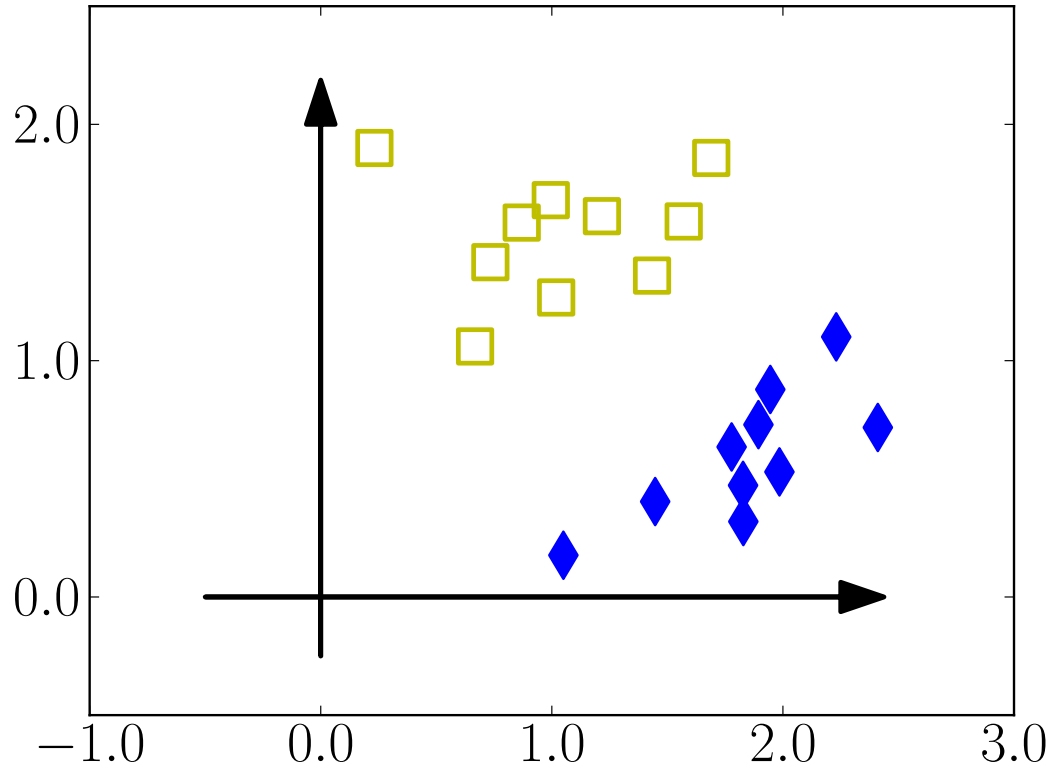
$$\boldsymbol{X}^t = \begin{pmatrix} 1.8 & 0.3 \\ 2.0 & 0.5 \\ 1.8 & 0.5 \\ 1.8 & 0.6 \\ 1.9 & 0.7 \\ 2.4 & 0.7 \\ 1.9 & 0.9 \\ 1.4 & 0.4 \\ 2.6 & 0.2 \\ 2.2 & 1.1 \\ 1.5 & 1.5 \\ 0.7 & 1.4 \\ 1.2 & 1.6 \\ 1.0 & 1.7 \\ 0.7 & 1.1 \\ 1.0 & 1.3 \\ 0.2 & 1.2 \\ 1.7 & 1.9 \\ 0.9 & 1.6 \\ 1.6 & 1.6 \end{pmatrix} \quad \boldsymbol{y}^t = \begin{pmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$



$$w^t = \boldsymbol{y}\boldsymbol{X}^t(\boldsymbol{X}\boldsymbol{X}^t)^{-1}$$

Does it work?

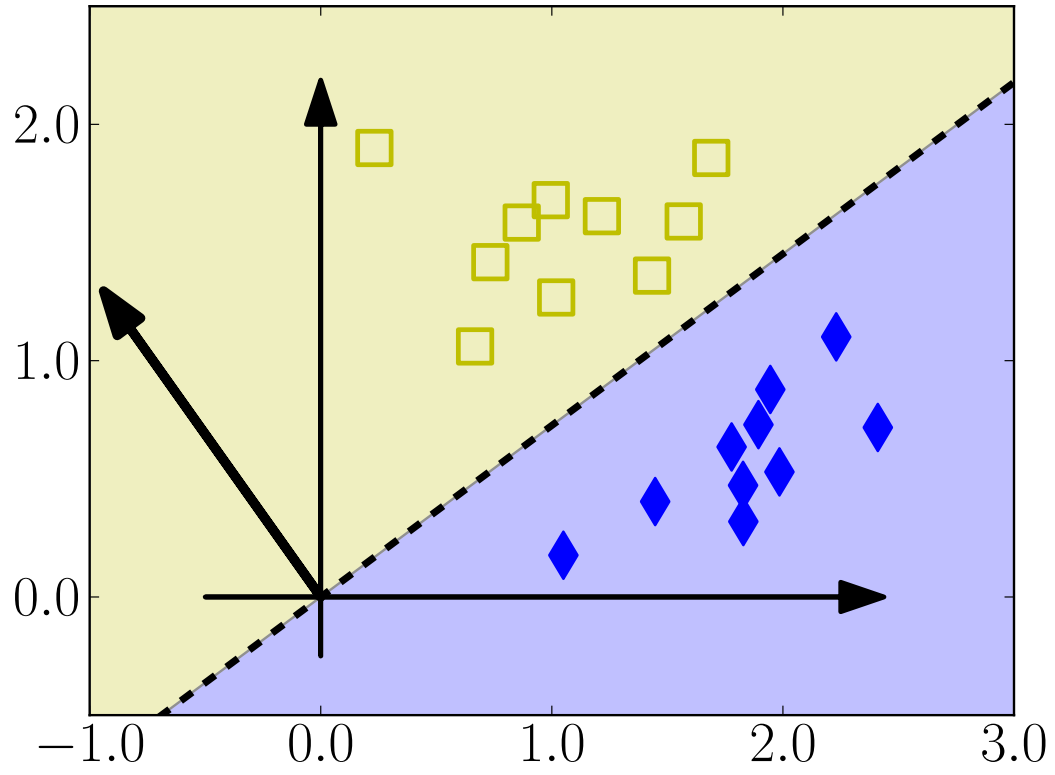
$$\boldsymbol{X}^t = \begin{pmatrix} 1.8 & 0.3 \\ 2.0 & 0.5 \\ 1.8 & 0.5 \\ 1.8 & 0.6 \\ 1.9 & 0.7 \\ 2.4 & 0.7 \\ 1.9 & 0.9 \\ 1.4 & 0.4 \\ 2.6 & 0.2 \\ 2.2 & 1.1 \\ 1.5 & 1.5 \\ 0.7 & 1.4 \\ 1.2 & 1.6 \\ 1.0 & 1.7 \\ 0.7 & 1.1 \\ 1.0 & 1.3 \\ 0.2 & 1.2 \\ 1.7 & 1.9 \\ 0.9 & 1.6 \\ 1.6 & 1.6 \end{pmatrix} \quad \boldsymbol{y}^t = \begin{pmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$



$$w^t = \boldsymbol{y}\boldsymbol{X}^t(\boldsymbol{X}\boldsymbol{X}^t)^{-1} = \begin{pmatrix} -7.97 \\ 9.37 \end{pmatrix} \begin{pmatrix} 47.82 & 27.75 \\ 27.75 & 28.33 \end{pmatrix}^{-1} = \begin{pmatrix} -0.83 \\ 1.14 \end{pmatrix}$$

Does it work?

$$\boldsymbol{X}^t = \begin{pmatrix} 1.8 & 0.3 \\ 2.0 & 0.5 \\ 1.8 & 0.5 \\ 1.8 & 0.6 \\ 1.9 & 0.7 \\ 2.4 & 0.7 \\ 1.9 & 0.9 \\ 1.4 & 0.4 \\ 2.6 & 0.2 \\ 2.2 & 1.1 \\ 1.5 & 1.5 \\ 0.7 & 1.4 \\ 1.2 & 1.6 \\ 1.0 & 1.7 \\ 0.7 & 1.1 \\ 1.0 & 1.3 \\ 0.2 & 1.2 \\ 1.7 & 1.9 \\ 0.9 & 1.6 \\ 1.6 & 1.6 \end{pmatrix} \quad \boldsymbol{y}^t = \begin{pmatrix} +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ +1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix}$$



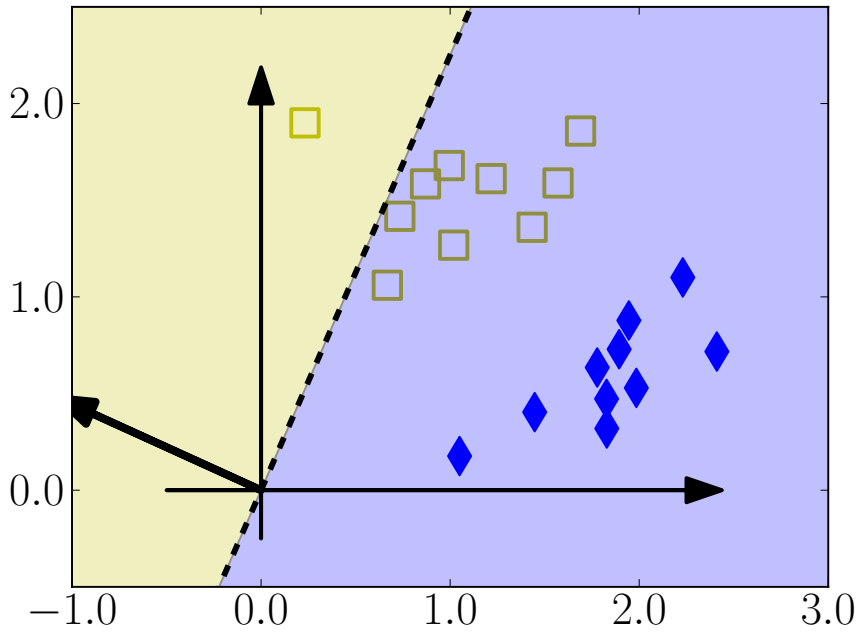
$$w^t = \boldsymbol{y}\boldsymbol{X}^t(\boldsymbol{X}\boldsymbol{X}^t)^{-1} = \begin{pmatrix} -7.97 \\ 9.37 \end{pmatrix} \begin{pmatrix} 47.82 & 27.75 \\ 27.75 & 28.33 \end{pmatrix}^{-1} = \begin{pmatrix} -0.83 \\ 1.14 \end{pmatrix}$$

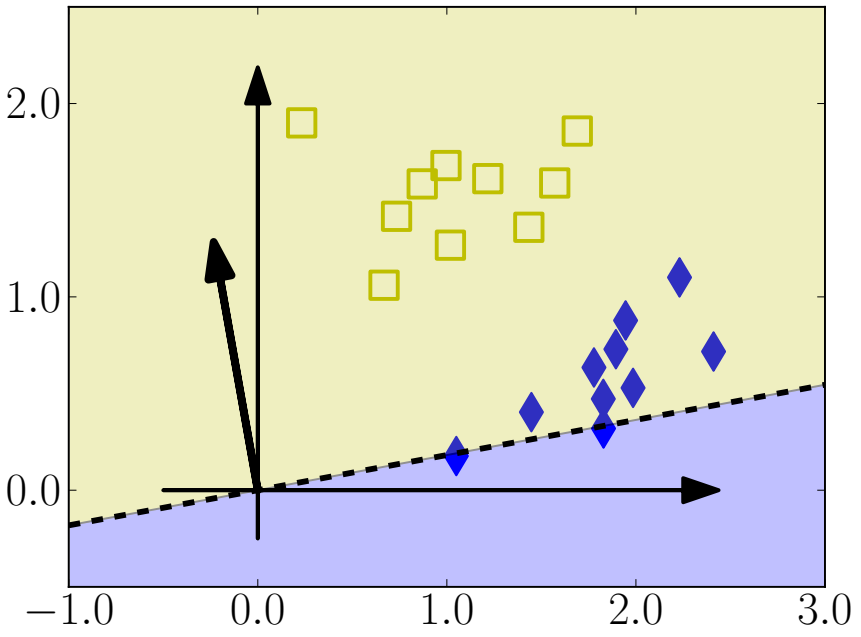
Nice. But is it the **best** w?

Criteria for Linear Classification

What properties should an optimal w have?

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





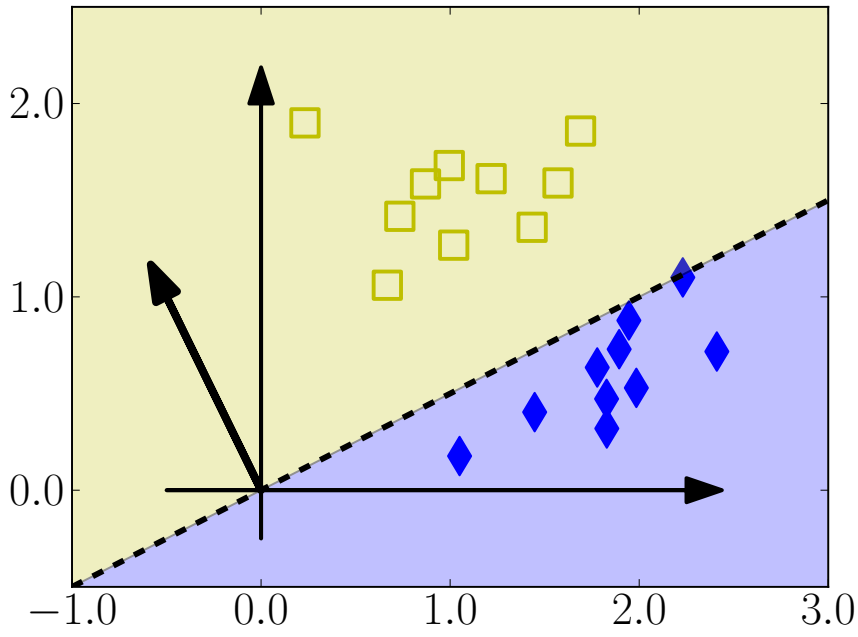
Are these the best? No, they misclassify many examples.

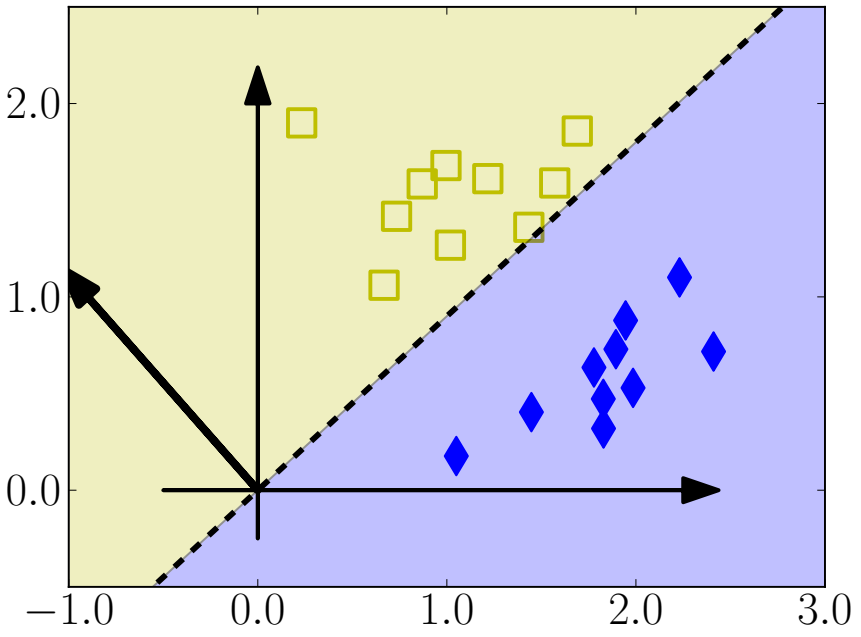
Criterion 1: Enforce $\operatorname{sign}\langle w, x_i \rangle = y_i$ for $i = 1, \ldots, n$.

Criteria for Linear Classification

What properties should an optimal w have?

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



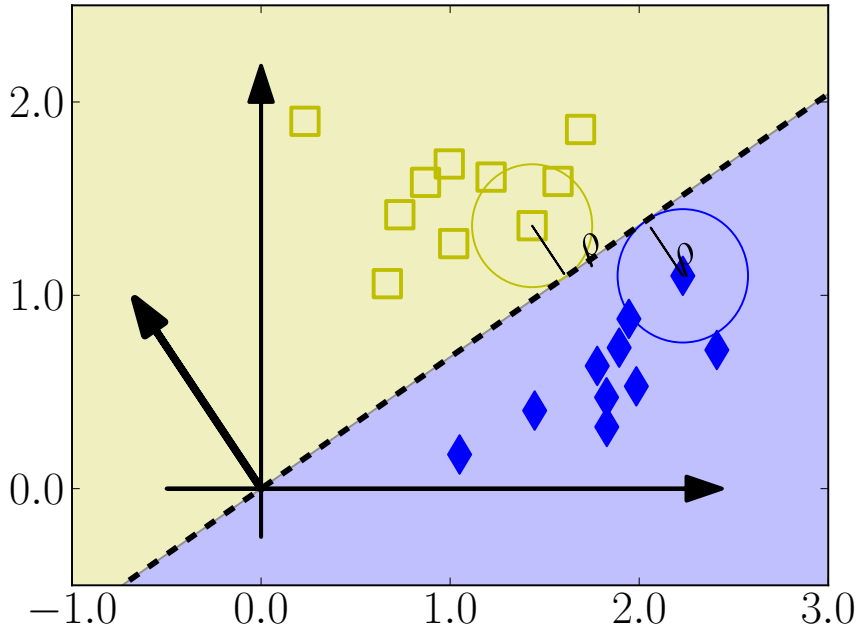


Are these the best? No, they would be "risky" for future samples.

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

Criteria for Linear Classification

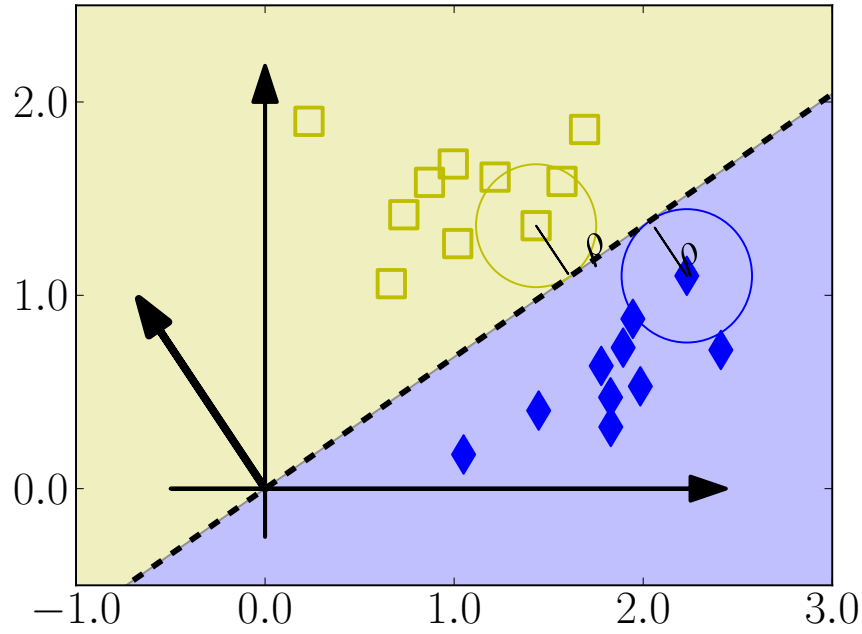
Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Assume that future samples are *similar* to current ones. What's the best w?

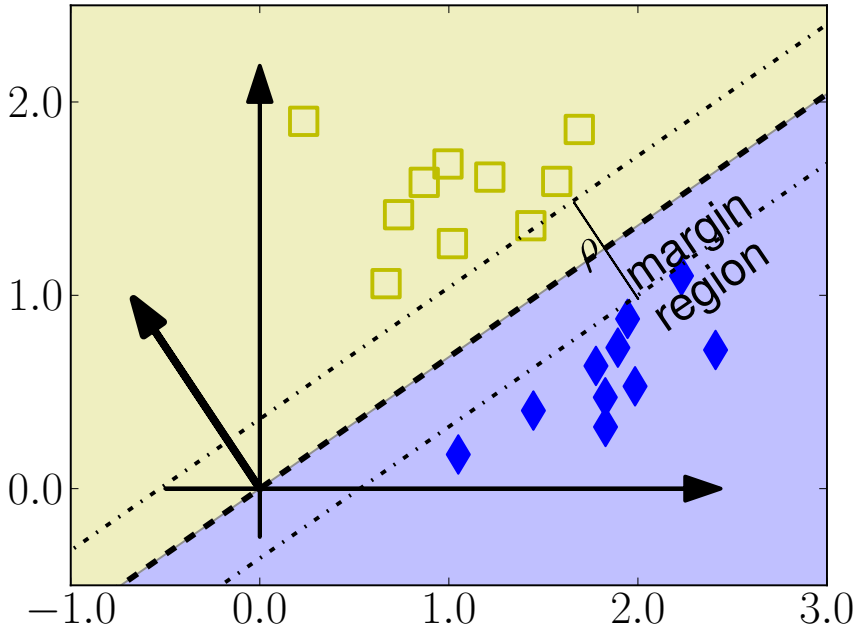


Maximize "robustness": use w such that we can maximally perturb the input samples without introducing misclassifications.

Criteria for Linear 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 "robustness": use w such that we can maximally perturb the input samples without introducing misclassifications.

Central quantity:

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

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

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

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

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

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 $f(x) = \langle w, x\rangle$.

Maximum Margin Classification

We can rewrite this as a *minimization problem*:

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

subject to

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

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

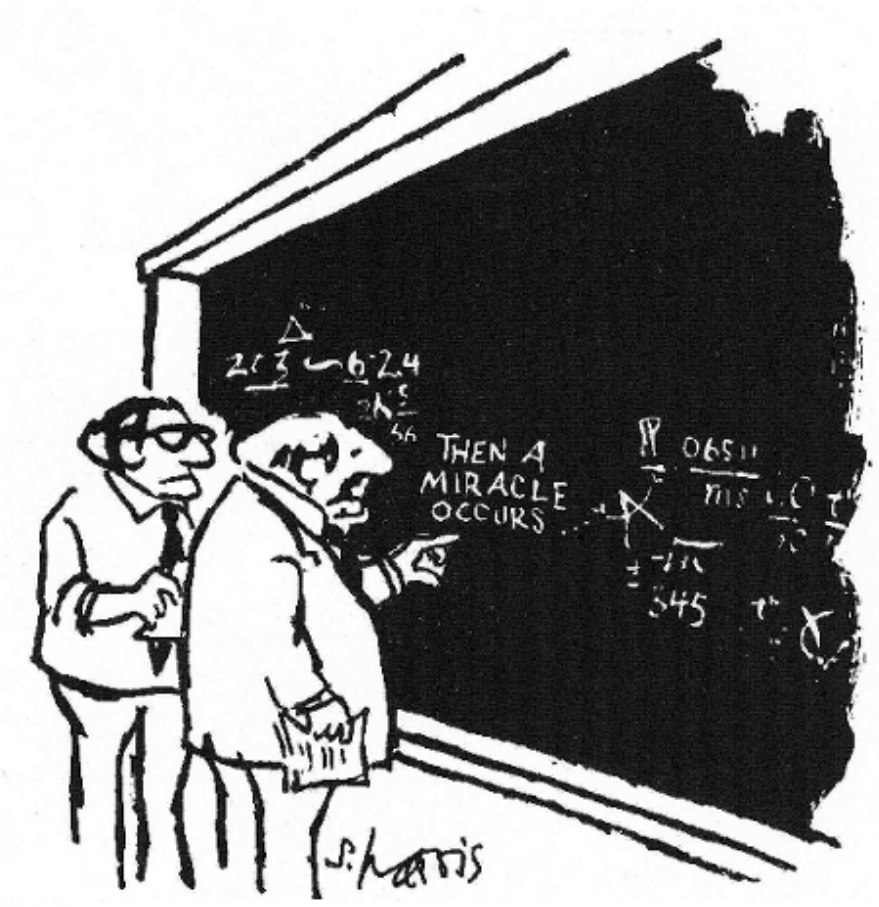
Maximum Margin Classifier (MMC)

Commercial Break: Objective Functions

Why is everything an optimization problem?

Why all the formulas?

Why not simply teach algorithms?



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

Commercial Break: Objective Functions

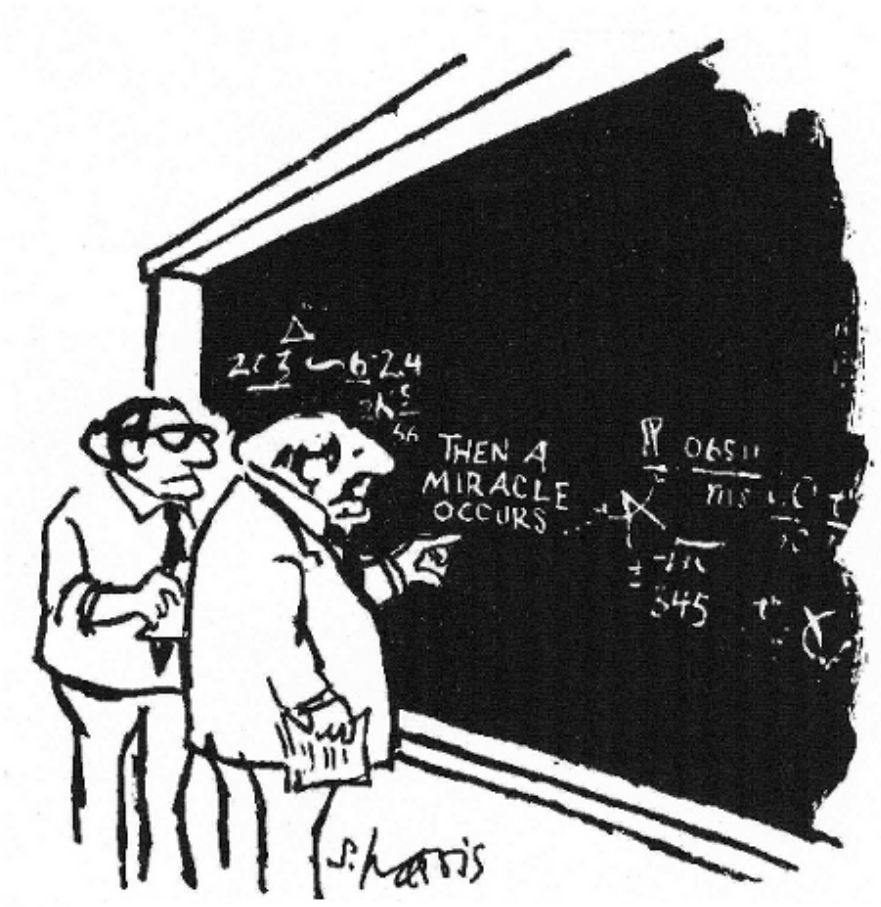
Why is everything an optimization problem?

Why all the formulas?

Why not simply teach algorithms?

Because...

- we want to separate between:
  - what is our ideal goal?
    = objective function
  - (how) do we achieve it?
    = optimization method



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

Commercial Break: Objective Functions

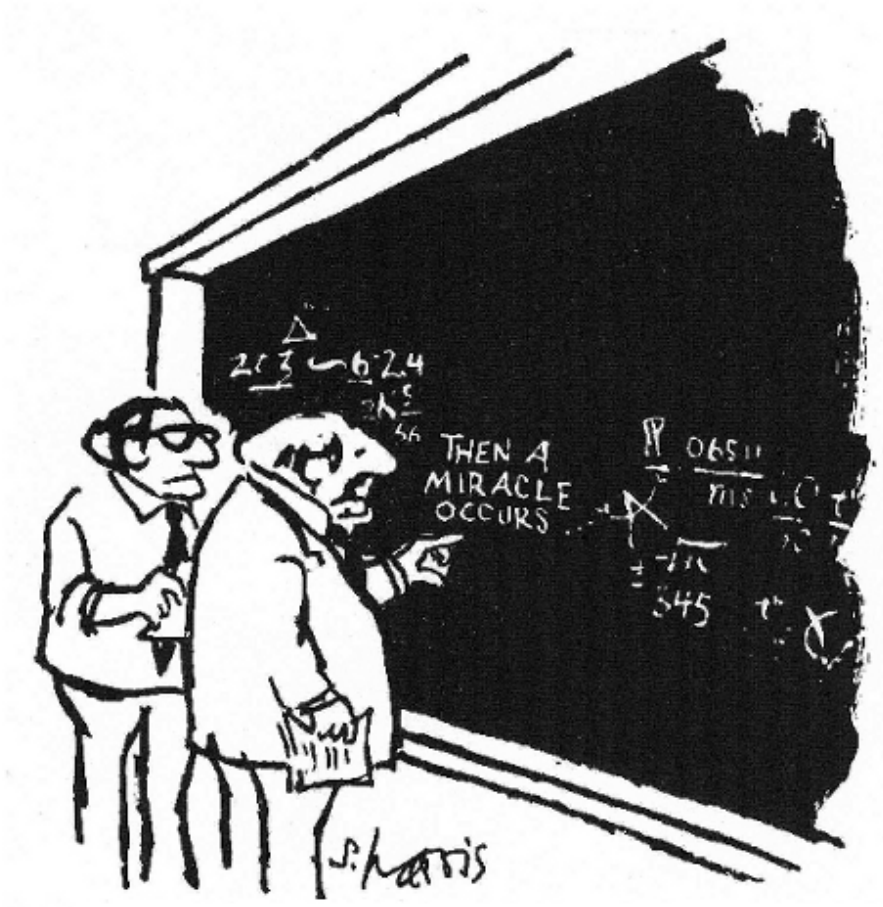
Why is everything an optimization problem?

Why all the formulas?

Why not simply teach algorithms?

Because...

- we want to separate between:
  - what is our ideal goal?
    = objective function
  - (how) do we achieve it?
    = optimization method
- defining a goal helps in **understanding** the problem
- mathematical formulation allows **re-using existing algorithms** (developed for different tasks)



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

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:

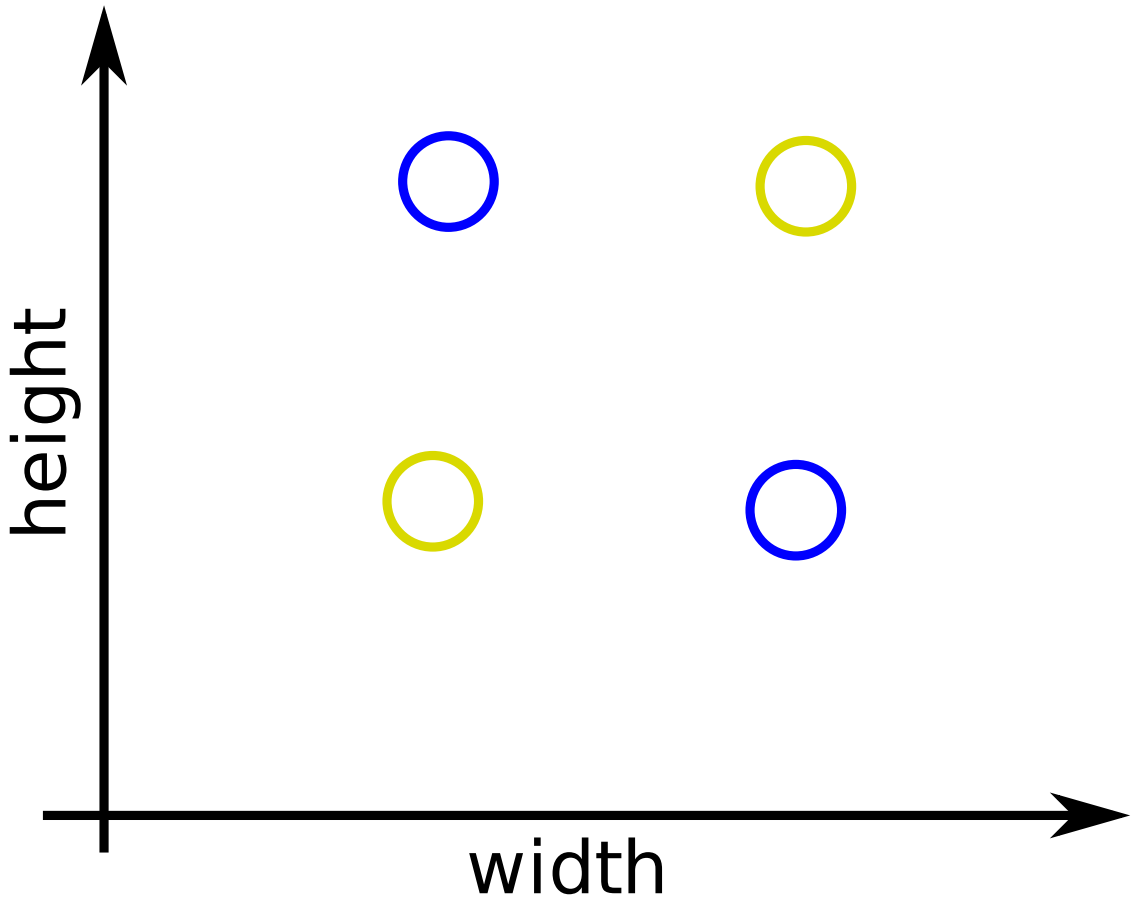
- The objective function is differentiable and *convex*.
- The constraints are all linear.

We can find the *globally* optimal w in $O(nd^2 + d^3)$ (often faster).

- There are no local minima.
- We have a definite stopping criterion.

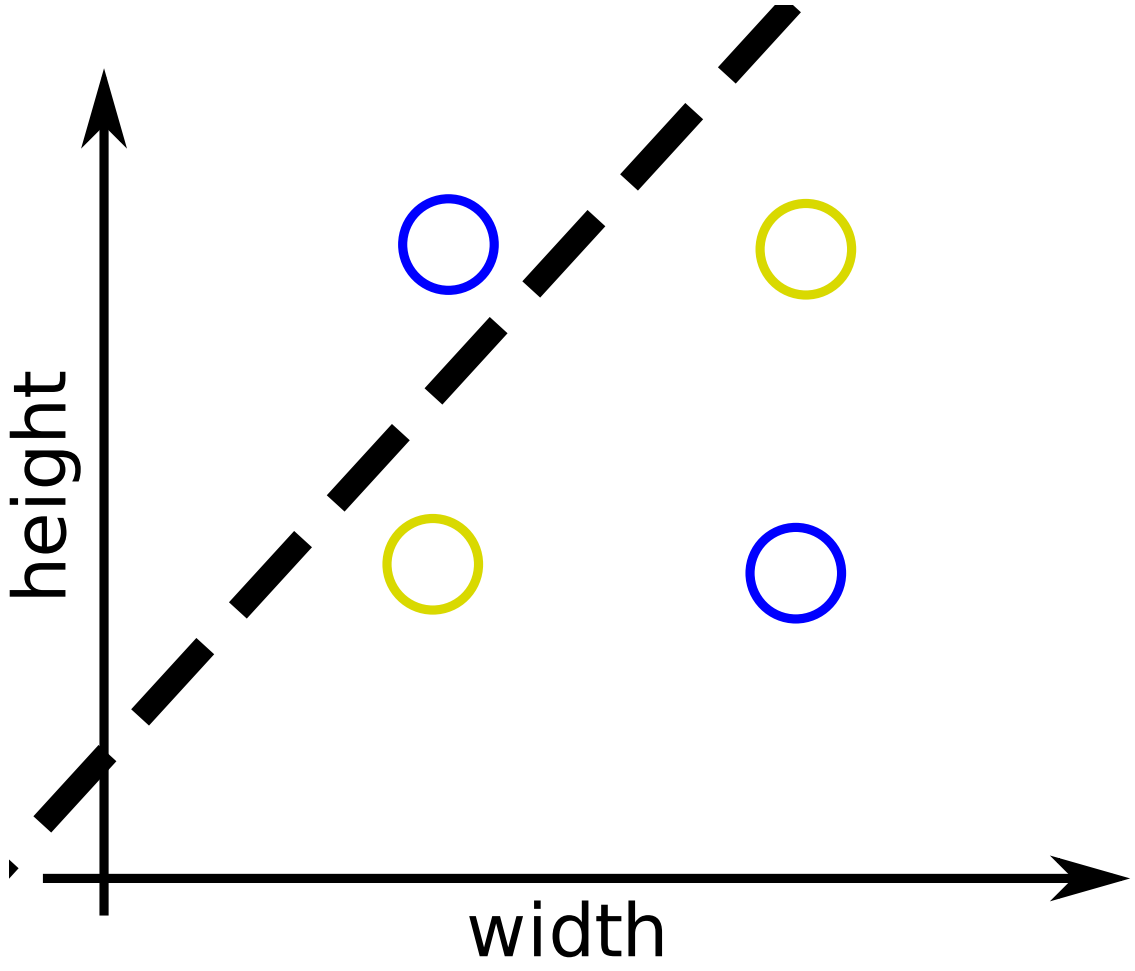
Linear Separability

What is the best w for this dataset?



Linear Separability

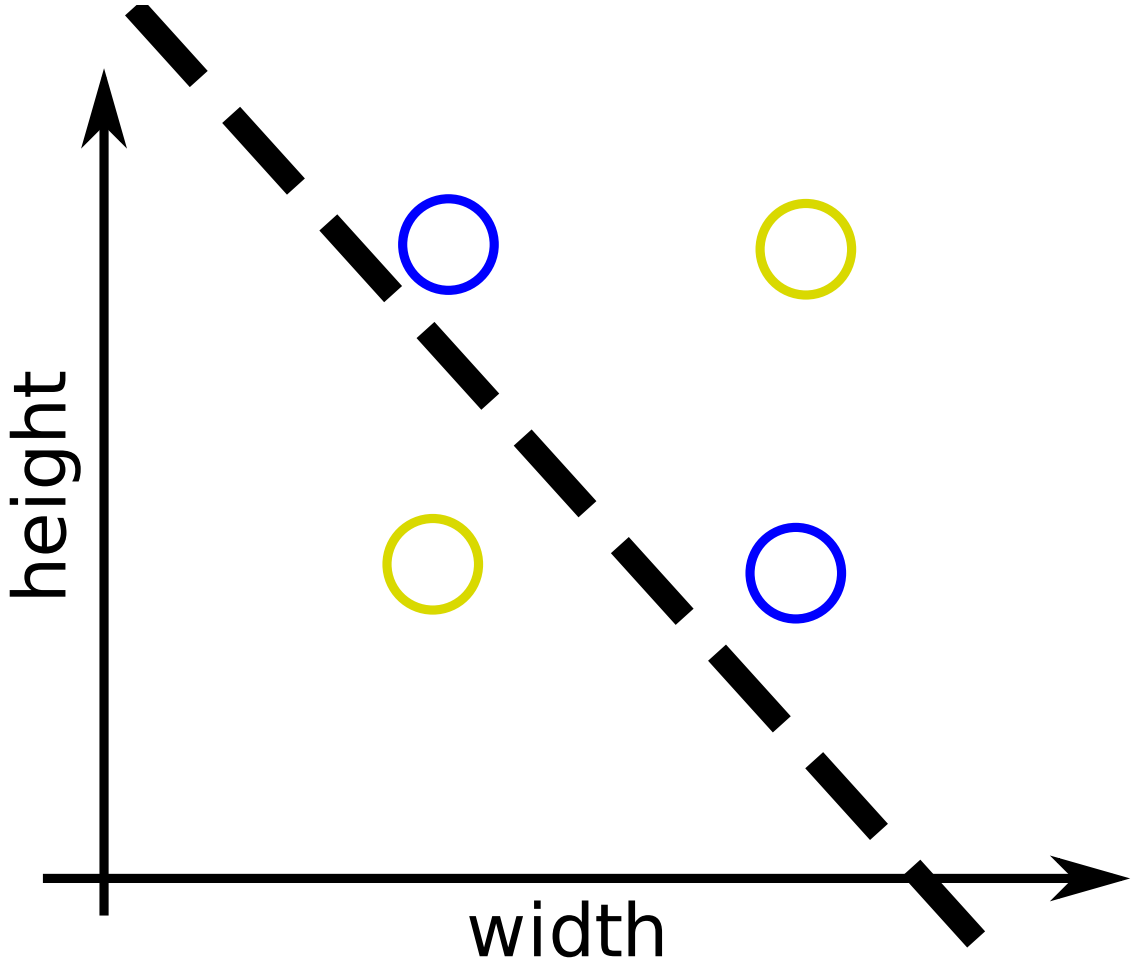
What is the best w for this dataset?



Not this.

Linear Separability

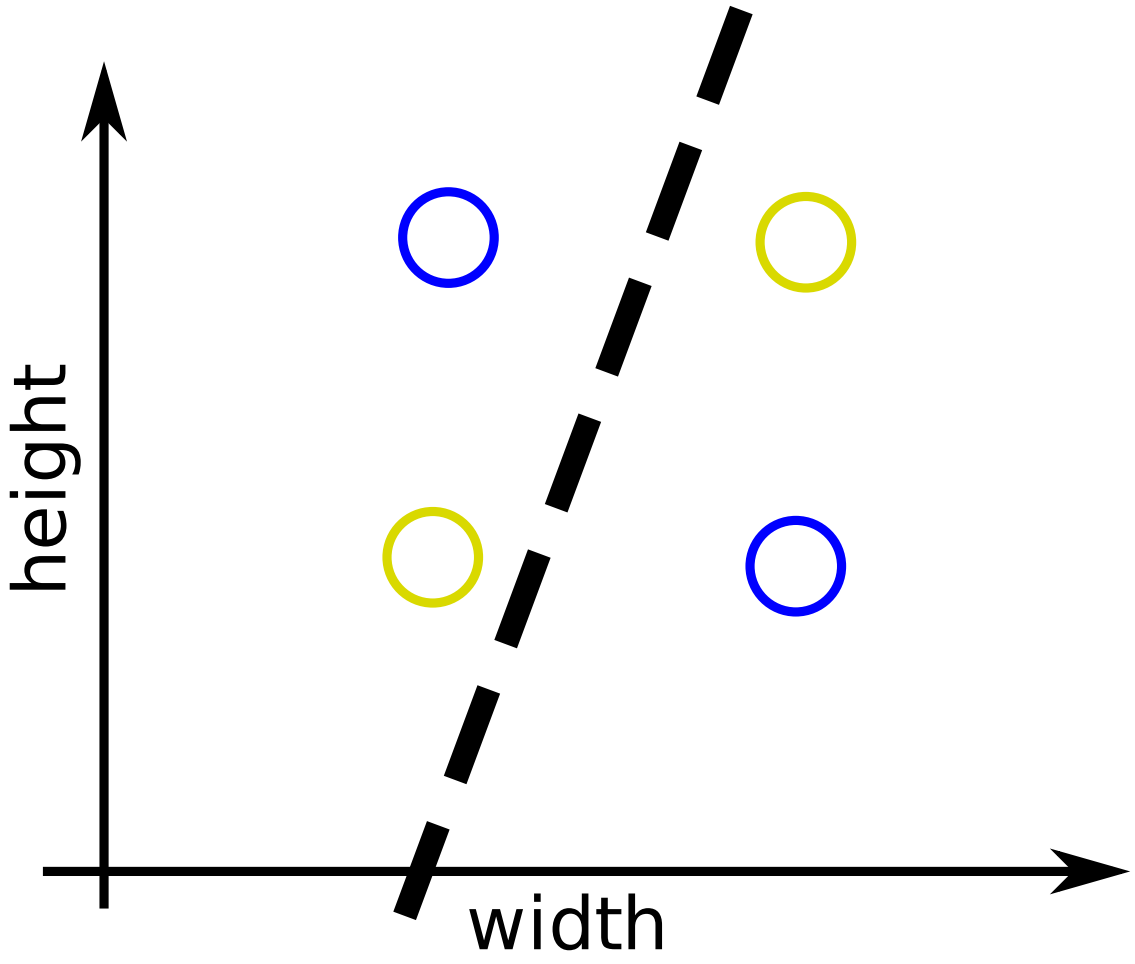
What is the best w for this dataset?



Not this.

Linear Separability

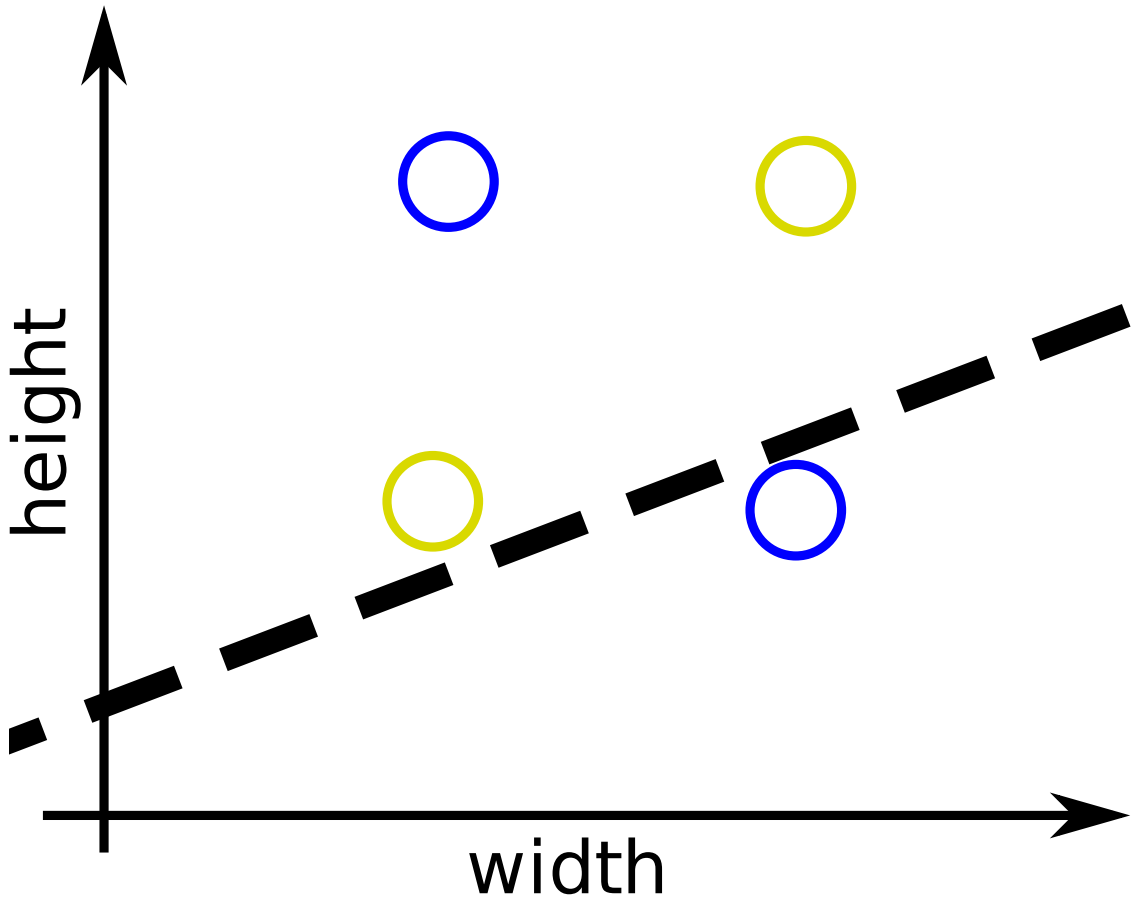
What is the best w for this dataset?



Not this.

Linear Separability

What is the best w for this dataset?



Not this.

Linear Separability

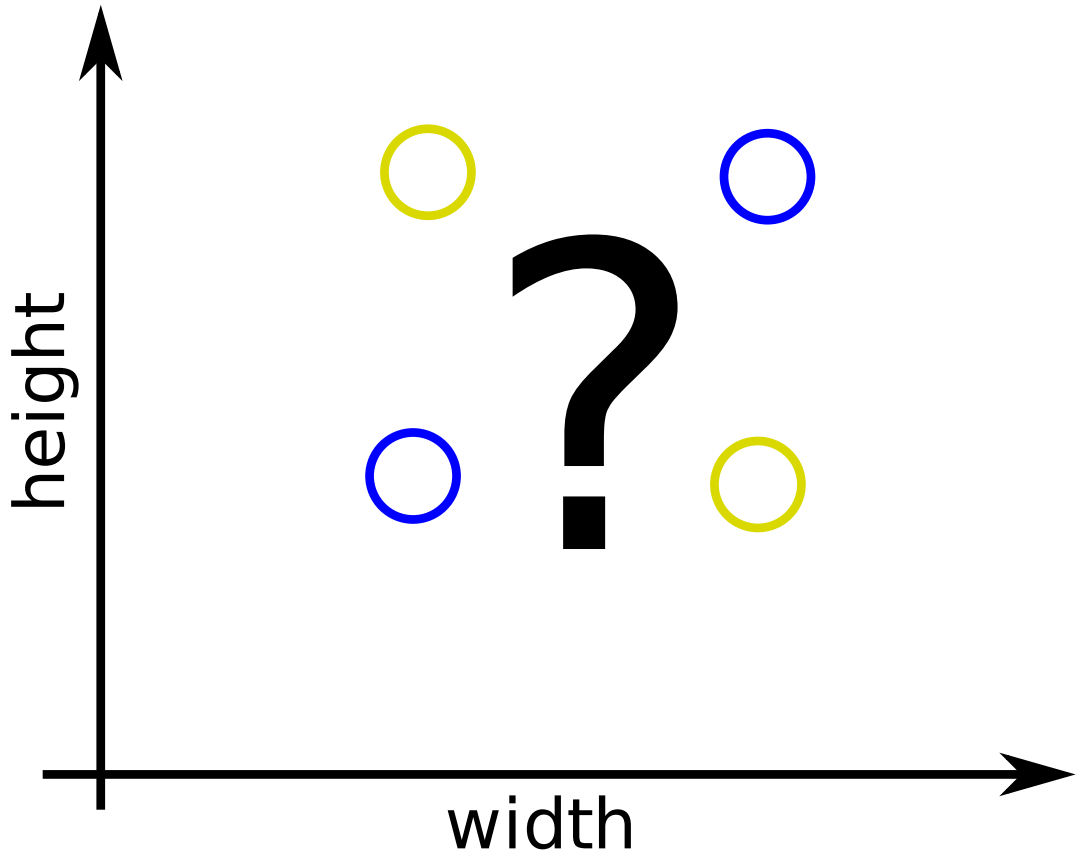
The MMC problem

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

subject to

$$\begin{aligned} y_1 \langle w, x_1 \rangle &\geq 1 \\ y_2 \langle w, x_2 \rangle &\geq 1 \\ y_3 \langle w, x_3 \rangle &\geq 1 \\ y_4 \langle w, x_4 \rangle &\geq 1 \end{aligned}$$

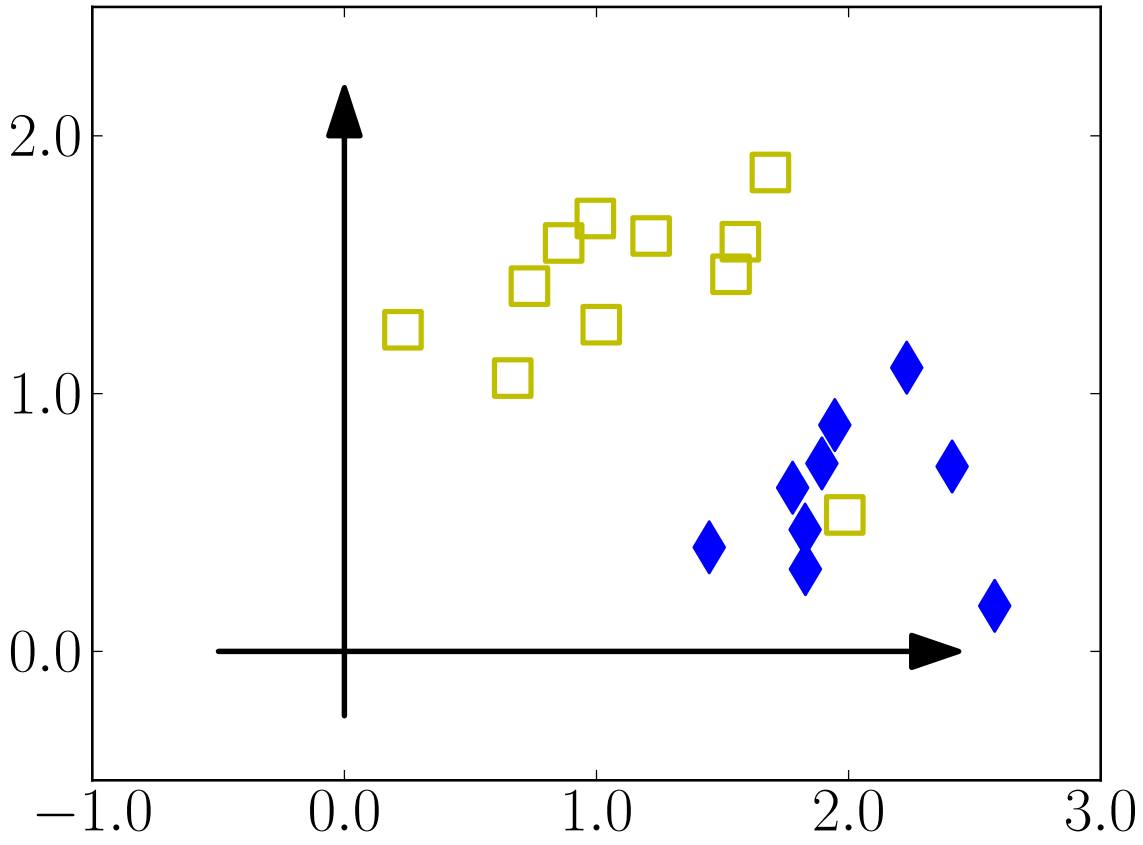
has **no solution**.
The constraints contradict each other!



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

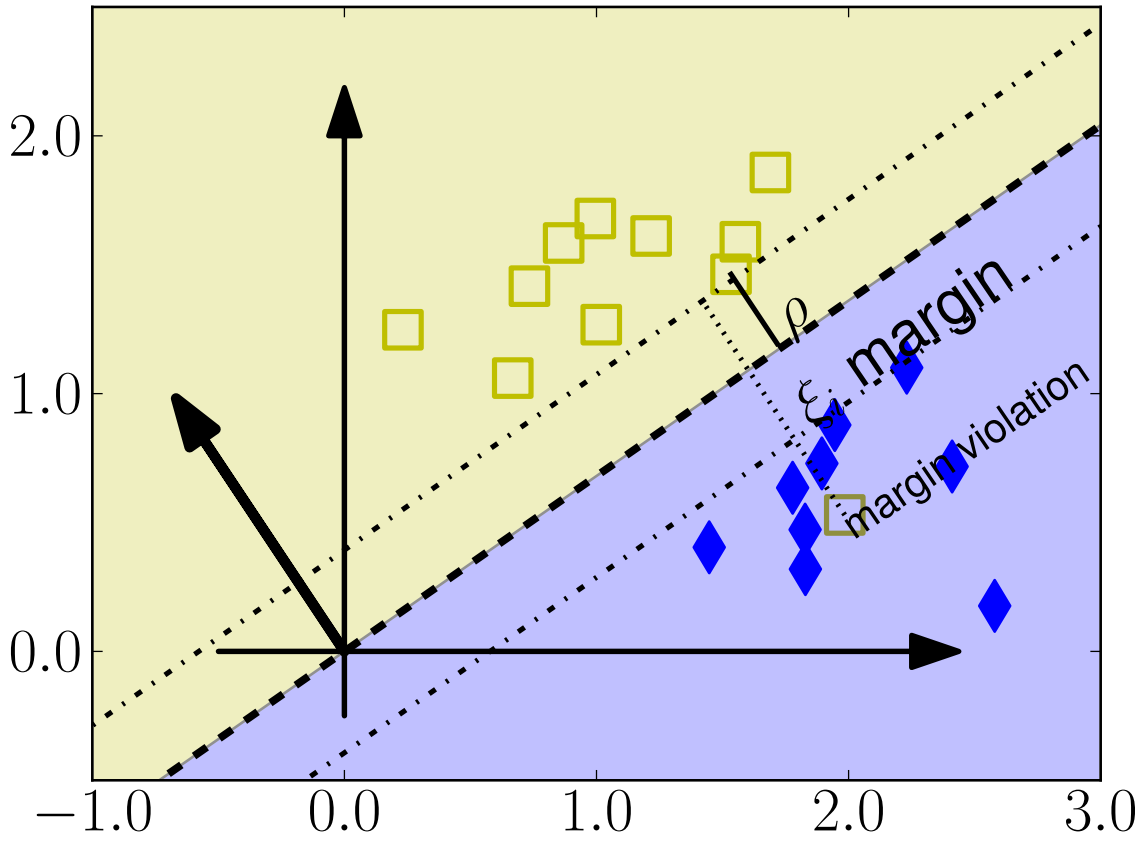
Linear Separability

What is the best w for this dataset?



Linear Separability

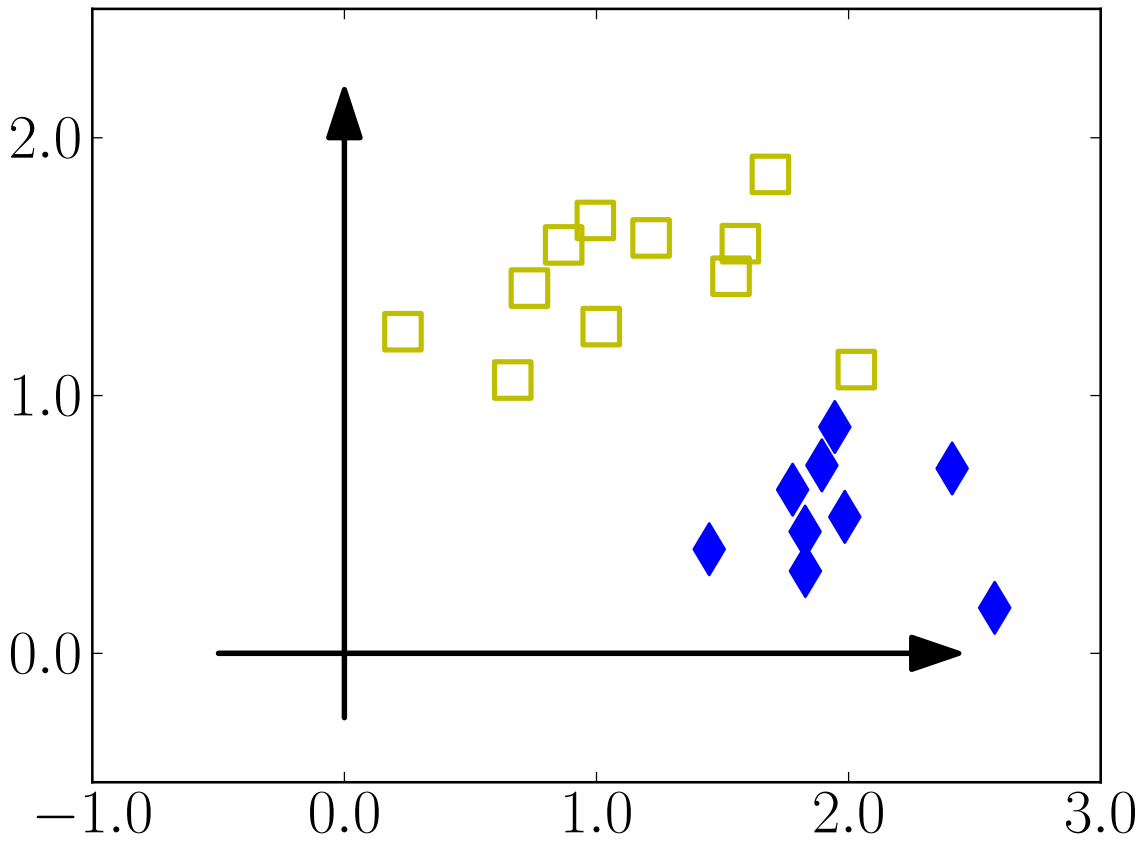
What is the best w for this dataset?



Possibly this one, even though one sample is misclassified.

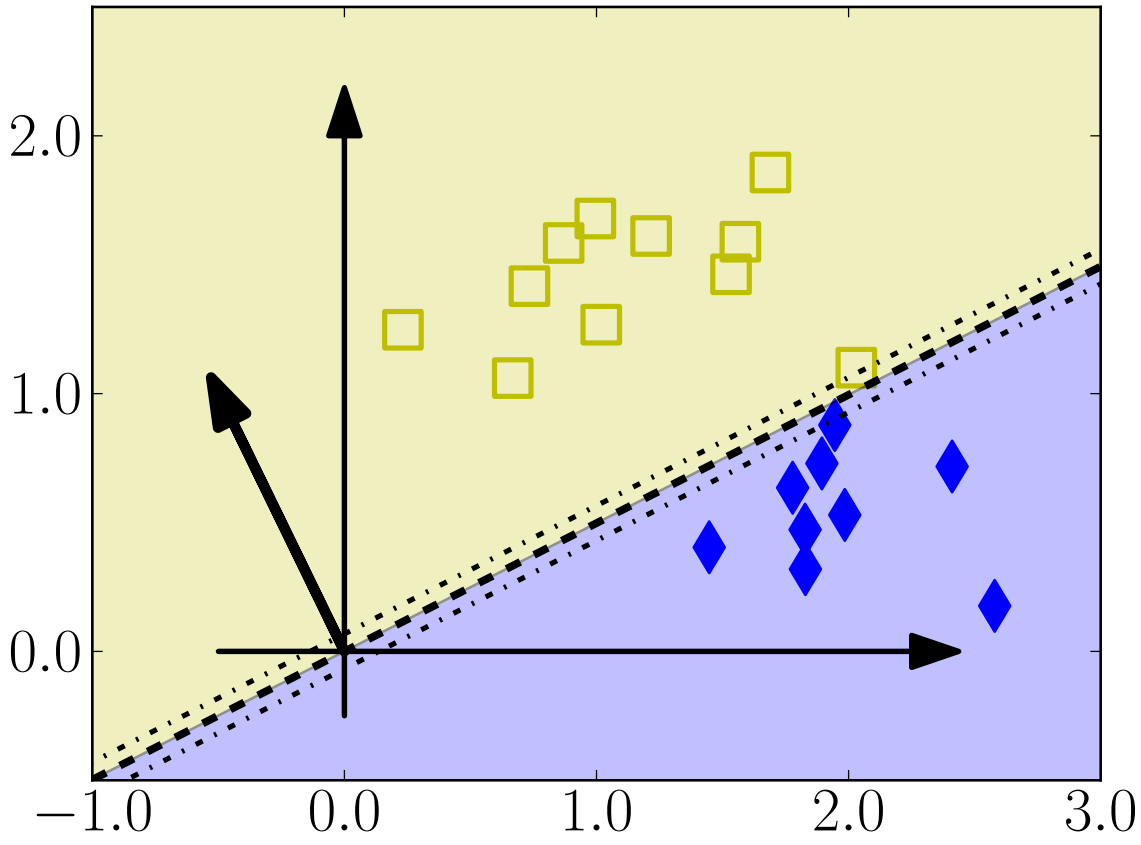
Linear Separability

What is the best w for this dataset?



Linear Separability

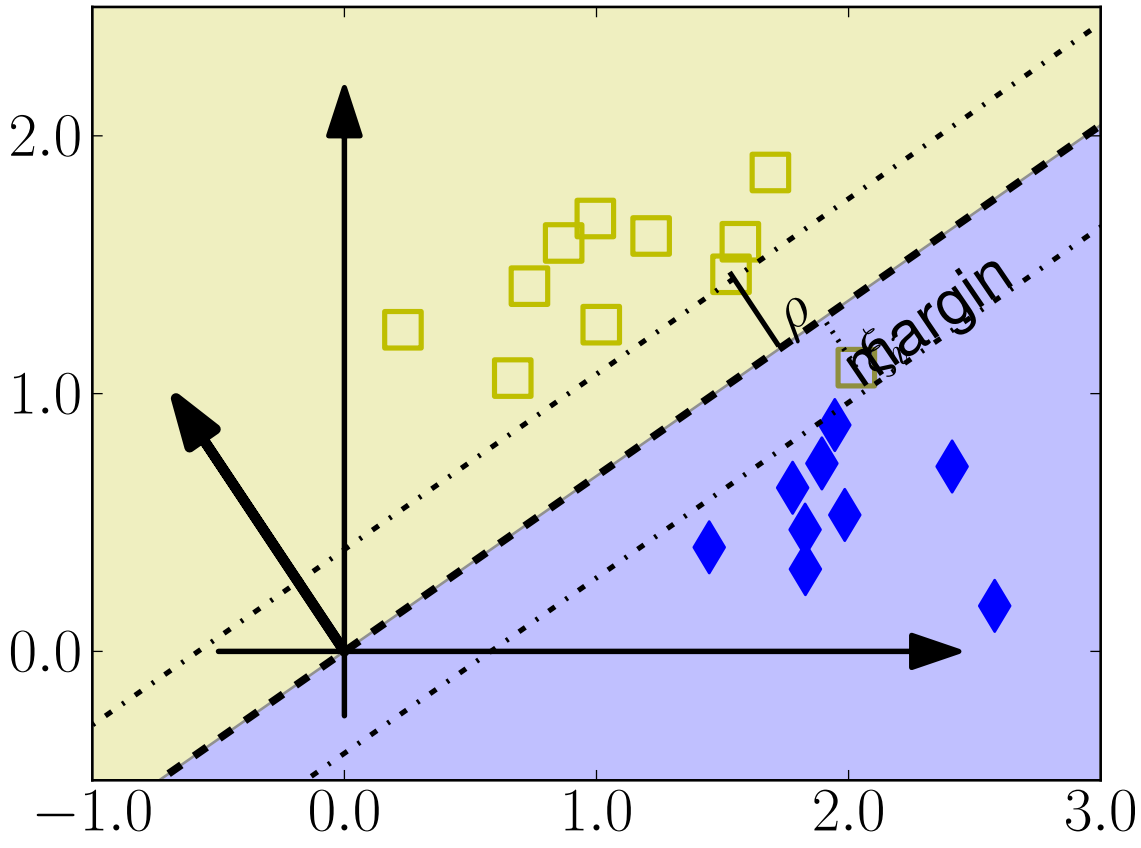
What is the best w for this dataset?



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

Soft-Margin Classification

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

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

Linear Support Vector Machine (linear SVM)

Soft-Margin Classification

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

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

Linear Support Vector Machine (linear SVM)

- 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 errors on training set
- Note: The problem is still convex and efficiently solvable.

Note: This is not a hack. There's strong theoretical justifications, e.g. generalization bounds.
see e.g. [J. Shaw-Taylor, N. Cristianini: "Kernel Methods for Pattern Analysis", Cambridge University Press, 2004]

Solving for Soft-Margin Solution

Reformulate constraints into objective functions with *Hinge* loss:

$$\min_{w \in \mathbb{R}^d} \|w\|^2 + \; C \sum_{i=1}^{n} \Big[1 - y_i \langle w, x_i \rangle \Big]_+$$

where $[t]_+ := \max\{0, t\}$.

- Now unconstrained optimization, but non-differentiable function
- Solve, e.g., by *subgradient-descent*,
- Currently most efficient: *stochastic gradient descent*

[O. Chapelle, *"Training a support vector machine in the primal"*, Neural Computation, 2007]
[T. Joachims, *"Training linear SVMs in linear time"*, ACM SIGKDD, 2006]
[S. Shalev-Shwartz, Y. Singer, N. Srebro, *"Pegasos: Primal estimated sub-gradient solver for SVM"*, ICML, 2007]
[L. Bottou, *"Large-scale machine learning with stochastic gradient descent"*, Compstat, 2010]

Linear SVMs in Practice

Efficient software packages:

- **liblinear:** `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`
- **SVMperf:** `http://www.cs.cornell.edu/People/tj/svm_light/svm_perf.html`
- also: **Pegasos:**, `http://www.cs.huji.ac.il/~shais/code/`
- also: **sgd:**, `http://leon.bottou.org/projects/sgd`

Training time:

- approximately **linear** in data dimensionality
- approximately **linear** in number of training examples,

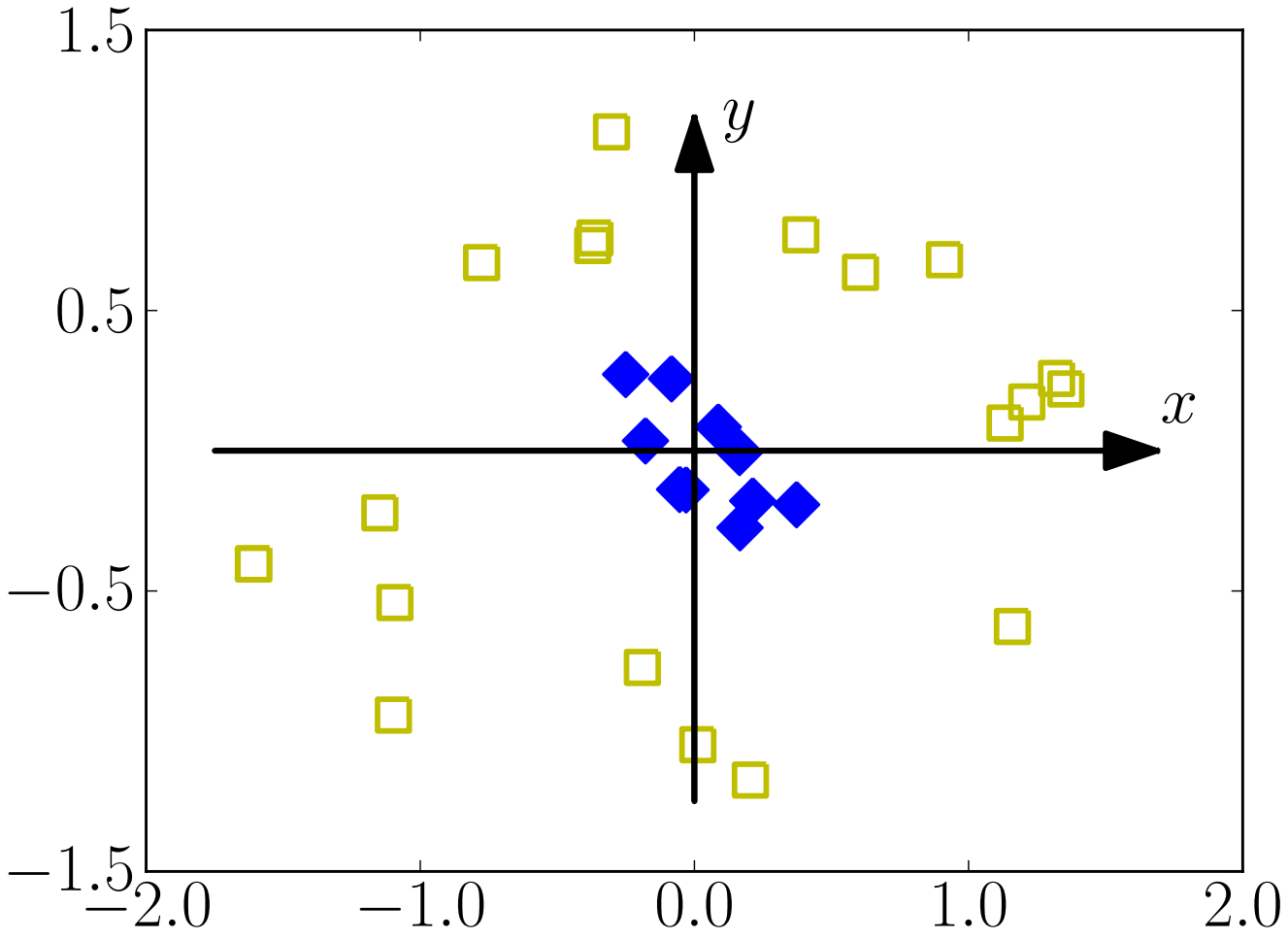
Evaluation time (per test example):

- **linear** in data dimensionality
- **independent** of number of training examples

Linear SVMs are currently the most frequently used classifiers in Computer Vision.

Linear Separability

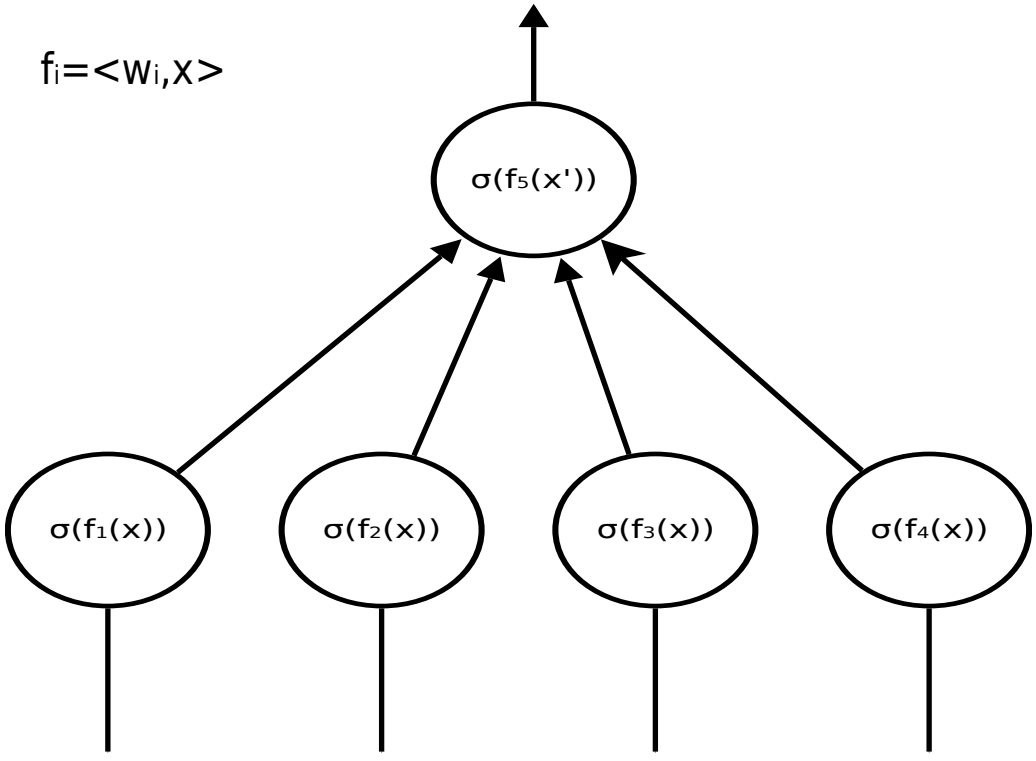
So, what is the best soft-margin w for this dataset?



None. We need something non-linear!

Non-Linear Classification: Stacking

Idea 1) Use classifier outputs as input to other classifier:

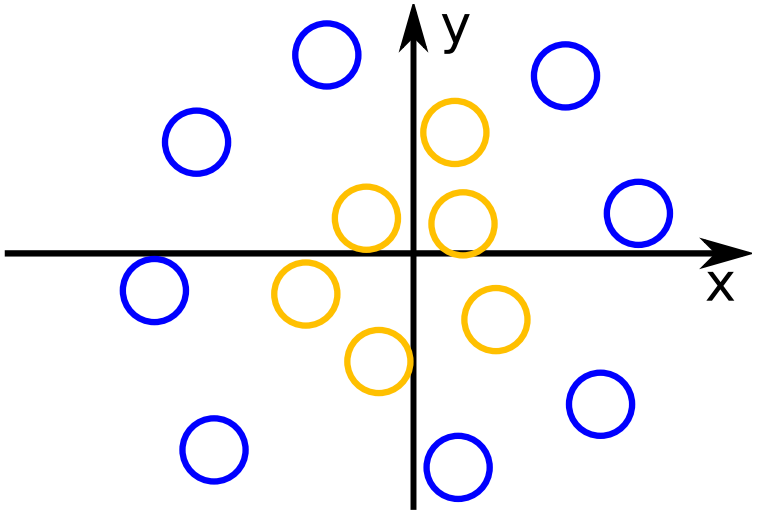


Multilayer Perceptron aka "Neural Network" aka "Deep Learning"
Boosting, **Decision Trees**, **Random Forests**
$\Rightarrow$ decisions depend non-linearly on x and w_j.

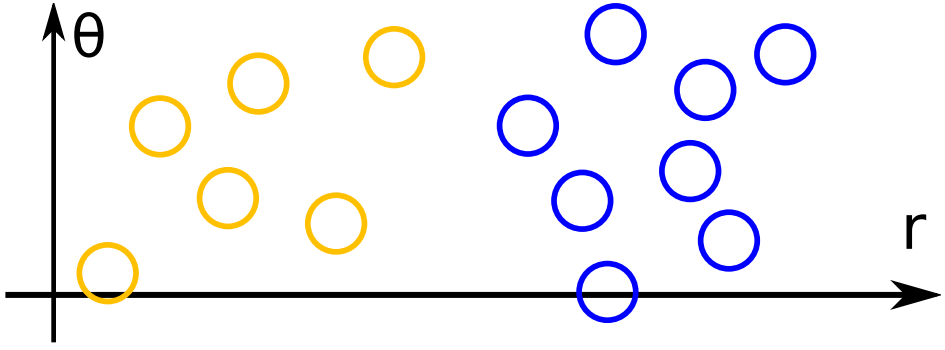
Non-linearity: Data Preprocessing

Idea 2) Preprocess the data:

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

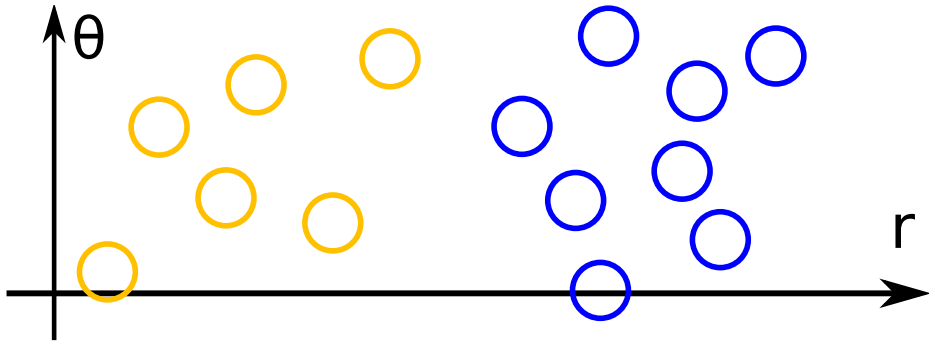


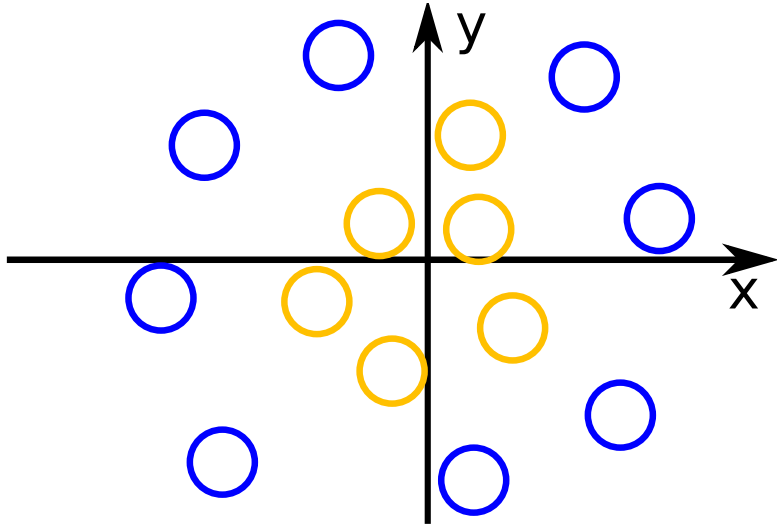
This one is:



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

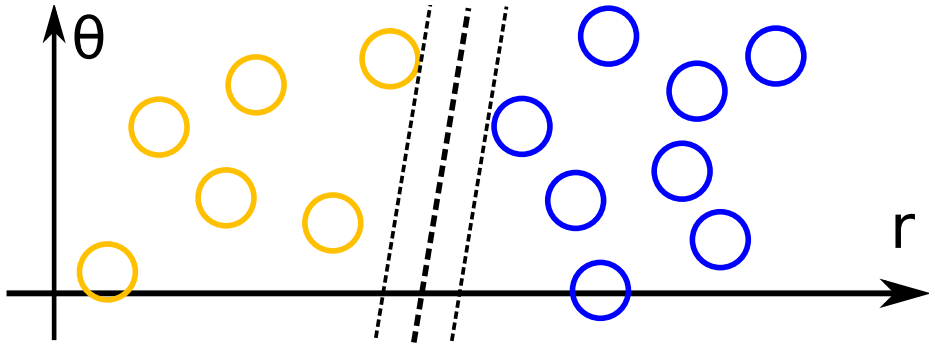
Non-linearity: Data Preprocessing

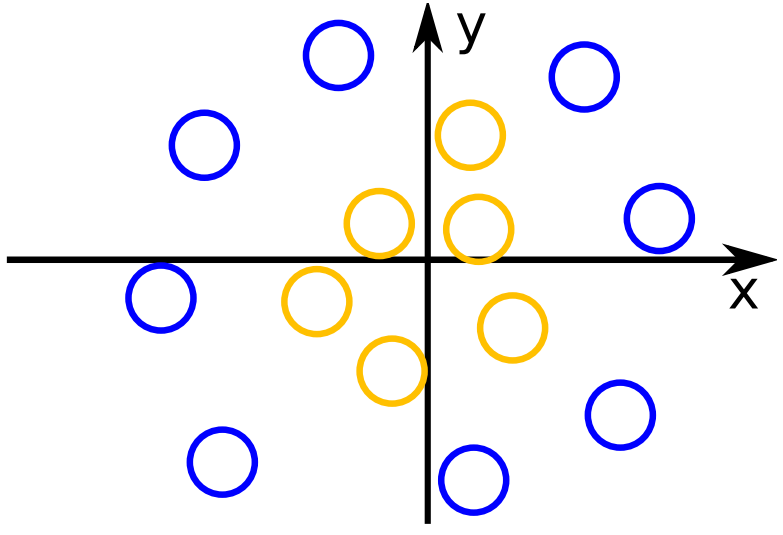




Non-linearity: Data Preprocessing

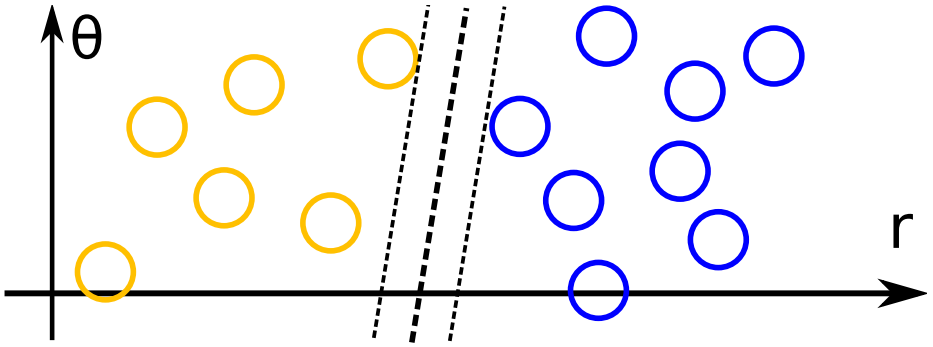
Linear separation



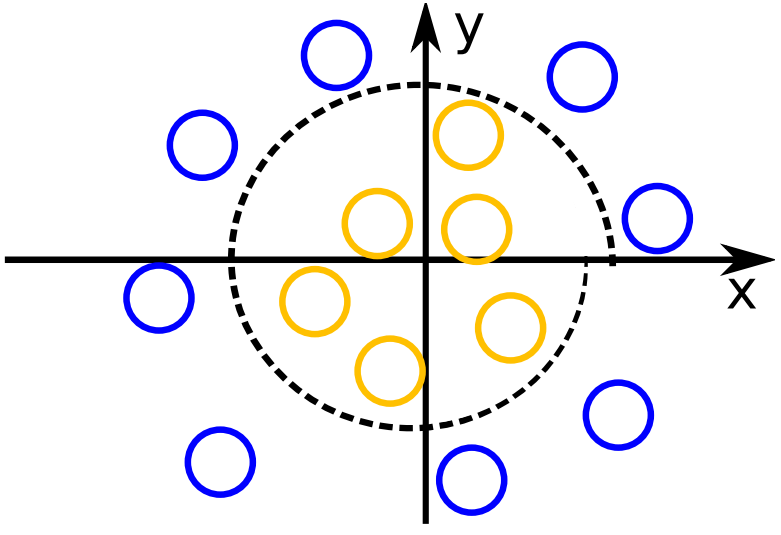


Non-linearity: Data Preprocessing

Linear separation

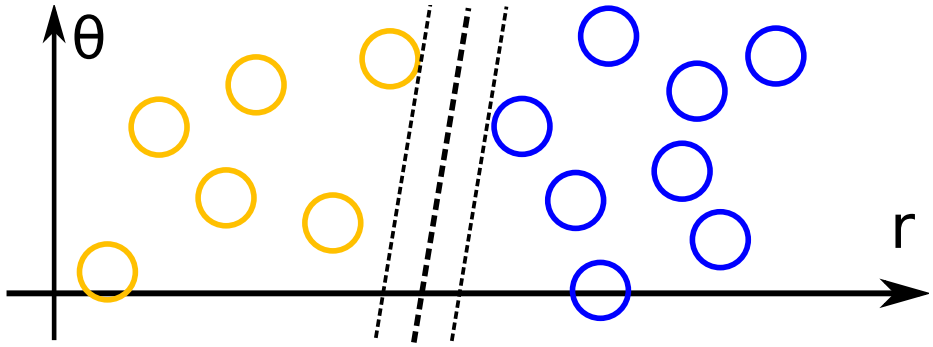


Non-linear separation

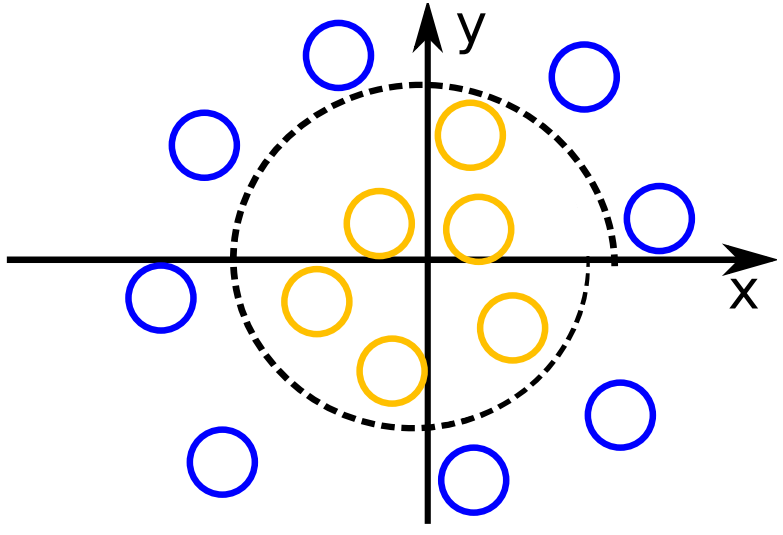


Non-linearity: Data Preprocessing

Linear separation



Non-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 inner 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:

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

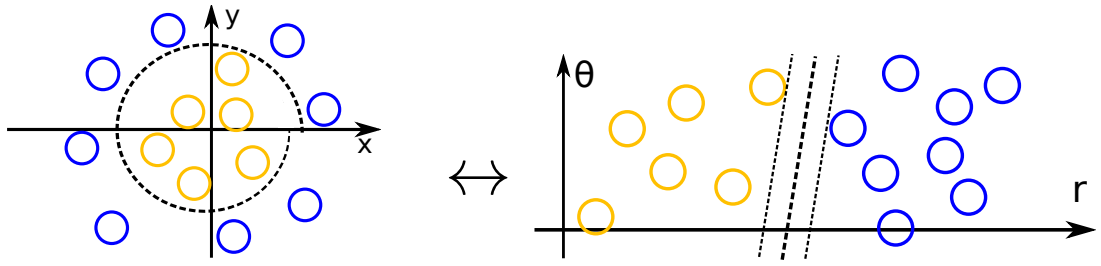
- Distance map:

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

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

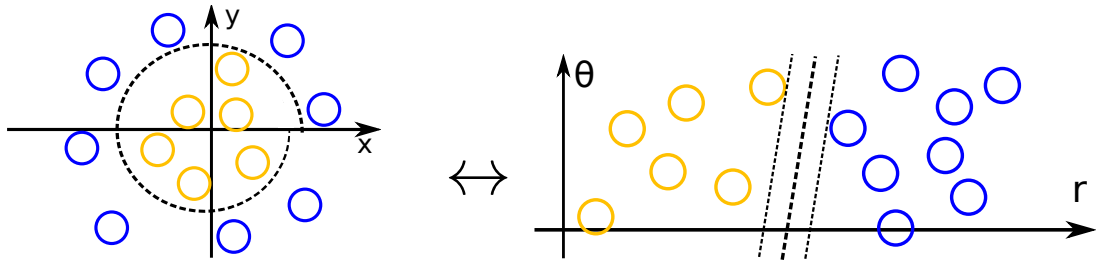
Is this enough?

In this example, a coordinate change was enough to make the problem solvable linearly. **Does this trick always work?**



Is this enough?

In this example, a coordinate change was enough to make the problem solvable linearly. **Does this trick always work?**



Answer: If we do it right, yes!

Lemma

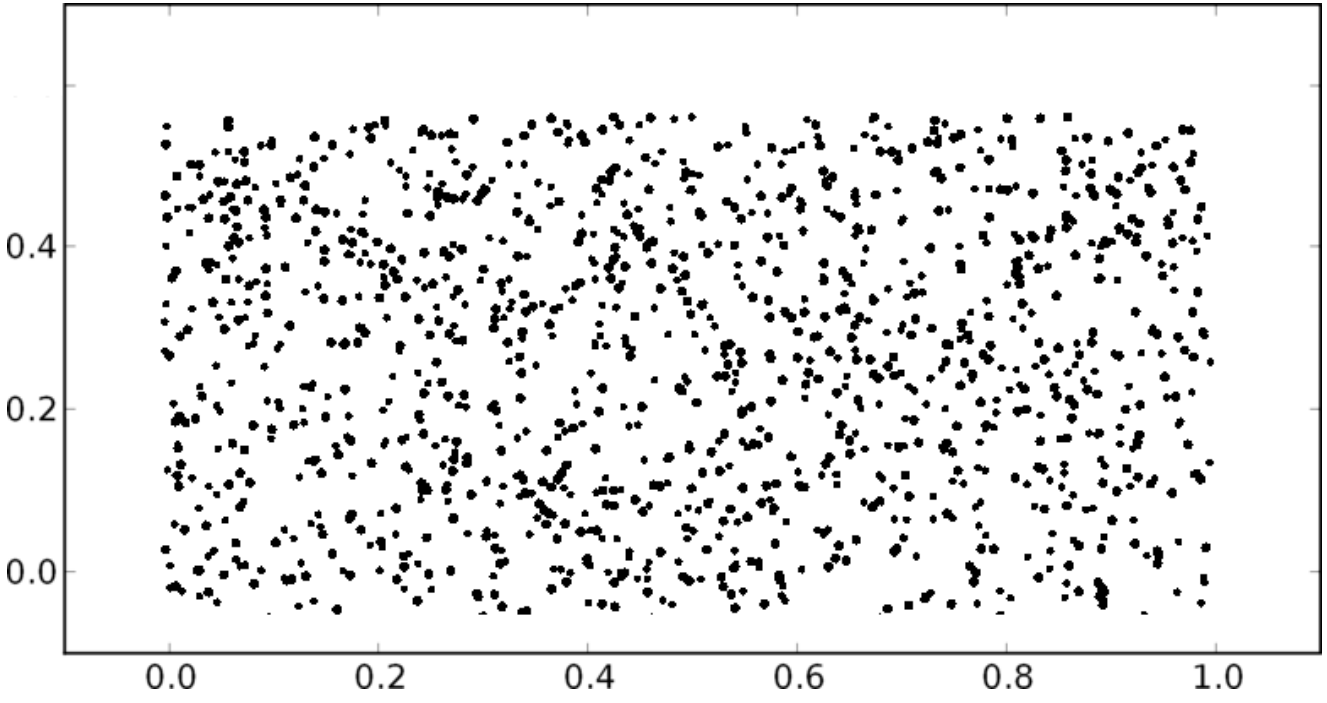
Let $(x_i)_{i=1,\dots,n}$ with $x_i \neq x_j$ for $i \neq j$. Let $\varphi : \mathbb{R}^k \to \mathbb{R}^m$ be a feature map. If the set $\varphi(x_i)_{i=1,\dots,n}$ is linearly independent, then the points $\varphi(x_i)_{i=1,\dots,n}$ are linearly separable.

Lemma

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

Is this enough?

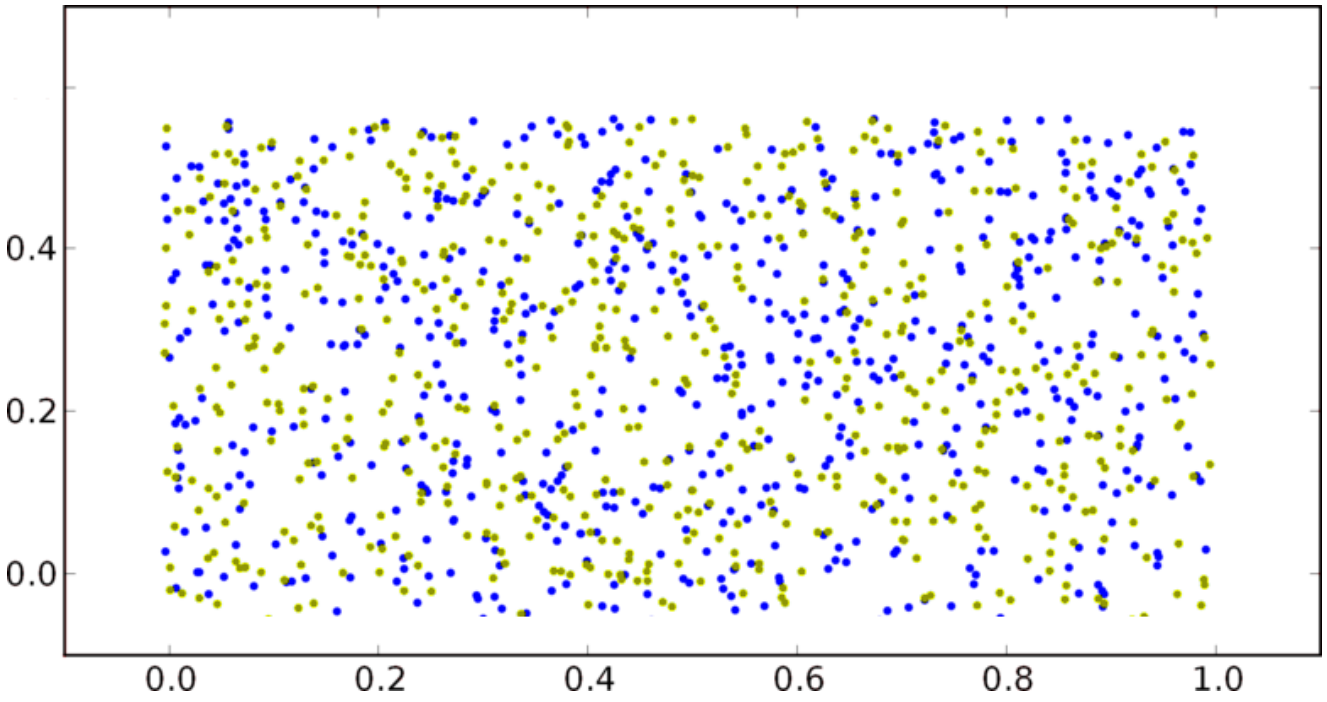
Caveat: We can separate *any* set, not just one with "reasonable" y_i:



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

Never rely on training accuracy! Don't forget to regularize!

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

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

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \quad \text{subject to } y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i$$

then $w = \sum_j \alpha_j \varphi(x_j)$ for some $\alpha \in \mathbb{R}^n$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \quad \text{subject to } y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i$$

$$\text{then } w = \textstyle\sum_j \alpha_j \varphi(x_j) \text{ for some } \alpha \in \mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1), \ldots, \varphi(x_n)\} = \{\sum_j \alpha_j \varphi(x_j) : \alpha \in \mathbb{R}^n\} \subset \mathbb{R}^m$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \quad \text{subject to } y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i$$

$$\text{then } w = \textstyle\sum_j \alpha_j \varphi(x_j) \text{ for some } \alpha \in \mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1), \ldots, \varphi(x_n)\} = \{\sum_j \alpha_j \varphi(x_j) : \alpha \in \mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u \in \mathbb{R}^m : \langle u, v \rangle = 0 \text{ for all } v \in V\}$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w \in \mathbb{R}^m, \xi_i \in \mathbb{R}^+} \|w\|^2 + C \sum_{i=1}^{n} \xi_i \quad \text{subject to } y_i \langle w, \varphi(x_i) \rangle \geq 1 - \xi_i$$

$$\text{then } w = \textstyle\sum_j \alpha_j \varphi(x_j) \text{ for some } \alpha \in \mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1), \ldots, \varphi(x_n)\} = \{\sum_j \alpha_j \varphi(x_j) : \alpha \in \mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u \in \mathbb{R}^m : \langle u, v \rangle = 0 \text{ for all } v \in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V \in V$, $w_{V^\perp} \in V^\perp$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Now we would like to show $w_{V^\perp} = 0$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Now we would like to show $w_{V^\perp} = 0$. What if $w_{V^\perp}\neq 0$?

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Now we would like to show $w_{V^\perp}=0$. What if $w_{V^\perp}\neq 0$?

- $\|w_{V^\perp}\|^2 > 0 \Rightarrow \|w\|^2 > \|w_V\|^2$.

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Now we would like to show $w_{V^\perp}=0$. What if $w_{V^\perp}\neq 0$?

- $\|w_{V^\perp}\|^2 > 0 \Rightarrow \|w\|^2 > \|w_V\|^2$.
  w_V solves $(*)$ with same ξ_i but smaller norm. Contradiction!

Representer Theorem: Proof

Theorem: w solves

$$(*) \qquad \min_{w\in\mathbb{R}^m,\xi_i\in\mathbb{R}^+} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to } y_i\langle w,\varphi(x_i)\rangle \geq 1-\xi_i$$

$$\text{then } w=\textstyle\sum_j \alpha_j\varphi(x_j) \text{ for some } \alpha\in\mathbb{R}^n.$$

Proof:

- $V := \operatorname{span}\{\varphi(x_1),\ldots,\varphi(x_n)\} = \{\sum_j \alpha_j\varphi(x_j) : \alpha\in\mathbb{R}^n\} \subset \mathbb{R}^m$.
- $V^\perp := \{u\in\mathbb{R}^m : \langle u,v\rangle = 0 \text{ for all } v\in V\}$.
- decompose $w = w_V + w_{V^\perp}$ with $w_V\in V$, $w_{V^\perp}\in V^\perp$.
- from $\langle w_{V^\perp},\varphi(x_i)\rangle = 0$ follows $\langle w_V,\varphi(x_i)\rangle = \langle w,\varphi(x_i)\rangle$.
- from $\langle w_V, w_{V^\perp}\rangle = 0$ follows $\|w\|^2 = \|w_V\|^2 + \|w_{V^\perp}\|^2$.

Now we would like to show $w_{V^\perp}=0$. What if $w_{V^\perp}\neq 0$?

- $\|w_{V^\perp}\|^2 > 0 \Rightarrow \|w\|^2 > \|w_V\|^2$.
  w_V solves $(*)$ with same ξ_i but smaller norm. Contradiction!

So $w_{V^\perp}=0$, ergo $w = w_V \in V$. q.e.d.

Kernel Trick

Rewrite the optimization using the representer theorem: insert $w = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$, minimize over α_i instead of w:

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

subject to

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

Kernel Trick

Rewrite the optimization using the representer theorem: insert $w = \sum_{j=1}^{n} \alpha_j \varphi(x_j)$, 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

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

The former m-dimensional optimization is now n-dimensional.

Kernel Trick

Use $\|w\|^2 = \langle w, w \rangle$ and linearity of $\langle \cdot, \cdot \rangle$:

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

subject to

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

Kernel Trick

Use $\|w\|^2 = \langle w, w \rangle$ and linearity of $\langle \cdot, \cdot \rangle$:

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

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

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

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

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

To train, we only need to know the **kernel matrix** K

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

To evaluate on new data x, we need values $k(x_1, x), \ldots, k(x_n, x)$:

$$f(x) = \langle w, \varphi(x) \rangle = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$

Dualization

More elegant: dualization with Langrian multipliers α_i

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

subject to

$$\sum_{i=1}^{n} y_i \alpha_i = 0$$

Support-Vector Machine (SVM)

Optimization be solved numerically by any **quadratic program (QP)** solver but specialized software packages are more efficient.

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

1) Memory usage:
   - Storing $\varphi(x_1), \ldots, \varphi(x_n)$ requires $O(nm)$ memory.
   - Storing $k(x_1, x_1), \ldots, k(x_n, x_n)$ requires $O(n^2)$ memory.

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

1) Memory usage:

- Storing $\varphi(x_1), \ldots, \varphi(x_n)$ requires $O(nm)$ memory.
- Storing $k(x_1, x_1), \ldots, k(x_n, x_n)$ requires $O(n^2)$ memory.

2) 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: comparing angles $(x \in [0, 2\pi])$

$$\varphi : x \mapsto (\cos(x), \sin(x)) \in \mathbb{R}^2$$

$$\begin{aligned} \langle \varphi(x_i), \varphi(x_j) \rangle &= \langle\ (\cos(x_i), \sin(x_i)), (\cos(x_j), \sin(x_j))\ \rangle \\ &= \cos(x_i)\cos(x_j) + \sin(x_i)\sin(x_j) \end{aligned}$$

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

1) Memory usage:

- Storing $\varphi(x_1), \ldots, \varphi(x_n)$ requires $O(nm)$ memory.
- Storing $k(x_1, x_1), \ldots, k(x_n, x_n)$ requires $O(n^2)$ memory.

2) 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: comparing angles $(x \in [0, 2\pi])$

$$\varphi : x \mapsto (\cos(x), \sin(x)) \in \mathbb{R}^2$$

$$\begin{aligned} \langle \varphi(x_i), \varphi(x_j) \rangle &= \langle\ (\cos(x_i), \sin(x_i)), (\cos(x_j), \sin(x_j))\ \rangle \\ &= \cos(x_i)\cos(x_j) + \sin(x_i)\sin(x_j)\ \ = \cos(x_i - x_j) \end{aligned}$$

  Equivalently, but faster, without φ:

$$k(x_i, x_j) := \cos(x_i - x_j)$$

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

3) Flexibility:

- One can think of kernels as *measures of similarity*.
- 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 $\langle \varphi(x), \varphi(x') \rangle$?

3) Flexibility:

- One can think of kernels as *measures of similarity*.
- 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 videos on YouTube"* or $\mathcal{X} =$ *"all permutations of* $\{1, \ldots, k\}$*"*, or $\mathcal{X} =$ *"the internet"*.

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.
  Example: $\varphi(x) = \Big($*"# of red pixels in image* x*", green,blue* $\Big)$.

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.
  Example: $\varphi(x) = \Big($*"# of red pixels in image* x*", green,blue* $\Big)$.

- Any norm $\|.\| : V \to \mathbb{R}^m$ that fulfills the *parallelogram equation*
  - $\|x + y\|^2 + \|x - y\|^2 = 2\|x\|^2 + 2\|y\|^2$

  induces a kernel by *polarization*:
  - $k(x, y) := \frac{1}{2}\Big(\|x + y\|^2 - \|x\|^2 - \|y\|^2\Big)$.

  Example: $\mathcal{X} =$ *time series with bounded values*, $\|x\|^2 = \sum_{t=1}^{\infty} \frac{1}{2^t} x_t$

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.
  Example: $\varphi(x) = \Big($*"# of red pixels in image x", green,blue* $\Big)$.

- Any norm $\|.\| : V \to \mathbb{R}^m$ that fulfills the *parallelogram equation*
  - $\|x + y\|^2 + \|x - y\|^2 = 2\|x\|^2 + 2\|y\|^2$

  induces a kernel by *polarization*:
  - $k(x, y) := \frac{1}{2}\Big(\|x + y\|^2 - \|x\|^2 - \|y\|^2\Big)$.

  Example: $\mathcal{X} =$ *time series with bounded values*, $\|x\|^2 = \sum_{t=1}^{\infty} \frac{1}{2^t} x_t$

- If $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is *conditionally positive definite*, i.e.
  - $\sum_{i,j=1}^{n} t_i k(x_i, x_j) t_j \geq 0$ for any $t \in \mathbb{R}^n$ with $\sum_i t_i = 0$

  for $x_1, \ldots, x_n \in \mathcal{X}$ for any $n \in \mathbb{N}$, then
  - $k(x, x') := \exp(-d(x, x'))$ is a positive kernel.

  Example: $d(x, x') = \|x - x'\|_{L^2}^2$. $k(x, x') = e^{-\|x - x'\|_{L^2}^2}$

Constructing Kernels

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.
- if k is a kernel, then $\exp(k)$ is a kernel.

Constructing Kernels

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.
- if k is a kernel, then $\exp(k)$ is a kernel.

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* a.k.a. *RBF* $k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$ with $\sigma > 0$,

Constructing Kernels

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.
- if k is a kernel, then $\exp(k)$ is a kernel.

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* a.k.a. *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

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.
- if k is a kernel, then $\exp(k)$ is a kernel.

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* a.k.a. *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*

Not an example: $\tanh\left(a\langle x, x' \rangle + b\right)$ is *not* positive definite.

Caveat

So what, if $k(x, x')$ isn't positive definite?

Remember, SVM training is solving this optimization:

$$\min_{\alpha_i \in \mathbb{R}^+} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \qquad \text{subject to some constraints.}$$

If $k(x, x')$ is not positive definite:

- the matrix $K = \Big(k(x_i, x_j)\Big)_{i,j}$ can have negative eigenvalues,
- let $\boldsymbol{v} = (v_1, \ldots, v_n)$ be an eigenvector with eigenvalue $\lambda < 0$,
- for $\alpha_i = y_i v_i$, we have $\sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j k(x_i, x_j) = \lambda \|v\|^2$,
- for $\|v\| \to \infty$, we have $\lambda \|v\|^2 \to -\infty$, because $\lambda < 0$.

The optimization might diverge, SVM training would fail.

(but not always, it depends on the data $\to$ unexplanable behavior for future users)

Nonlinear SVMs in Practice

SVMs with non-linear kernel are commonly used for small to medium sized Computer Vision problems.

- Software packages:
  - **libSVM**: `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
  - **SVMlight**: `http://svmlight.joachims.org/`
- Training time is
  - typically **cubic** in number of training examples.
- Evaluation time:
  - typically **linear** in number of training examples.
- Classification accuracy is typically higher than with linear SVMs.

Summary – Classification with Kernels

- (Generalized) linear classification with SVMs
  - conceptually simple, but powerful by using kernels

Summary – Classification with Kernels

- (Generalized) linear classification with SVMs
  - conceptually simple, but powerful by using kernels
- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space

Summary – Classification with Kernels

- (Generalized) linear classification with SVMs
  - conceptually simple, but powerful by using kernels
- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space
- Kernelization is implicit application of a feature map
  - The method can become non-linear in the original data.
  - The method is still linear in *some* feature space.
    $\Rightarrow$ still somewhat intuitive/interpretable

Summary – Classification with Kernels

- (Generalized) linear classification with SVMs
  - conceptually simple, but powerful by using kernels
- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space
- Kernelization is implicit application of a feature map
  - The method can become non-linear in the original data.
  - The method is still linear in *some* feature space.
    $\Rightarrow$ still somewhat intuitive/interpretable
- We can build new kernels from
  - explicit inner products
  - distances
  - existing kernels

Summary – Classification with Kernels

- (Generalized) linear classification with SVMs
  - conceptually simple, but powerful by using kernels
- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space
- Kernelization is implicit application of a feature map
  - The method can become non-linear in the original data.
  - The method is still linear in *some* feature space.
    $\Rightarrow$ still somewhat intuitive/interpretable
- We can build new kernels from
  - explicit inner products
  - distances
  - existing kernels
- Kernels can be defined for arbitrary input data, not just vectors.

What did we not see?

We have skipped a large part of theory on kernel methods:

- Optimization
  - Dualization
- Numerics
  - Algorithms to train SVMs
- Statistical Interpretations
  - What are our assumptions on the samples?
- Generalization Bounds
  - Theoretic guarantees on what accuracy the classifier will have!

This and much more in standard references, e.g.

- Schölkopf, Smola: *"Learning with Kernels"*, MIT Press (50€/60$)
- Shawe-Taylor, Cristianini: *"Kernel Methods for Pattern Analysis"*, Cambridge University Press (60€/75$)

Kernels for Computer Vision

Kernels in Computer Vision

Classification
- Optical Character Recognition
- Face Recognition
- Object Recognition
- Object Category Recognition
- Action Classification
- Sign Language Recognition
- Image Retrieval

We can tackle all these by training an SVM, if

- we have training data,
- we have a suitable kernel function.

What's our data?

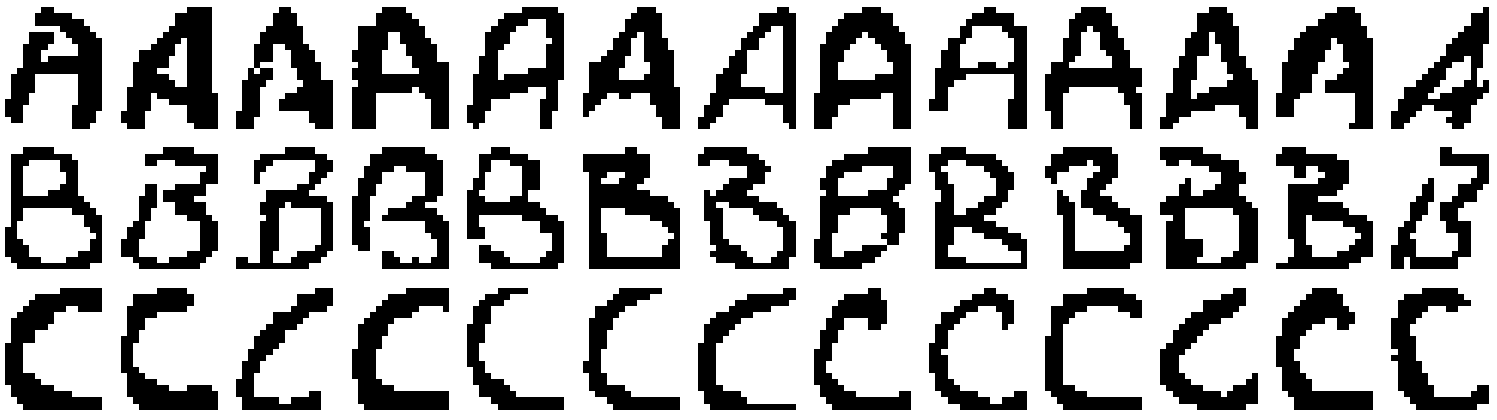
So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- Face Recognition
- Object Recognition
- Object Category Recognition
- Action Classification
- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- **Optical Character Recognition**



- Face Recognition
- Object Recognition
- Object Category Recognition
- Action Classification
- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems. Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- **Face Recognition**

- Object Recognition
- Object Category Recognition
- Action Classification
- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

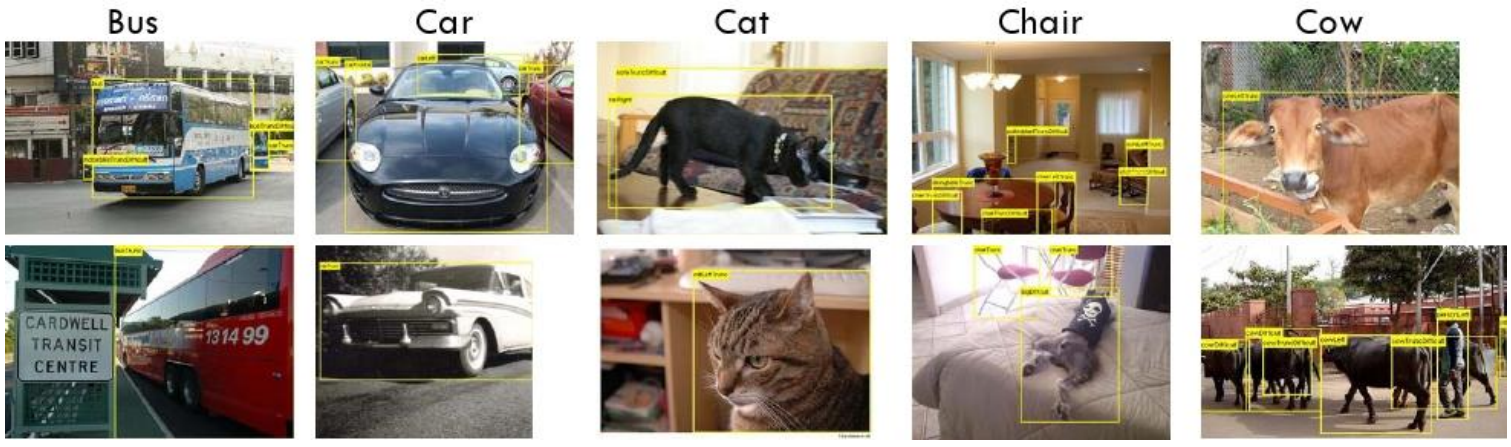
- Optical Character Recognition
- Face Recognition
- **Object Recognition**

- Object Category Recognition
- Action Classification
- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- Face Recognition
- Object Recognition
- **Object Category Recognition**



- Action Classification
- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- Face Recognition
- Object Recognition
- Object Category Recognition
- **Action Classification**

- Sign Language Recognition
- Image Retrieval

What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- Face Recognition
- Object Recognition
- Object Category Recognition
- Action Classification
- **Sign Language Recognition**

- Image Retrieval

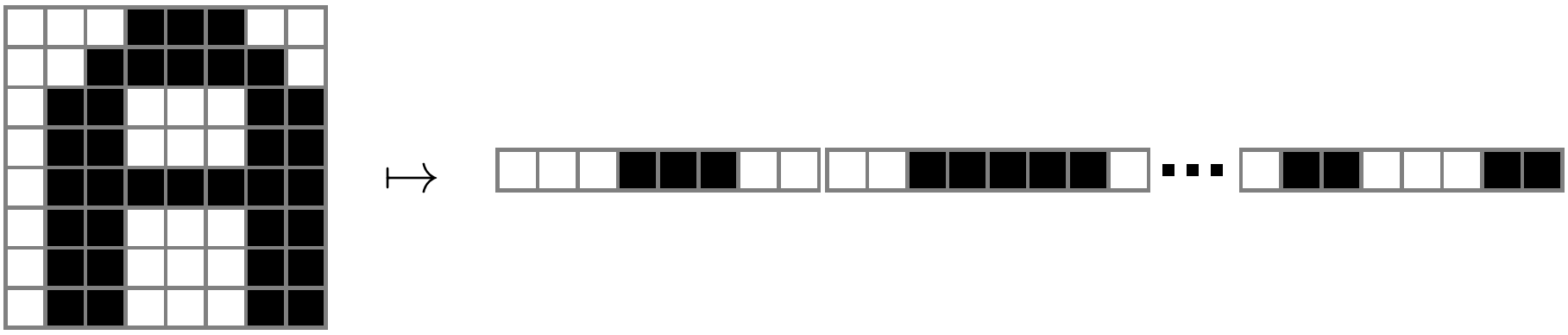
What's our data?

So far, we looked at arbitrary samples $x \in \mathcal{X}$, and abstract problems.
Now we look at typical problems where x_i are *images*:

- Optical Character Recognition
- Face Recognition
- Object Recognition
- Object Category Recognition
- Action Classification
- Sign Language Recognition
- **Image Retrieval**

Kernels on top of Pixel Representations

The easiest way to treat images is as pixel vectors:



$$\mapsto \quad 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, \cdots, 0, 1, 1, 0, 0, 0, 1, 1$$

- each image is $N \times M$ pixels, pixels are values in $[0, 255]$ or $[0, 1]$ $\Rightarrow$ images $x_1, \ldots, x_n$ are vectors in $\mathbb{R}^{NM}$.

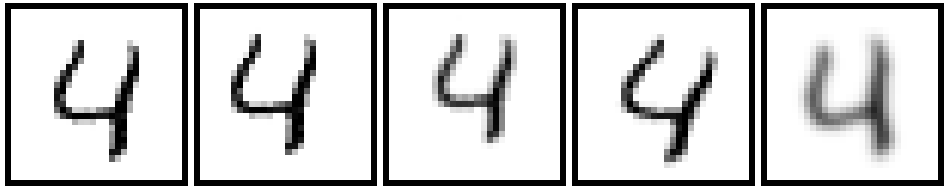
Every kernel for vectors can be used, e.g.

- $k_{lin}(x, x') = \langle x, x' \rangle = \sum_{j=1}^{NM} x^j x'^j$
- $k_{gauss}(x, x') = \exp\left(-\frac{1}{\gamma}\|x - x'\|^2\right)$ with $\gamma > 0$

Need for Invariance

Treating images directly as vectors often leads to bad classification:

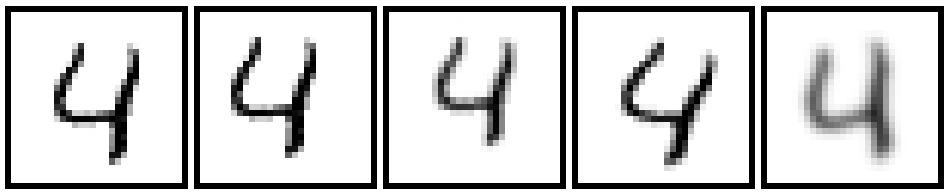
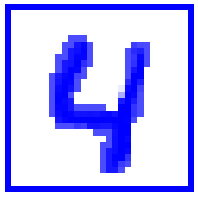
- Visually similar images can have large distance as vectors.

train: 

Need for Invariance

Treating images directly as vectors often leads to bad classification:

- Visually similar images can have large distance as vectors.

train:  test: 

- $\|x_{tst} - x_i\|$ is large for all $i = 1, \ldots, n$,

  $\Rightarrow \quad k(x_i, x_{tst}) = \exp\left(-\frac{1}{2\sigma^2}\|x_{tst} - x_i\|^2\right) \approx 0,$

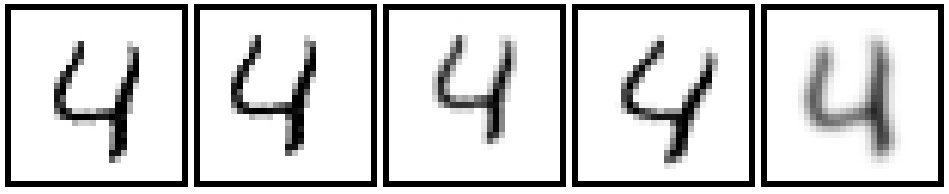
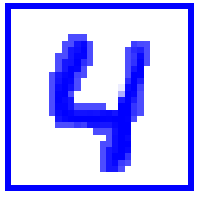
  $\Rightarrow \quad f(x_{tst}) = \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_{tst}) \approx 0,$

  $\Rightarrow$ Decision of the SVM classifier is unrealiable.

Need for Invariance

Treating images directly as vectors often leads to bad classification:

- Visually similar images can have large distance as vectors.

train:  test: 

- $\|x_{tst} - x_i\|$ is large for all $i = 1, \ldots, n$,

  $\Rightarrow \quad k(x_i, x_{tst}) = \exp\left(-\frac{1}{2\sigma^2}\|x_{tst} - x_i\|^2\right) \approx 0,$

  $\Rightarrow \quad f(x_{tst}) = \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_{tst}) \approx 0,$

  $\Rightarrow$ Decision of the SVM classifier is unrealiable.

- $k(x, x')$ should be large, iff x is visually similar to x'.

Encoding Invariance

Visual similarity is **invariant** to

- translations
- (small) rotations
- (small) scale changes
- blur
- brightness/contrast
- *in OCR:* stroke width
- *in natural images:* illuminant color
- and many more...

Encoding Invariance

Visual similarity is **invariant** to

- translations
- (small) rotations
- (small) scale changes
- blur
- brightness/contrast
- *in OCR:* stroke width
- *in natural images:* illuminant color
- and many more...

Proposed Solutions:

1) normalize the image before computing the pixel-wise distance
2) transform the image to an invariant representation
3) encode invariance into the kernel function

Encoding Invariance

Visual similarity is **invariant** to

- translations
- (small) rotations
- (small) scale changes
- blur
- brightness/contrast
- *in OCR:* stroke width
- *in natural images:* illuminant color
- and many more...

Proposed Solutions:

1) normalize the image before computing the pixel-wise distance
2) transform the image to an invariant representation
3) encode invariance into the kernel function

There is also another fix:

0) forget about invariance, but use more training data

0) More Training Data

SVM decision function is like a smart nearest neighbor classifier:

$$f(x_{tst}) = \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_{tst}) \qquad \text{with } 0 \leq \alpha_j \leq C.$$

- $f(x_{tst})$ is linear combination of y_j weighted by $\alpha_j k(x_j, x_{tst})$.
- only samples similar to x_{tst} (i.e. $k(x_j, x_{tst}) \gg 0$) influence $f(x_{tst})$
- if *none* of the training sample is similar to x, then $f(x_{tst}) \approx 0$
  $\rightarrow$ unreliable decision

More (independent) training data increase the coverage of the feature space and the chance of a reliable decision (like in k-NN)

0) Image Jittering / Virtual Samples

Increasing the training set:

- Usually, we cannot create new *independent* training data.
- But, we can create modifications of the original samples:
  - add noise,
  - apply geometric transformations (translate, rotate, mirror, ...)
- Assign distorted images the same labels as original images.

Advantage

- More *similar* prototypes also for translated/rotated test images.

Disadvantage

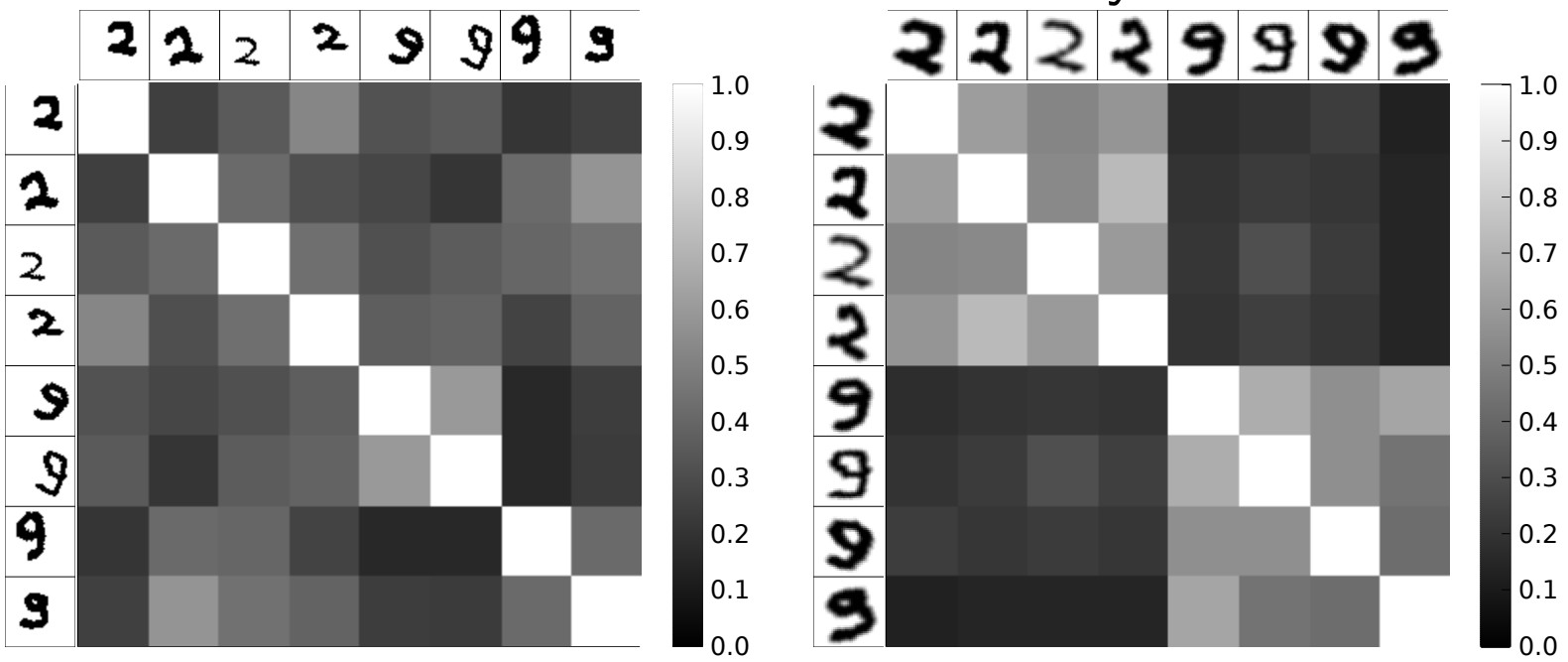
- We need to know procedures *how* to distort the images (without influencing the class it belongs to)
- In kernel methods, more data usually means slower.

1) Image Normalization

Apply an image-to-image transform that removes variations, e.g.

- resize all images to fixed size
- perform brightness/contrast normalization
- translate all images to same *center of gravity*
- rotate all images to same orientation
- calculate a *gradient/edge image*

Example: normalization leads to better similarity scores:



1) Image Normalization

Apply an image-to-image transform that removes variations, e.g.

- resize all images to fixed size
- perform brightness/contrast normalization
- translate all images to same *center of gravity*
- rotate all images to same orientation
- calculate a *gradient/edge image*

Advantage

- Output images show much less unwanted variance.

Disadvantage

- We need an *inverse map* that removes the variations.
- Typically works only for constrained domains or small images.

2) Invariant Representations

We want to apply a kernel afterwards anyway, so can apply transformations that do not output images:

- Pick any map: $\psi : \{\textit{images}\} \to \{\textit{feature representations}\}$, e.g.
  - filter responses, e.g. Fourier, Gabor, ...
  - calculate **histograms** of properties, e.g. color histogram

  such that the output is robust against unwanted image variations.

Advantage

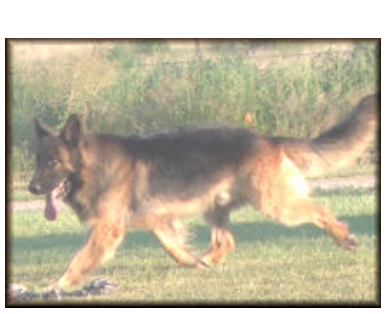
- we can easily remove many sources of unwanted image variance

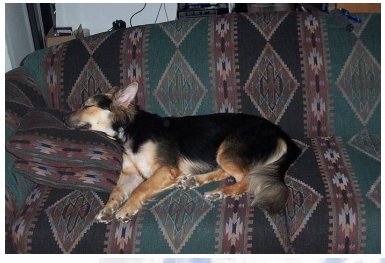
Disadvantage

- one might accidentally discard *important* sources of variance, e.g. $\mathbf{6}$ and $\mathbf{9}$ are hard to distinguish without image geometry

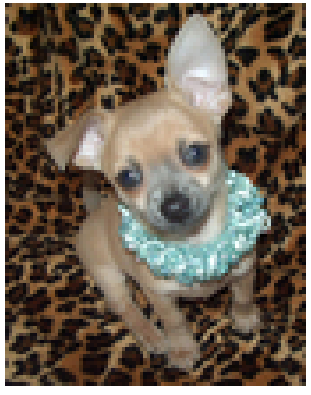
Image Representation by Local Regions

For natural images, variations even within a class can be huge:







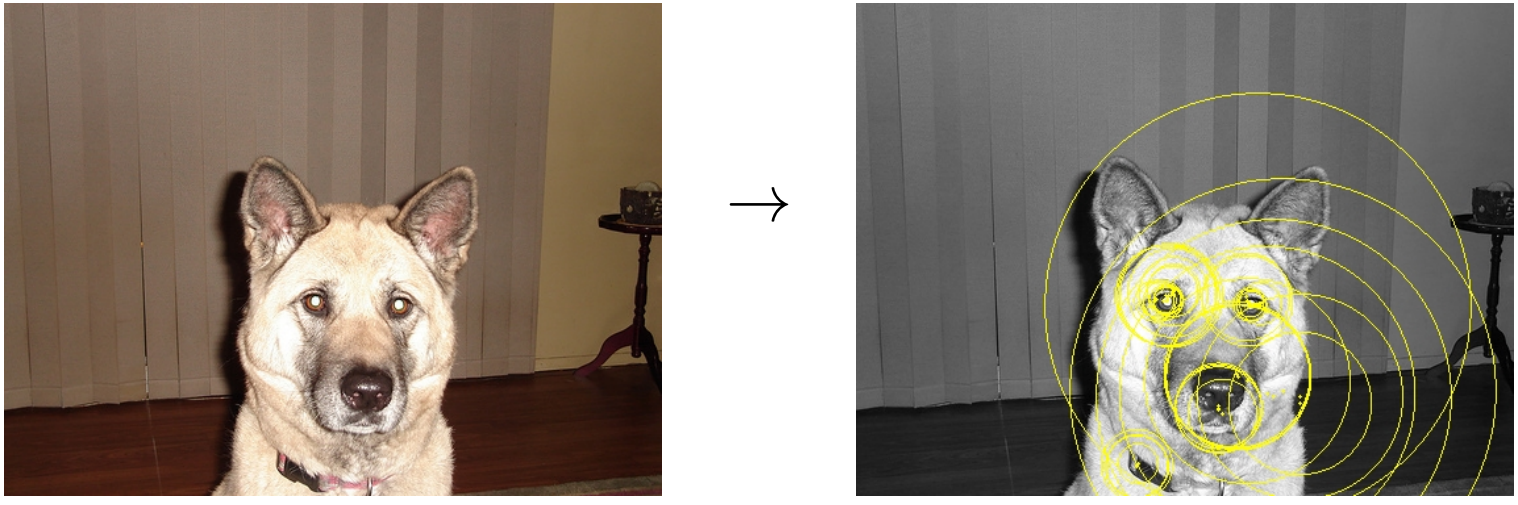
- a large part of the image is background, often more than 50%
- brightness, color etc. vary strongly over image regions.

Instead of *one* global image representation, we form many local ones.

Interest Point Operators

Typical procedure:

- Some normalization, e.g. global contrast, color correction
- Apply an interest point operator:

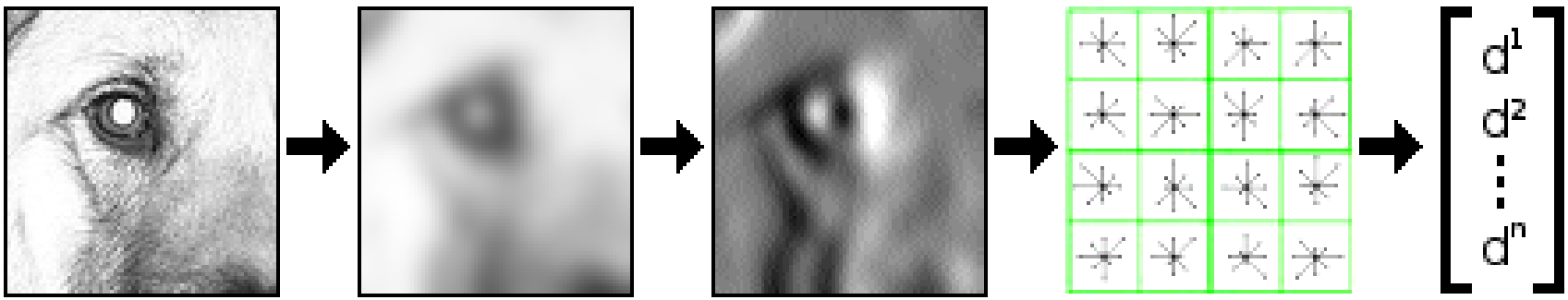


- Output: a list of regions $(\theta_j, x_j, y_j, r_j, \alpha_j)_{j=1,\ldots,n_i}$
  - θ_j is confidence or "interestingness"
  - (x_j, y_j) is the center of the region
  - σ_j is the scale of the region (e.g. radius of a circle)
  - α_j is the orientation of the region
- the number of regions n_i depends on the image I_i

Sparse Local Descriptors, e.g. SIFT

Each region of interest is a *small image*. Calculate an *invariant representation* of it:

- Normalize the region's size, optionally also orientation
- Smoothen the image (removes noise)
- Calculate the gradient image (invariance to global brightness)
- Form localized histograms of gradient directions over grid cells
- Normalize the total histogram e.g. by its L^2-norm



Output: one fixed-length descriptor per region: $d_j \in \mathbb{R}^{128}$

[D. Lowe, "Object recognition from local scale-invariant features", ICCV 1999]

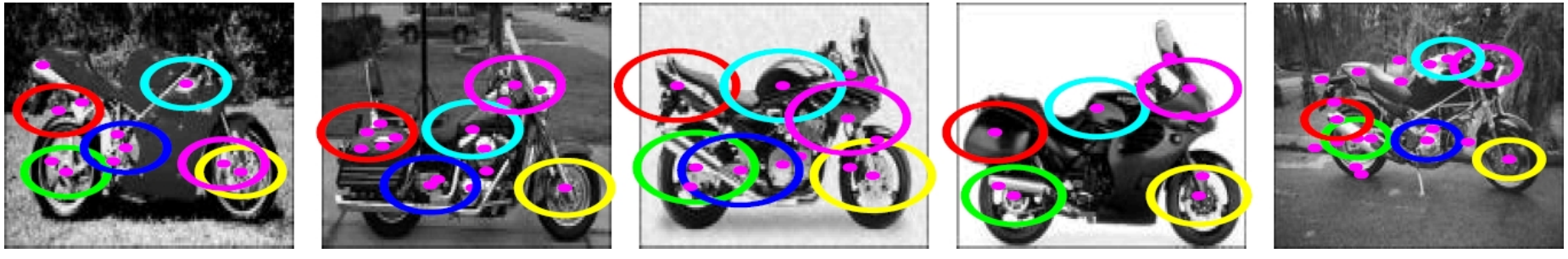
Object Classification using Local Regions

Use the *set of regions* as new representation for images:

$$I_i \mapsto (\theta_i, x_i, y_i, r_i, \alpha_i, d_i)_{i=1,\ldots,n_i}$$

Original Idea:

- Each region is a "part" of the object,
  - face: eyes, nose, mouth.
  - motorbike: wheels, saddle, handles
- An object is characterized by few characteristic parts: $n_i \approx 5$–10



[R. Fergus, P. Perona, A. Zisserman, "Object class recognition by unsupervised scale-invariant learning", CVPR 2003]

Object Classification using Local Regions

Use the *set of regions* as new representation for images:

$$I_i \mapsto (\theta_i, x_i, y_i, r_i, \alpha_i, d_i)_{i=1,\ldots,n_i}$$

Problem:

- Detecting high-level object parts in images is difficult.
- Detected regions turn out to be rather low-level: corner, "blobs"
- Similar regions occur in very different objects,
  e.g. eyes vs. wheels vs. polka dots

Bag of Local Regions

Alternative Approaches:

1) Find better "parts" e.g. by grouping of the local regions
   - probabilistic part-based models, topic models,...
2) Don't rely on few important parts, but on *statistics* of many unreliable ones: $n_i = 1000 - 50000$

Define a **kernel** between local representations with many elements:

- We know how to compare descriptors $d_i \in \mathbb{R}^d$, e.g. as vectors.
- We don't know how to compare their spatial arrangement.

Simplest ideas:

- Discard the spatial arrangements, keep only the descriptors:

$$I_i \mapsto D_i := \{d_i\}_{i=1,\ldots,n_i}$$

- Each image is represented as a set or *bag* of descriptors.

Set Kernels

To compare *sets of vectors* with each other, we define *set kernels*. Typically, they are built from simpler components, e.g. by counting

$$k_{count}(D_i, D_j) := \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} \delta_{d_i = d_j}$$

or by averaging

$$k_{sum}(D_i, D_j) := \frac{1}{n_i n_j} \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} k(d_i, d_j) \qquad \text{(geometric mean)}$$

$$k_{prod}(D_i, D_j) := \left(\prod_{i=1}^{n_i} \prod_{j=1}^{n_j} k(d_i, d_j) \right)^{\frac{1}{n_i n_j}} \qquad \text{(harmonic mean)}$$

[D. Haussler. Convolutional kernels on discrete structures. Technical Report UC Santa Cruz, 1999]

Matching "Kernel"

When many features are just background, it can be more robust not to average, but to "match":

$$d_{match}(D_i, D_j) := \frac{1}{2}\left[\hat{k}(D_i, D_j) + \hat{k}(D_j, D_i)\right]$$

$$\text{with} \quad \hat{k}(D_i, D_j) := \frac{1}{n_i}\sum_{i=1}^{n_i} \max_{j=1,\ldots,n_j} k(d_i, d_j)$$

Unfortunately, *set kernels* in general have some problems:

- They are slow to compute: kernel matrix $O(N^2 n^2 d)$
  - kernel matrix of N samples: $O(N^2)$ kernel evaluations.
  - Each kernel evaluation: compare all pairs of elements in $D_i \times D_j$: $O(n_i \cdot n_j)$
  - Each comparison: at least $O(d)$: for d-dim vectors

- Also, the match-"kernel" $d_{match}(D_i, D_j)$ is *not* positive definite.

[C. Wallraven, B. Caputo, A.B.A. Graf. "Recognition with local features: the kernel recipe", ICCV 2003.]

Breakthrough: Pyramid Match Kernel

Don't calculate all pairwise distances.
Approximate them by discretizing the feature space.

- lay a grid over the feature space
- count as a "match" if two descriptors fall into the same bin

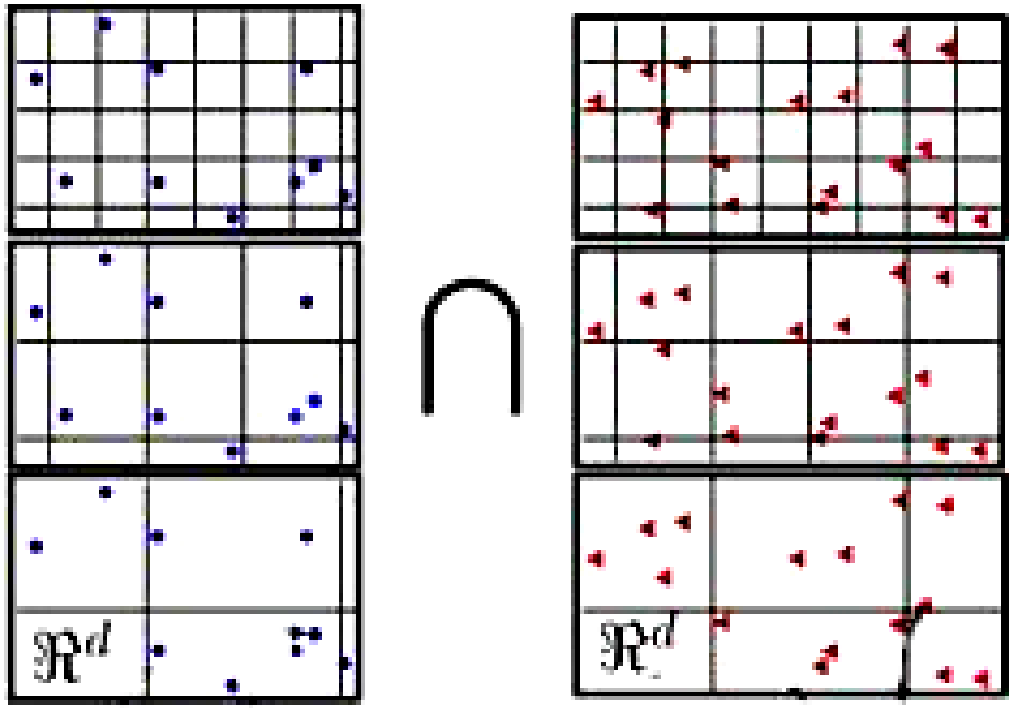
How to choose a grid size? Don't! Use a *pyramid* of grids:

- lowest level: each feature is in a bin of its own
- next level: merge neighboring cells per dimension into one cell (2^d bins in total)
- highest level: all features lie in the same bin

[K. Grauman, T. Darrell. "The pyramid match kernel: Discriminative classification with sets of image features", CVPR 2005]

Pyramid Match Kernel

- Each dot is a descriptor.
- Sum up intersection of how many descriptors fall in each bin.



Pre-compute histograms $h^{l,k}(D_i)$: number of descriptors in bin (l,k)

- Match-score of the bins: $\min\left(h^{l,k}(D_i), h^{l,k}(D_j)\right)$
- Total score is weighted sum of match-scores:

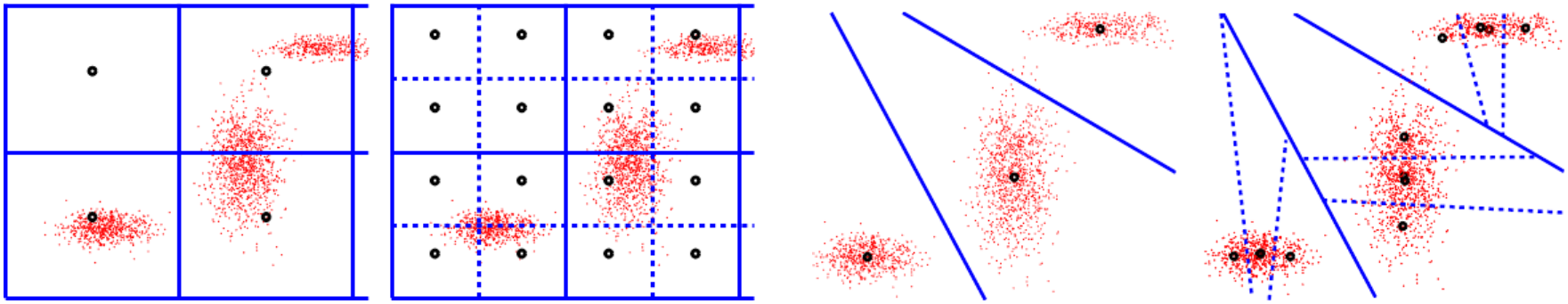
$$k_{PMK}(x,x') = \sum_l 2^l \sum_k \min\left(h^{l,k}(D_i), h^{l,k}(D_j)\right)$$

- Reduced complexity $O(nd\log R)$ where R is a "data radius"

Improvements by Codebook Usage

With many descriptors per image, they will form clusters. A rectangular grid is wasteful,

- most bins are empty.
- bins cut through natural clusters.



Better do a discretization that respects clusters in data:

- Cluster the set of all descriptors into K *codebook centers*.
- Assign each descriptor to its nearest codebook entry.
- Represent each image (set of descriptors) by a histogram of which codebook entry appeared how often.

Bag of Visual Words Representation

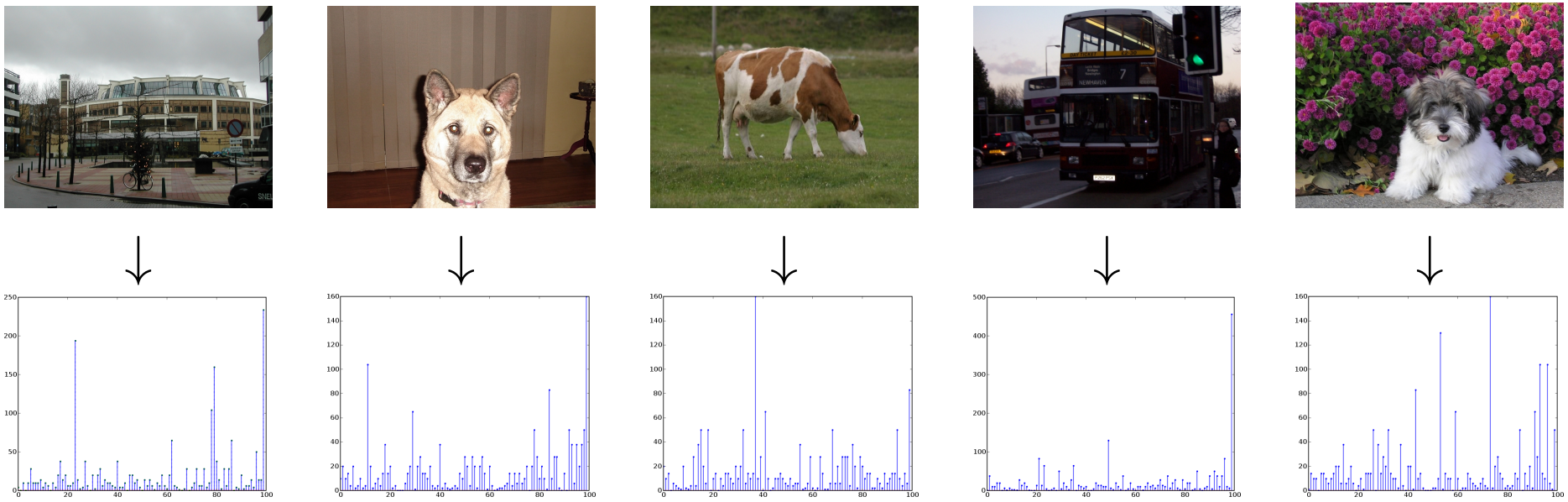
Generalized framework:

- Represent images as histograms over *any* codebook
- Compare two images by *any* histogram kernel

Bag-of-Visual-Words Representation

aka

Bag-of-Features Representation



Kernels for Comparing Histograms

Many *kernels* have been suggest for histogram data:

- We can treat K-bin histograms h, h' as vectors in $\mathbb{R}^K$:
  - $k(h, h') = \sum_j h_j h'_j$ *linear kernel*
  - $k(h, h') = (c + \sum_j h_j h'_j)^d$ *polynomial kernel*
  - $k(h, h') = \exp\left(-\frac{1}{\gamma} \sum_j \|h_j - h'_j\|^2\right)$ *Gaussian kernel*

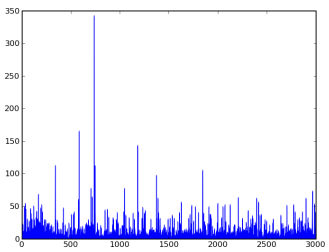
- If we normalize them (to sum 1), we can treat histograms as discrete probability distributions:
  - $k_{HI}(h, h') = \sum_j \min(h_j, h'_j)$.
  - $k_{bhattacharya}(h, h') = \sum_j \sqrt{h_j h'_j}$
  - $k_{symKL}(h, h') = \exp\left(-\frac{1}{2}(KL(h|h') + KL(h'|h))\right)$
    where $KL(h|h') = \sum_j h_j \log \frac{h_j}{h'_j}$
  - $k_{\chi^2}(h, h') = \exp\left(-\frac{1}{\gamma}\chi^2(h, h')\right)$ with $\chi^2(h, h') = \sum_j \frac{(h_j - h'_j)^2}{h_j + h'_j}$.

[M. Hein, O. Bousquet, "Hilbertian metrics and positive definite kernels on probability measures", AISTATS 2005]

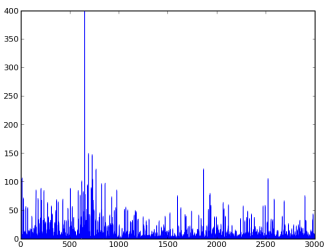
Robust Kernels

Especially k_{HI} and k_{χ^2} seem to work very well for computer vision:

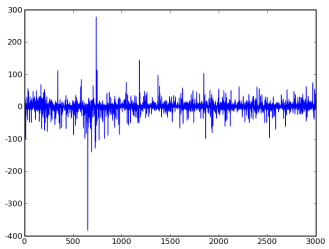
$$k_{HI}(h, h') = \sum_j \min(h_j, h'_j) \qquad k_{\chi^2}(h, h') = \exp\left(-\frac{1}{\gamma}\sum_j \frac{(h_j - h'_j)^2}{h_j + h'_j}\right).$$



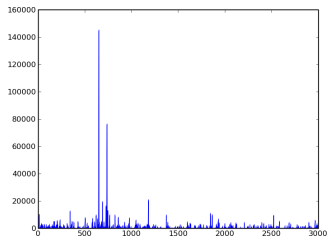
h_j



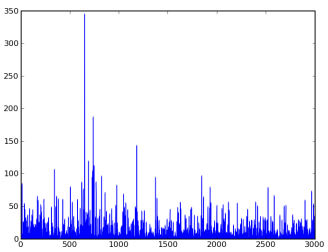
h'_j



$h_j - h'_j$



$(h_j - h'_j)^2$



$\frac{(h_j - h'_j)^2}{h_j + h'_j}$

- Feature-histograms have few large and many small entries.
- Quadratic measures (L^2 or Gaussian kernel) concentrate on the largest differences: 3 bins (out of 3000) contribute 25%
- $1st$-order (HI or χ^2-kernel) consider bins more balanced: 3 largest terms contribute 3.5%

Tricks of the Trade: Normalization

We formed histograms as raw counts of feature points:

- Their number of entries can vary strongly:
  - Normalize each histogram by its L^1 or L^2 norm:

$$\hat{h}_j := \frac{1}{n_1} h_j \qquad \text{with } n_1 = \sum_j h_j$$

$$\hat{h}_j := \frac{1}{n_2} h_j \qquad \text{with } n_2 = \sqrt{\sum_j h_j^2}$$

  - Suppress strong peaks in the histogram by non-linear *preprocessing*, e.g.

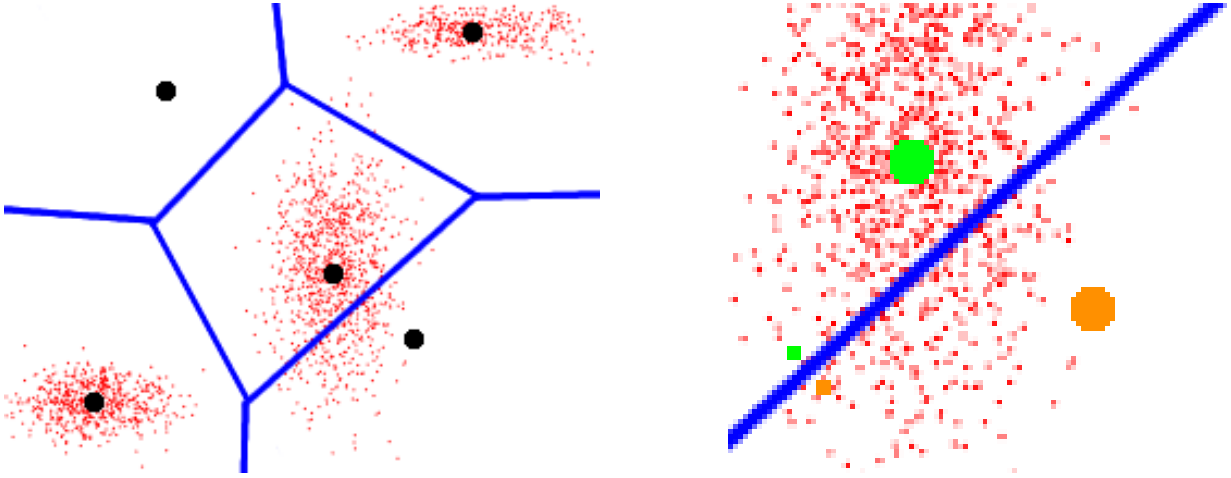
$$\hat{h}_j := \sqrt{h_j}, \qquad \hat{h}_j := \sqrt[3]{h_j}, \qquad \hat{h}_j := \begin{cases} 1 & \text{for } h_j > \theta_j \\ 0 & \text{else.} \end{cases}$$

- No golden rule how to normalize/preproces
  - rough hint: adjust normalization+kernel for scaling invariance.

Beyond Bag of Visual Words

The bag of visual word concept has drawbacks:

1) Quantization artifacts



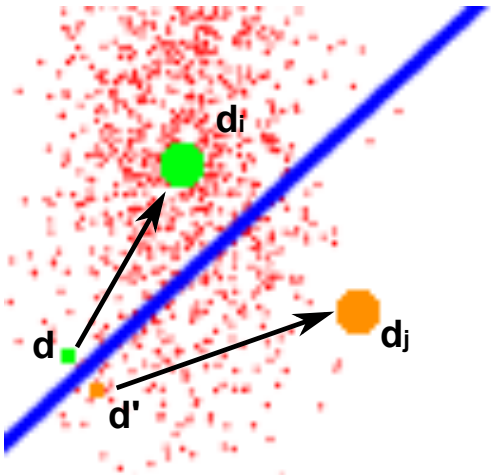
2) No geometric information

1) Soft quantization

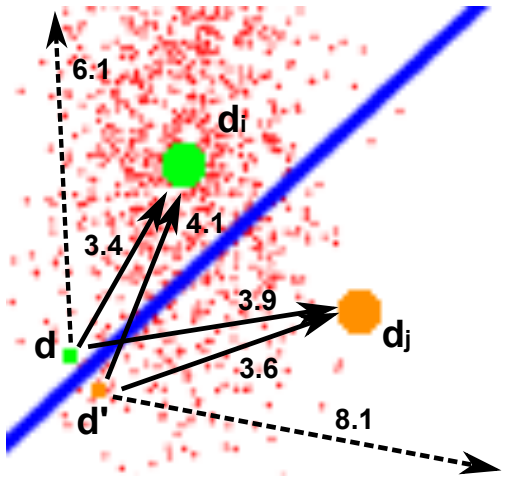
Problem: similar features are assigned to different bins

Idea: soft quantization

- for each feature vector d, compute k nearest codebook entries
- compute weights: $a_k = \exp(-\lambda \| d - d_k \|^2)$, $\tilde{a}_k = a_k / \sum_l a_l$
- soft-assign d to k histogram bins with weights $\tilde{a}_k$



$$h(d) = (0,\, 0,\, 1,\, 0, \dots)$$
$$h(d') = (0,\, 1,\, 0,\, 0, \dots)$$
$$k_{HI}(h(d), h(d')) = 0$$



$$h(d) = (0.1,\, 0.4,\, 0.5,\, 0.0, \dots)$$
$$h(d') = (0.0,\, 0.5,\, 0.4,\, 0.1, \dots)$$
$$k_{HI}(h(d), h(d')) = 0.8$$

[Farquhar, Szedmak, Meng, Shawe-Taylor. Improving "bag-of-keypoints" image categorisation. TR Southampton, 2005]

1') Gaussian Mixture Model

Idea: Use **Gaussian Mixture Model** instead of histogram:

- Histograms represent *probability distributions*.
- Gaussian mixture models do this as well, but more powerfully:

$$p_{GMM}(d) = \sum_{k=1}^{K} a_k \mathcal{G}(d; \mu_k, \Sigma_k) \quad \text{with} \sum_k a_k = 1.$$

with $\mathcal{G}(d; \mu_k, \Sigma_k) = \frac{1}{(2\pi|\Sigma|)^{m/2)}} e^{-\frac{1}{2}(d-\mu_k)^t \Sigma_k^{-1}(d-\mu_k)}$.
(can also include more than descriptors, e.g. (x, y)-coordinates)

- bag-of-words: use k-means to create codebook $d_1, \ldots, d_K$
- GMM: *expectation maximization (EM)* to learn maximum-likelihood parameters, $\lambda = \{a_k, \mu_k, \Sigma_k\}_k$,

1') Gaussian Mixture Model

Idea: Use **Gaussian Mixture Model** instead of histogram:

- Histograms represent *probability distributions*.
- Gaussian mixture models do this as well, but more powerfully:

$$p_{GMM}(d) = \sum_{k=1}^{K} a_k \mathcal{G}(d; \mu_k, \Sigma_k) \quad \text{with} \sum_k a_k = 1.$$

with $\mathcal{G}(d; \mu_k, \Sigma_k) = \frac{1}{(2\pi|\Sigma|)^{m/2)}} e^{-\frac{1}{2}(d-\mu_k)^t \Sigma_k^{-1}(d-\mu_k)}$.
(can also include more than descriptors, e.g. (x, y)-coordinates)

- bag-of-words: use k-means to create codebook $d_1, \ldots, d_K$
- GMM: *expectation maximization (EM)* to learn maximum-likelihood parameters, $\lambda = \{a_k, \mu_k, \Sigma_k\}_k$,

New problem: how to compare images w.r.t. a GMM?

1') Fisher Kernel

Using the GMM we can compute the **Fisher Kernel**:

For image $x = \{d_1, \ldots, d_m\}$, compute *log-likelihood gradient vector*

$$G_\lambda(x) := \frac{1}{m} \sum_{i=1}^{m} \nabla_\lambda \log p_{GMM}(d_i)$$

and the Fisher information matrix

$$F_\lambda := \mathbb{E}_{d \sim p_{GMM}(d)} \left\{ [\nabla_\lambda \log p_{GMM}(d)][\nabla_\lambda \log p_{GMM}(d)]^t \right\}$$

Then the Fisher kernel between images x and x' is:

$$k(x, x') := G_\lambda(x)^t F_\lambda^{-1} G_\lambda(x')$$

Looks terrible, but is actually not that bad (at least for diagonal Σ).

[T. Jaakkola, D. Haussler: Exploiting generative models in discriminative classifiers. NIPS 1999]
[F. Perronnin and C. Dance, Fisher Kernels on Visual Vocabularies for Image Categorization, CVPR 2007]

2) Incorporating Geometry

- For some object classes, *geometry* is a very strong cue:



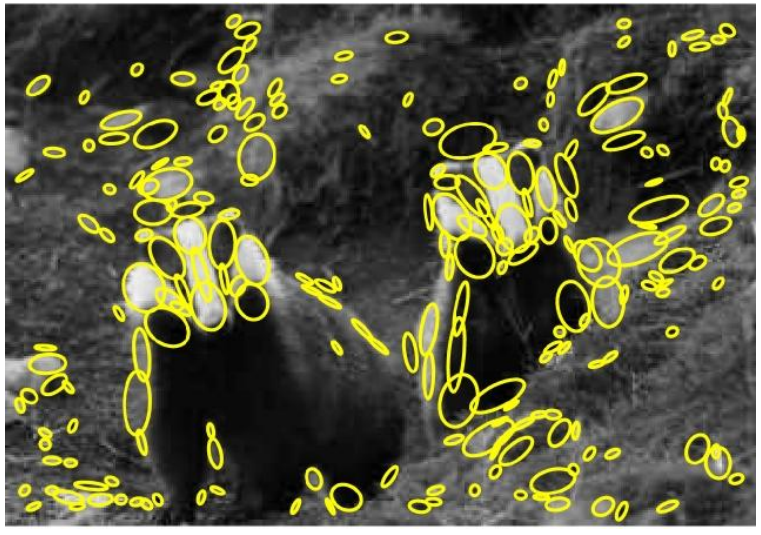

- The bag-of-features representation ignores all spatial relations:
- These images would get very similar representations:

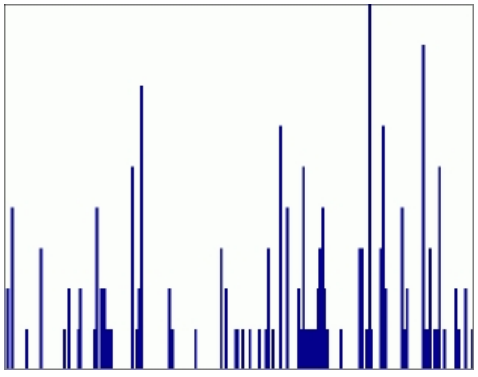


- How can we incorporate the spatial layout?

Spatial Pyramid Features

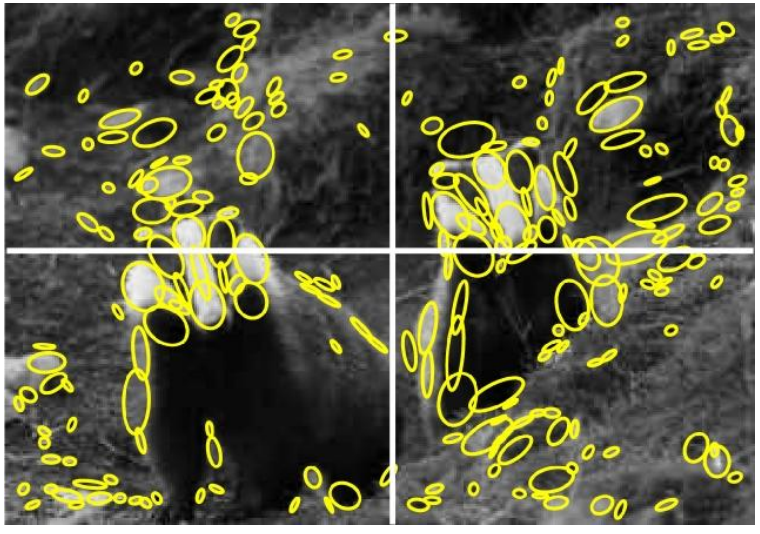
Local histograms trick: *bag-of-features* representation for subregions

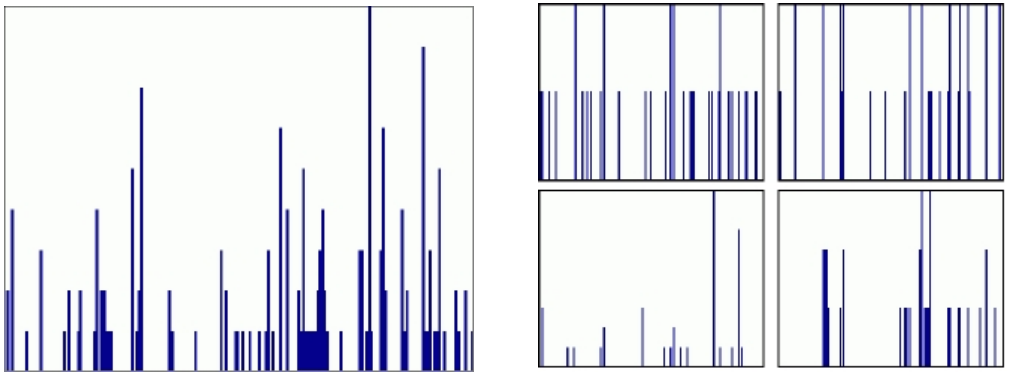




Spatial Pyramid Features

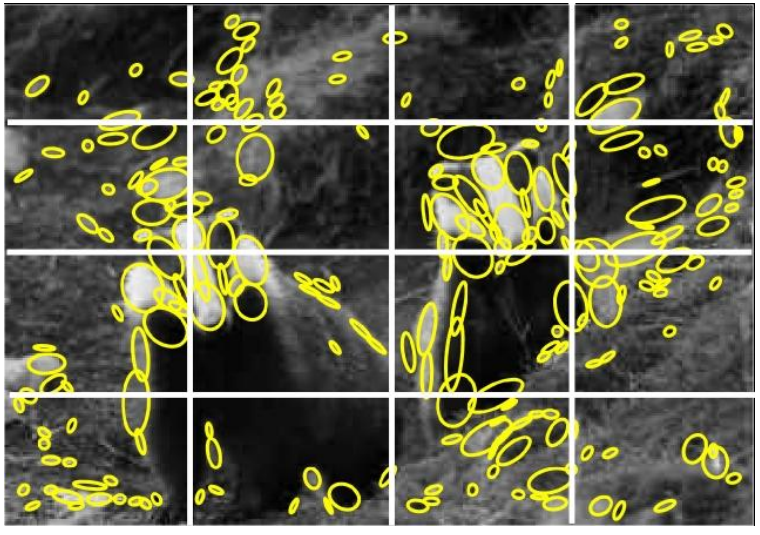
Local histograms trick: *bag-of-features* representation for subregions

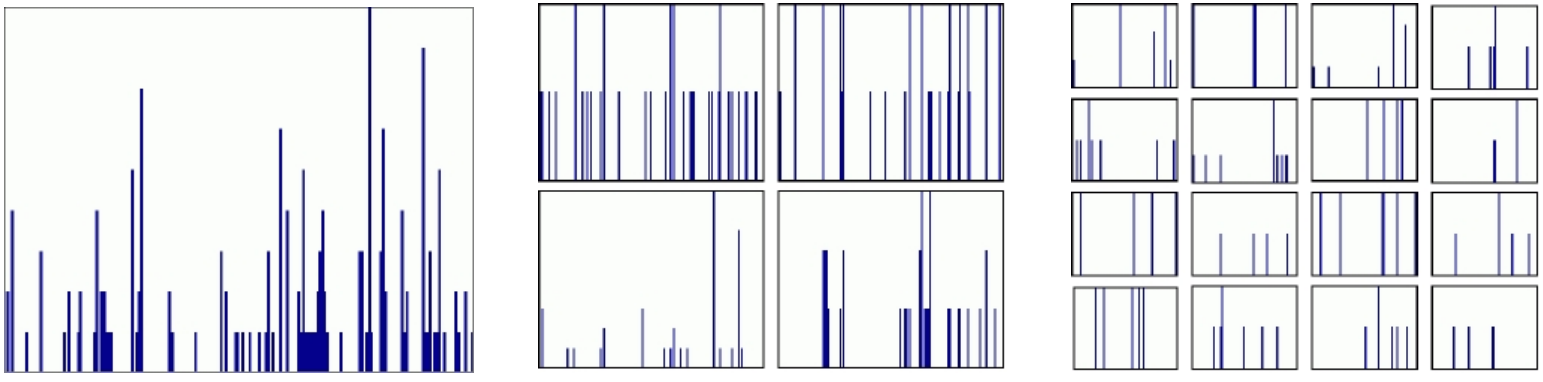




Spatial Pyramid Features

Local histograms trick: *bag-of-features* representation for subregions





$h^{0,0}$ $h^{1,0}, \ldots, h^{1,3}$ $h^{2,0}, \ldots, h^{2,15}$

Combine per-level kernels: $k_{pyramid}(h, h') = \frac{1}{2^L} \sum_{l,k} 2^l \, k(h^{l,k}, h'^{l,k})$

Examples: Scene Classification

- SIFT descriptors, 3×1 spatial pyramid, χ^2-kernel SVM

[S. Lazebnik, C. Schmid, J. Ponce: "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories", CVPR 2006]

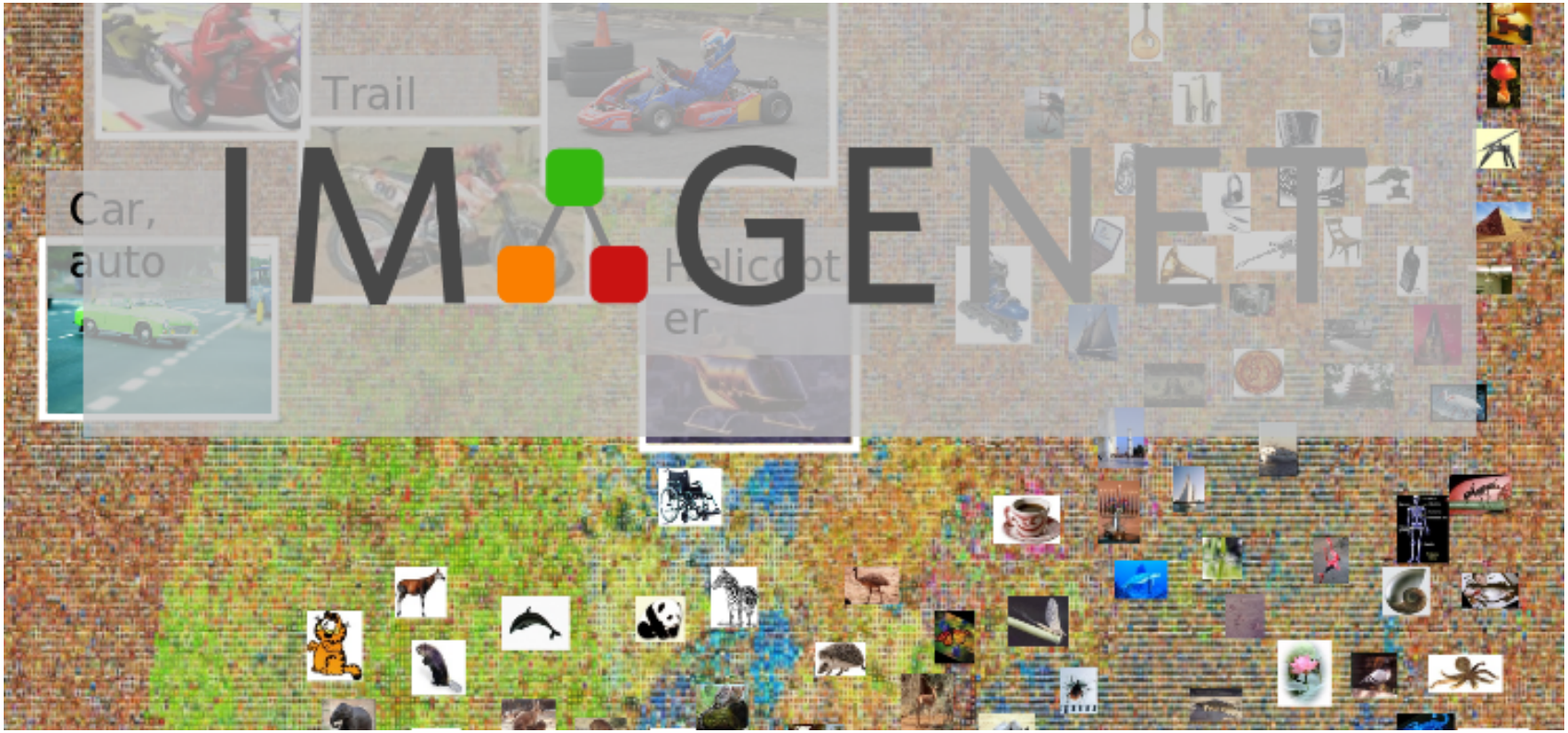
Examples: Action Classification

- Spatio-temporal descriptors, spatio-temporal pyramid, χ^2-RBF-kernel SVM

[I. Laptev, M. Marszalek, C. Schmid, B. Rozenfeld, "Learning realistic human actions from movies", CVPR 2008]

Examples: Object Classification

- colorSIFT descriptors, Fisher kernel (explicit φ), linear SVM

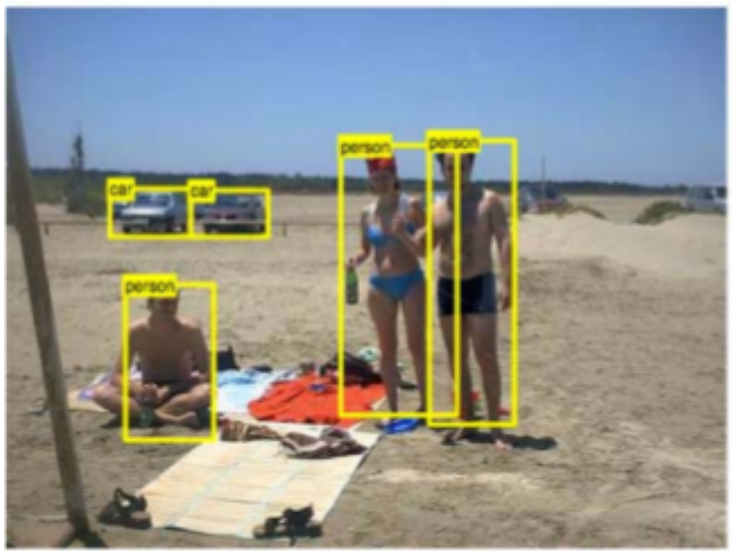


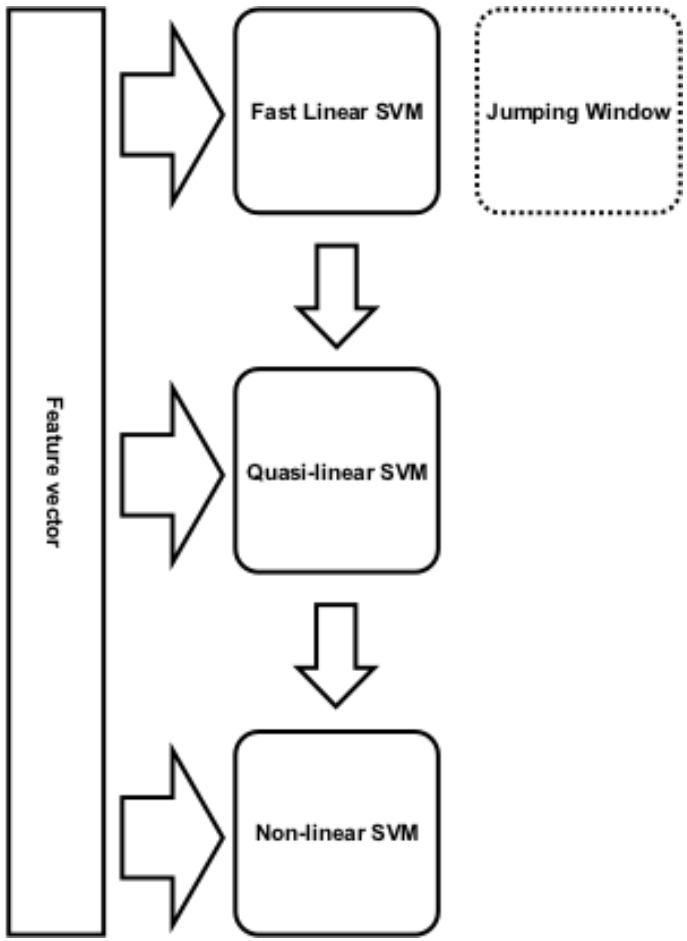
(> 10 Mio images, > 10000 classes)

[F. Perronnin, J. Sanchez and T. Mensink, "Improving the Fisher kernel for Large-Scale Image Classification", ECCV, 2010]

Examples: Object Detection/Localization

- three stage cascade: linear, χ^2 (explicit φ), RBF-χ^2-SVM
- combine multiple features: SIFT BOW, PHOG, SSIM, ...





[A. Vedaldi, A. Zisserman, *"Efficient Additive Kernels via Explicit Feature Maps"*, CVPR 2010]

Linear or Non-linear?

Observation 1: Linear SVMs are **very fast** in training and evaluation.

Observation 2: Non-linear kernel SVMs give better results, but do not scale well (with respect to number of training examples)

Can we combine the strengths of both approaches?

Linear or Non-linear?

Observation 1: Linear SVMs are **very fast** in training and evaluation.

Observation 2: Non-linear kernel SVMs give better results, but do not scale well (with respect to number of training examples)

Can we combine the strengths of both approaches?

Yes! By (approximately) going back to explicit feature maps.

[A. Rahimi, *"Random Features for Large-Scale Kernel Machines"*, NIPS, 2008]
[A. Vedaldi, A. Zisserman, *"Efficient additive kernels via explicit feature maps"*, PAMI 2012]

(Approximate) Explicit Feature Maps

Reminder:

- For every kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists an implicit $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle.$$

- In case that $\varphi : \mathcal{X} \to \mathbb{R}^D$, training a kernelized SVMs yields the same prediction function as
  - preprocessing the data: make every x into a $\varphi(x)$,
  - training a linear SVM on the new data.

(Approximate) Explicit Feature Maps

Reminder:

- For every kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists an implicit $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle.$$

- In case that $\varphi : \mathcal{X} \to \mathbb{R}^D$, training a kernelized SVMs yields the same prediction function as
  - preprocessing the data: make every x into a $\varphi(x)$,
  - training a linear SVM on the new data.

Problem: φ is generally unknown, and it does not map to $\mathbb{R}^D$.

(Approximate) Explicit Feature Maps

Reminder:

- For every kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists an implicit $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle.$$

- In case that $\varphi : \mathcal{X} \to \mathbb{R}^D$, training a kernelized SVMs yields the same prediction function as
  - preprocessing the data: make every x into a $\varphi(x)$,
  - training a linear SVM on the new data.

Problem: φ is generally unknown, and it does not map to $\mathbb{R}^D$.

Idea: Approximate $\varphi : \mathcal{X} \to \mathcal{H}$ by explicit $\tilde{\varphi} : \mathcal{X} \to \mathbb{R}^D$ such that

$$k(x, x') \approx \langle \tilde{\varphi}(x), \tilde{\varphi}(x') \rangle$$

State of the art for large-scale non-linear learning.

Example: exact explicit feature map for Hellinger kernel

$$k_H(x, x') = \sum_{i=1}^{d} \sqrt{x_i x_i'} \qquad \text{for } x = (x_1, \ldots, x_d).$$

Define

$$\varphi_H(x) := \left(\sqrt{x_1}, \ldots, \sqrt{x_d}\right)$$

then

$$\langle \varphi_H(x), \varphi_H(x') \rangle = \sum_{i=1}^{d} \sqrt{x_i}\,\sqrt{x_i'} = \sum_{i=1}^{d} \sqrt{x_i x_i'} = k_H(x, x')$$

Example: exact explicit map for Fisher-kernel

$$k_{\mathit{Fisher}}(x, x') = G_\lambda(x)^t F_\lambda^{-1} G_\lambda(x')$$

with $$G_\lambda(x) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\lambda \log p_{GMM}(d_i) \qquad \text{for } x = \{d_1, \ldots, d_m\},$$

$$F_\lambda = \mathbb{E}_{d \sim p_{GMM}(d)} \left\{ [\nabla_\lambda \log p_{GMM}(d)] [\nabla_\lambda \log p_{GMM}(d)]^t \right\}.$$

$$\varphi(x) := (F_\lambda)^{-1/2} G_\lambda(x) \qquad (F_\lambda \text{ symmetric and pos.def.})$$

Then $$\begin{aligned} \langle \varphi(x), \varphi(x') \rangle &= [F_\lambda^{-1/2} G_\lambda(x)]^t F_\lambda^{-1/2} G_\lambda(x') \\ &= G_\lambda(x) F_\lambda^{-1/2} F_\lambda^{-1/2} G_\lambda(x') \\ &= k_{\mathit{Fisher}}(x, x') \end{aligned}$$

Example: approximate feature map for Histgram Intersection

$$k_{HI}(x, x') = \sum_{i=1}^{d} \min(x_i, x_i') \qquad \text{for } x = (x_1, \ldots, x_d) \in [0,1]^d$$

Let $\varphi : [0,1] \to \{f : [0,1] \to [0,1]\}$: $\varphi(a) = H_a(x) = [\![x < a]\!]$. Then

$$\min(a, b) = \int_0^1 H_a(x) H_b(x)\ dx$$

Approximate H by finite-dimensional $h : [0,1] \to \mathbb{R}^m$

$$h(a) = \frac{1}{\sqrt{m}} (\underbrace{1, 1, \ldots, 1}_{k}, 0, \ldots, 0) \quad \text{for } \frac{k}{m} \leq a < \frac{k+1}{m}$$

Then $\quad k_{HI}(x, x') \approx \sum_{i=1}^{d} \langle h(x_i), h(x_i') \rangle_{\mathbb{R}^m}$

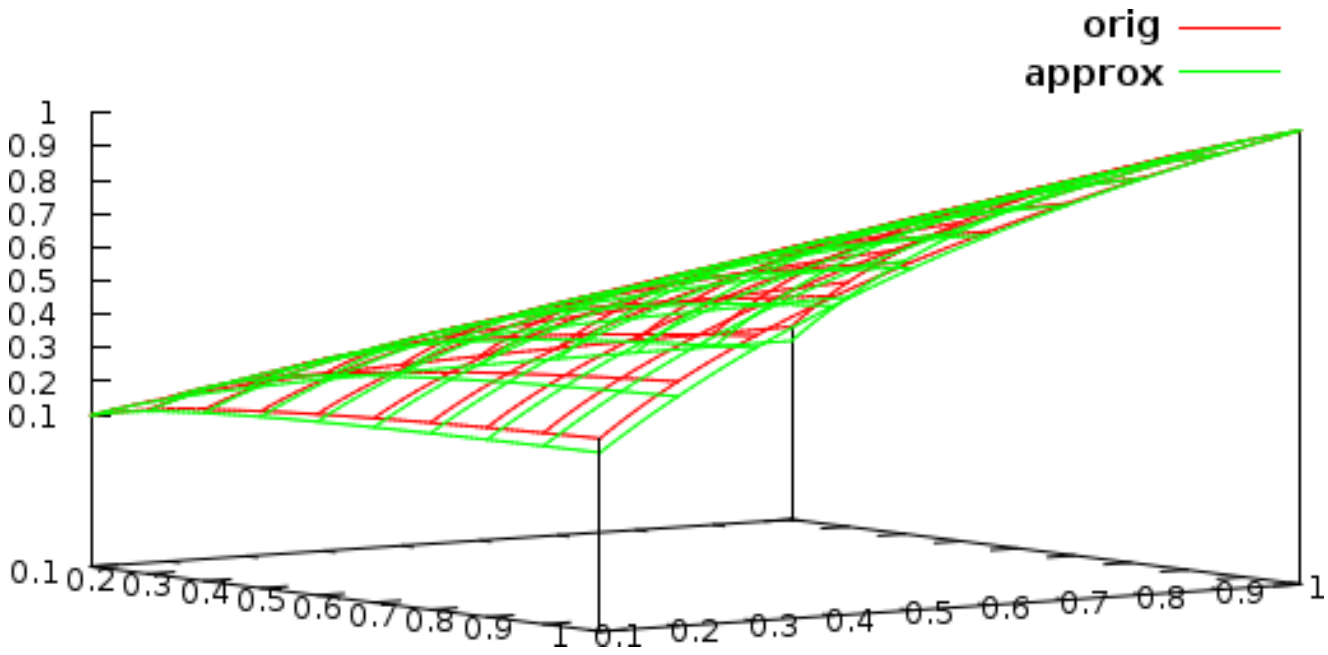
[S. Maji and A. C. Berg, "Max-margin additive classifiers for detection", ICCV 2009]

Example: approximate feature map for χ^2-kernel

$$k_{\chi^2}(x, x') = \sum_{i=1}^{d} \frac{2x_i x_i'}{x_i + x_i'} \quad \approx \quad \langle \tilde{\varphi}(x), \tilde{\varphi}(x') \rangle \quad \text{for}$$

$$\tilde{\varphi}(x) := \Big(h(x_1), \ldots, h(x_d) \Big) \in \mathbb{R}^{3d}, \quad \text{with}$$

$$h(a) := \Big(0.8\sqrt{a},\ 0.6\sqrt{a}\cos(0.6 \log a),\ 0.6\sqrt{a}\sin(0.6 \log a) \Big) \in \mathbb{R}^3$$



[A. Vedaldi, A. Zisserman, *"Efficient Additive Kernels via Explicit Feature Maps"*, CVPR 2010]

Kernels for Computer Vision

- We can see Kernels as similarity measures
  - Visually similar objects x, x' should have large $k(x, x')$,
  - Visually dissimilar objects x, x' should have small $k(x, x')$.
- Most pixel-based kernel function are too strict.
- Good kernels integrate domain knowledge, e.g. invariances.

How to Integrate Invariance in Kernels

- normalization
- invariant representation + "normal kernel"
  - (local) histograms for (partial) geometric invariance

Successful Kernel Functions for Histograms

- first order in difference of entries: "heavy tail" distribution
  - histogram intersection kernel, RBF-χ^2-kernel
  - explicit feature maps make (some) non-linear SVMs efficient

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

Remark: Model Selection makes a difference!

- Action Classification, KTH dataset

Method	**Accuracy** (on test data)
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)

Message: never rely on default parameters!

Kernel Parameter Selection

Rule of thumb for kernel parameters

- For generalize Gaussian kernels:

$$k(x, x') = \exp(-\frac{1}{2\gamma} d^2(x, x'))$$

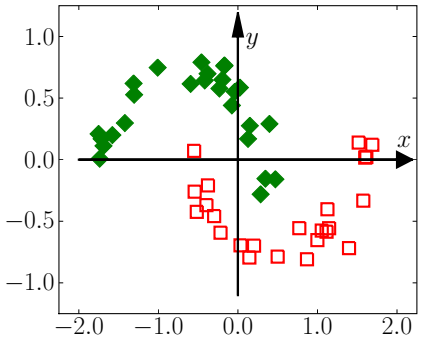
  with any distance d, set

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

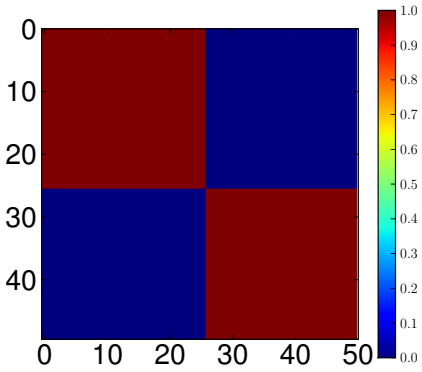
- Many variants:
  - *mean* instead of *median*
  - only $d(x_i, x_j)$ with $y_i \neq y_j$...
- In general, if there are several classes, then the *kernel matrix*:

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

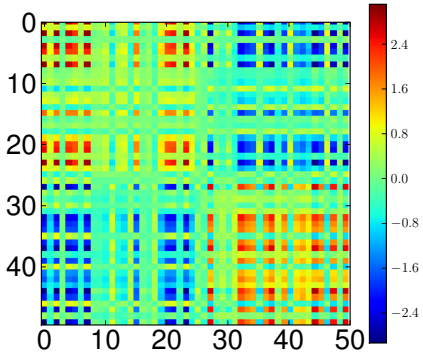
  should have a *block structure* w.r.t. the classes.



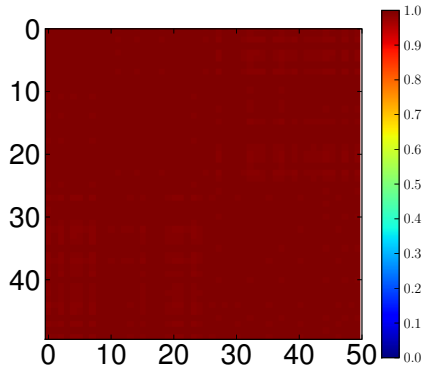
two moons



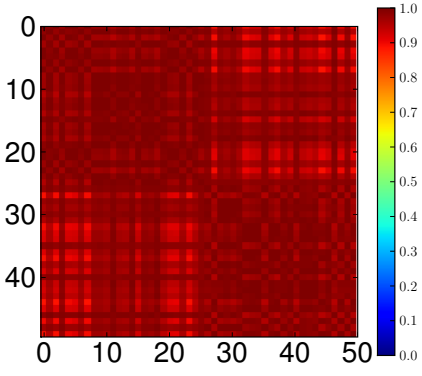
label “kernel”



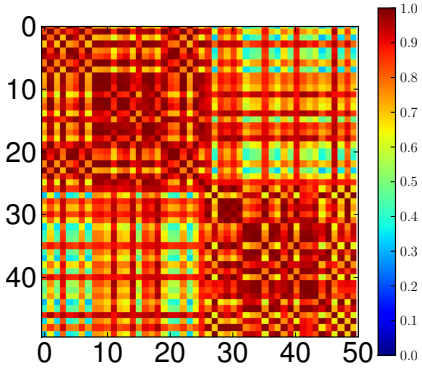
linear



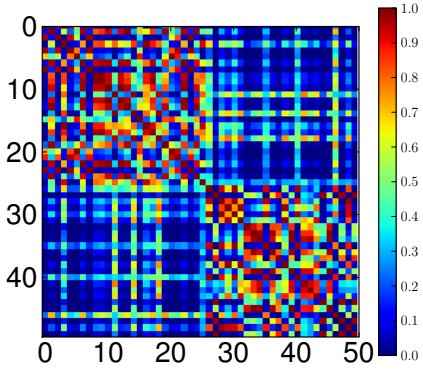
Gauss: $\gamma = 0.001$



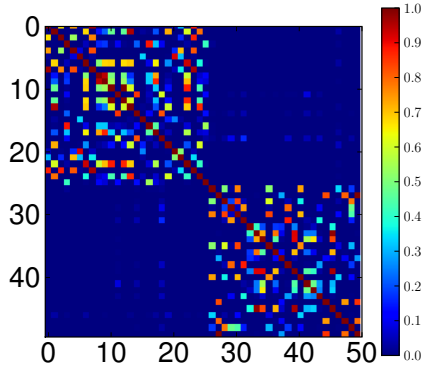
$\gamma = 0.01$



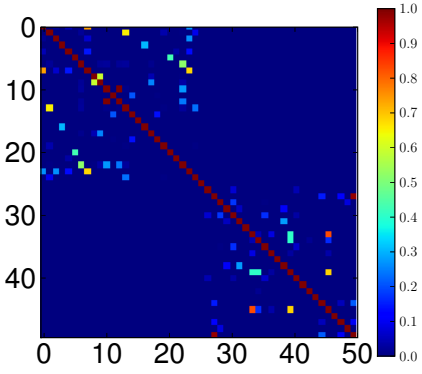
$\gamma = 0.1$



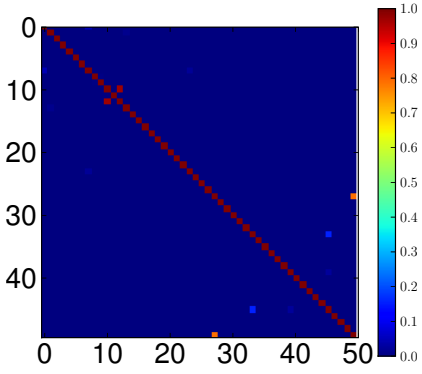
$\gamma = 1$



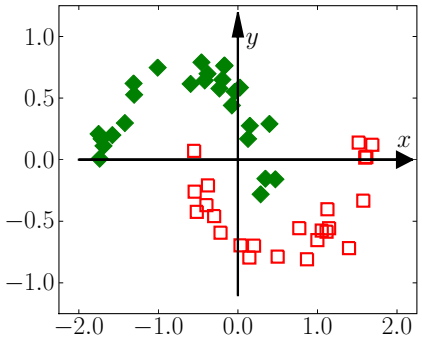
$\gamma = 10$



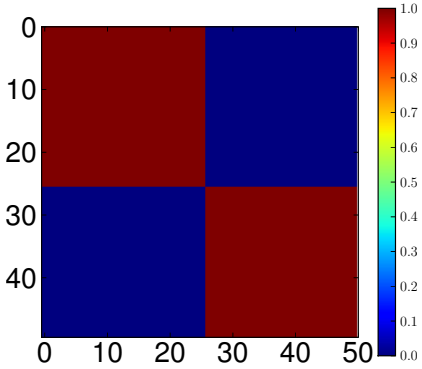
$\gamma = 100$



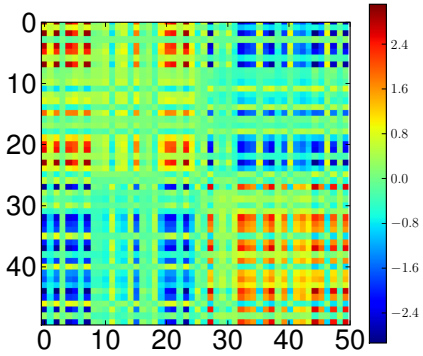
$\gamma = 1000$



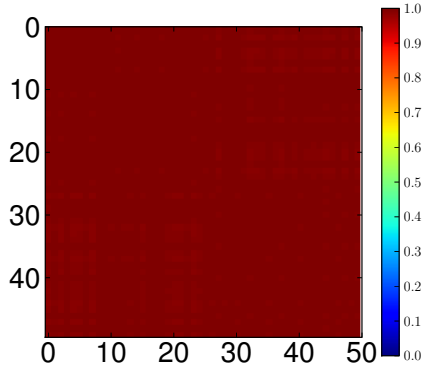
two moons



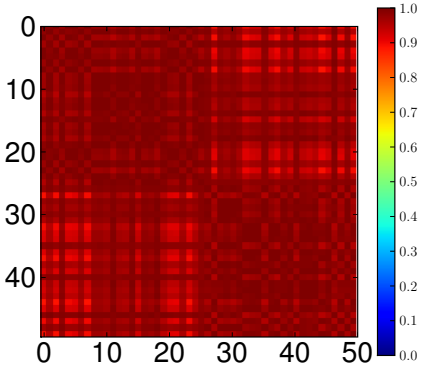
label “kernel”



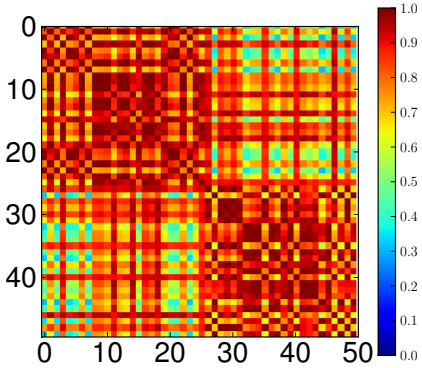
linear



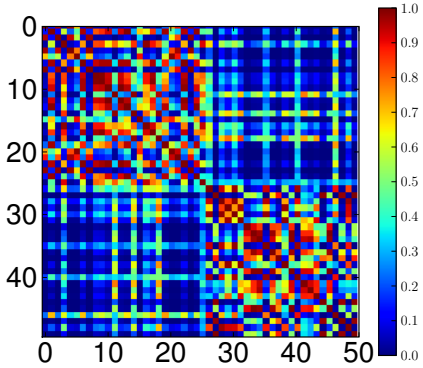
Gauss: $\gamma = 0.001$



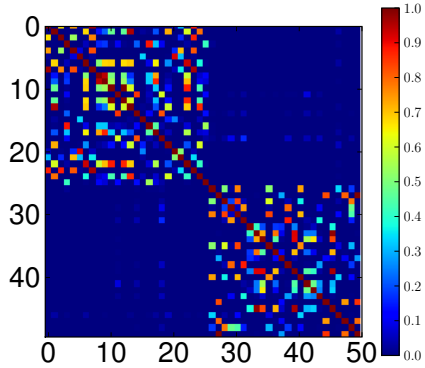
$\gamma = 0.01$



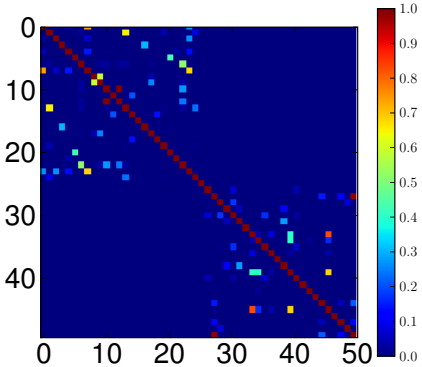
$\gamma = 0.1$



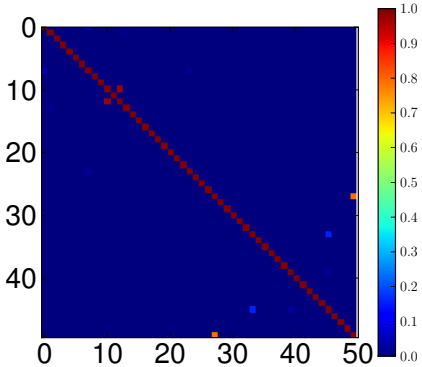
$\gamma = 1$



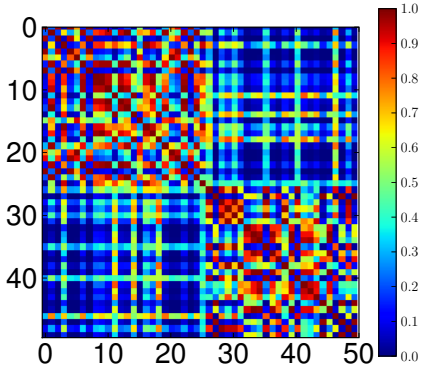
$\gamma = 10$



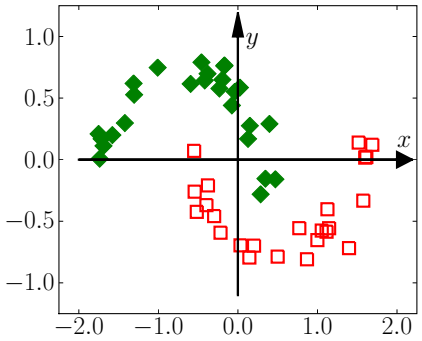
$\gamma = 100$



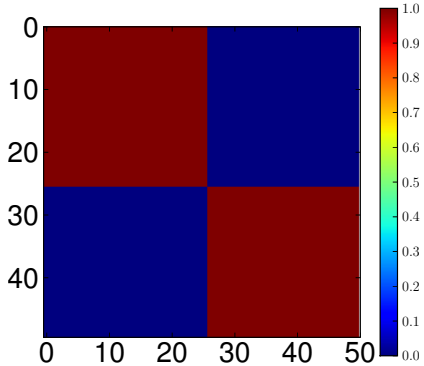
$\gamma = 1000$



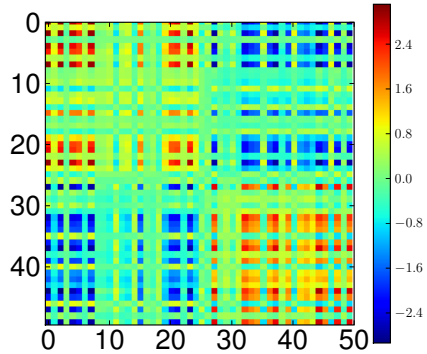
$\gamma = 0.6$
rule of thumb



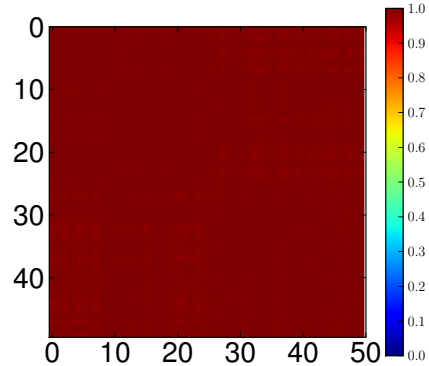
two moons



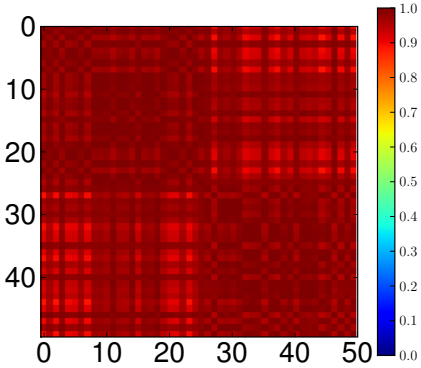
label "kernel"



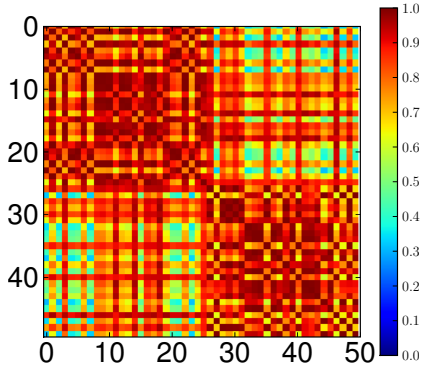
linear



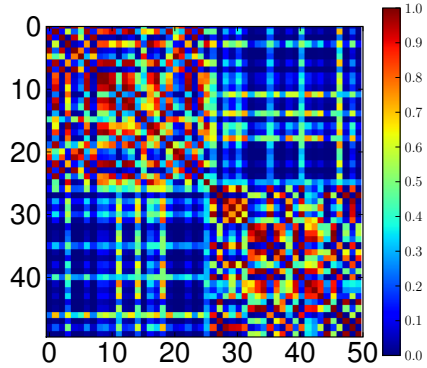
Gauss: $\gamma = 0.001$



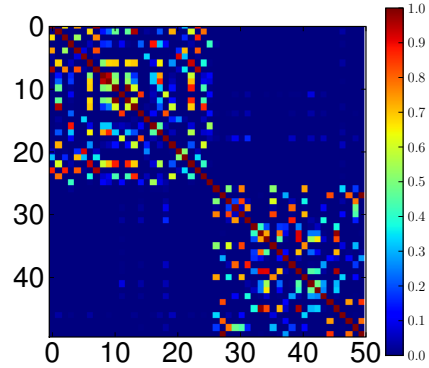
$\gamma = 0.01$



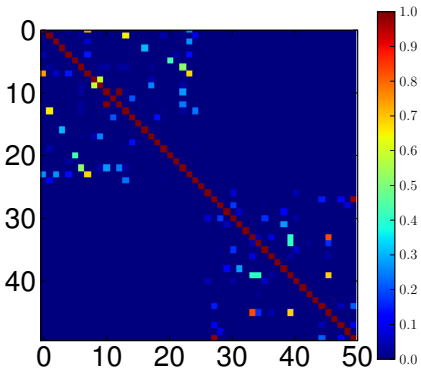
$\gamma = 0.1$



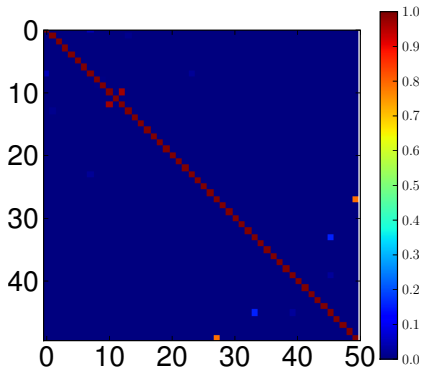
$\gamma = 1$



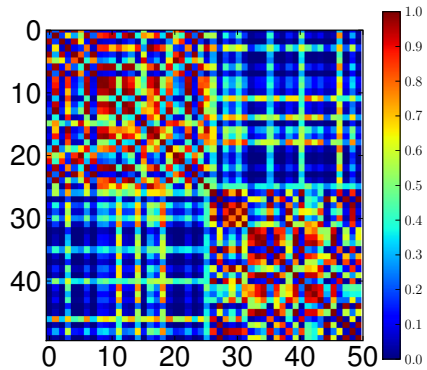
$\gamma = 10$



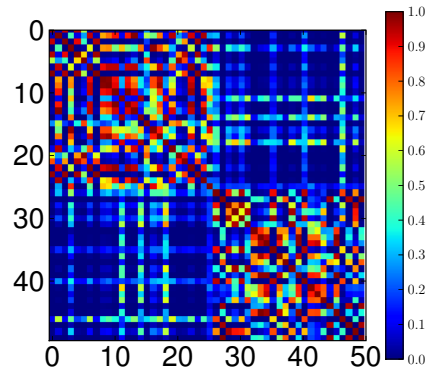
$\gamma = 100$



$\gamma = 1000$



$\gamma = 0.6$
rule of thumb



$\gamma = 1.6$
5-fold CV

Kernel Selection $\leftrightarrow$ Kernel Combination

Is really one of the kernels *best*?

- Kernels are typically 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 really one of the kernels *best*?

- Kernels are typically 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 combined feature spaces can we best
  - *explain the training data*, and
  - achieve a *large margin* between the classes?

Multiple Kernel Learning

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

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

  subject to

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

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

Learning kernel weights and SVM classifier together lie this is called **Multiple-Kernel-Learning** (MKL).

Combining Good Kernels

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

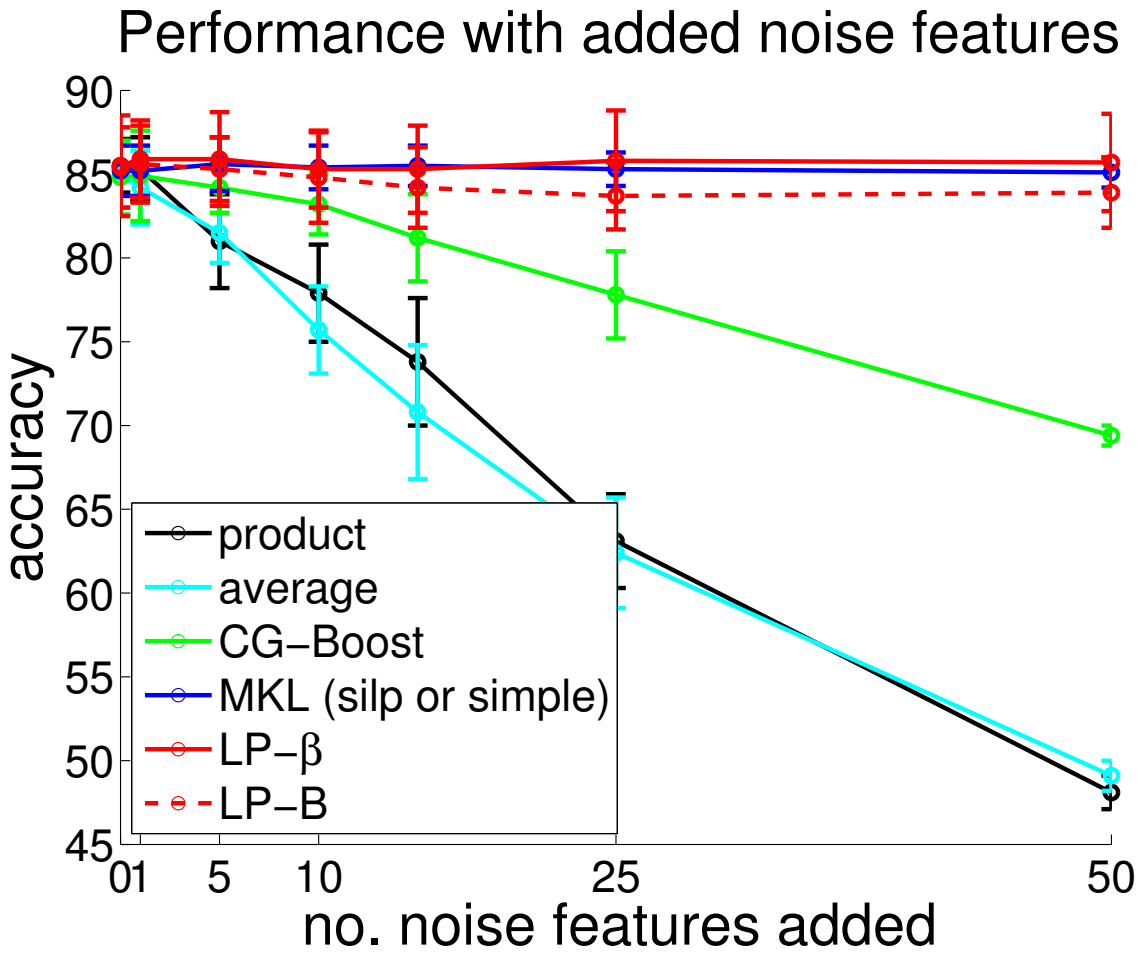
Single features			Combination methods		
Method	Accuracy	Time	Method	Accuracy	Time
Colour	60.9 ± 2.1	3 s	product	85.5 ± 1.2	2 s
Shape	70.2 ± 1.3	4 s	averaging	84.9 ± 1.9	10 s
Texture	63.7 ± 2.7	3 s	CG-Boost	84.8 ± 2.2	1225 s
HOG	58.5 ± 4.5	4 s	MKL (SILP)	85.2 ± 1.5	97 s
HSV	61.3 ± 0.7	3 s	MKL (Simple)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

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

Message: Never use MKL without comparing to simpler baselines!

Combining Good and Bad kernels

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



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

Summary

Kernel Selection and Combination

- Model selection is important to achive highest accuracy
- Combining features is often superior to selecting one
- Kernel combination is a principled way for feature combination:
  - Simple techniques often work as well as complicated ones.
  - Always try *single best*, *average*, and *product* first.

Other Kernel Methods

Multiclass Classification – One-versus-rest SVM

Classification problems with M classes:

- Training samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$,
- Training labels $\{y_1, \ldots, y_n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \ldots, M\}$.

Multiclass Classification – One-versus-rest SVM

Classification problems with M classes:

- Training samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$,
- Training labels $\{y_1, \ldots, y_n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \ldots, M\}$.

One-versus-rest SVM:

- train one SVM F_m for each class m:
  - all samples with class label m are positive examples
  - all other samples are negative examples
- classify by finding maximal response

$$f(x) = \operatorname*{argmax}_{m=1,\ldots,M} F_m(x)$$

Advantage: easy to implement, works well.
Disadvantage: with many classes, training sets become unbalanced.

Multiclass Classification – All-versus-all SVM

Classification problems with M classes:

- Training samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$,
- Training labels $\{y_1, \ldots, y_n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \ldots, M\}$.

Multiclass Classification – All-versus-all SVM

Classification problems with M classes:

- Training samples $\{x_1, \ldots, x_n\} \subset \mathcal{X}$,
- Training labels $\{y_1, \ldots, y_n\} \subset \{1, \ldots, M\}$,
- Task: learn a prediction function $f : \mathcal{X} \to \{1, \ldots, M\}$.

All-versus-all SVM:

- train one SVM for each pair of classes F_{ij} for each $1 \leq i < j \leq M$, total $\frac{m(m-1)}{2}$ SVMs
- classify by voting

$$f(x) = \operatorname*{argmax}_{m=1,\ldots,M} \#\{i \in \{1, \ldots, M\} : F_{m,i}(x) > 0\},$$

(writing $F_{j,i} = -F_{i,j}$ for $j > i$ and $F_{j,j} = 0$)

Advantage: SVM problems stay small and balanced, also works well.
Disadvantage: Number of classifiers grows quadratically in classes.

The "Kernel Trick"

Many other algorithms besides SVMs can make use of kernels:

> Any algorithm that uses the data only in form of **inner products** can be *kernelized*.

How to *kernelize* an algorithm:

- Write the algorithms in terms of only inner products.
- Replace all inner products by kernel function evaluations.

The resulting algorithm will do the same as the linear version, but in the (hidden) feature space $\mathcal{H}$.

Caveat: working in $\mathcal{H}$ is not a guarantee for better performance.
A good choice of k and model-selection are important!

Linear Regression

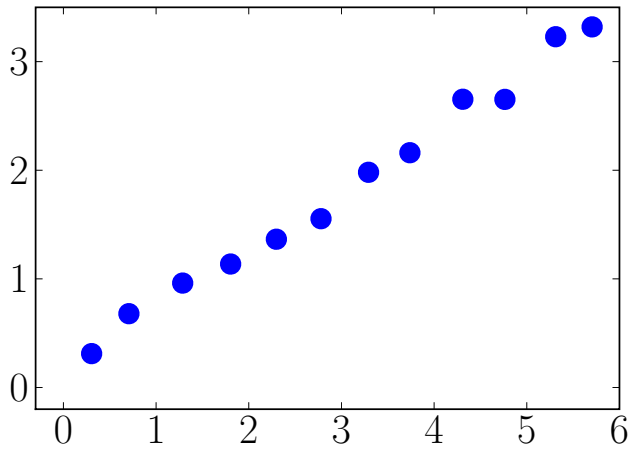
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

$$e_i := (y_i - \langle w, x_i \rangle)^2$$

Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$



Linear Regression

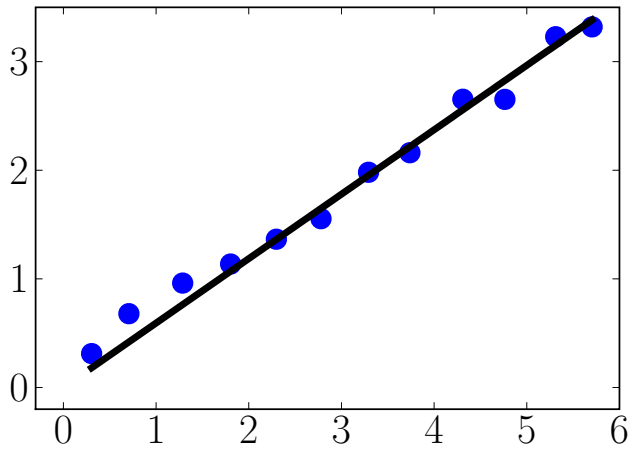
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

$$e_i := (y_i - \langle w, x_i \rangle)^2$$

Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$



Very popular, because it has a closed form solution!

$$w = X(\lambda I_n + X^T X)^{-1} y$$

with I_n is the $n \times n$ identity matrix, $X = (x_1, \ldots, x_n)^t$ is the data matrix, $y = (y_1, \ldots, y_n)^t$ is the target vector.

Linear Regression

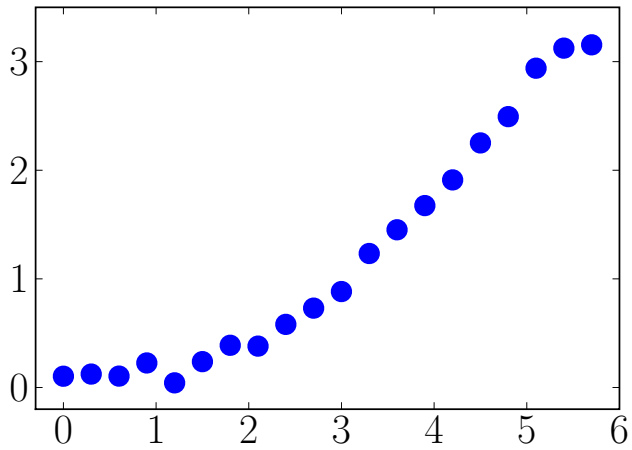
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

$$e_i := (y_i - \langle w, x_i \rangle)^2$$

Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$



What about non-linear?

Very popular, because it has a closed form solution!

$$w = X(\lambda I_n + X^T X)^{-1} y$$

with I_n is the $n \times n$ identity matrix, $X = (x_1, \ldots, x_n)^t$ is the data matrix, $y = (y_1, \ldots, y_n)^t$ is the target vector.

Linear Regression

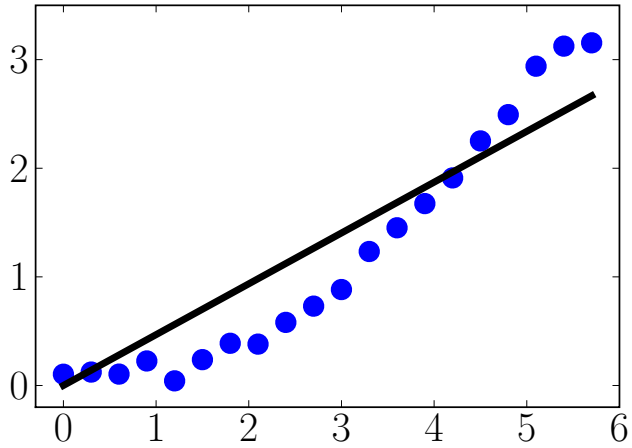
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

$$e_i := (y_i - \langle w, x_i \rangle)^2$$

Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$



What about non-linear?

Very popular, because it has a closed form solution!

$$w = X(\lambda I_n + X^T X)^{-1} y$$

with I_n is the $n \times n$ identity matrix, $X = (x_1, \ldots, x_n)^t$ is the data matrix, $y = (y_1, \ldots, y_n)^t$ is the target vector.

Nonlinear Regression

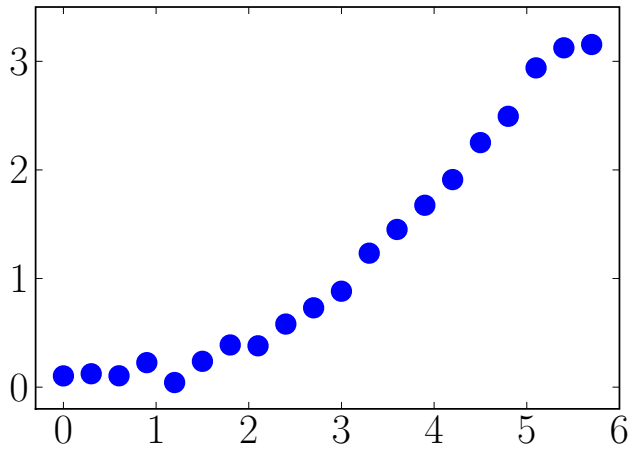
Given $x_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, $i = 1, \ldots, n$, and $\varphi : \mathcal{X} \to \mathcal{H}$. Find an approximating function $f(x) = \langle w, \varphi(x) \rangle_{\mathcal{H}}$ (non-linear in x, linear in w).

Interpolation error:

$$e_i := (y_i - \langle w, \varphi(x_i) \rangle_{\mathcal{H}})^2$$

Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$

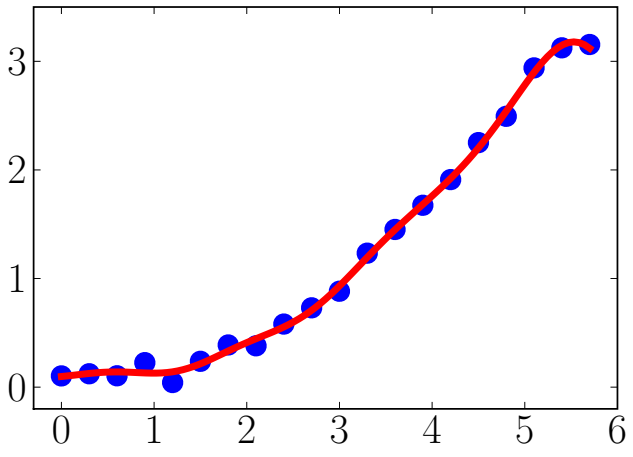


Nonlinear Regression

Given $x_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, $i = 1, \ldots, n$, and $\varphi : \mathcal{X} \to \mathcal{H}$. Find an approximating function $f(x) = \langle w, \varphi(x) \rangle_{\mathcal{H}}$ (non-linear in x, linear in w).

Interpolation error:

$$e_i := (y_i - \langle w, \varphi(x_i) \rangle_{\mathcal{H}})^2$$



Solve for $\lambda \geq 0$:

$$\min_{w \in \mathbb{R}^n} \sum_i e_i + \lambda \|w\|^2$$

Closed form solution is still valid:

$$w = \Phi(\lambda I_n + \Phi^T \Phi)^{-1} y$$

with I_n is the $n \times n$ identity matrix, $\Phi = (\varphi(x_1), \ldots, \varphi(x_n))^t$.

Example: Kernel Ridge Regression

What if $\mathcal{H}$ and $\varphi : \mathcal{X} \to \mathcal{H}$ are given implicitly by kernel function?

We cannot store the closed form solution vector $w \in \mathcal{H}$:

$$w = \Phi(\lambda I_n + \Phi^T \Phi)^{-1} y$$

but we can still calculate $f : \mathcal{X} \to \mathbb{R}$:

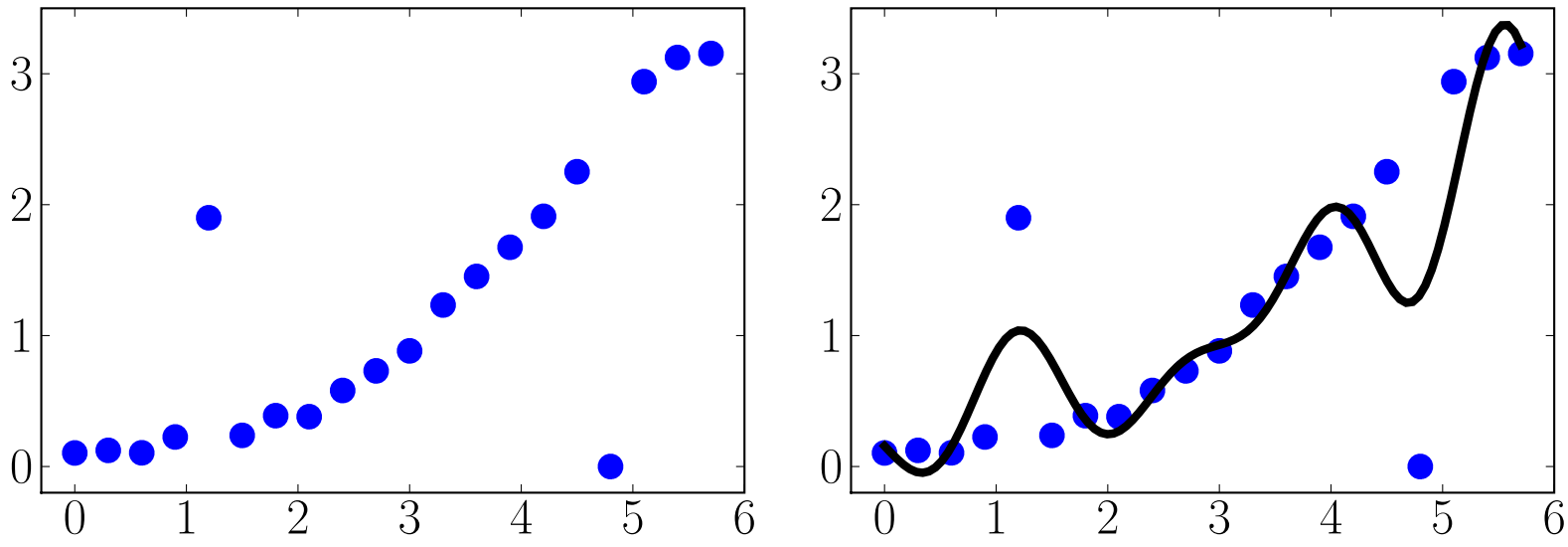
$$\begin{aligned} f(x) &= \langle w, \varphi(x) \rangle \\ &= \langle \Phi(\lambda I_n + \underbrace{\Phi^T \Phi}_{=K})^{-1} y, \varphi(x) \rangle \\ &= y^t (\lambda I_n + K)^{-1} \kappa(x) \end{aligned}$$

where $\kappa(x) = \Big(k(x_1, x), \ldots, k(x_n, x)\Big)^t$.

Kernel Ridge Regression

Nonlinear Regression

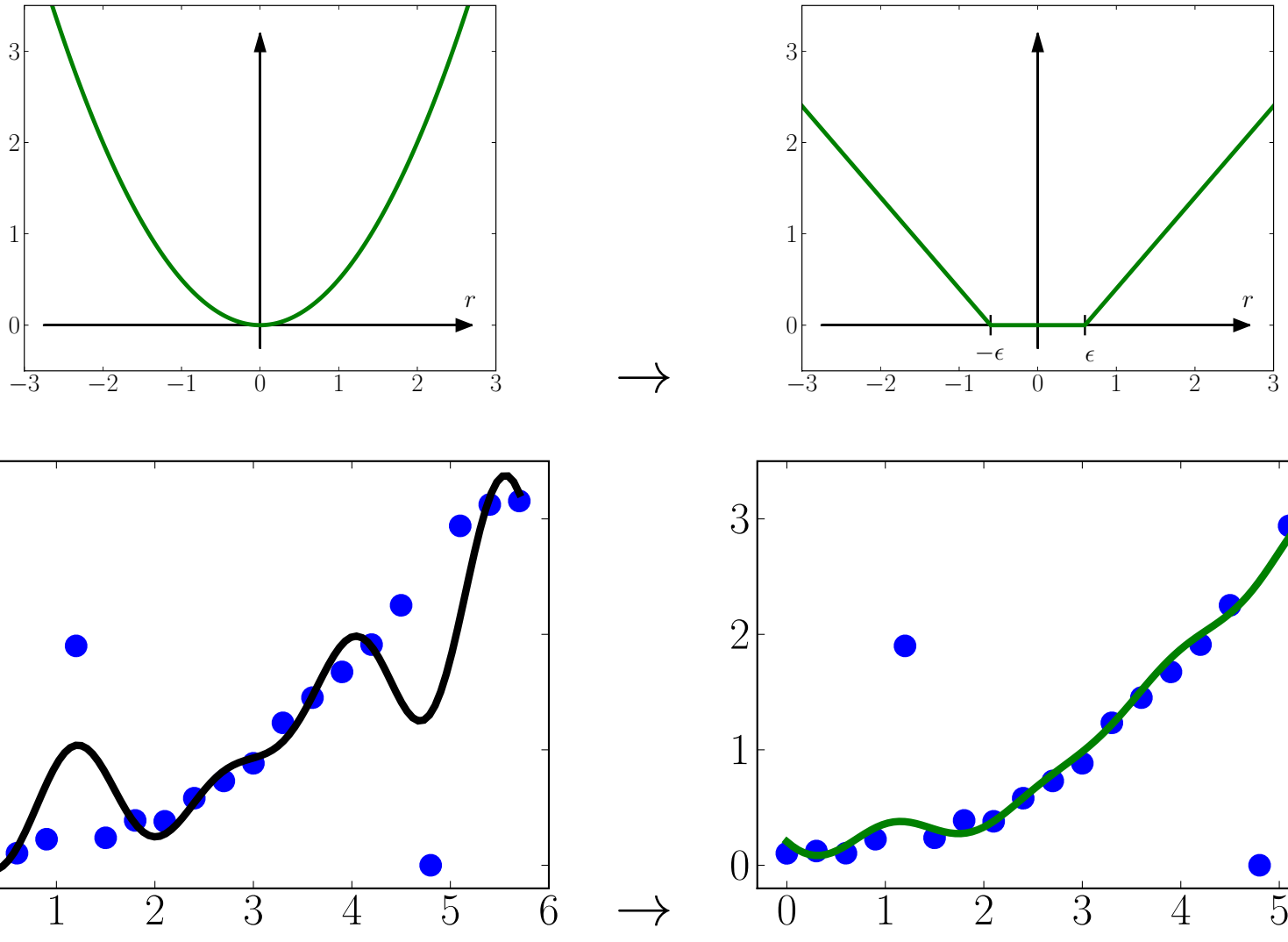
Like Least-Squared Regression, (Kernel) Ridge Regression is sensitive to outliers:



because the quadratic loss function penalized large residue.

Nonlinear Regression

Support Vector Regression with ϵ-insensitive loss is more robust:



Support Vector Regression

Optimization problem similar to support vector machine:

$$\min_{\substack{w\in\mathcal{H},\\ \xi_1,\ldots,\xi_n\in\mathbb{R}^+,\\ \xi'_1,\ldots,\xi'_n\in\mathbb{R}^+}} \quad \|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi'_i)$$

subject to

$$\begin{aligned} y_i - \langle w, \varphi(x_i)\rangle &\leq \epsilon + \xi_i, && \text{for } i = 1,\ldots,n,\\ \langle w, \varphi(x_i)\rangle - y_i &\leq \epsilon + \xi'_i, && \text{for } i = 1,\ldots,n. \end{aligned}$$

Again, we can use a Representer Theorem and get $w = \sum_j \alpha_j \varphi(x_j)$.

Support Vector Regression

Optimization problem similar to support vector machine:

$$\min_{\substack{\alpha_1,\ldots,\alpha_n\in\mathbb{R},\\ \xi_1,\ldots,\xi_n\in\mathbb{R}^+,\\ \xi'_1,\ldots,\xi'_n\in\mathbb{R}^+}} \quad \sum_{i,j=1}^{n} \alpha_i\alpha_j\langle\varphi(x_i),\varphi(x_j)\rangle + C\sum_{i=1}^{n}(\xi_i+\xi'_i)$$

subject to

$$y_i - \textstyle\sum_j \alpha_j\langle\varphi(x_j),\varphi(x_i)\rangle \le \epsilon + \xi_i, \qquad \text{for } i = 1,\ldots,n,$$
$$\textstyle\sum_j \alpha_j\langle\varphi(x_j),\varphi(x_i)\rangle - y_i \le \epsilon + \xi'_i, \qquad \text{for } i = 1,\ldots,n.$$

Rewrite in terms of *kernel evaluations* $k(x,x') = \langle\varphi(x),\varphi(x')\rangle$.

Support Vector Regression

Optimization problem similar to support vector machine:

$$\min_{\substack{\alpha_1,\ldots,\alpha_n\in\mathbb{R},\\ \xi_1,\ldots,\xi_n\in\mathbb{R}^+,\\ \xi'_1,\ldots,\xi'_n\in\mathbb{R}^+}} \quad \sum_{i,j=1}^{n} \alpha_i\alpha_j k(x_i,x_j) + C\sum_{i=1}^{n}(\xi_i+\xi'_i)$$

subject to

$$y_i - \sum\nolimits_j \alpha_j k(x_j,x_i) \le \epsilon + \xi_i, \qquad \text{for } i = 1,\ldots,n,$$
$$\sum\nolimits_j \alpha_j k(x_j,x_i) - y_i \le \epsilon + \xi'_i, \qquad \text{for } i = 1,\ldots,n.$$

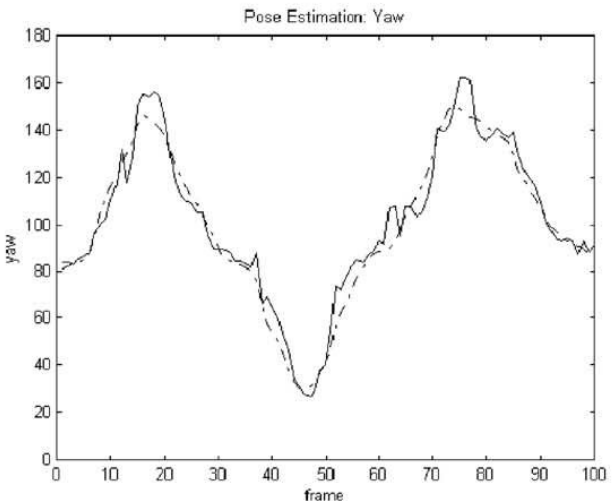
Regression function

$$f(x) = \langle w, \varphi(x)\rangle = \sum\nolimits_j \alpha_j k(x_j,x)$$

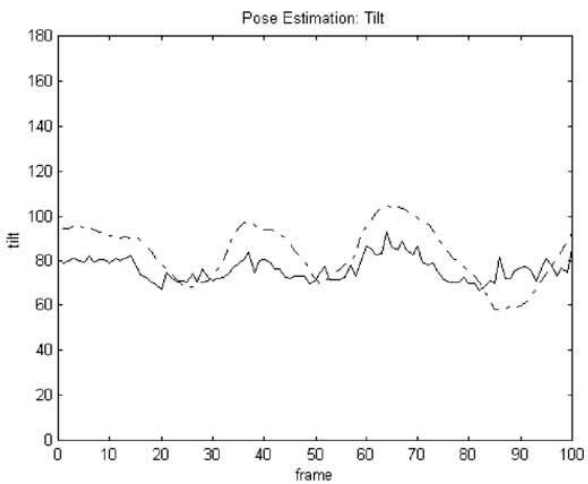
Example – Head Pose Estimation

- Detect faces in image
- Compute gradient representation of face region
- Train support vector regression for *yaw*, *tilt* (separately)

(a) Sample frames of a test sequence



(b) Yaw estimation



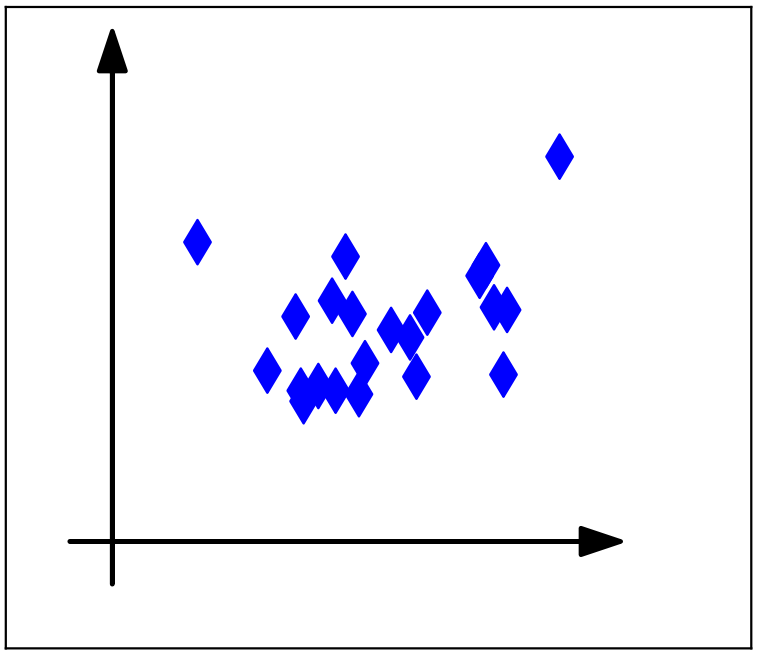
(c) Tilt estimation

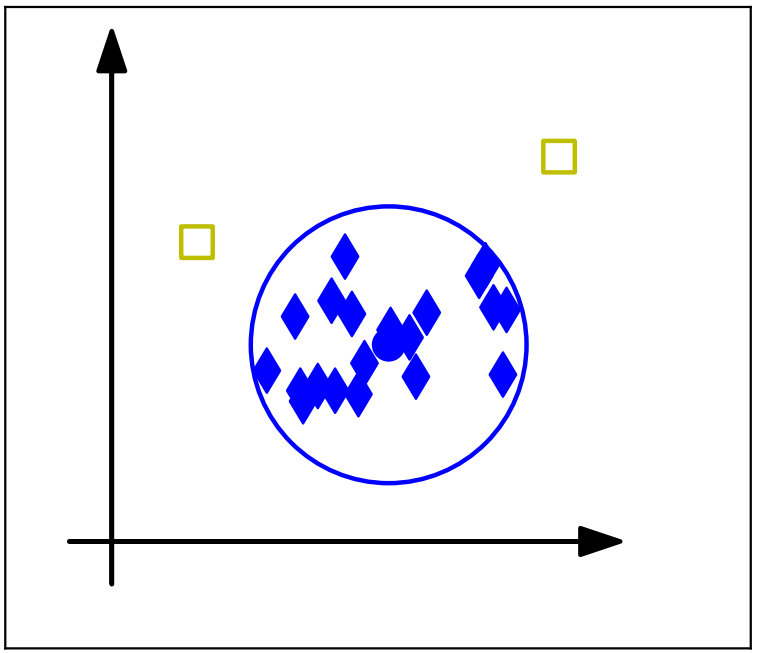
[Li, Gong, Sherra, Liddell, "Support vector machine based multi-view face detection and recognition", IVC 2004]

Outlier/Anomaly Detection in $\mathbb{R}^d$

For unlabeled data, we are interested to detect *outliers*, i.e. samples that lie far away from most of the other samples.

- For samples $x_1, \ldots, x_n$ find the smallest ball (center c, radius R) that contains "most" of the samples.





Outlier/Anomaly Detection in $\mathbb{R}^d$

For unlabeled data, we are interested to detect *outliers*, i.e. samples that lie far away from most of the other samples.

- For samples $x_1, \ldots, x_n$ find the smallest ball (center c, radius R) that contains "most" of the samples.
- Solve

$$\min_{R \in \mathbb{R}, c \in \mathbb{R}^n, \xi_i \in \mathbb{R}^+} R^2 + \frac{1}{\nu n} \sum_i \xi_i$$

  subject to

$$\|x_i - c\|^2 \leq R^2 + \xi_i \quad \text{for } i = 1, \ldots, n.$$

- $\nu \in (0, 1)$ upper bounds the number of "outliers".

Outlier/Anomaly Detection in Arbirary Inputs

Use a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with an implicit feature map $\varphi : \mathcal{X} \to \mathcal{H}$. Do outlier detection for $\varphi(x_1), \ldots, \varphi(x_n)$:

- Find the small ball (center $c \in \mathcal{H}$, radius R) that contains "most" of the samples:
- Solve

$$\min_{R \in \mathbb{R}, c \in \mathcal{H}, \xi_i \in \mathbb{R}^+} R^2 + \frac{1}{\nu n} \sum_i \xi_i$$

subject to

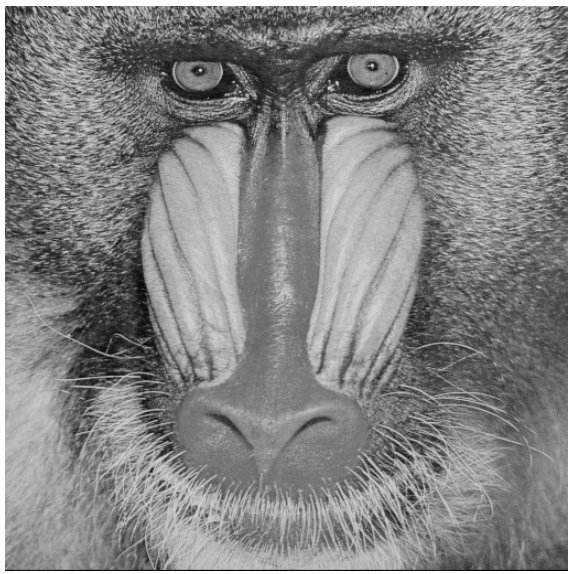
$$\|\varphi(x_i) - c\|^2 \leq R^2 + \xi_i \quad \text{for } i = 1, \ldots, n.$$

- Representer theorem: $c = \sum_j \alpha_j \varphi(x_j)$, and everything can be written using only $k(x_i, x_j)$.

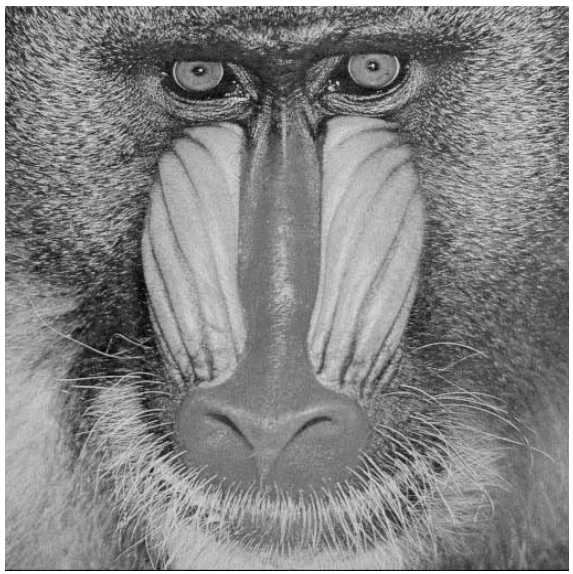
Support Vector Data Description

Example – Steganalysis

- **Steganography:** hide data in other data (e.g. in images)
  - e.g.: flip some least significant bits
- **Steganalysis:** given any data, find out if data is hidden



original



with 23300 hidden bits

- compute image statistics (color wavelet coefficients)
- train SVDD with RBF-kernel
- identified outlier images are suspicious candidates

[Lyu, Farid. "Steganalysis using color wavelet statistics and one-class support vector machines", SPIE EI, 2004]

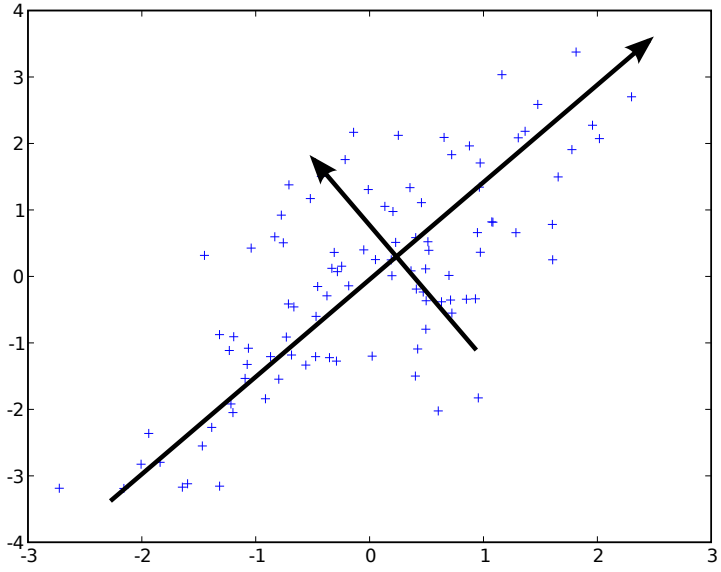
Principle Component Analysis (PCA)

Given samples $x_i \in \mathbb{R}^d$, PCA finds the directions of maximal covariance. Without loss of generality assume that $\sum_i x_i = 0$.

- The PCA directions $e_1, \ldots, e_d$ are the *eigenvectors* of the covariance matrix

$$C = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^t$$

sorted by their eigenvalue.



- We can express x_i in PCA-space by $P(x_i) = \sum_{j=1}^{K} \langle x_i, e_j \rangle e_j$.
- Lower-dim. coordinate mapping: $x_i \mapsto \begin{pmatrix} \langle x_i, e_1 \rangle \\ \langle x_i, e_2 \rangle \\ \ldots \\ \langle x_i, e_m \rangle \end{pmatrix} \in \mathbb{R}^m$

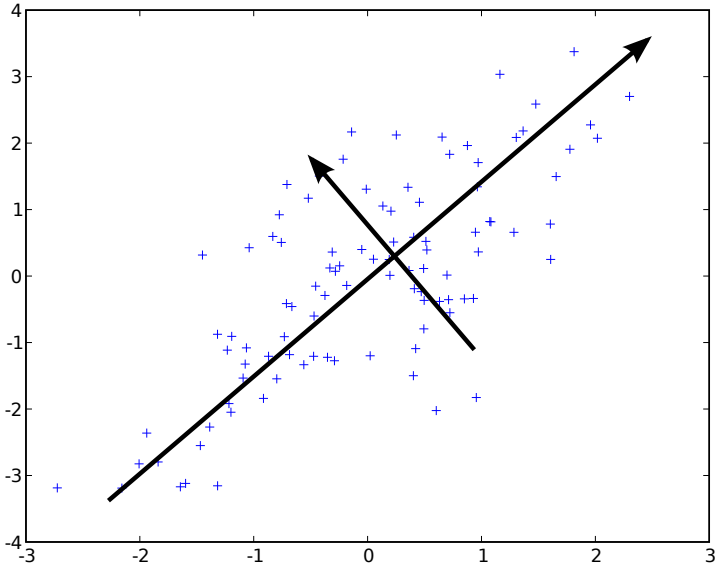
Kernel-PCA

Given samples $x_i \in \mathcal{X}$, kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with an implicit feature map $\varphi : \mathcal{X} \to \mathcal{H}$. Do PCA in the Hilbert space $\mathcal{H}$.

- The kernel-PCA directions $e_1, \ldots, e_d$ are the eigenvectors of the covariance operator

$$C = \frac{1}{n} \sum_{i=1}^{n} \varphi(x_i) \varphi(x_i)^t$$

sorted by their eigenvalue.



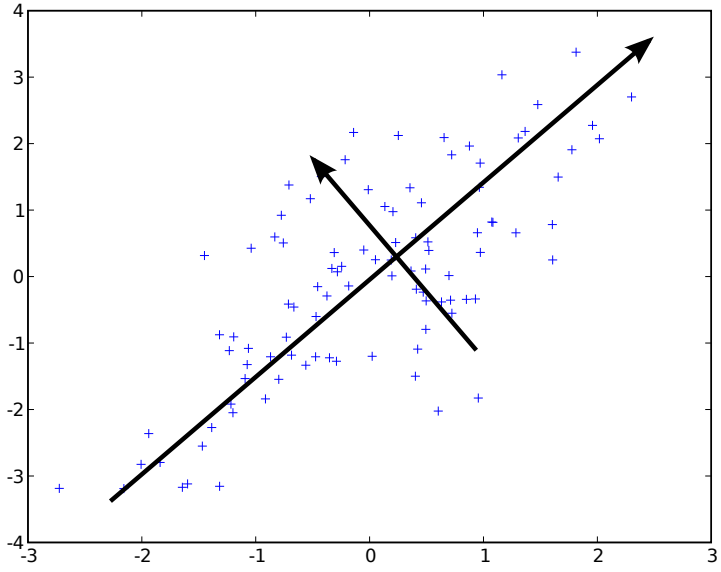
- Lower-dim. coordinate mapping: $x_i \mapsto \begin{pmatrix} \langle \varphi(x_i), e_1 \rangle \\ \langle \varphi(x_i), e_2 \rangle \\ \ldots \\ \langle \varphi(x_i), e_m \rangle \end{pmatrix} \in \mathbb{R}^m$

Kernel-PCA

Given samples $x_i \in \mathcal{X}$, kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with an implicit feature map $\varphi : \mathcal{X} \to \mathcal{H}$. Do PCA in the Hilbert space $\mathcal{H}$.

- Equivalently, we can use the eigenvectors e'_j and eigenvalues λ_j of

$$K = \left(\langle \varphi(x_i), \varphi(x_j) \rangle\right)_{i,j=1,\ldots,n}$$
$$= \left(k(x_i, x_j)\right)_{i,j=1,\ldots,n}$$

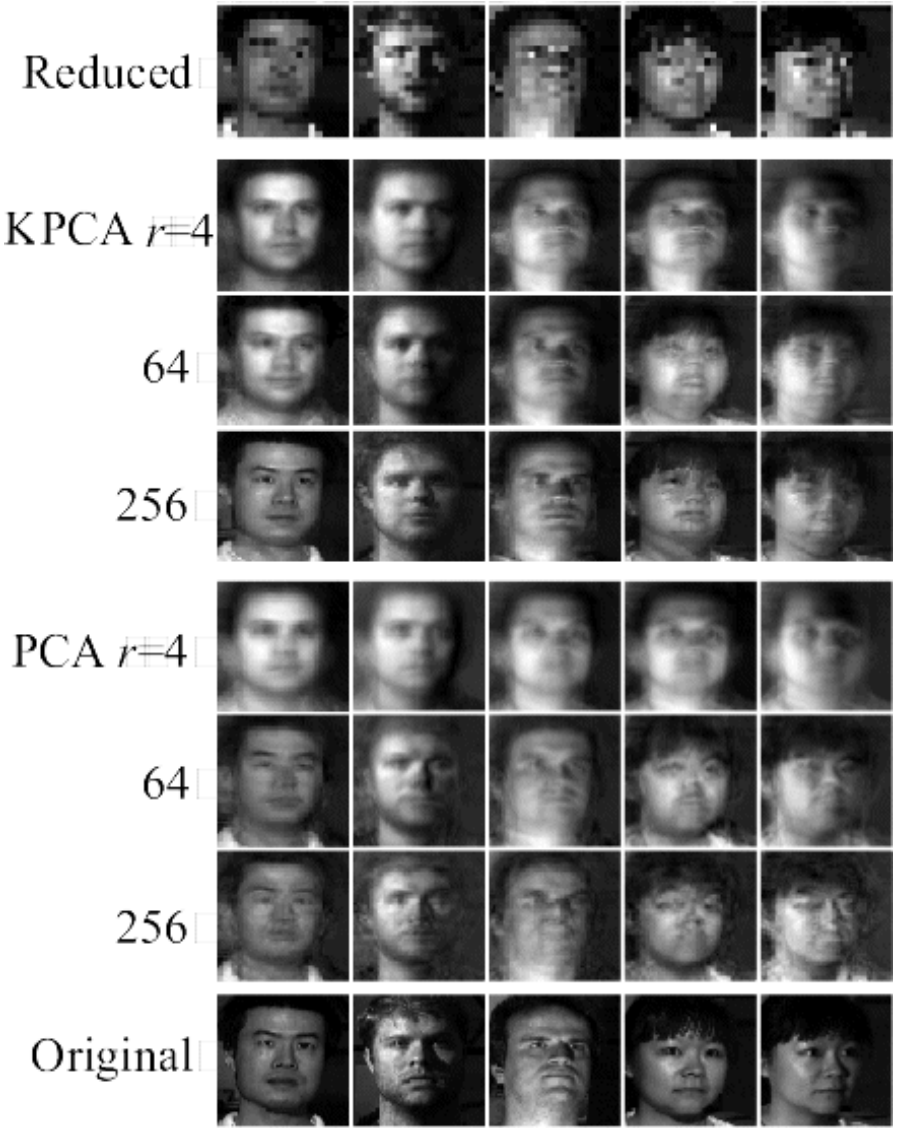


- Coordinate mapping: $x_i \mapsto \left(\sqrt{\lambda_1}\, e'^i_1, \ldots, \sqrt{\lambda_K}\, e'^i_K\right)$.

Kernel-PCA

Example – Image Superresolution

- Collect high-res face images
- Use KPCA with RBF-kernel to learn non-linear subspaces
- For new low-res image:
  - scale to target high resolution
  - project to closest point in face subspace



reconstruction in r dimensions

[Kim, Jung, Kim, "Face recognition using kernel principal component analysis", Signal Processing Letters, 2002.]

Kernel k-Means Clustering

Kernel-PCA is more than just a non-linear version of PCA:

- PCA maps $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, e.g. to remove noise dimensions.
- Kernel-PCA maps $\mathcal{X} \to \mathbb{R}^{d'}$, so it provides a vectorial representation also of non-vectorial data!

Kernel k-Means Clustering

Kernel-PCA is more than just a non-linear version of PCA:

- PCA maps $\mathbb{R}^d$ to $\mathbb{R}^{d'}$, e.g. to remove noise dimensions.
- Kernel-PCA maps $\mathcal{X} \to \mathbb{R}^{d'}$, so it provides a vectorial representation also of non-vectorial data!

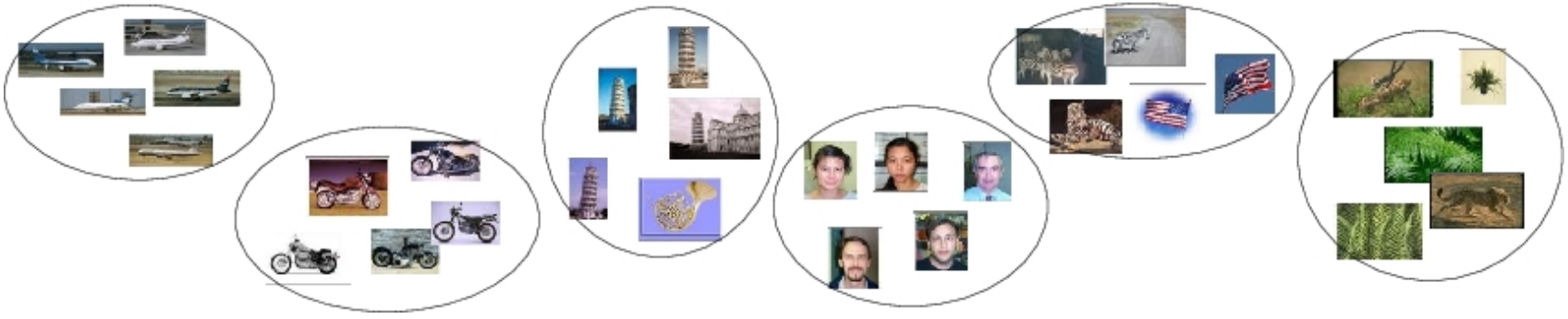
We can apply algorithms that only work in vector spaces, e.g. K-Means:

- Given $x_1, \ldots, x_n \in \mathcal{X}$.
- Choose a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.
- Apply kernel-PCA to obtain vectorial $v_1, \ldots, v_n \in \mathbb{R}^{d'}$.
- Cluster $v_1, \ldots, v_n \in \mathbb{R}^{d'}$ using K-Means.

$\Rightarrow x_1, \ldots, x_n$ are clustered based on the similarity defined by k.

Example – Unsupervised Object Categorization

Automatically group together images that show similar objects



- Represent images by bag of words histograms
- Perform Kernel k-Means Clustering
  - observation: clusters get better if we use a good image kernel (e.g. χ^2) instead of plain k-means (linear kernel)

[Tuytelaars, Lampert, Blaschko, Buntine, "Unsupervised object discovery: a comparison", IJCV, 2009.]

Learning Structured Outputs

From Arbitrary Inputs to Arbitrary Outputs

With kernels, we can handle "arbitrary" input spaces:

- we only need a pairwise similarity measure for objects:
  - images, e.g.
  - gene sequences, e.g. string kernels
  - graphs, e.g. random walk kernels
- We can learn mappings

$$f : \mathcal{X} \to \{-1, +1\} \quad \text{or} \quad f : \mathcal{X} \to \mathbb{R}.$$

What about arbitrary *output* spaces?

- We know: kernels correspond to feature maps: $\varphi : \mathcal{X} \to \mathcal{H}$.
- But: we cannot *invert* φ, there is no $\varphi^{-1} : \mathcal{H} \to \mathcal{X}$.
- Kernels do not readily help to construct

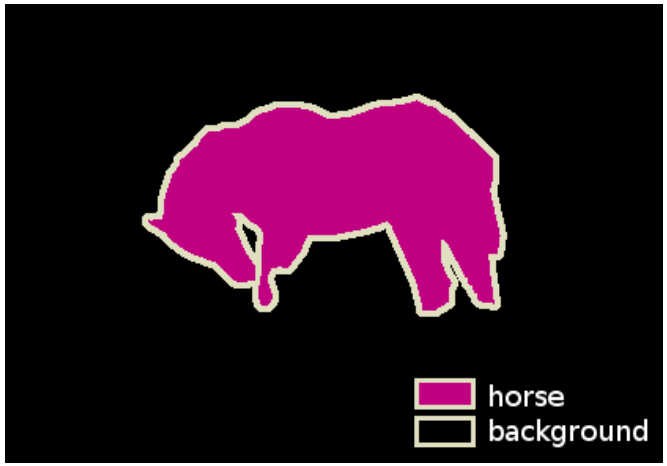
$$f : \mathcal{X} \to \mathcal{Y} \qquad \text{with } \mathcal{Y} \neq \mathbb{R}.$$

What would we like to predict?

- Natural Language Processing:
  - Automatic Translation (output: sentences)
  - Sentence Parsing (output: parse trees)
- Bioinformatics:
  - Secondary Structure Prediction (output: bipartite graphs)
  - Enzyme Function Prediction (output: path in a tree)
- Robotics:
  - Planning (output: sequence of actions)
- **Computer Vision**
  - Image Segmentation (output: segmentation mask)
  - Human Pose Estimation (output: positions of body parts)
  - Image Retrieval (output: ranking of images in database)

Computer Vision Example: Semantic Image Segmentation

$\mapsto$



input: images output: segmentation masks

- input space $\mathcal{X} = \{\textit{images}\} \mathrel{\hat{=}} [0, 255]^{3 \cdot M \cdot N}$
- output space $\mathcal{Y} = \{\textit{segmentation masks}\} \mathrel{\hat{=}} \{0, 1\}^{M \cdot N}$
- (structured output) prediction function: $f : \mathcal{X} \to \mathcal{Y}$

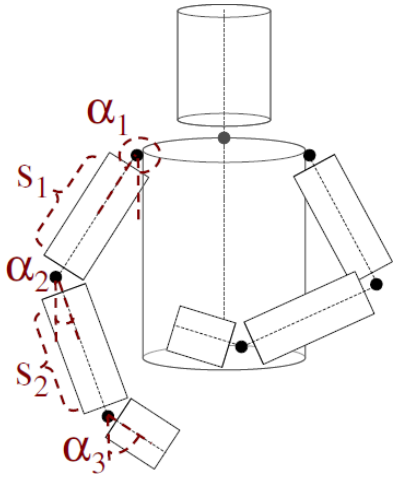
$$f(x) := \operatorname*{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- energy function $E(x, y) = \sum_i w_i^\top \varphi_u(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi_p(y_i, y_j)$

Images: [M. Everingham et al. "The PASCAL Visual Object Classes (VOC) challenge", IJCV 2010]

Computer Vision Example: Human Pose Estimation

input: image



body model

output: model fit

- input space $\mathcal{X} = \{\textit{images}\}$
- output space $\mathcal{Y} = \{\textit{positions/angles of } K \textit{ body parts}\} \hat{=} \mathbb{R}^{4K}$.
- prediction function: $f : \mathcal{X} \to \mathcal{Y}$

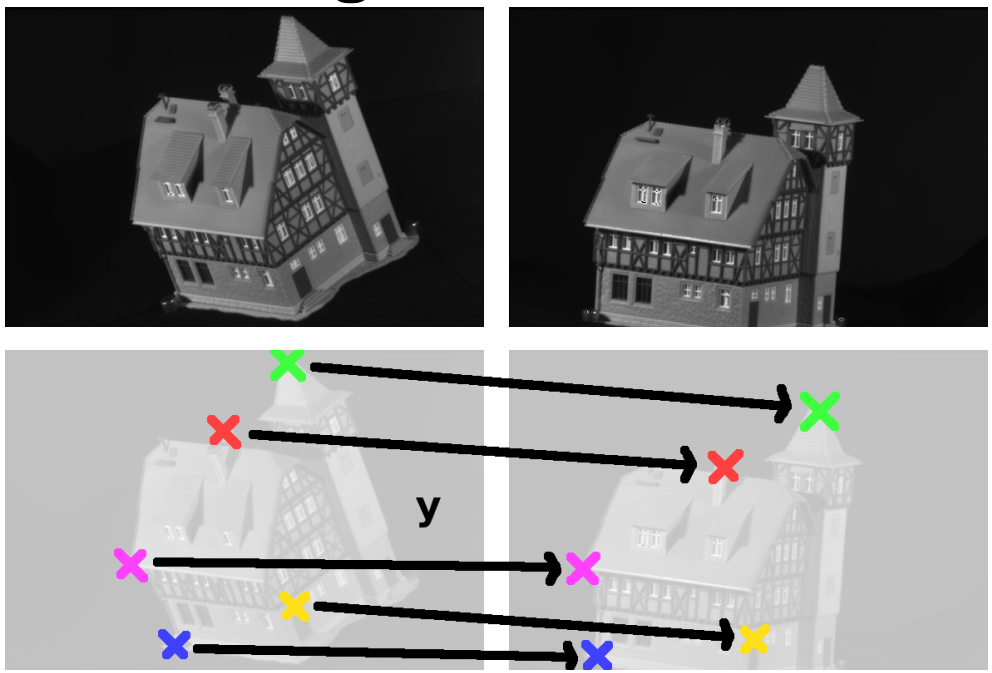
$$f(x) := \operatorname*{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- energy $E(x, y) = \sum_i w_i^\top \varphi_{\textit{fit}}(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi_{\textit{pose}}(y_i, y_j)$

Images: [Ferrari, Marin-Jimenez, Zisserman: "Progressive Search Space Reduction for Human Pose Estimation", CVPR 2008.]

Computer Vision Example: Point Matching

input: image pairs



output: mapping $y : x_i \leftrightarrow y(x_i)$

- prediction function: $f : \mathcal{X} \rightarrow \mathcal{Y}$

$$f(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

- scoring function $F(x, y) =$
$\sum_i w_i^\top \varphi_{\textit{sim}}(x_i, y(x_i)) + \sum_{i,j} w_{ij}^\top \varphi_{\textit{dist}}(x_i, x_j, y(x_i), y(x_j)) +$
$\sum_{i,j,k} w_{ijk}^\top \varphi_{\textit{angle}}(x_i, x_j, x_k, y(x_i), y(x_j), y(x_k))$

[J. McAuley et al.: "Robust Near-Isometric Matching via Structured Learning of Graphical Models", NIPS, 2008]

Computer Vision Example: Object Localization

input:
image

output:
object position
$(left, top$
$right, bottom)$

- input space $\mathcal{X} = \{images\}$
- output space $\mathcal{Y} = \mathbb{R}^4$ bounding box coordinates
- prediction function: $f : \mathcal{X} \to \mathcal{Y}$

$$f(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

- scoring function $F(x, y) = w^\top \varphi(x, y)$ where $\varphi(x, y) = h(x|_y)$ is a feature vector for an image region, e.g. bag-of-visual-words.

[M. Blaschko, C. Lampert: "Learning to Localize Objects with Structured Output Regression", ECCV, 2008]

Computer Vision Examples: Summary

Image Segmentation

$$y = \underset{y \in \{0,1\}^N}{\operatorname{argmin}} E(x, y) \qquad E(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j)$$

Pose Estimation

$$y = \underset{y \in \mathbb{R}^{4K}}{\operatorname{argmin}} E(x, y) \qquad E(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j)$$

Point Matching

$$y = \underset{y \in \Pi_n}{\operatorname{argmax}} F(x, y) \quad F(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j) + \sum_{i,j,k} w_{ijk}^\top \varphi(y_i, y_j, y_k)$$

Object Localization

$$y = \underset{y \in \mathbb{R}^4}{\operatorname{argmax}} F(x, y) \qquad F(x, y) = w^\top \varphi(x, y)$$

Grand Unified View

Predict structured output by maximization

$$y = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

of a compatibility function

$$F(x, y) = \langle w, \varphi(x, y) \rangle$$

that is linear in a parameter vector w.

A generic structured prediction problem

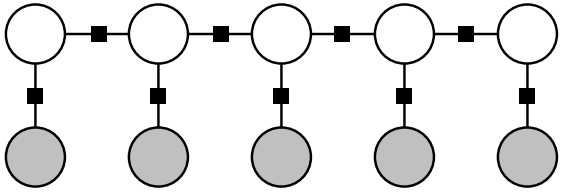
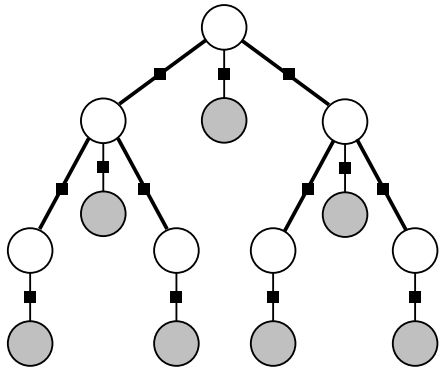
- $\mathcal{X}$: arbitrary input domain
- $\mathcal{Y}$: structured output domain, decompose $y = (y_1, \ldots, y_k)$
- Prediction function $f : \mathcal{X} \to \mathcal{Y}$ by

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

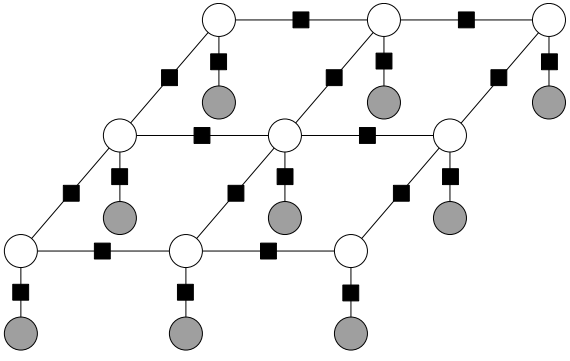
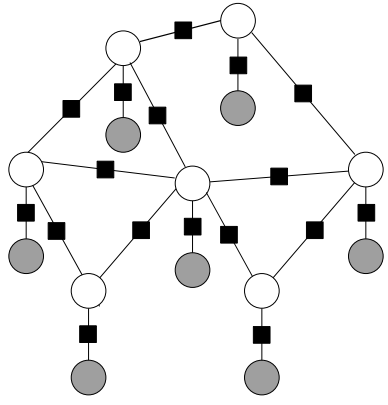
- Compatiblity function (or negative of "energy")

$$\begin{aligned} F(x, y) &= \langle w, \varphi(x, y) \rangle \\ &= \sum_{i=1}^{k} w_i^\top \varphi_i(y_i, x) \qquad \textbf{unary terms} \\ &\quad + \sum_{i,j=1}^{k} w_{ij}^\top \varphi_{ij}(y_i, y_j, x) \qquad \textbf{binary terms} \\ &\quad + \quad \ldots \quad \textbf{higher order terms} \text{ (sometimes)} \end{aligned}$$

Yesterday's and tomorrow's lectures: how to solve $\operatorname{argmax}_y F(x, y)$.

chain   tree

- loop-free graphs: Viterbi decoding / dynamic programming

grid   arbitrary graph

- loopy graphs: GraphCuts, or approximate inference (e.g. LBP)

Today: how to learn a good function $F(x, y)$ from training data.

Parameter Learning in Structured Models

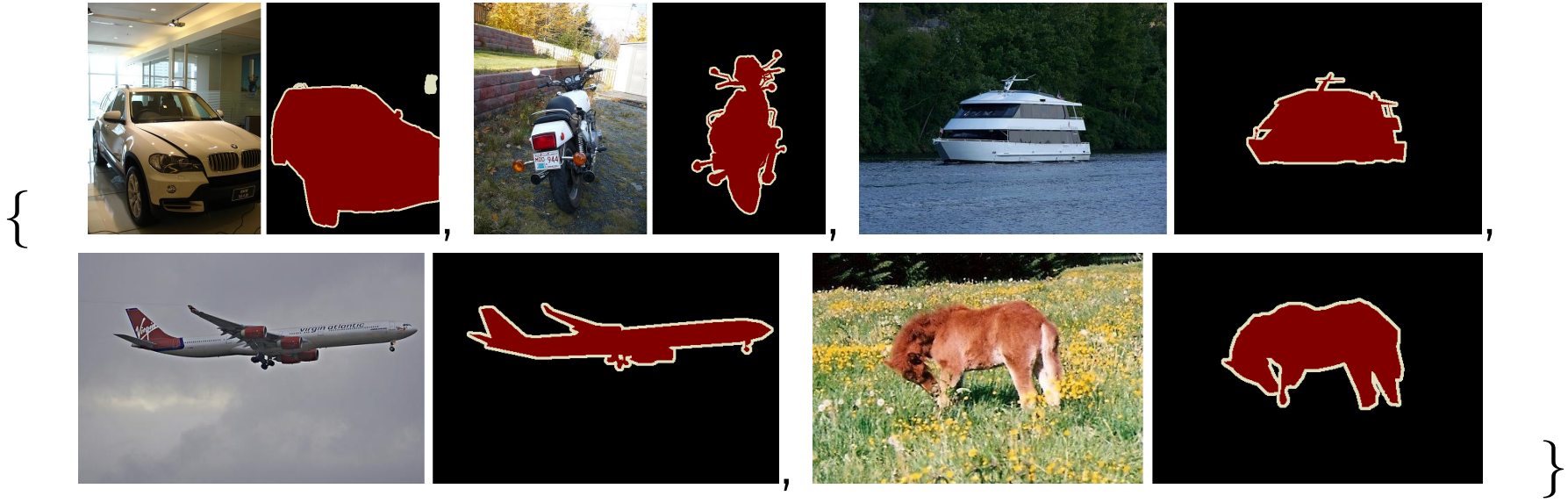
- Given: parametric model (family): $F(x, y) = \langle w, \varphi(x, y) \rangle$
- Given: prediction method: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$
- Not given: parameter vector w (high-dimensional)

Parameter Learning in Structured Models

- Given: parametric model (family): $F(x, y) = \langle w, \varphi(x, y) \rangle$
- Given: prediction method: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$
- Not given: parameter vector w (high-dimensional)

Supervised Training:

- Given: example pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$. typical inputs with "the right" outputs for them.



- Task: determine "good" w

Loss function

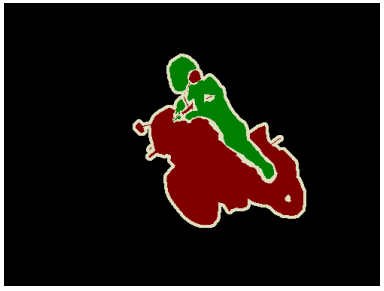
What make a solution "good"?

- Define a *loss function*

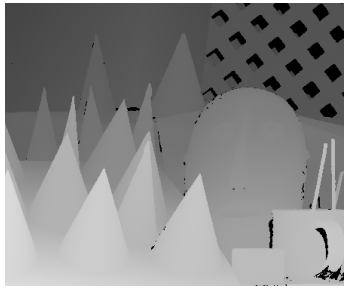
$$\Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^{+},$$

such that $\Delta(y, \bar{y})$ measures the loss/cost incurred by predicting $\bar{y}$ when y is correct.

- The *loss function* is application dependent:



number of mislabeled pixels



total depth error

number of wrong body parts

bounding box area overlap

Structured Output SVM

Two criteria for decision function f:

- 1) Correctness: Ensure $f(x_i) = y_i$ for $i = 1, \ldots, n$.
- 2) Robustness: f should also work if x_i are perturbed.

Structured Output SVM

Two criteria for decision function f:

- 1) Correctness: Ensure $f(x_i) = y_i$ for $i = 1, \ldots, n$.
- 2) Robustness: f should also work if x_i are perturbed.

Translated to structured prediction $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$:

- 1) Ensure for $i = 1, \ldots, n$,

$$\begin{aligned} & \operatorname*{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x_i, y) \rangle = y_i, \\ \Leftrightarrow \quad & \langle w, \varphi(x_i, y_i) \rangle \geq \epsilon + \langle w, \varphi(x_i, y) \rangle \qquad \text{for all } y \in \mathcal{Y} \setminus \{y_i\}. \end{aligned}$$

- 2) Enforce *large margin*, minimize $\|w\|^2$.

Structured Output SVM

Optimization Problem:

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

subject to, for $i = 1, \ldots, n$,

$$\langle w, \varphi(x_i, y_i) \rangle \geq \Delta(y_i, y) + \langle w, \varphi(x_i, y) \rangle - \xi_i, \quad \text{for all } y \in \mathcal{Y} \setminus \{y_i\}.$$

Structured Output SVM

Optimization Problem:

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

subject to, for $i = 1, \ldots, n$,

$$\langle w, \varphi(x_i, y_i) \rangle \geq \Delta(y_i, y) + \langle w, \varphi(x_i, y) \rangle - \xi_i, \quad \text{for all } y \in \mathcal{Y} \setminus \{y_i\}.$$

- $\Delta(y_i, y) \geq 0$: Loss function ("predicted y, correct would be y_i")
- Optimization problem very similar to normal (soft-margin) SVM
  - quadratic in w, linear in ξ
  - constraints linear in w and ξ
  - **convex**
- But there are $n(|\mathcal{Y}| - 1)$ constraints!
  - numeric optimization needs some tricks
  - computationally expensive

Example: A "Real" Multiclass SVM

- $\mathcal{Y} = \{1, 2, \ldots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise.} \end{cases}$.
- $\varphi(x, y) = \Big([\![y = 1]\!]\Phi(x),\ [\![y = 2]\!]\Phi(x),\ \ldots,\ [\![y = K]\!]\Phi(x) \Big)$
  $= \Phi(x) e_y^\top$ with e_y=y-th unit vector

Example: A "Real" Multiclass SVM

- $\mathcal{Y} = \{1, 2, \ldots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise.} \end{cases}$
- $\varphi(x, y) = \Big([\![y = 1]\!]\Phi(x),\ [\![y = 2]\!]\Phi(x),\ \ldots,\ [\![y = K]\!]\Phi(x) \Big)$
  $= \Phi(x) e_y^\top$ with e_y=y-th unit vector

Solve:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$$

subject to, for $i = 1, \ldots, n$,

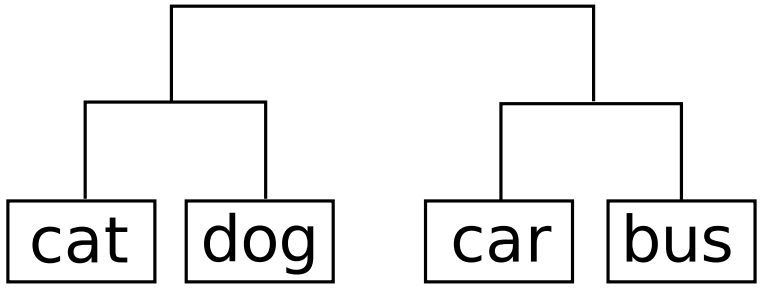
$$\langle w, \varphi(x_i, y_i)\rangle \geq 1 + \langle w, \varphi(x_i, y)\rangle - \xi_i \quad \text{for all } y \in \mathcal{Y} \setminus \{y_i\}.$$

Classification: $f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \ \langle w, \varphi(x, y)\rangle$

Crammer-Singer Multiclass SVM

Example: Hierarchical Multiclass Classification

Loss function can reflect hierarchy:



$$\Delta(y, y') := \frac{1}{2}(\text{distance in tree})$$

$$\Delta(\texttt{cat}, \texttt{cat}) = 0, \quad \Delta(\texttt{cat}, \texttt{dog}) = 1, \quad \Delta(\texttt{cat}, \texttt{bus}) = 2, \quad etc.$$

Solve:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \varphi(x_i, y_i)\rangle \geq \Delta(y_i, y) + \langle w, \varphi(x_i, y)\rangle - \xi_i \quad \text{for all } y \in \mathcal{Y} \setminus \{y_i\}.$$

Example: Object Localization

input: image

output: object position $(left, top\ right, bottom)$

- $\varphi(x, y) = \Phi(x|_y)$ feature vector of *image inside box region*
- $\Delta(y, y') := \frac{\text{area } y \cap y'}{\text{area } y \cup y'}$ (*box overlap*).
- $F(x, y) = \langle w, \varphi(x, y)$: quality score for region y in image x

$$\langle w, \varphi(x_i, y_i) \rangle \geq \Delta(y_i, y) + \langle w, \varphi(x_i, y) \rangle - \xi_i$$

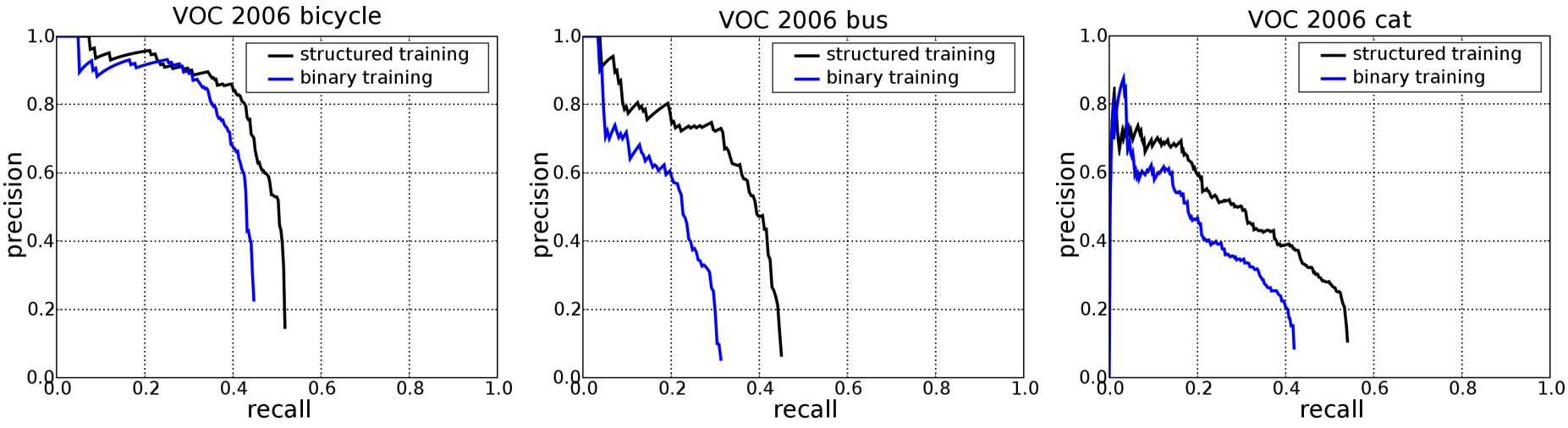
- Interpretation:
  - correct location must have largest score of all regions
  - almost correct wrong regions can have similar score
  - non-overlapping wrong ones must have clearly lower score

[M. Blaschko, C. Lampert: "Learning to Localize Objects with Structured Output Regression", ECCV, 2008]

Example: Object Localization Results

Experiments on PASCAL VOC 2006 dataset:

- Compare S-SVM with conventional training for sliding windows
- Identical setup: same features, same image-kernel, etc.



Precision–recall curves for VOC 2006 `bicycle`, `bus` and `cat`.

- Structured prediction training improved precision and recall.

Structured SVMs in Practice

Structured SVMs more and more popular for CV problems.

- Software packages:
  - **SVMstruct**: `http://svmlight.joachims.org/svm_struct.html`
  - **BMRM**: `http://users.rsise.anu.edu.au/~chteo/BMRM.html`
  - **grante**: `http://www.nowozin.net/sebastian/grante/`
- Needs application specific adaption
  - feature map, loss function, how to solve argmax
- Typically used with explicit feature map, not kernelized.
- Training is often **slow**,
  - requires several passes through training data,
  - one argmax step per training image per pass
- Prediction require solving one argmax problem per test image

Summary

Structured-Output SVM

- Task: predict $f : \mathcal{X} \to \mathcal{Y}$ instead of $f : \mathcal{X} \to \mathbb{R}$.
- Key idea:
  - learn $F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ with $F = \langle w, \varphi(x, y) \rangle$
  - predict via $f(x) := \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$.
- Convex optimization problem, similar to SVM
  - but very many constraints: computationally expensive
- Field of active research, many open questions
  - how to accelerate training?
  - how to handle complicated φ ("higher order terms")
  - how to combine S-SVM with approximate inference?
  - ...

Overall Summary

Learning with Kernels

- Kernels are at the same time
  1) Similarity measure between (arbitrary) objects,
  2) Scalar products in a (hidden) vector space.
- We obtain non-linear version of originally linear algorithms:
  - classification, regression, outlier detection, dim.reduction, ...
- We can learn with complicated inputs and outputs:
  - kernels for arbitrary inputs, structured output SVM, ...

Kernels in Computer Vision

- Kernels allow us to integrate domain knowledge, i.e. invariances.

Kernel Selection/Combination

- Kernels allow us to combine heterogenous information sources,
  - color, shape, text captions, ...

Ad: Positions at IST Austria, Vienna

More info: **www.ist.ac.at**

IST Austria Graduate School

- join with MSc **or BSc**
- 1(2) + 3 yr PhD program
  - **Computer Vision/Machine Learning** (me, Vladimir Kolmogorov)
  - Computer Graphics (C. Wojtan)
  - Comp. Topology (H. Edelsbrunner)
  - Game Theory (K. Chatterjee)
  - Software Verification (T. Henzinger)
  - Cryptography (K. Pietrzak)
  - Comp. Neuroscience (G. Tkacik)
  - Statistics (C. Uhler), and more...
- fully funded positions

Postdoc Positions in my Group

- see `http://www.ist.ac.at/~chl`

Internships: ask me!