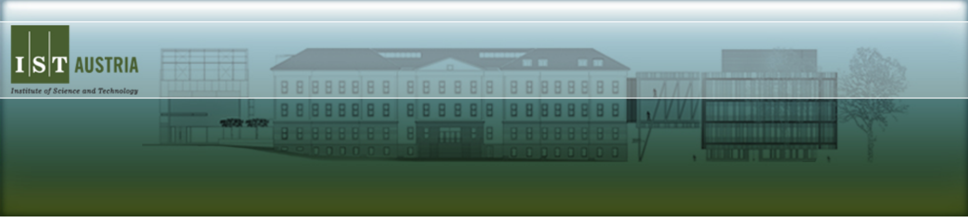



# Intro to Supervised Learning

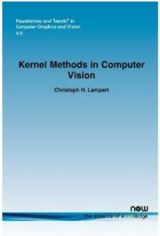
Christoph Lampert
IST Austria (Institute of Science and Technology Austria)
Vienna, Austria

ENS/INRIA Summer School
Visual Recognition and Machine Learning
Paris 2013

## Slides available on my home page

http://www.ist.ac.at/~chl

## More details on Max-Margin / Kernel Methods




Foundations and Trends in Computer Graphics and Vision,

www.nowpublishers.com/

Also as PDFs on my homepage

**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**

**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**

**Automatic systems that analyzes some aspects of visual data**

→

→

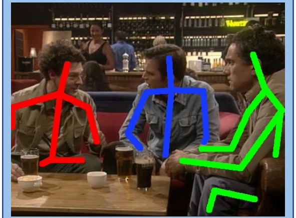
- three people
- one table
- three glasses

**Object Recognition**

**Automatic systems that analyzes some aspects of visual data**

$\rightarrow$

$\rightarrow$



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

**Pose Estimation**

**Classification/ Regression**
today

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

**Structured Prediction**
today/Wednesday

- Pose Estimation
- Stereo Reconstruction
- Image Denoising
- Semantic Image Segmentation

Outlier Detection

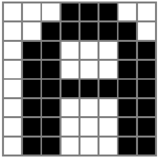
- Anomaly Detection in Videos
- Video Summarization

Clustering

- Image Duplicate Detection

Classification

- …
- Optical Character Recognition
- …

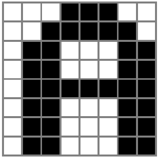


- It's difficult to *program* a solution to this.

```
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.
  - When a new font occurs: re-train, don't re-program

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

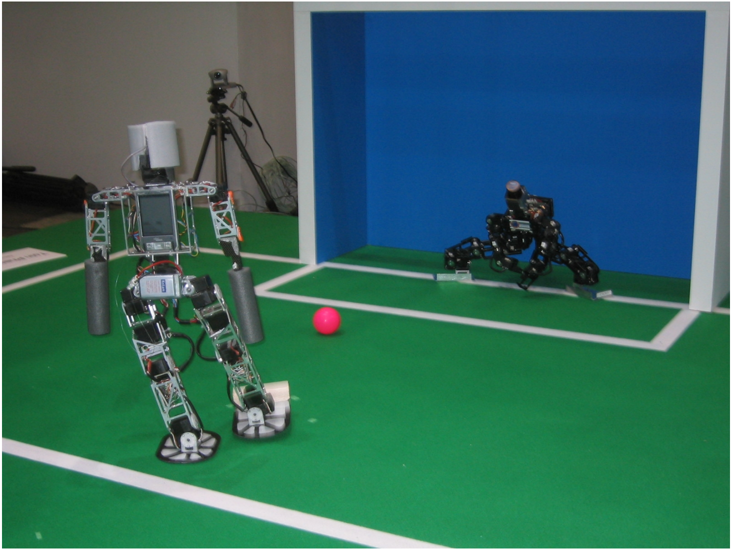

- It's ~~difficult~~ impossible to *program* a solution to this.

```
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.

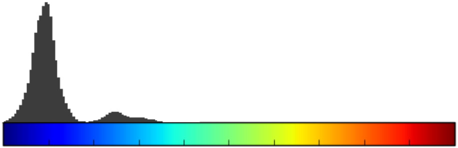
# Classification

# Example – RoboCup

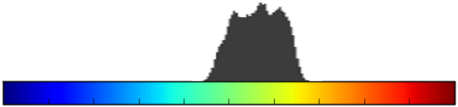


**Goal: blue** **Floor: green/white** **Ball: red**

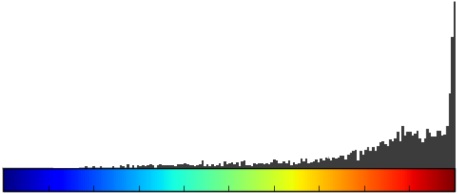
**Goal: blue**

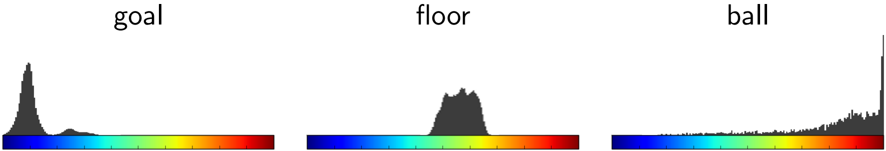


**Floor: green**



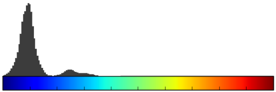
**Ball: red**



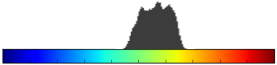

goal
floor
ball

# Example – RoboCup

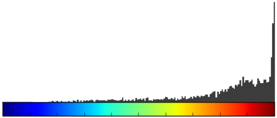
goal



floor

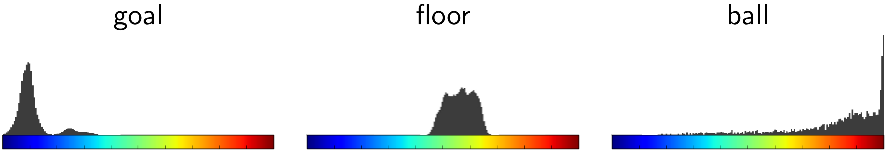


ball

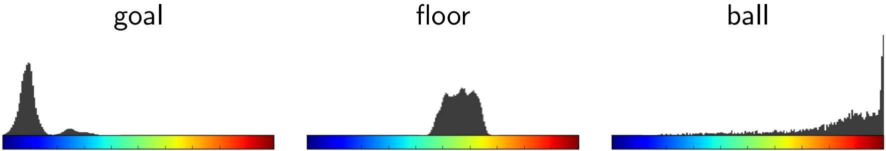


New object: 

# Example – RoboCup

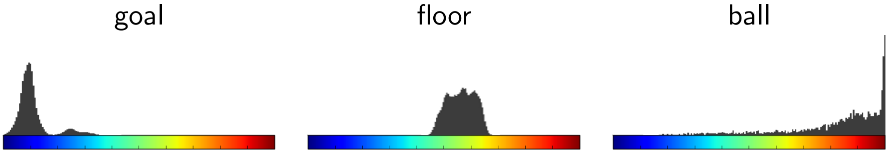



New object:  $\rightarrow$ ball




New object: → ball

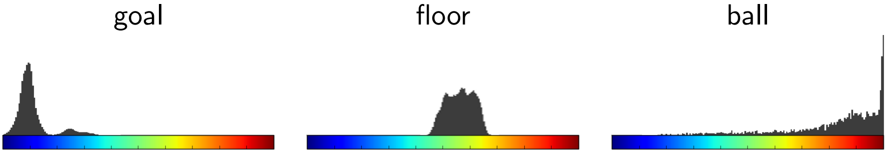
New object:




New object:  $\rightarrow$ ball

New object:  $\rightarrow$ floor

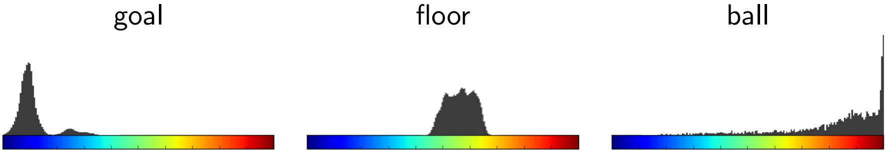
# Example – RoboCup




New object:  → ball

New object:  → floor

New object: 

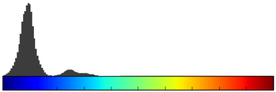



New object:  $\rightarrow$ ball

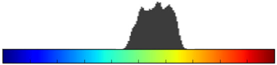
New object:  $\rightarrow$ floor

New object:  $\rightarrow$ goal

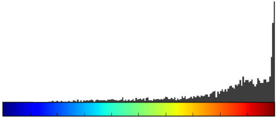
goal



floor



ball

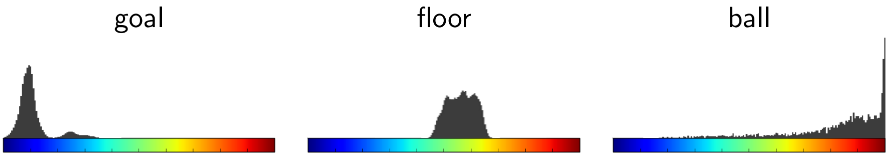


New object:  → ball

New object:  → floor

New object:  → goal

New object: 




New object:  → ball

New object:  → floor

New object:  → goal

New object:  → floor

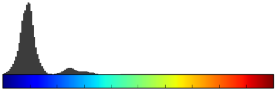
# Bayesian Decision Theory

**Notation...**

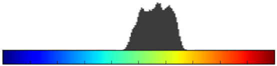
- data: $x \in \mathcal{X} = \mathbb{R}^d$, (here: colors, $d = 3$)
- labels: $y \in \mathcal{Y} = \{$goal, floor, ball$\}$, (here: object classes)
- goal: classification rule $g : \mathcal{X} \to \mathcal{Y}$.

Histograms: class-conditional probability densities $p(x|y)$. For any $y \in \mathcal{Y}$

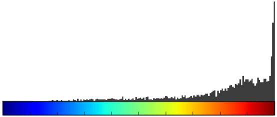
$$\forall x \in \mathcal{X} : p(x|y) \geq 0 \qquad\qquad \sum_{x \in \mathcal{X}} p(x|y) = 1$$



$p(x|y = \text{goal})$



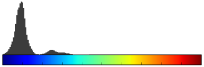
$p(x|y = \text{floor})$



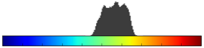
$p(x|y = \text{ball})$

Maximum Likehood Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

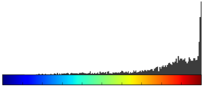
Assume: fourth class: *sun*, but occurs only outdoors



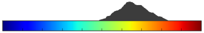
$p(x|y = \text{goal})$



$p(x|y = \text{floor})$



$p(x|y = \text{ball})$

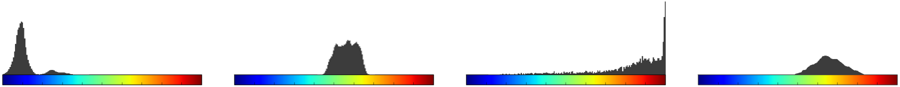


$p(x|y = \text{sun})$

Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

# Bayesian Decision Theory

Assume: fourth class: *sun*, but occurs only outdoors

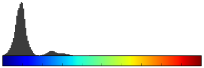


$p(x|y = \text{goal})$ $p(x|y = \text{floor})$ $p(x|y = \text{ball})$ $p(x|y = \text{sun})$

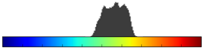
Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

New object:  $\rightarrow$ ball

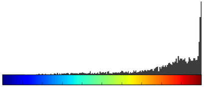
Assume: fourth class: *sun*, but occurs only outdoors



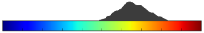
$p(x|y = \text{goal})$



$p(x|y = \text{floor})$



$p(x|y = \text{ball})$



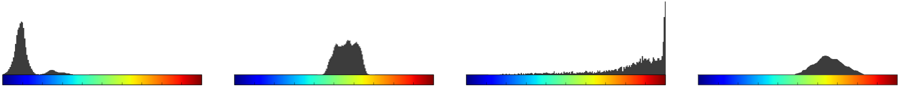
$p(x|y = \text{sun})$

Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

New object:  $\rightarrow$ ball

New object:  $\rightarrow$ floor

Assume: fourth class: *sun*, but occurs only outdoors



$p(x|y = \text{goal})$ $p(x|y = \text{floor})$ $p(x|y = \text{ball})$ $p(x|y = \text{sun})$

Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

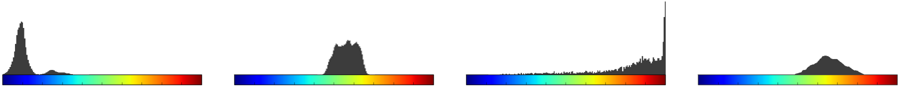
New object:  $\rightarrow$ ball

New object:  $\rightarrow$ floor

New object:  $\rightarrow$ goal

# Bayesian Decision Theory

Assume: fourth class: *sun*, but occurs only outdoors



$p(x|y = \text{goal})$ $p(x|y = \text{floor})$ $p(x|y = \text{ball})$ $p(x|y = \text{sun})$

Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

New object:  $\rightarrow$ ball

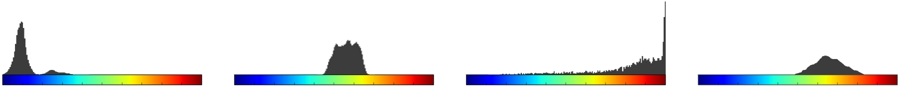
New object:  $\rightarrow$ floor

New object:  $\rightarrow$ goal

New object:  $\rightarrow$ sun

# Bayesian Decision Theory

Assume: fourth class: *sun*, but occurs only outdoors



$p(x|y = \text{goal})$ $p(x|y = \text{floor})$ $p(x|y = \text{ball})$ $p(x|y = \text{sun})$

Maximum Likehood (ML) Rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(x|y)$

New object:  $\rightarrow$ ball

New object:  $\rightarrow$ floor

New object:  $\rightarrow$ goal

New object:  $\rightarrow$ sun

**We must take into account how likely it is to see a class at all!**

**Notation:**

- class conditional densities: $p(x|y)$ for all $y \in \mathcal{Y}$
- class priors: $p(y)$ for all $y \in \mathcal{Y}$
- goal: decision rule $g : \mathcal{X} \to \mathcal{Y}$ that results in fewest mistakes

For any input $x \in \mathcal{X}$:

$$p(\textit{mistake}|x) = \sum_{y \in \mathcal{Y}} p(y|x)[\![g(x) \neq y]\!] \qquad [\![P]\!] = \begin{cases} 1 & \text{if } P = \texttt{true} \\ 0 & \textit{otherwise} \end{cases}$$

$$p(\textit{no mistake}|x) = \sum_{y \in \mathcal{Y}} p(y|x)[\![g(x) = y]\!] = p(\, g(x)|x \,)$$

**Notation:**

- class conditional densities: $p(x|y)$ for all $y \in \mathcal{Y}$
- class priors: $p(y)$ for all $y \in \mathcal{Y}$
- goal: decision rule $g : \mathcal{X} \to \mathcal{Y}$ that results in fewest mistakes

For any input $x \in \mathcal{X}$:

$$p(\textit{mistake}|x) = \sum_{y \in \mathcal{Y}} p(y|x) [\![ g(x) \neq y ]\!] \qquad [\![ P ]\!] = \begin{cases} 1 & \text{if } P = \texttt{true} \\ 0 & \textit{otherwise} \end{cases}$$

$$p(\textit{no mistake}|x) = \sum_{y \in \mathcal{Y}} p(y|x) [\![ g(x) = y ]\!] = p(\, g(x)|x \,)$$

Optimal decision rule: $g(x) = \mathrm{argmax}_{y \in \mathcal{Y}}\, p(y|x)$ "Bayes classifier"

How to get "class posterior" $p(y|x)$?

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \qquad \text{(Bayes' rule)}$$

- $p(x|y)$: class conditional density (here: histograms)
- $p(y)$: class priors, e.g. for indoor RoboCup
  $p(\text{floor}) = 0.6, \quad p(\text{goal}) = 0.3, \quad p(\text{ball}) = 0.1, \quad p(\text{sun}) = 0$
- $p(x)$: probability of seeing data $x$

Equivalent rules:

$$\begin{aligned} g(x) &= \operatorname*{argmax}_{y\in\mathcal{Y}} p(y|x) \;=\; \operatorname*{argmax}_{y\in\mathcal{Y}} \frac{p(x|y)p(y)}{p(x)} \\ &= \operatorname*{argmax}_{y\in\mathcal{Y}} p(x|y)p(y) \\ &= \operatorname*{argmax}_{y\in\mathcal{Y}} p(x,y) \end{aligned}$$

Special case: binary classification, $\mathcal{Y} = \{-1, +1\}$

$$\operatorname*{argmax}_{y\in\mathcal{Y}} p(y|x) = \begin{cases} +1 & \text{if } p(+1|x) > p(-1|x), \\ -1 & \text{if } p(+1|x) \leq p(-1|x). \end{cases}$$

Equivalent rules:

$$\begin{aligned} g(x) &= \operatorname*{argmax}_{y\in\mathcal{Y}} p(y|x) \\ &= \operatorname{sign}\ \big(p(+1|x) - p(-1|x)\big) \\ &= \operatorname{sign}\ \log \frac{p(+1|x)}{p(-1|x)} \end{aligned}$$

With $\operatorname{sign}(t) := \begin{cases} +1 & \text{if } t > 0, \\ -1 & \text{otherwise.} \end{cases}$.

Not all mistakes are equally bad:

- mistake *opponent goal* as *your goal*:
  You don't shoot, missed opportunity to score: *bad*
- mistake *your goal* as *opponent goal*:
  You shoot, score own-goal: *much worse*!

Formally:

- **loss function**, $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$
- $\Delta(y, \bar{y})$ = cost of predicting $\bar{y}$ if $y$ is correct.

$\Delta_{\text{goals}}$:

| $y \setminus \bar{y}$ | opponent | own |
|---|---|---|
| opponent | 0 | 2 |
| own | 10 | 0 |

- Convention: $\Delta(y, y) = 0$ for all $y \in \mathcal{Y}$ (correct decision has $0$ loss)

Reminder: $\Delta(y, \bar{y}) =$ cost of predicting $\bar{y}$ if $y$ is correct.

Optimal decision: choose $g : \mathcal{X} \to \mathcal{Y}$ to **minimize the expected loss**

$$L_\Delta(y; x) = \sum_{\bar{y} \neq y} p(\bar{y}|x)\Delta(\bar{y}, y) \quad = \sum_{\bar{y} \in \mathcal{Y}} p(\bar{y}|x)\Delta(\bar{y}, y) \qquad (\Delta(y, y) = 0)$$

$$g(x) = \operatorname*{argmin}_{y \in \mathcal{Y}} L_\Delta(y; x) \qquad \text{pick label of smallest expected loss}$$

Reminder: $\Delta(y, \bar{y})$ = cost of predicting $\bar{y}$ if $y$ is correct.

Optimal decision: choose $g : \mathcal{X} \to \mathcal{Y}$ to **minimize the expected loss**

$$L_\Delta(y; x) = \sum_{\bar{y} \neq y} p(\bar{y}|x)\Delta(\bar{y}, y) \quad = \sum_{\bar{y} \in \mathcal{Y}} p(\bar{y}|x)\Delta(\bar{y}, y) \qquad (\Delta(y, y) = 0)$$

$$g(x) = \operatorname*{argmin}_{y \in \mathcal{Y}} L_\Delta(y; x)$$ pick label of smallest expected loss

Special case: $\Delta(y, \bar{y}) = [\![ y \neq y ]\!]$. E.g. (for 3 labels)

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$g_\Delta(x) = \operatorname*{argmin}_{y \in \mathcal{Y}} L_\Delta(y) = \operatorname*{argmin}_{y \in \mathcal{Y}} \sum_{\bar{y} \neq y} p(y|x)[\![ y \neq y ]\!]$$

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

($\to$ Bayes classifier)

# Learning Paradigms

Given: training data $\{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$

**Approach 1)** Generative Probabilistic Models

1) Use training data to obtain an estimate $p(x|y)$ for any $y \in \mathcal{Y}$
2) Compute $p(y|x) \propto p(x|y)p(y)$
3) Predict using $g(x) = \operatorname{argmin}_y \sum_{\bar{y}} p(\bar{y}|x)\Delta(\bar{y}, y)$.

**Approach 2)** Discriminative Probabilistic Models

1) Use training data to estimate $p(y|x)$ directly.
2) Predict using $g(x) = \operatorname{argmin}_y \sum_{\bar{y}} p(\bar{y}|x)\Delta(\bar{y}, y)$.

**Approach 3)** Loss-minimizing Parameter Estimation

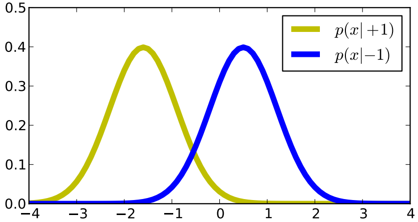
1) Use training data to search for best $g : \mathcal{X} \to \mathcal{Y}$ directly.

This is what we did in the RoboCup example!

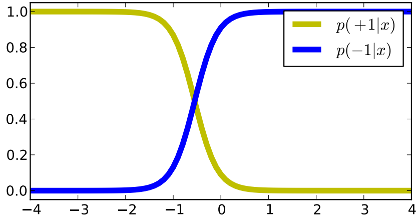
- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, x_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- For each $y \in \mathcal{Y}$, build model for $p(x|y)$ of $X_y := \{x_i \in X : y_i = y\}$
  - **Histogram:** if $x$ can have only few discrete values.
  - **Kernel Density Estimator:** $p(x|y) \propto \sum_{x_i \in X_y} k(x_i, x)$
  - **Gaussian:** $p(x|y) = \mathcal{G}(x; \mu_y, \Sigma_y) \propto \exp(-\frac{1}{2}(x - \mu_y)^\top \Sigma_y^{-1}(x - \mu_y))$
  - **Mixture of Gaussians:** $p(x|y) = \sum_{k=1}^{K} \pi_y^k \mathcal{G}(x; \mu_y^k, \Sigma_y^k)$

This is what we did in the RoboCup example!

- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, x_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- For each $y \in \mathcal{Y}$, build model for $p(x|y)$ of $X_y := \{x_i \in X : y_i = y\}$
  - **Histogram:** if $x$ can have only few discrete values.
  - **Kernel Density Estimator:** $p(x|y) \propto \sum_{x_i \in X_y} k(x_i, x)$
  - **Gaussian:** $p(x|y) = \mathcal{G}(x; \mu_y, \Sigma_y) \propto \exp(-\frac{1}{2}(x-\mu_y)^\top \Sigma_y^{-1}(x-\mu_y))$
  - **Mixture of Gaussians:** $p(x|y) = \sum_{k=1}^K \pi_y^k \mathcal{G}(x; \mu_y^k, \Sigma_y^k)$




class conditional densities (Gaussian)




class posteriors for $p(+1) = p(-1) = \frac{1}{2}$

This is what we did in the RoboCup example!

- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, x_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- For each $y \in \mathcal{Y}$, build model for $p(x|y)$ of $X_y := \{x_i \in X : y_i = y\}$
  - **Histogram:** if $x$ can have only few discrete values.
  - **Kernel Density Estimator:** $p(x|y) \propto \sum_{x_i \in X_y} k(x_i, x)$
  - **Gaussian:** $p(x|y) = \mathcal{G}(x; \mu_y, \Sigma_y) \propto \exp(-\frac{1}{2}(x - \mu_y)^\top \Sigma_y^{-1}(x - \mu_y))$
  - **Mixture of Gaussians:** $p(x|y) = \sum_{k=1}^{K} \pi_y^k \mathcal{G}(x; \mu_y^k, \Sigma_y^k)$

Typically: $\mathcal{Y}$ small, i.e. few possible labels,

$\mathcal{X}$ low-dimensional, e.g. RGB colors, $\mathcal{X} = \mathbb{R}^3$

But: large $\mathcal{Y}$ is possible with right tools $\rightarrow$ *"Intro to graphical models"*

Most popular: **Logistic Regression**

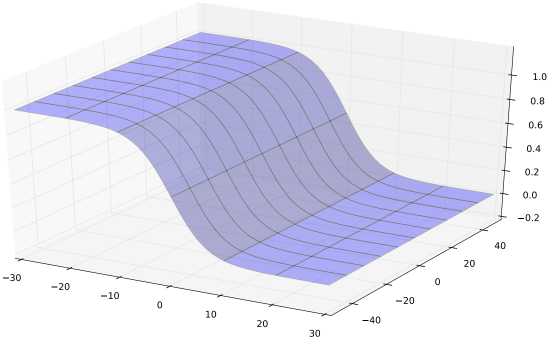
- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- To simplify notation: assume $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{\pm 1\}$
- **Parametric model:**

$$p(y|x) = \frac{1}{1 + \exp(-y\, w^\top x)} \qquad \text{with free parameter } w \in \mathbb{R}^d$$

Most popular: **Logistic Regression**

- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- To simplify notation: assume $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{\pm 1\}$
- **Parametric model:**

$$p(y|x) = \frac{1}{1 + \exp(-y\, w^\top x)} \qquad \text{with free parameter } w \in \mathbb{R}^d$$



Most popular: **Logistic Regression**

- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- To simplify notation: assume $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{\pm 1\}$
- **Parametric model:**

$$p(y|x) = \frac{1}{1 + \exp(-y\, w^\top x)} \qquad \text{with free parameter } w \in \mathbb{R}^d$$

- Find $w$ by maximizing the conditional data likelihood

$$\begin{aligned} w &= \operatorname*{argmax}_{w \in \mathbb{R}^d} \prod_{i=1}^{n} p(y_i|x_i) \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log\big(1 + \exp(-y_i\, w^\top x_i)\big) \end{aligned}$$

Extensions to very large $\mathcal{Y}$ $\quad\rightarrow$ *"Structured Outputs (Wednesday)"*

- Training data $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, x_n\}$. $X \times Y \subset \mathcal{X} \times \mathcal{Y}$
- Simplify: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{\pm 1\}$, $\Delta(y, \bar{y}) = [\![ y \neq \bar{y} ]\!]$
- Choose **hypothesis class:** (which classifiers do we consider?)

$$\mathcal{H} = \{g : \mathcal{X} \to \mathcal{Y}\} \qquad \text{(e.g. all linear classifiers)}$$

- Expected loss of a classifier $h : \mathcal{X} \to \mathcal{Y}$ on a sample $x$

$$L(g, x) = \sum_{y \in \mathcal{Y}} p(y|x) \Delta(\, y, g(x)\,)$$

- Expected overall loss of a classifier:

$$\begin{aligned} L(g) &= \sum_{x \in \mathcal{X}} p(x) L(g, x) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \Delta(\, y, g(x)\,) = \mathbb{E}_{x,y} \Delta(y, g(x)) \end{aligned}$$

- Task: find "best" $g$ in $\mathcal{H}$, i.e. $g := \operatorname{argmin}_{g \in \mathcal{H}} L(g)$

Note: for simplicity, we always write $\sum_x$. When $\mathcal{X}$ is infinite (i.e. almost always), read this as $\int_{\mathcal{X}} dx$

Part II:
$\mathcal{H} = \{$**linear classifiers**$\}$

Part III:
$\mathcal{H} = \{$**nonlinear classifiers**$\}$

Part IV (if there's time):
**Multi-class Classification**

# Notation...

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

# Notation...

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

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

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

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

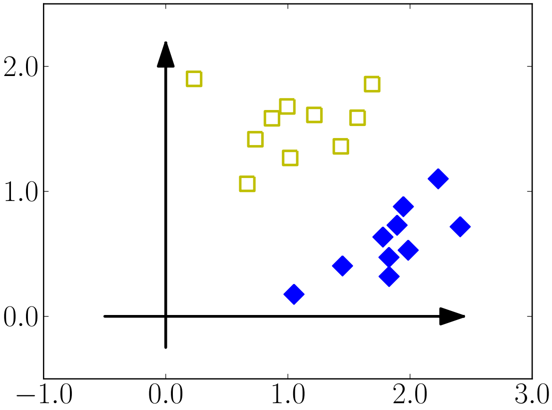
$$f(x) = \langle \hat{w}, \hat{x} \rangle$$

(inner/scalar product in $\mathbb{R}^{d+1}$)

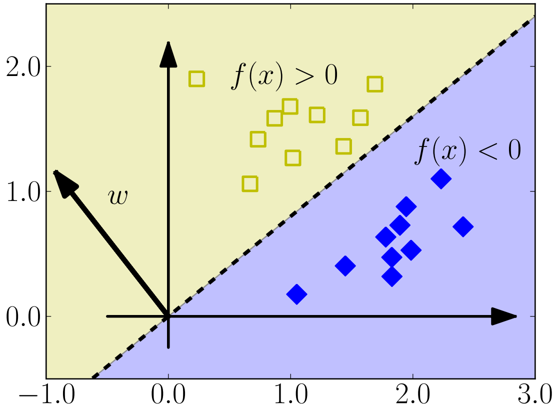
(also: $\hat{w} \cdot \hat{x}$ or $\hat{w}^\top \hat{x}$)

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

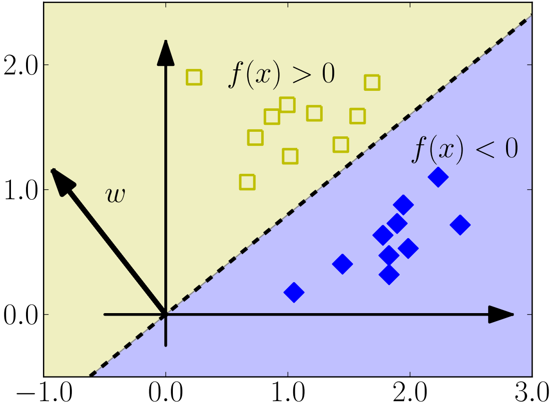
Given $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$.



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



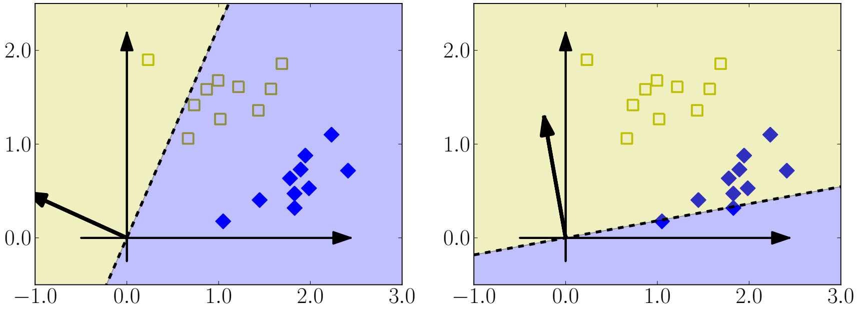
Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. Any $w$ partitions the data space into two half-spaces by means of $f(x) = \langle w, x \rangle$.




"What's the **best** $w$?"

**What properties should an optimal $w$ have?**

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

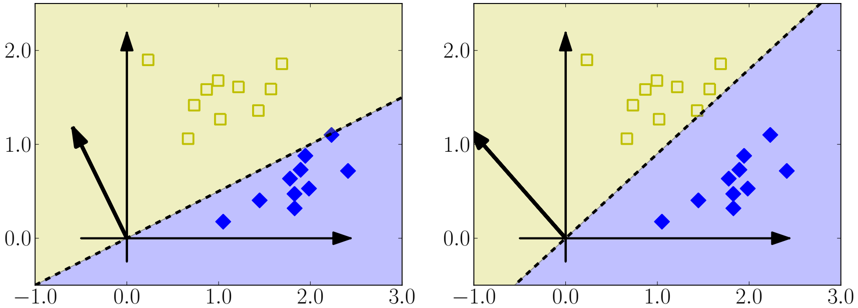



Are these the best? No, they misclassify many examples.

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

**What properties should an optimal $w$ have?**

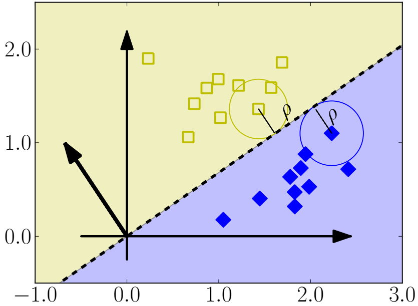
Given $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_n\}$. What's the best $w$?




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

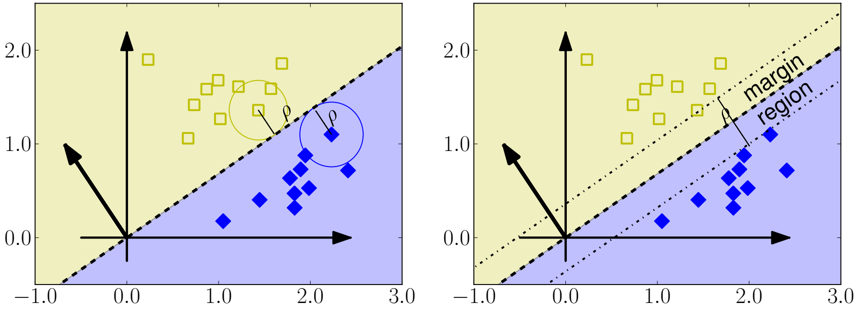
$\boxed{\textbf{Criterion 2}\text{: Ensure } \operatorname{sign}\langle w, x\rangle = y \text{ for future } (x, y) \text{ as well.}}$

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.

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:

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

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

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

subject to

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

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

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

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

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

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)**

From the view of optimization theory

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

subject to

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

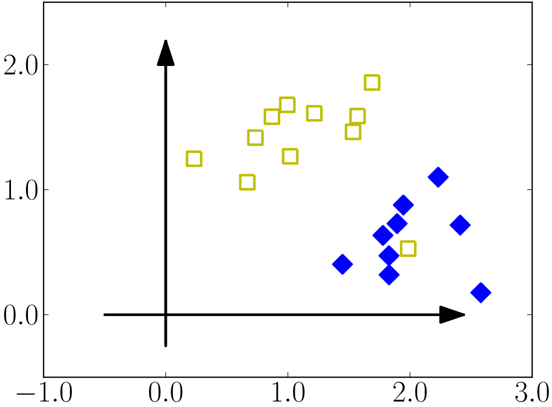
is rather easy:

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

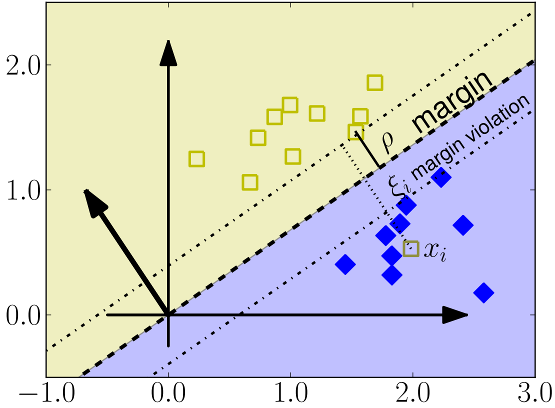
We can find the *globally* optimal $w$ in $O(d^3)$ (usually much faster).

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

What is the best $w$ for this dataset?

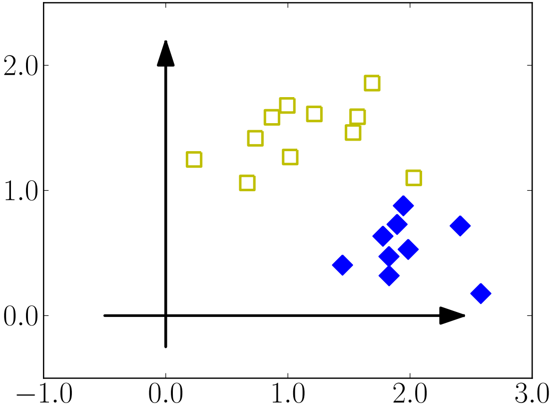


What is the best $w$ for this dataset?

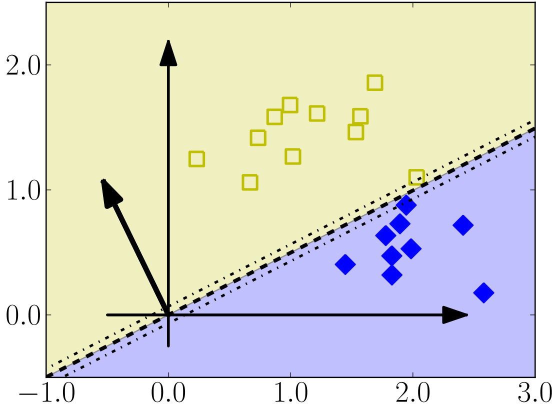



Possibly this one, even though one sample is misclassified.

What is the best $w$ for this dataset?

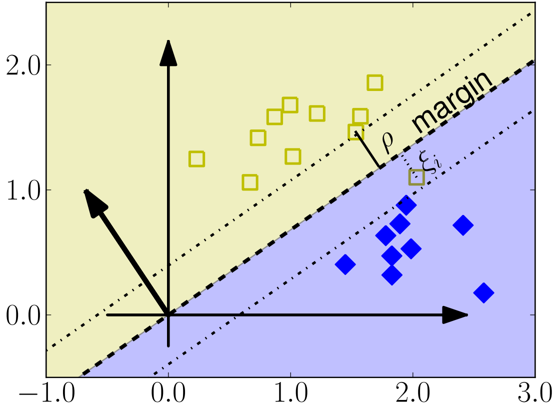


What is the best $w$ for this dataset?




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

What is the best $w$ for this dataset?




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

Mathematically, we formulate the trade-off by *slack*-variables $\xi_i$:

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

subject to

$$y_i \langle w, x_i \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n,$$
$$\xi_i \geq 0 \qquad \text{for } i = 1, \ldots, n.$$

**Linear Support Vector Machine (linear SVM)**

Mathematically, we formulate the trade-off by *slack*-variables $\xi_i$:

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

subject to

$$y_i \langle w, x_i \rangle \geq 1 - \xi_i \qquad \text{for } i = 1, \ldots n,$$
$$\xi_i \geq 0 \qquad \text{for } i = 1, \ldots, n.$$

**Linear Support Vector Machine (linear SVM)**

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

# Solving for Soft-Margin Solution

Reformulate:

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

subject to

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

$$\xi_i \geq 0 \qquad \text{for } i = 1,\dots,n.$$

We can read off the optimal values $\xi_i = \max\{0,\ 1 - y_i\langle w, x_i\rangle\}$.

Equivalent optimization problem (with $\lambda = 1/C$):

$$\min_{w\in\mathbb{R}^d} \lambda\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\max\{0,\ 1 - y_i\langle w, x_i\rangle\}$$

- Now unconstrained optimization, but non-differentiable
- Solve efficiently, e.g., by *subgradient* method

→ *"Large-scale visual recognition"* (Thursday)

# 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`
- see also: **Pegasos:**, `http://www.cs.huji.ac.il/~shais/code/`
- see 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.**

Geometric intuition is nice, but are there any *guarantees*?

- SVM solution is $g(x) = \operatorname{sign} f(x)$ for $f(x) = \langle w, x \rangle$ with

$$w = \operatorname*{argmin}_{w \in \mathbb{R}^d} \ \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, \ 1 - y_i \langle w, x_i \rangle\}$$

- What we really wanted to minimized is *expected loss*:

$$g = \operatorname*{argmin}_{h \in \mathcal{H}} \ \mathbb{E}_{x,y} \ \Delta(y, g(x))$$

with $\mathcal{H} = \{ \ g(x) = \operatorname{sign} f(x) \ | \ f(x) = \langle w, x \rangle$ for $w \in \mathbb{R}^d \}$.

**What's the relation?**

Geometric intuition is nice, but are there any *guarantees*?

- SVM solution is $g(x) = \operatorname{sign} f(x)$ for $f(x) = \langle w, x \rangle$ with

$$w = \operatorname*{argmin}_{w \in \mathbb{R}^d} \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0,\ 1 - y_i \langle w, x_i \rangle\}$$

- What we really wanted to minimized is *expected loss*:

$$g = \operatorname*{argmin}_{h \in \mathcal{H}} \mathbb{E}_{x,y}\ \Delta(y, g(x))$$

with $\mathcal{H} = \{\ g(x) = \operatorname{sign} f(x)\ \mid\ f(x) = \langle w, x \rangle$ for $w \in \mathbb{R}^d\}$.

**What's the relation?**

Geometric intuition is nice, but are there any *guarantees*?

- SVM solution is $\boxed{g(x) = \operatorname{sign} f(x)}$ for $f(x) = \langle w, x \rangle$ with

$$w = \operatorname*{argmin}_{w \in \mathbb{R}^d} \; \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0, \; 1 - y_i \langle w, x_i \rangle\}$$

- What we really wanted to minimized is *expected loss*:

$$g = \operatorname*{argmin}_{h \in \mathcal{H}} \; \mathbb{E}_{x,y} \; \Delta(y, g(x))$$

with $\mathcal{H} = \{ \boxed{g(x) = \operatorname{sign} f(x)} \mid f(x) = \langle w, x \rangle$ for $w \in \mathbb{R}^d \}$.

**What's the relation?**

Geometric intuition is nice, but are there any *guarantees*?

- SVM solution is $g(x) = \operatorname{sign} f(x)$ for $f(x) = \langle w, x \rangle$ with

$$\boxed{w = \operatorname*{argmin}_{w \in \mathbb{R}^d} \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \max\{0,\ 1 - y_i \langle w, x_i \rangle\}}$$

- What we really wanted to minimized is *expected loss*:

$$\boxed{g = \operatorname*{argmin}_{h \in \mathcal{H}} \mathbb{E}_{x,y}\, \Delta(y, g(x))}$$

with $\mathcal{H} = \{\, g(x) = \operatorname{sign} f(x) \ \mid\ f(x) = \langle w, x \rangle \text{ for } w \in \mathbb{R}^d \}$.

**What's the relation?**

SVM training is an example of **Regularized Risk Minimization**.

General form:

$$\min_{f\in\mathcal{F}} \quad \underbrace{\Omega(f)}_{\text{regularizer}} \quad + \quad \underbrace{\frac{1}{n}\sum_{i=1}^{n}\ell(y_i, f(x_i))}_{\text{loss on training set: 'risk'}}$$

Support Vector Machine:

$$\min_{w\in\mathbb{R}^d} \quad \lambda\|w\|^2 \quad + \quad \frac{1}{n}\sum_{i=1}^{n}\max\{0, 1 - y_i\langle w, x_i\rangle\}$$

- $\mathcal{F} = \{f(x) = \langle w, x\rangle | w \in \mathbb{R}^d\}$
- $\Omega(f) = \|w\|^2$ for any $f(x) = \langle w, x\rangle$
- $\ell(y, f(x)) = \max\{0, 1 - yf(x)\}$ *(Hinge loss)*

Observation 1: **The empirical loss approximates the expected loss.**
For $i.i.d.$ training examples $(x_1, y_1), \dots, (x_n, y_n)$:

$$\mathbb{E}_{x,y}\big(\Delta(y, g(x))\big) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y)\Delta(y, g(x)) \approx \frac{1}{n}\sum_{i=1}^{n} \Delta(\, y_i, g(x_i)\,)$$

Observation 1: **The empirical loss approximates the expected loss.**
For $i.i.d.$ training examples $(x_1, y_1), \dots, (x_n, y_n)$:

$$\mathbb{E}_{x,y}\big(\Delta(y, g(x))\big) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y)\Delta(y, g(x)) \approx \frac{1}{n}\sum_{i=1}^{n} \Delta(\, y_i, g(x_i)\,)$$

Observation 2: **The Hinge loss upper bounds the 0/1-loss.**
For $\Delta(y, \bar{y}) = [\![y \neq \bar{y}]\!]$ and $g(x) = \operatorname{sign}\langle w, x\rangle$ one has

$$\Delta(\, y, g(x)\,) = [\![y\langle w, x\rangle < 0]\!] \;\leq\; \max\{0, 1 - y\langle w, x\rangle\}$$

Observation 1: **The empirical loss approximates the expected loss.**
For $i.i.d.$ training examples $(x_1, y_1), \dots, (x_n, y_n)$:

$$\mathbb{E}_{x,y}\big(\Delta(y, g(x))\big) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y)\Delta(y, g(x)) \approx \frac{1}{n}\sum_{i=1}^{n} \Delta(\, y_i, g(x_i)\,)$$

Observation 2: **The Hinge loss upper bounds the 0/1-loss.**
For $\Delta(y, \bar{y}) = [\![y \neq \bar{y}]\!]$ and $g(x) = \operatorname{sign}\langle w, x\rangle$ one has

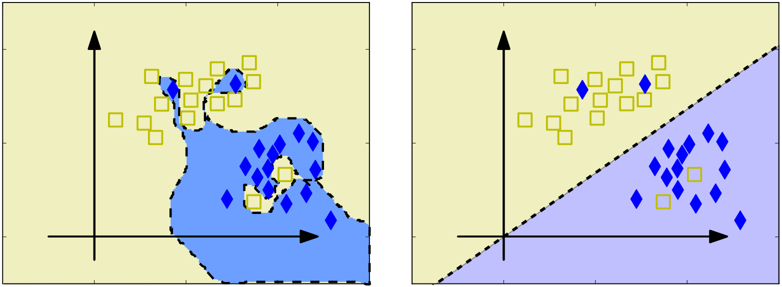
$$\Delta(\, y, g(x)\,) = [\![y\langle w, x\rangle < 0]\!] \ \leq \ \max\{0, 1 - y\langle w, x\rangle\}$$

Combination:

$$\mathbb{E}_{x,y}\big(\Delta(y, g(x))\big) \quad \lesssim \quad \frac{1}{n}\sum_{i} \max\{0, 1 - y_i\langle w, x_i\rangle\}$$

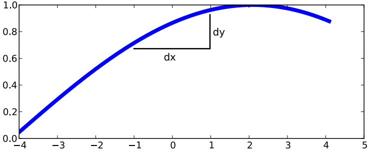
Intuition: small "risk" term in SVM $\rightarrow$ few mistakes in the future

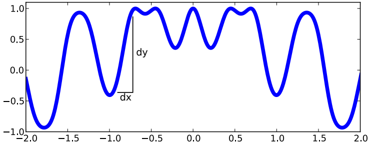
Observation 3: **Only minimizing the loss term can lead to overfitting.**



We want classifiers that have small loss, but are **simple** enough to generalize.

Ad-hoc definition: a function $f : \mathbb{R}^d \to \mathbb{R}$ is *simple*, if it not very sensitive to the exact input




sensitivity is measured by slope: $f'$

For linear $f(x) = \langle w, x \rangle$, slope is $\|\nabla_x f\| = \|w\|$:

**Minimizing $\|w\|^2$ encourages "simple" functions**

---

Formal results, including proper bounds on the generalization error: e.g.

[Shawe-Taylor, Cristianini: "Kernel Methods for Pattern Analysis", Cambridge U Press, 2004]

There are many other RRM-based classifiers, including variants of SVM:

**L1-regularized Linear SVM**

$$\min_{w \in \mathbb{R}^d} \quad \lambda \|w\|_{L^1} \quad + \quad \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \langle w, x_i \rangle\}$$

$\|w\|_{L^1} = \sum_{j=1}^{d} |w_j|$ encourages **sparsity**

- learned weight vector $w$ will have many zero entries
- acts as *feature selector*
- evaluation $f(x) = \langle w, x \rangle$ becomes more efficient

Use if you have *prior knowledge* that optimal classifier should be sparse.

**SVM with squared slacks / squared Hinge loss**

$$\min_{w\in\mathbb{R}^d} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{n}\sum_{i=1}^{n}\xi_i^2$$

subject to $\qquad y_i\langle w, x_i\rangle \geq 1-\xi_i \quad$ and $\quad \xi_i \geq 0.$

Equivalently:

$$\min_{w\in\mathbb{R}^d} \qquad \lambda\|w\|_{L^1} \quad + \quad \frac{1}{n}\sum_{i=1}^{n}(\max\{0, 1-y_i\langle w, x_i\rangle\})^2$$

Also has a max-margin interpretation, but objective is *once differentiable*.

**Least-Squares SVM** aka **Ridge Regression**

$$\min_{w\in\mathbb{R}^d} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{n}\sum_{i=1}^{n}(1-y_i\langle w,x_i\rangle)^2$$

Loss function: $\ell(y,f(x))=(y-f(x))^2$ *"squared loss"*

- Easier to optimize than regular SVM: closed-form solution for $w$

$$w=y^\top(\lambda\mathsf{Id}+XX^\top)^{-1}X^\top$$

- But: loss does not really reflect *classification*:
  $\ell(y,f(x))$ can be big, even if $\operatorname{sign} f(x)=y$

**Regularized Logistic Regression**

$$\min_{w\in\mathbb{R}^d} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(\,-y_i\langle w,x_i\rangle\,))$$

Loss function: $\ell(\,y,f(x)\,)=\log(1+\exp(\,-y_i\langle w,x_i\rangle))$ *"logistic loss"*

- Smooth ($C^\infty$-differentiable) objective
- Often similar results to SVM

# Summary – Linear Classifiers

## (Linear) Support Vector Machines

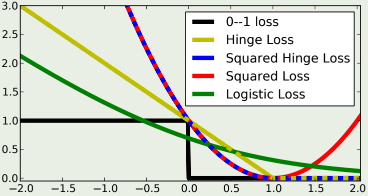
- geometric intuition: maximum margin classifier
- well understood theory: regularized risk minimization

## (Linear) Support Vector Machines

- geometric intuition: maximum margin classifier
- well understood theory: regularized risk minimization

## Many variants of losses and regularizers

- first: try $\Omega(\cdot) = \|\cdot\|^2$
- encourage sparsity: $\Omega(\cdot) = \|\cdot\|_{L^1}$
- differentiable losses:
  easier numeric optimization



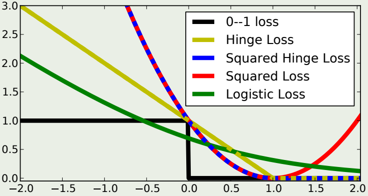
# Summary – Linear Classifiers

## (Linear) Support Vector Machines

- geometric intuition: maximum margin classifier
- well understood theory: regularized risk minimization

## Many variants of losses and regularizers

- first: try $\Omega(\cdot) = \|\cdot\|^2$
- encourage sparsity: $\Omega(\cdot) = \|\cdot\|_{L^1}$
- differentiable losses:
  easier numeric optimization

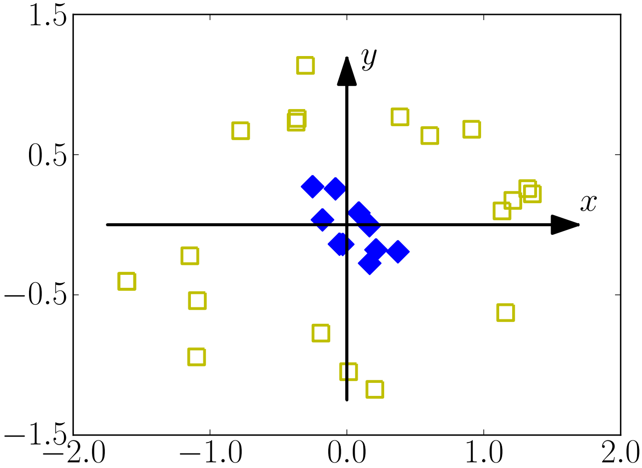



## Fun fact: different losses often have similar empirical performance

- don't blindly believe claims *"My classifier is the best."*

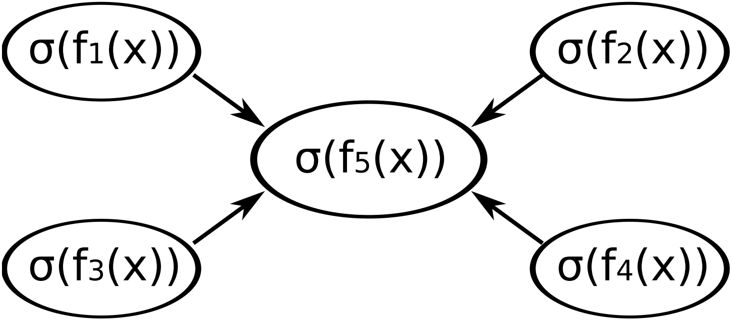
# Nonlinear Classification

What is the best linear classifier for this dataset?




None. We need something nonlinear!

**Idea 1) Combine multiple linear classifiers into nonlinear classifier**



## Boosting

Situation:

- we have many simple classifiers (typically linear), $h_1, \dots, h_k : \mathcal{X} \to \{\pm 1\}$
- none of them is particularly good

Method:

- construct stronger nonlinear classifier:

$$g(x) = \operatorname{sign} \sum_j \alpha_j h_j(x) \quad \text{with } \alpha_j \in \mathbb{R}$$

- typically: iterative construction for finding $\alpha_1, \alpha_2, \dots$

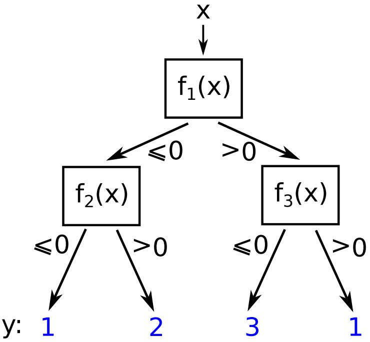
Advantage:

- very easy to implement

Disadvantage:

- computationally expensive to train
- finding base classifiers can be hard

## Decision Trees




Advantage:

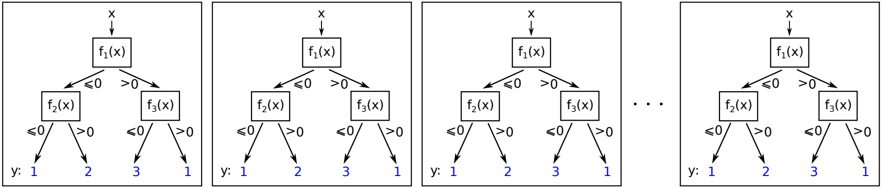
- easy to interpret
- handles multi-class situation

Disadvantage:

- by themselves typically worse results than other modern methods

[Breiman, Friedman, Olshen, Stone, "Classification and regression trees", 1984]

## Random Forest




Method:

- construct many decision trees **randomly** (under some constraints)
- classify using majority vote

Advantage:

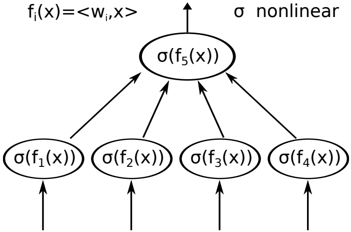
- conceptually easy
- works surprisingly well

Disadvantage:

- computationally expensive to train
- expensive at test time if forest has many trees

[Breiman, "Random Forests", 2001]

## Artificial Neural Network / Multilayer Perceptron / Deep Learning




Multi-layer architecture:

- first layer: inputs $x$
- each layer $k$ evaluates $f_1^k, \dots, f_m^k$ feeds output to next layer
- last layer: output $y$

Advantage:

- biologically inspired $\rightarrow$ easy to explain to non-experts
- efficient at evaluation time

Disadvantage:

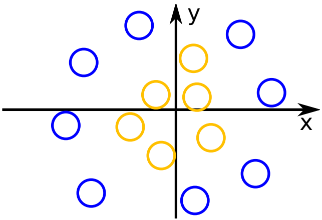
- non-convex optimization problem
- many design parameters, few theoretic results

$\rightarrow$ *"Deep Learning" (Tuesday)*

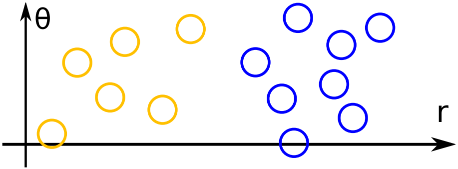
[Rumelhart, Hinton, Williams, "Learning Internal Representations by Error Propagation", 1986]

**Idea 2) Preprocess the data**

This dataset is not *linearly separable*:




This one is separable:

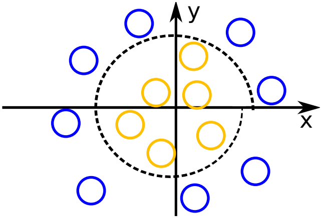



But: both are *the same dataset*!

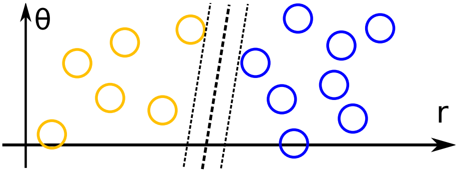
Top: Cartesian coordinates. Bottom: polar coordinates

**Idea 2) Preprocess the data**

*Nonlinear separation*:




Linear Separation




Linear classifier in polar space acts nonlinearly in Cartesian space.

Given

- $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$.
- Given any (nonlinear) feature map $\phi : \mathbb{R}^k \to \mathbb{R}^m$.

Solve the minimization for $\phi(x_1), \dots, \phi(x_n)$ instead of $x_1, \dots, x_n$:

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

subject to

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

- 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 $f(x) = \langle w, \phi(x) \rangle$ is *linear* in $w$, but *nonlinear* in $x$.

- Polar coordinates:

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

- $d$-th degree polynomials:

$$\phi : \left(x_1, \ldots, x_n\right) \mapsto \left(1, x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, \ldots, x_1^d, \ldots, x_n^d\right)$$

- Distance map:

$$\phi : \vec{x} \mapsto \left(\|\vec{x} - \vec{p}_i\|, \ldots, \|\vec{x} - \vec{p}_N\|\right)$$

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

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

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

subject to

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

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

# Representer Theorem

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

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

subject to

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

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*

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

[Schölkopf, Smola, "Learning with Kernels", 2001]

Rewrite the optimization using the representer theorem:

- insert $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$ everywhere,
- minimize over $\alpha_i$ instead of $w$.

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

subject to

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

Rewrite the optimization using the representer theorem:

- insert $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$ everywhere,
- minimize over $\alpha_i$ instead of $w$.

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

subject to

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

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

Rewrite the optimization using the representer theorem:

- insert $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$ everywhere,
- minimize over $\alpha_i$ instead of $w$.

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

subject to

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

Rewrite the optimization using the representer theorem:

- insert $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$ everywhere,
- minimize over $\alpha_i$ instead of $w$.

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

subject to

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

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

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

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

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

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

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** $K \in \mathbb{R}^{n \times n}$

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

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

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

More elegant: dualize using Lagrangian multipliers

$$\max_{\alpha_i \in \mathbb{R}^+} \qquad -\frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) + \sum_{i=1}^{n} \alpha_i$$

subject to

$$0 \leq \alpha_i \leq \frac{C}{n} \qquad \text{for } i = 1, \dots, n$$

**Support-Vector Machine (SVM)**

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

# Why use $\boldsymbol{k(x, x')}$ instead of $\langle \boldsymbol{\phi(x), \phi(x')} \rangle$?

**1) Memory usage:**

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

## Why use $\boldsymbol{k(x, x')}$ instead of $\langle \boldsymbol{\phi(x), \phi(x')} \rangle$?

**1) Memory usage:**

- Storing $\phi(x_1), \dots, \phi(x_n)$ requires $O(nm)$ memory.
- Storing $k(x_1, x_1), \dots, k(x_n, x_n)$ 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 $\phi(x_i)$ and then $\langle \phi(x_i), \phi(x_j) \rangle$.

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

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

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

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

**1) Memory usage:**

- Storing $\phi(x_1), \dots, \phi(x_n)$ requires $O(nm)$ memory.
- Storing $k(x_1, x_1), \dots, k(x_n, x_n)$ 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 $\phi(x_i)$ and then $\langle \phi(x_i), \phi(x_j) \rangle$.

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

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

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

  Equivalently, but faster, without $\phi$:

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

**3) Flexibility:**

- One can think of kernels as *measures of similarity*.
- Any similarity measure $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ can be used, as long as it is
  - **symmetric**: $k(x', x) = k(x, x')$ for all $x, x' \in \mathcal{X}$
  - **positive definite**: for any set of points $x_1, \ldots, x_n \in \mathcal{X}$

$$K_{ij} = (k(x_i, x_j))_{i,j=1,\ldots,n}$$

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

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

- Using *functional analysis* one can show that for these $k(x, x')$, a feature map $\phi : \mathcal{X} \to \mathcal{F}$ exists, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{F}}$

We can interpret the kernelized SVM as loss and regularizer:

$$\min_{\alpha_i \in \mathbb{R}, \xi_i \in \mathbb{R}^+} \underbrace{\sum_{j,k=1}^{n} \alpha_j \alpha_k k(x_j, x_k)}_{\text{regularizer}} + \frac{C}{n} \sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j k(x_j, x_i)\}}_{\text{Hinge loss}}$$

for

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

Data dependent hypothesis class

$$\mathcal{H} = \{\sum_{i=1}^{n} \alpha_i k(x_i, x) : \alpha \in \mathbb{R}^n\} \qquad \text{for training set } x_1, \ldots, x_n.$$

Nonlinear functions, spanned by basis functions centered at training points.

## Popular kernel functions in Computer Vision

- **"Linear kernel"**: identical solution as linear SVM

$$k(x, x') = x^\top x' = \sum_{i=1}^{d} x_i x'_i$$

- **"Hellinger kernel"**: less sensitive to extreme value in feature vector

$$k(x, x') = \sum_{i=1}^{d} \sqrt{x_i\, x'_i} \qquad \text{for } x = (x_1, \dots, x_d) \in \mathbb{R}^d_+$$

- **"Histogram intersection kernel"**: very robust

$$k(x, x') = \sum_{i=1}^{d} \min(x_i, x'_i) \qquad \text{for } x \in \mathbb{R}^d_+$$

- **"$\chi^2$-distance kernel"**: good empirical results

$$k(x, x') = -\chi^2(x, x') = -\sum_{i=1}^{d} \frac{(x_i - x'_i)^2}{x_i + x'_i} \text{ for } x \in \mathbb{R}^d_+$$

## Popular kernel functions in Computer Vision

- **"Gaussian kernel"**: overall most popular kernel in Machine Learning

$$k(x, x') = \exp(\,-\lambda \|x - x'\|^2\,)$$

- **"(Exponentiated) $\chi^2$-kernel"**: best results in many benchmarks

$$k(x, x') = \exp(\,-\lambda \chi^2(x, x')\,) \qquad \text{for } x \in \mathbb{R}^d_+$$

- **"Fisher kernel"**: good results and allows for efficient training

$$k(x, x') = [\nabla p(x; \Theta)]^\top F^{-1} [\nabla p(x'; \Theta)]$$

  - $p(x; \Theta)$ is generative model of the data, i.e. Gaussian Mixture Model
  - $\nabla p$ is gradient of the density function w.r.t. the parameter $\Theta$
  - $F$ is the *Fisher Information Matrix*

[Perronnin, Dance "Fisher Kernels on Visual Vocabularies for Image Categorization", 2007]

**SVMs with nonlinear 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.

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

**Observation 2:** Nonlinear 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?

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

**Observation 2:** Nonlinear 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.

[Maji, Berg, Malik, "*Classification using intersection kernel support vector machines is efficient*", CVPR 2008]
[Rahimi, "*Random Features for Large-Scale Kernel Machines*", NIPS, 2008]

**Core Facts**

- For every positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists (implicit) $\phi : \mathcal{X} \to \mathcal{F}$ such that

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

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

**Problem:** $\phi$ is generally unknown, and $\dim \mathcal{F} = \infty$ is possible

# (Approximate) Explicit Feature Maps

**Core Facts**

- For every positive definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists (implicit) $\phi : \mathcal{X} \to \mathcal{F}$ such that

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

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

**Problem:** $\phi$ is generally unknown, and $\dim \mathcal{F} = \infty$ is possible

**Idea:** Find **approximate** $\tilde{\phi} : \mathcal{X} \to \mathbb{R}^D$ such that

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

# Explicit Feature Maps

For some kernels, we can find an explicit feature map:

## Example: Hellinger kernel

$$k_H(x,x') = \sum_{i=1}^{d} \sqrt{x_i x_i'} \qquad \text{for } x \in \mathbb{R}_+^d.$$

Set $\phi_H(x) := \big(\sqrt{x_1}, \ldots, \sqrt{x_d}\big)$:

$$\langle \phi_H(x), \phi_H(x') \rangle_{\mathbb{R}^d} \quad = \quad \sum_{i=1}^{d} \sqrt{x_i}\sqrt{x_i'} \quad = \quad k_H(x,x')$$

We can train a linear SVM on $\sqrt{x}$ instead of a kernelized SVM with $k_H$.

When there is no exact feature map, we can look for approximations:

**Example: $\chi^2$-distance kernel**

$$k_{\chi^2}(x, x') = \sum_{i=1}^{d} \frac{x_i x'_i}{x_i + x'_i}$$

set $\phi(x) := \left(\sqrt{x_i}, \sqrt{2\pi x_i}\cos(\log x_i), \sqrt{2\pi x_i}\sin(\log x_i)\right)_{i=1,\dots,d}$

$$\langle \phi(x), \phi(x') \rangle_{\mathbb{R}^{3d}} \approx k_{\chi^2}(x, x')$$

**Current state-of-the-art in large-scale nonlinear learning.**

[A. Vedaldi, A. Zisserman, *"Efficient Additive Kernels via Explicit Feature Maps"*, TPAMI 2011]

# Other Supervised Learning Methods

## Multiclass SVMs

# Multiclass SVMs

What if $\mathcal{Y} = \{1, \dots, K\}$ with $K > 2$?

Some classifiers works naturally also for multi-class

- Nearest Neigbhor, Random Forests, ...

SVMs don't. We need to modify them:

- Idea 1: decompose multi-class into several binary problems
  - One-versus-Rest
  - One-versus-One
- Idea 2: generalize SVM objective to multi-class situation
  - Crammer-Singer SVM

Most common: **One-vs-Rest (OvR)** training

- For each class $y$, train a separate binary SVM, $f_y : \mathcal{X} \to \mathbb{R}$.
  - Positive examples: $X_+ = \{x_i : y_i = y\}$
  - Negative examples: $X_- = \{x_i : y_i \neq y\}$ (aka *"the rest"*)
- Final decision: $g(x) = \mathrm{argmax}_{y \in \mathcal{Y}} f_y(x)$

Advantage:

- easy to implement
- works well, if implemented correctly

Disadvantage:

- Training problems often unbalanced, $|X_-| \gg |X_+|$
- ranges of the $f_y$ are no calibrated to each other.

# Reductions: Multiclass SVM to Binary SVMs

Also popular: **One-vs-One (OvO)** training

- For each pair of classes $y \neq y'$, train a separate binary SVM, $f_{yy'} : \mathcal{X} \to \mathbb{R}$.
  - Positive examples: $X_+ = \{x_i : y_i = y\}$
  - Negative examples: $X_- = \{x_i : y_i = y'\}$ (aka *"the rest"*)
- Final decision: majority vote amongst all classifiers

Advantage:

- easy to implement
- training problems approximately balanced

Disadvantage:

- number of SVMs to train grows *quadratically* in $|\mathcal{Y}|$
- less intuitive decision rule

## Crammer-Singer SVM

Standard setup:

- $f_y(x) = \langle w, x \rangle$ (also works kernelized)
- decision rule: $g(x) = \operatorname{argmax}_{y \in \mathcal{Y}} f_y(x)$
- $0/1$-loss: $\Delta(y, \bar{y}) = [\![ y \neq \bar{y} ]\!]$

What's a good multiclass *loss function*?

$$
\begin{aligned}
g(x^i) = y^i \quad &\Leftrightarrow \quad y^i = \operatorname*{argmax}_{y \in \mathcal{Y}} f_y(x^i) \\
&\Leftrightarrow \quad f_{y^i}(x^i) > \max_{y \neq y^i} f_y(x^i) \\
&\Leftrightarrow \quad \underbrace{f_{y^i}(x^i) - \max_{y \neq y^i} f_y(x^i)}_{\text{takes role of } y\langle w, x \rangle} > 0
\end{aligned}
$$

$$
\ell(\, y^i, f_1(x^i), \dots, f_K(x^i)\,) = \max\{0, 1 - \big(f_{y^i}(x^i) - \max_{y \neq y^i} f_y(x^i)\big)\}
$$

Regularizer: $$\Omega(f_1,\dots,f_K) = \sum_{k=1}^{K} \|w_k\|^2$$

Together:

$$\min_{w_1,\dots,w_K\in\mathbb{R}^d} \sum_{k=1}^{K} \|w_k\|^2 + \frac{C}{n}\sum_{i=1}^{n} \max\{0, 1 - \big(f_{y^i}(x^i) - \max_{y\neq y^i} f_y(x^i)\big)\}$$

Equivalently:

$$\min_{\substack{w_1,\dots,w_K\in\mathbb{R}^d\\ \xi_1,\dots,xi_n\in\mathbb{R}^+}} \sum_{k=1}^{K} \|w_k\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$$

subject to, for $i = 1,\dots,n$, $\quad f_{y^i}(x^i) - \max_{y\neq y^i} f_y(x^i) \geq 1 - \xi_i$.

Interpretation:

- One-versus-Rest: correct class has margin at least $1$ to origin.
- Cramer-Singer: correct class has margin at least $1$ to all other classes

# Summary – Nonlinear Classification

- Many technique based on *stacking*:
  - boosting, random forests, deep learning, ...
  - powerful, but sometimes hard to train (non-convex $\rightarrow$ local optima)

- Generalized linear classification with SVMs
  - conceptually simple, but powerful by using kernels
  - convex optimization, solvable to global optimality

# Summary – Nonlinear Classification

- Many technique based on *stacking*:
  - boosting, random forests, deep learning, ...
  - powerful, but sometimes hard to train (non-convex $\rightarrow$ local optima)
- Generalized linear classification with SVMs
  - conceptually simple, but powerful by using kernels
  - convex optimization, solvable to global optimality
- Kernelization is implicit application of a feature map
  - the method can become nonlinear in the original data
  - the method is still linear in parameter space

# Summary – Nonlinear Classification

- Many technique based on *stacking*:
  - boosting, random forests, deep learning, ...
  - powerful, but sometimes hard to train (non-convex $\rightarrow$ local optima)

- Generalized linear classification with SVMs
  - conceptually simple, but powerful by using kernels
  - convex optimization, solvable to global optimality

- Kernelization is implicit application of a feature map
  - the method can become nonlinear in the original data
  - the method is still linear in parameter space

- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space

# Summary – Nonlinear Classification

- Many technique based on *stacking*:
  - boosting, random forests, deep learning, ...
  - powerful, but sometimes hard to train (non-convex $\rightarrow$ local optima)
- Generalized linear classification with SVMs
  - conceptually simple, but powerful by using kernels
  - convex optimization, solvable to global optimality
- Kernelization is implicit application of a feature map
  - the method can become nonlinear in the original data
  - the method is still linear in parameter space
- Kernels are at the same time
  - similarity measures between arbitrary objects
  - inner products in a (hidden) feature space
- For large datasets, kernelized SVMs are inefficient
  - construct explicit feature map (approximate if necessary)

# 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 EUR/60$)
- Shawe-Taylor, Cristianini: *"Kernel Methods for Pattern Analysis"*, Cambridge University Press (60 EUR/75$)