Statistical Machine Learning

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2015/2016 // Lecture 6

Classifier Training and Evaluation

input data $\mathcal{D}$
input learning method A
  split $\mathcal{D} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place // do not look at it
  $g \leftarrow A[\mathcal{D}_{trn}]$ // learn a predictor from $\mathcal{D}_{trn}$
  apply g to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

In practice we often want more: not just train a classifier and evaluate it, but

- select the best algorithm out of multiple ones,
- select the best (hyper)parameters for a training algorithm.

We simulate the classifier evaluation step during the training procedure. This needs (at least) one additional data split:

Training and Selecting between Multiple Models

input data $\mathcal{D}$
input set of method $\mathcal{A} = \{A_1, \ldots, A_K\}$
  split $\mathcal{D} = \mathcal{D}_{trnval} \dot{\cup} \mathcal{D}_{tst}$ disjointly
  set aside $\mathcal{D}_{tst}$ to a safe place (do not look at it)

  split $\mathcal{D}_{trnval} = \mathcal{D}_{trn} \dot{\cup} \mathcal{D}_{val}$ disjointly
  for all models $A_i \in \mathcal{A}$ **do**
    $g_i \leftarrow A_i[\mathcal{D}_{trn}]$
    apply g_i to $\mathcal{D}_{val}$ and measure performance $E_{val}(A_i)$
  end for
  pick best performing A_i

  (optional) $g_i \leftarrow A_i[\mathcal{D}_{trnval}]$ // retrain on larger dataset
  apply g_i to $\mathcal{D}_{tst}$ and measure performance R_{tst}
output performance estimate R_{tst}

How to split? For example $1/3 : 1/3 : 1/3$ or 70% : 10% : 20%.

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Discussion.

- Each algorithm is trained on $\mathcal{D}_{trn}$ and evaluated on disjoint $\mathcal{D}_{val}$ ✓
- You select a predictor based on E_{val} (its performance on $\mathcal{D}_{val}$), only afterwards $\mathcal{D}_{tst}$ is used. ✓
- $\mathcal{D}_{tst}$ is used to evaluate the final predictor and nothing else. ✓

Problems.

- small $\mathcal{D}_{val}$ is bad: E_{val} could be bad estimate of g_A's true performance, and we might pick a suboptimal method.
- large $\mathcal{D}_{val}$ is bad: $\mathcal{D}_{trn}$ is much smaller than $\mathcal{D}_{trnval}$, so the classifier learned on $\mathcal{D}_{trn}$ might be much worse than necessary.
- retraining the best model on $\mathcal{D}_{trnval}$ might overcome that, but it comes at a risk: just because a model worked well when trained on $\mathcal{D}_{trn}$, this does not mean it'll also work well when trained on $\mathcal{D}_{trnval}$.

Leave-one-out Evaluation (for a single model/algorithm)

input algorithm A
input loss function ℓ
input data $\mathcal{D}$ (trnval part only: test part set aside earlier)
  for all $(x^i, y^i) \in \mathcal{D}$ **do**
    $g^{\neg i} \leftarrow A[\, \mathcal{D} \setminus \{(x^i, y^i)\}\,]$ // $\mathcal{D}_{trn}$ is $\mathcal{D}$ with i-th example removed
    $r^i \leftarrow \ell(y^i, g^{\neg i}(x^i))$ // $\mathcal{D}_{val} = \{(x^i, y^i)\}$, disjoint to $\mathcal{D}_{trn}$
  end for
output $R_{loo} = \frac{1}{n} \sum_{i=1}^{n} r^i$ (average leave-one-out risk)

Properties.

- Each r^i is a unbiased (but noisy) estimate of the risk $\mathcal{R}(g^{\neg i})$
- $\mathcal{D} \setminus \{(x^i, y^i)\}$ is almost the same as $\mathcal{D}$, so we can hope that each $g^{\neg i}$ is almost the same as $g = A[\mathcal{D}]$.
- Therefore, R_{loo} can be expected a good estimate of $\mathcal{R}(g)$

Problem: slow, trains n times on $n-1$ examples instead of once on n

Compromise: use fixed number of small $\mathcal{D}_{val}$

K-fold Cross Validation (CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  split $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$ into K equal sized disjoint parts
  for $k = 1, \ldots, K$ **do**
    $g^{\neg k} \leftarrow A[\,\mathcal{D} \setminus \mathcal{D}_k\,]$
    $r^k \leftarrow \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} \ell(y^i, g^{\neg k}(x))$
  end for
output $R_{K\text{-CV}} = \frac{1}{K} \sum_{k=1}^{n} r^k$ (K-fold cross-validation risk)

Observation.

- for $K = |\mathcal{D}|$ same as leave-one-out error.
- approximately k times increase in runtime.
- most common: $k = 10$ or $k = 5$.

Problem: training sets overlap, so the error estimates are correlated.
Exception: $K = 2$

5×2 Cross Validation (5×2-CV)

input algorithm A, loss function ℓ, data $\mathcal{D}$ (trnval part)
  for $k = 1, \ldots, 5$ **do**
    Split $\mathcal{D} = \mathcal{D}_1 \dot{\cup} \mathcal{D}_2$
    $g_1 \leftarrow A[\mathcal{D}_1]$,
    $r_1^k \leftarrow$ evaluate g_1 on $\mathcal{D}_2$
    $g_2 \leftarrow A[\mathcal{D}_2]$,
    $r_2^k \leftarrow$ evaluate g_2 on $\mathcal{D}_1$
    $r^k \leftarrow \frac{1}{2}(r_k^1 + r_k^2)$
  end for
output $E_{5\times 2} = \frac{1}{5} \sum_{k=1}^{5} r^k$

Observation.

- 5×2-CV is really the average of 5 runs of 2-fold CV
- very easy to implement: shuffle the data and split into halfs
- within each run the training sets are disjoint and the classifiers g_1 and g_2 are independent

Problem: training sets are smaller than in 5- or 10-fold CV.

Unbalanced Classes

If classes are *unbalanced* accuracy might not tell us much:

- $p(y = -1) = 0.99$, $p(y = +1) = 0.01$ $\rightarrow$ "always no" is 99% correct
- there might not be a better non-constant classifier

Two solutions:

- balancing
  - use only subset of the majority class to balance data (5:1, or 1:1)
- reweighting
  - multiple loss in optimization with class-dependent constant C_{y_i},

$$\frac{1}{|\mathcal{D}_+|} \sum_{(x_i,y_i)\in\mathcal{D}_+}^{n} \ell(\, y_i, f(x_i)\,) + \frac{1}{|\mathcal{D}_-|} \sum_{(x_i,y_i)\in\mathcal{D}_-}^{n} \ell(\, y_i, f(x_i)\,) + \Omega(f)$$

- treat as a retrieval problem

Classifiers for Information Retrieval Tasks

Some classification tasks are really rather *retrieval* tasks, e.g.

- database lookup: is an entry x relevant $(y = 1)$ or not $(y = -1)$?

A typical property:

- prediction is performed on a fixed database
- we have access to all elements of the test set at the same time
- positives $(y = 1)$ are important, negative $(y = -1)$ are a nuisanse
- we don't need all decisions, a few correct positives is enough

For a classifier $g(x) = \operatorname{sign} f(x)$ with $f(x) : \mathcal{X} \to \mathbb{R}$ (e.g., $f(x) = \langle w, x \rangle$), we interpret $f(x)$ as its *confidence*.

To produce K positive we return the test samples of highest confidence.

Equivalently, we decide by $g_\theta(x) = \operatorname{sign}(\, f(x) - \theta \,)$, for the right θ.

Other Ways to Evaluate Classifiers

Retrieval quality is often measure in terms of *precision* and *recall*:

Definition (Precision, Recall, F-Score)

For $\mathcal{Y} = \{\pm 1\}$, let $g : \mathcal{X} \to \mathcal{Y}$ a decision function and $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$ be a *database*.

Then we define

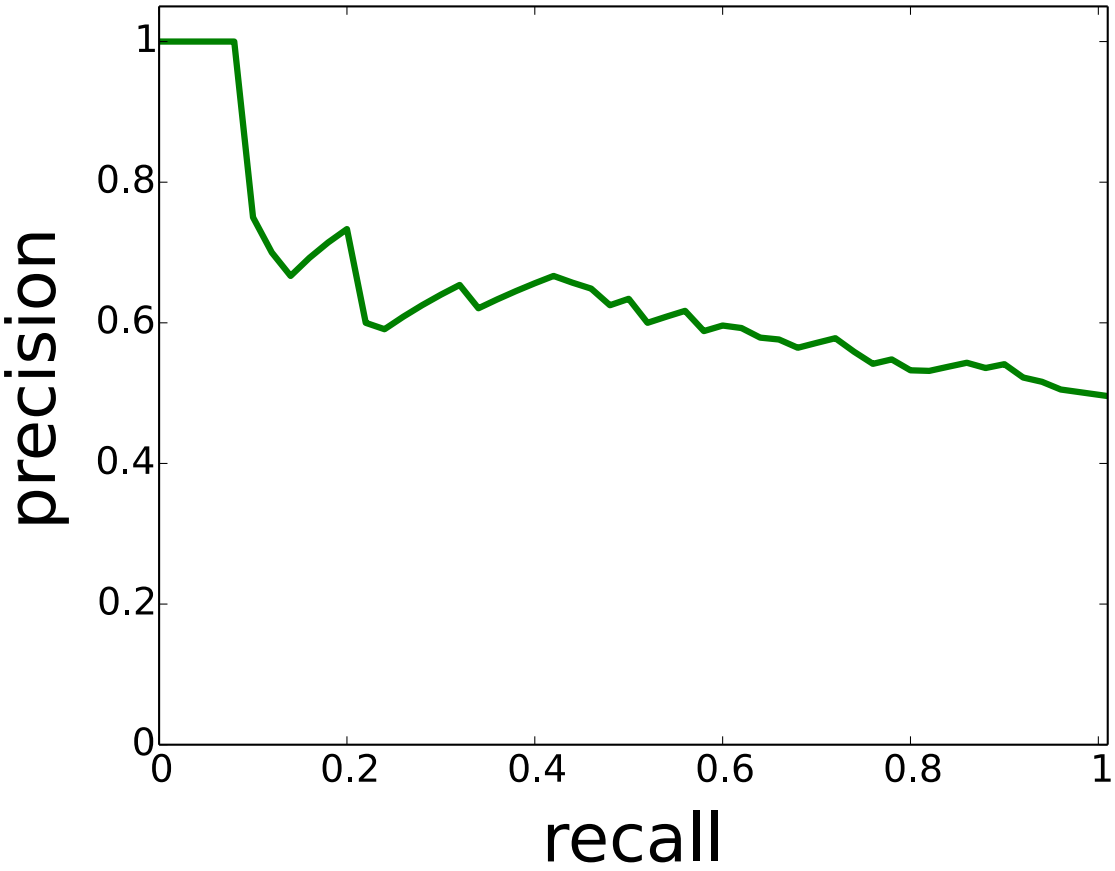
$$\textit{precision}(g) = \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } g(x^j) = 1}$$

$$\textit{recall}(g) = \frac{\textit{number of test samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of test samples with } y^j = 1}$$

$$F\textit{-score}(g) = 2\frac{\textit{precision}(g) \cdot \textit{recall}(g)}{\textit{precision}(g) + \textit{recall}(g)}$$

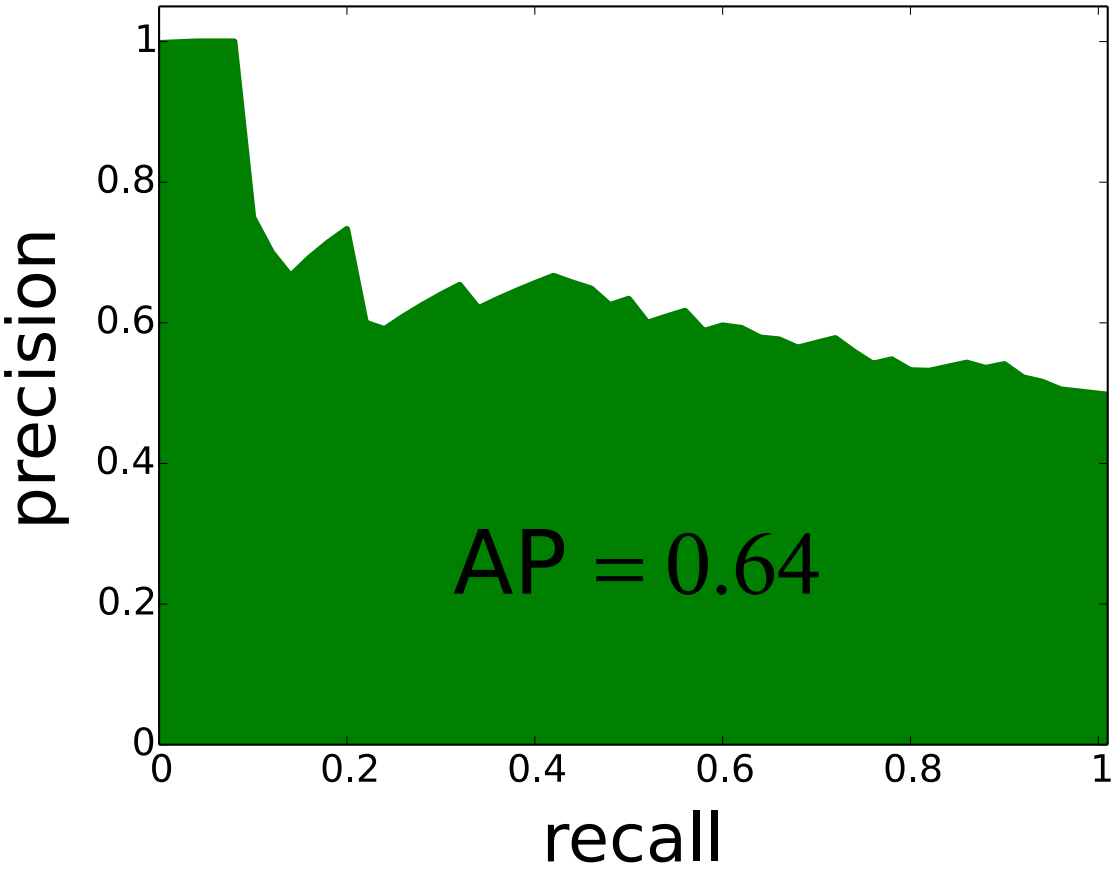
For different thresholds, θ, we obtain different precision and recall values.

They are summarized by a **precision-recall curve**:



For different thresholds, θ, we obtain different precision and recall values.

They are summarized by a **precision-recall curve**:



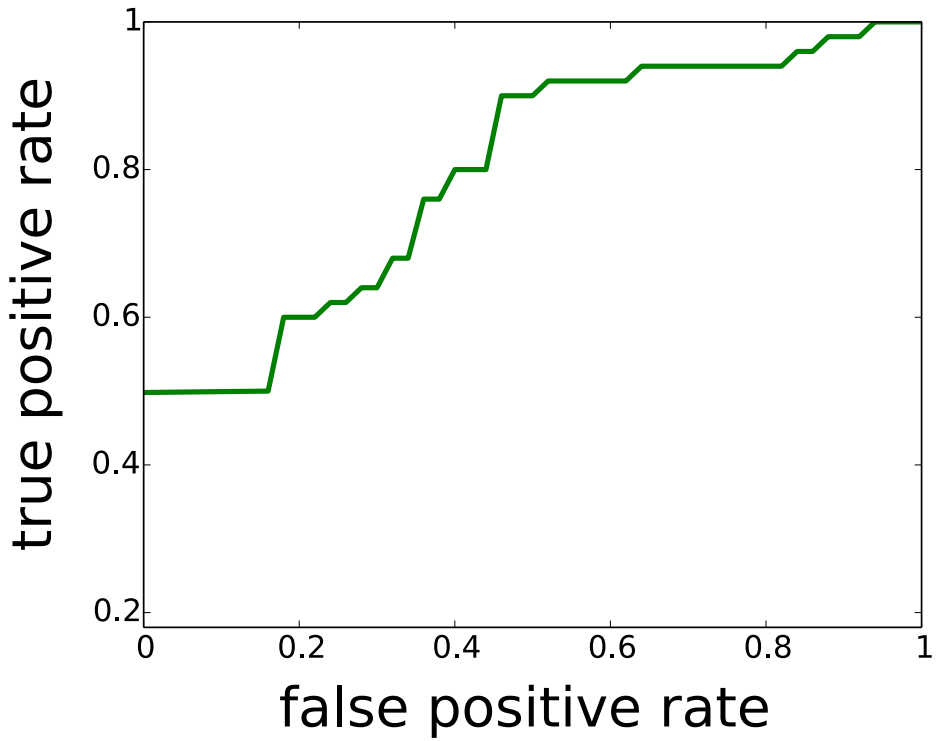
- If pressured, summarize into one number: **average precision**.
- Curve/value depends on class ratio: higher values for more positives

A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

$$\textit{true-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

$$\textit{false-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$

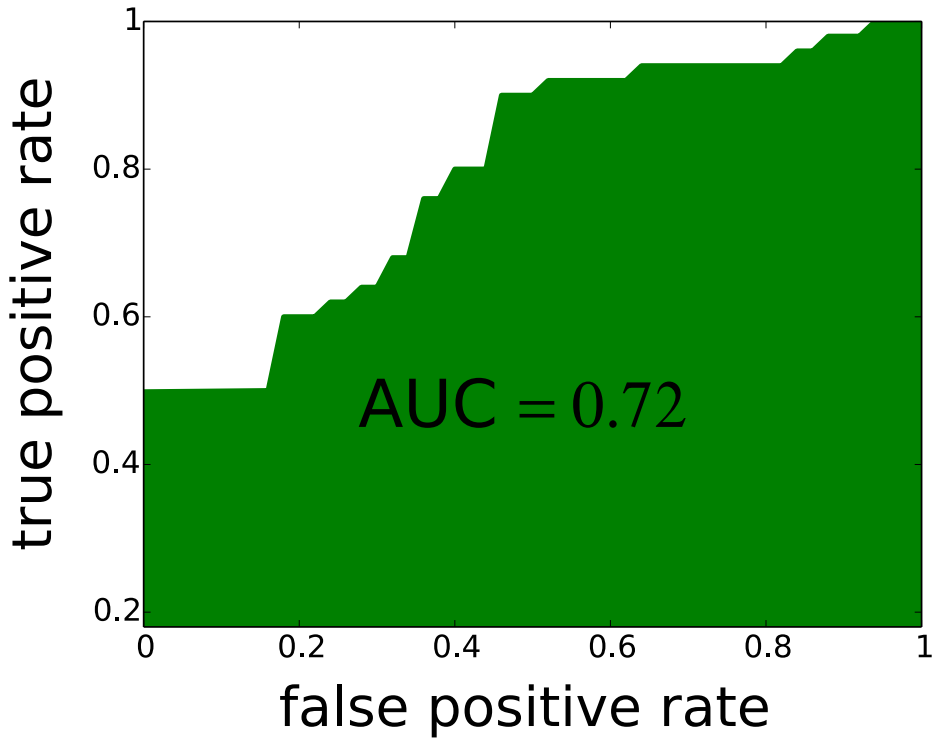


A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

$$\textit{true-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

$$\textit{false-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$



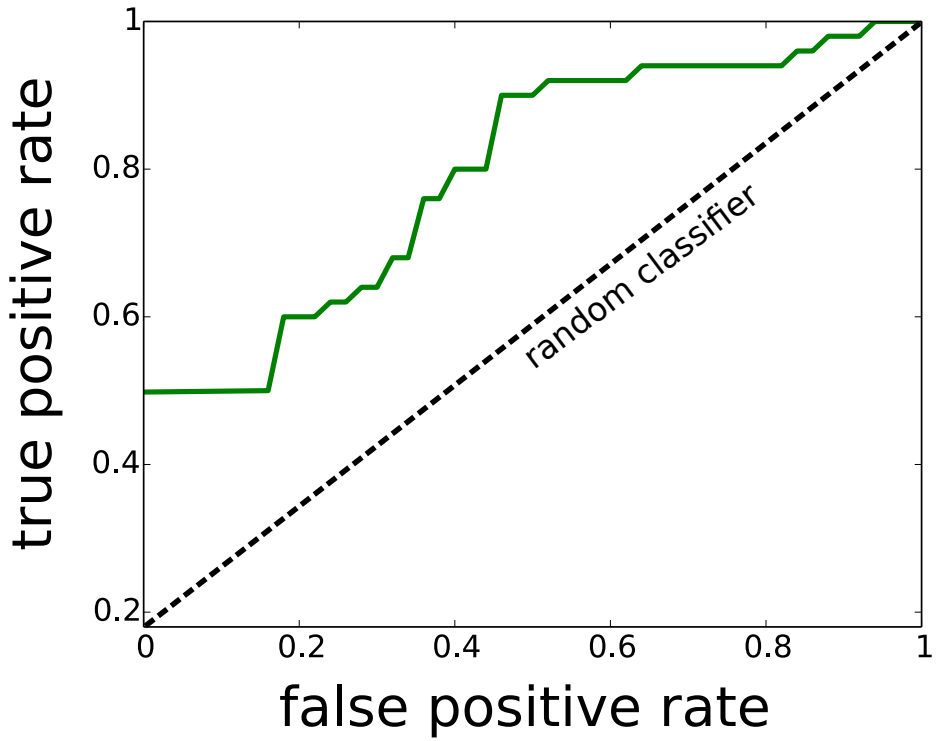
Summarize into: **area under ROC curve (AUC)**.

A similar role in different context:

Receiver Operating Characteristic (ROC) Curve

$$\textit{true-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = 1}{\textit{number of samples with } y^j = 1}$$

$$\textit{false-positive-rate}(g) = \frac{\textit{number of samples with } g(x^j) = 1 \textit{ and } y^j = -1}{\textit{number of samples with } y^j = -1}$$



Random classifier: $AUC = 0.5$, regardless of class proportions.

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

Are the differences just due to chance?

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

"HCRF" method is better	1
"Our" method is better	4
both methods are equal	0

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8
- Idea 2: sign test (like binomial before): `binom_test(1,5)=0.375`

Significance Using Multiple Datasets

Standard procedure in Machine Learning research:

- develop a new method
- compare it to results of previous methods on standard benchmarks

	HCRF	Our
SIFT-flow	31.22%	**27.73%**
MSRC-21	78.89%	**81.11%**
VOC 2008	20.13 %	**30.12%**
VOC 2009	42.43 %	**43.37%**
VOC 2010	30.13 %	**32.14%**

"HCRF" method is better	1
"Our" method is better	4
both methods are equal	0

Are the differences just due to chance?

- Idea 1: mean/std.err.: 40.6 ± 20.4 vs. 42.9 ± 19.8
- Idea 2: sign test (like binomial before): `binom_test(1,5)=0.375`
- Idea 3: take differences into account, not just the sign
  - H_0: differences have a symmetric distribution around zero

Wilcoxon signed rank test

Given: real values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$

- drop all cases with $a_i = b_i$, call the remaining number of pairs k
- compute $\delta_i = |a_i - b_i|$ and $s_i = \mathrm{sign}(a_i - b_i)$
- sort elements from smallest to larges δ_i
- compute rank, R_i, of each δ_i, ties get average of covered ranks
- compute statistics (sum of signed ranks)

$$W = \sum_{i=1}^{k} s_i R_i$$

- compare value to table, $W_{\text{critical},k}$ (large k: Gaussian approximation)

"HCRF" vs. "Our" example (5 datasets):

- $A = [31.22, 78.89, 20.13, 42.43, 30.13]$
  $B = [27.73, 81.11, 30.12, 43.37, 32.14]$
- `scipy.stats.wilcoxon(` A`,` B `) = 0.35`

Wilcoxon signed rank test

Given: real values $a_1, \ldots, a_m$ and $b_1, \ldots, b_m$

Class ID	Name	Number of samples (training)	Classification rate (%)	
			Proposed method	Proposed method with CA
1	10K (FT)	900 (100)	99.00	99.50
2	10K (FB)	900 (100)	98.25	100.00
3	10K (BT)	900 (100)	98.25	99.88
4	10K (BB)	900 (100)	97.88	99.75
5	5K (FT)	900 (100)	94.00	100.00
6	5K (FB)	900 (100)	98.38	99.75
7	5K (BT)	900 (100)	94.25	97.00
8	5K (BB)	900 (100)	97.00	98.25
9	1K (FT)	900 (100)	95.13	99.63
10	1K (FB)	900 (100)	93.75	99.75
11	1K (BT)	900 (100)	94.13	99.00
12	1K (BB)	900 (100)	95.75	98.75

- mean/std.err.: 96.5 ± 1.84, mean 2: 99.27 ± 0.86
- sign test: $+ : 12 \quad =: 0 \quad - : 0$, `binom_test(0,12)=0.0005`
- signed rank test: `wilcoxon(A,B)=0.0022`

Beyond Binary Classification

Multiclass Classification – One-versus-rest reduction

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 reduction

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 construction:

- train one binary classifier $g_c : \mathcal{X} \to \mathbb{R}$ for each class c:
  - all samples with class label c are positive examples
  - all other samples are negative examples
- classify by finding maximal response

$$f(x) = \operatorname*{\mathbf{argmax}}_{c=1,\ldots,M} g_c(x)$$

Advantage: easy to implement, parallel, works well in practice

Disadvantage: with many classes, training sets become unbalanced.
no explicit *calibration* of scores between different g_c

Multiclass Classification – All-versus-all reduction

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 reduction

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 construction:

- train one classifier, $g_{ij} : \mathcal{X} \to \mathbb{R}$, for each pair of classes $1 \leq i < j \leq M$, in total $\frac{m(m-1)}{2}$ prediction functions
- classify by voting

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

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

Advantage: small and balanced training problems, parallel, works well.

Disadvantage: number of classifiers grows quadratically in classes.

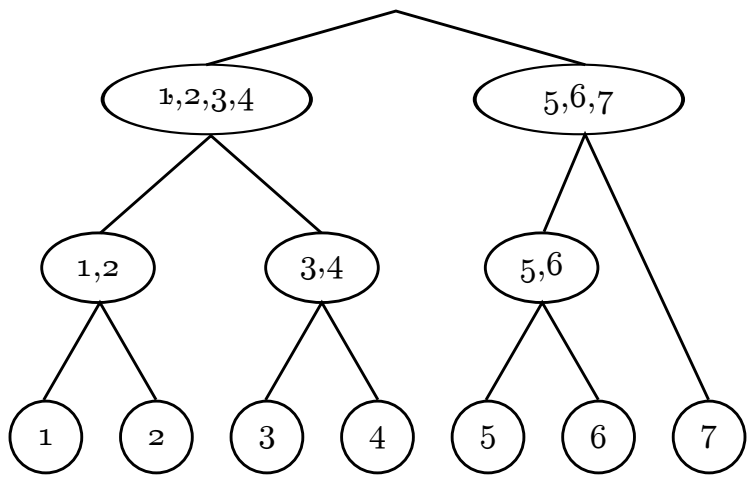
Multiclass Classification – Hierarchical

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 $f : \mathcal{X} \to \{1, \ldots, M\}$.

Hierarchical (tree) construction:

- construct binary tree with classes at leafs
- learn one classifier for each decision



Advantage: at most $\lceil \log_2 M \rceil$ classifier evaluation at test time

Disadvantage: not parallel, not robust to mistakes at any stage

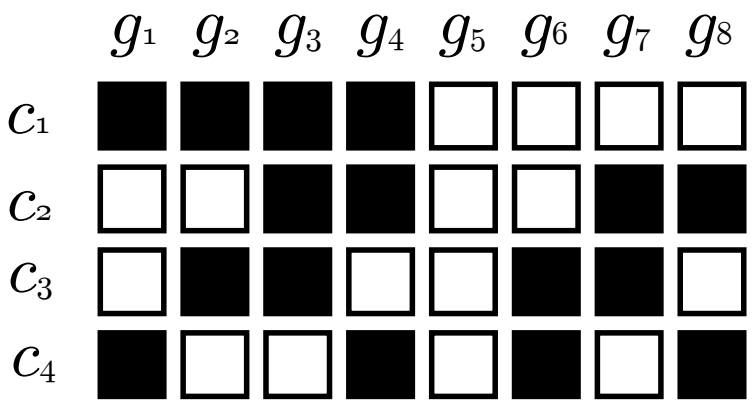
Multiclass Classification – Error Correcting Output Codes

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 $f : \mathcal{X} \to \{1, \ldots, M\}$.

Define a binary codeword for each class

- one classifier for codeword entry
- classify by comparing predictions to code words (exact or in some norm)



Advantage: parallel, trade off between speed and robustness

Disadvantage: optimal code design NP-hard

A Multiclass Support Vector Machines

- $\mathcal{X}$ anything, $\mathcal{Y} = \{1, 2, \ldots, M\}$,
- feature map $\phi : \mathcal{X} \to \mathcal{H}$ (explicit or implicit via kernel)
- training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- goal: learn functions $g_i(x) = \langle w_i, \phi(x) \rangle$ for $i = 1, \ldots, M$.

Enforce a margin between the correct and highest-ranked incorrect label:

$$\mathbf{min}_{w,\xi} \frac{1}{2} \sum_{k=1}^{M} \|w_k\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi^i$$

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

$$\langle w_{y^i}, \phi(x^i) \rangle \geq 1 + \langle w_k, \phi(x^i) \rangle - \xi^i, \quad \text{for all } k \neq y_i.$$

Prediction: $f(x) = \underset{k=1,\ldots,M}{\mathbf{argmax}} \ \langle w_k, \phi(x) \rangle$

Crammer-Singer Multiclass SVM

A Multiclass Support Vector Machines

Many different option for multi-class SVMs:

- One-versus-Rest
- One-versus-One
- ECOC
- Crammer-Singer
- ...

Which one is the best?

A Multiclass Support Vector Machines

Many different option for multi-class SVMs:

- One-versus-Rest
- One-versus-One
- ECOC
- Crammer-Singer
- ...

Which one is the best?

None (or all of them)!

- there's dozens of studies, they all disagree
- use whatever is available
- to implement yourself, One-versus-Rest is most popular, since it's simplest

More *Kernel*-Methods

Kernels are useful for many other things besides SVMs:

> 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

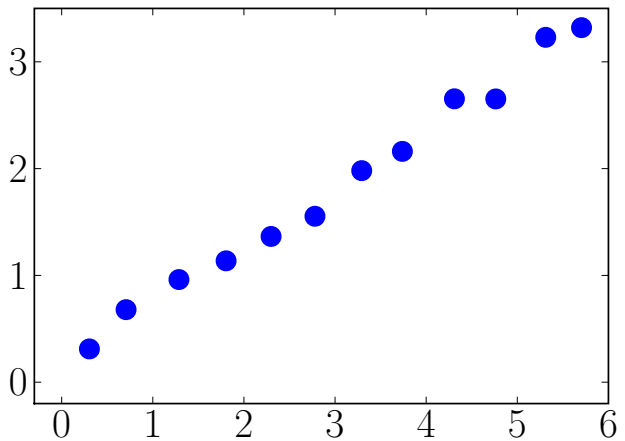
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



Linear Regression

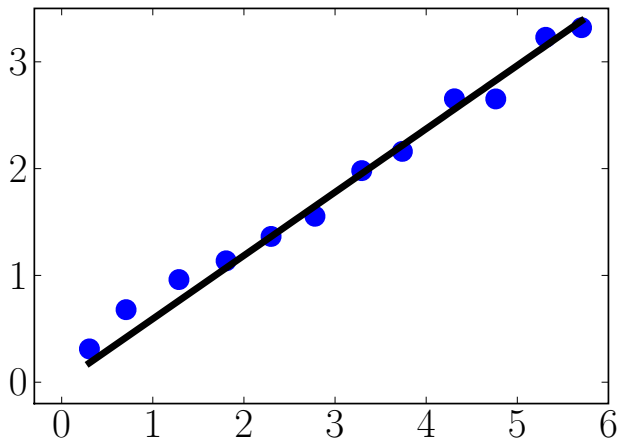
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



Very popular, because it has a closed form solution!

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

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

Linear Regression

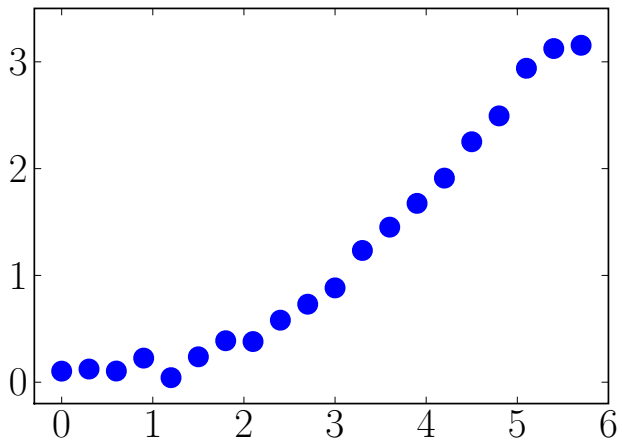
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



What about non-linear?

Very popular, because it has a closed form solution!

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

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

Linear Regression

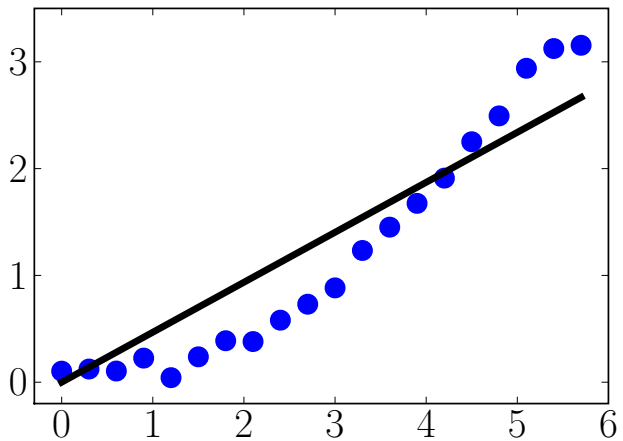
Given samples $x_i \in \mathbb{R}^d$ and function values $y_i \in \mathbb{R}$. Find a linear function $f(x) = \langle w, x \rangle$ that approximates the values.

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



What about non-linear?

Very popular, because it has a closed form solution!

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

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

Nonlinear Regression

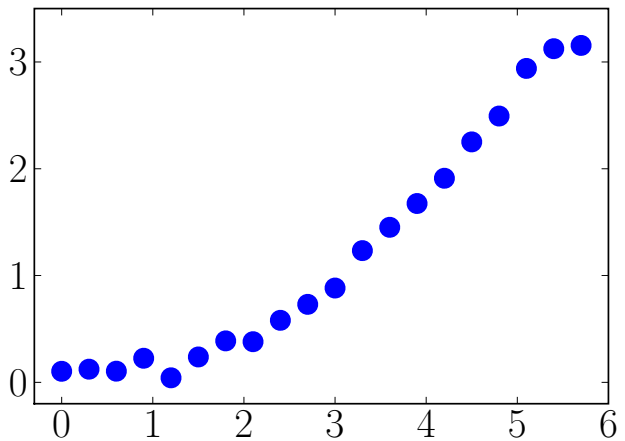
Given $x_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, $i = 1, \ldots, n$, and $\phi : \mathcal{X} \to \mathcal{H}$. Find an approximating function $f(x) = \langle w, \phi(x) \rangle_{\mathcal{H}}$ (non-linear in x, linear in w).

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



Nonlinear Regression

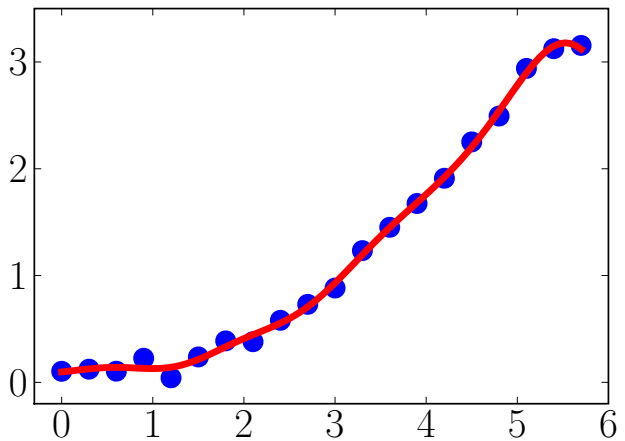
Given $x_i \in \mathcal{X}$, $y_i \in \mathbb{R}$, $i = 1, \ldots, n$, and $\phi : \mathcal{X} \to \mathcal{H}$. Find an approximating function $f(x) = \langle w, \phi(x) \rangle_{\mathcal{H}}$ (non-linear in x, linear in w).

Interpolation error:

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

Solve for $\lambda \geq 0$:

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



Closed form solution is still valid:

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

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

Example: Kernel Ridge Regression

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

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

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

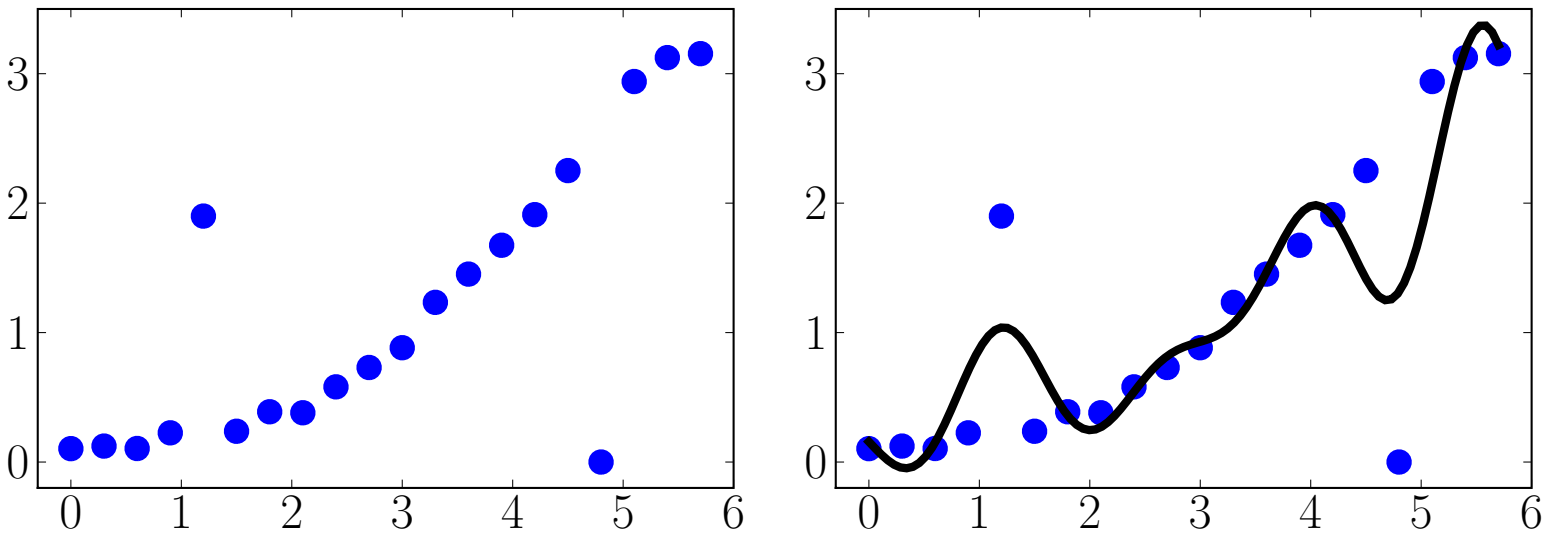
but we can still calculate $f : \mathcal{X} \to \mathbb{R}$:

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

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

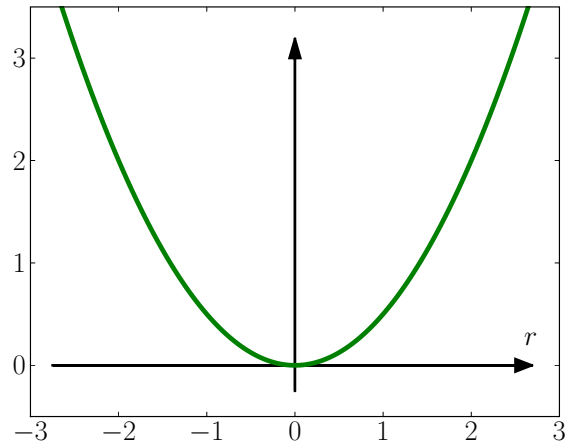
Kernel Ridge Regression

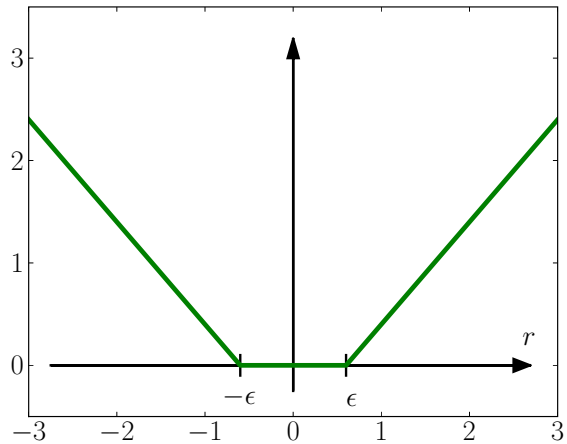
Like Least-Squared Regression, (Kernel) Ridge Regression is sensitive to outliers:

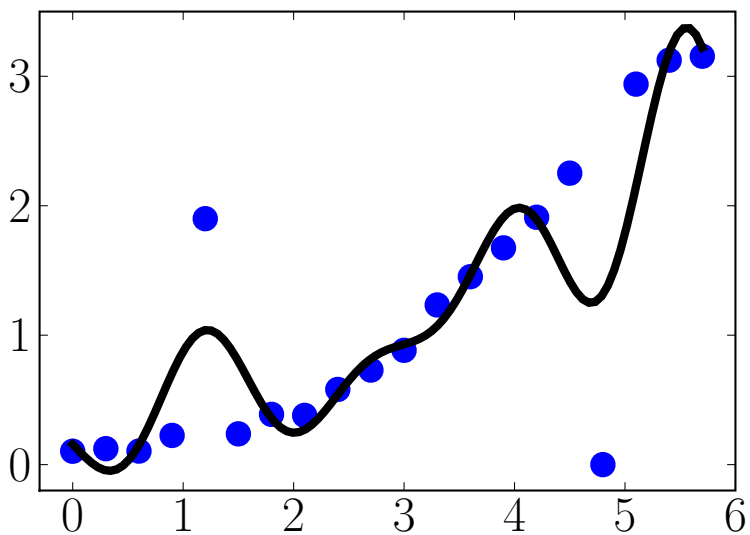


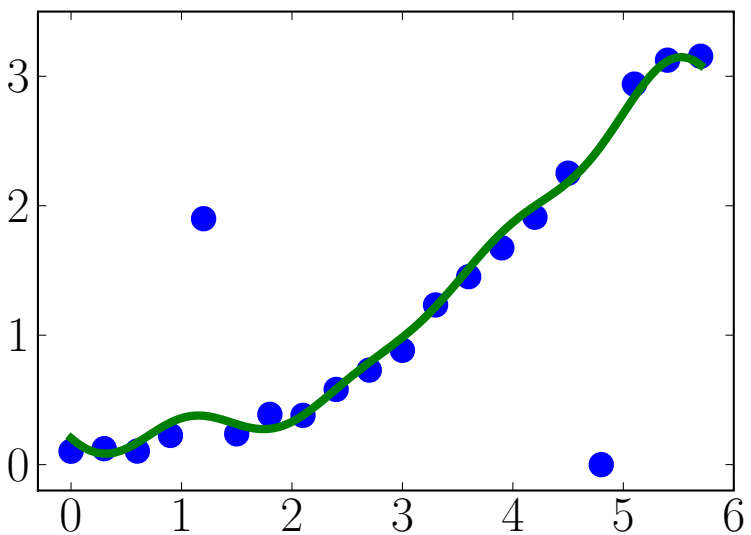
because the quadratic loss function penalized large residue.

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









Support Vector Regression

Optimization problem similar to support vector machine:

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

subject to

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

Dualization (or the "Representer Theorem") tell us that $w = \sum_j \alpha_j \phi(x_j)$.

Support Vector Regression

Optimization problem similar to support vector machine:

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

subject to

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

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

Support Vector Regression

Optimization problem similar to support vector machine:

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

subject to

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

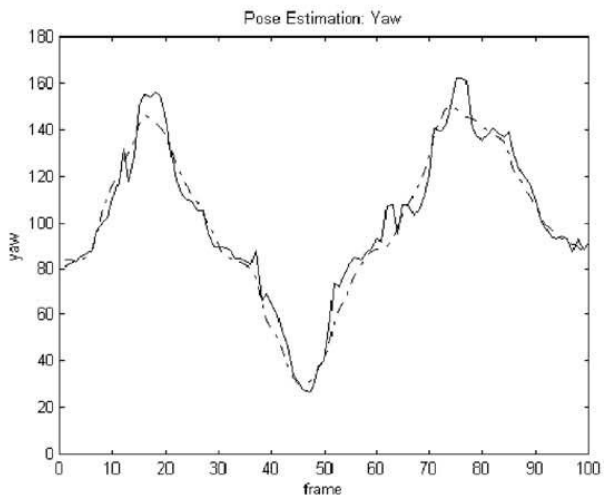
Regression function

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

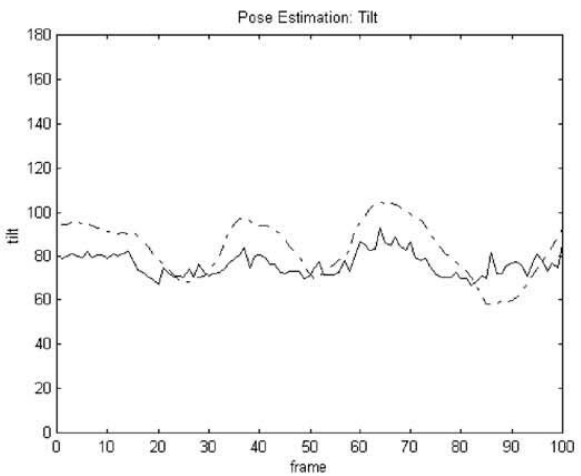
Example – Head Pose Estimation

- 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



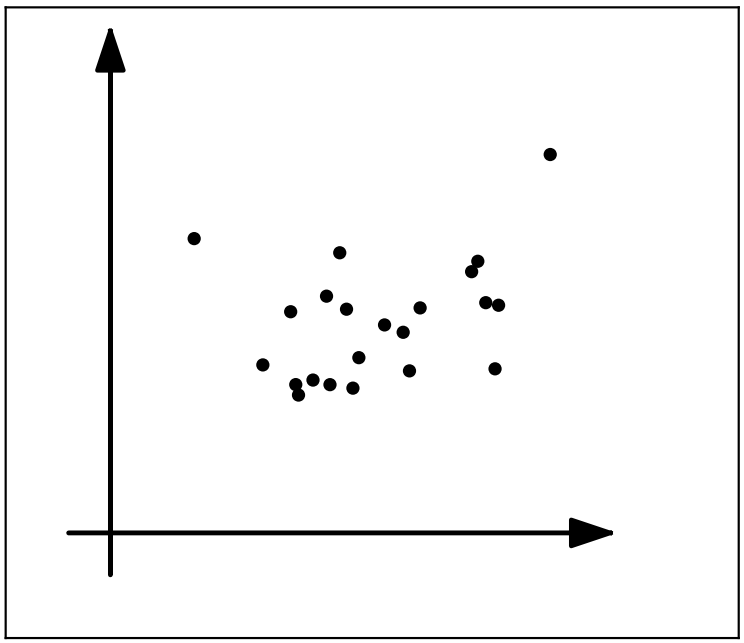
(c) Tilt estimation

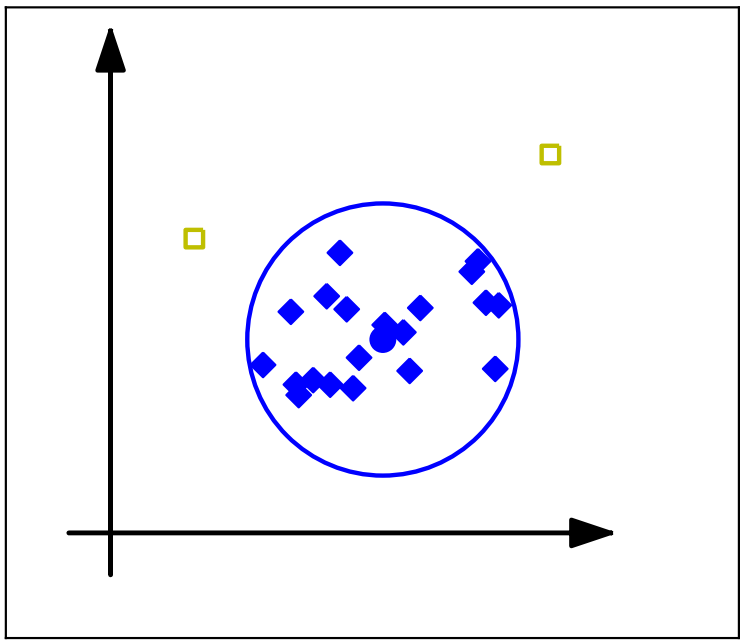
[Li, Gong, Sherra, Liddell, "Support vector machine based multi-view face detection and recognition", IVC 2004]

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

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

  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 $\phi : \mathcal{X} \to \mathcal{H}$.
Do outlier detection for $\phi(x_1), \ldots, \phi(x_n)$:

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

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

subject to

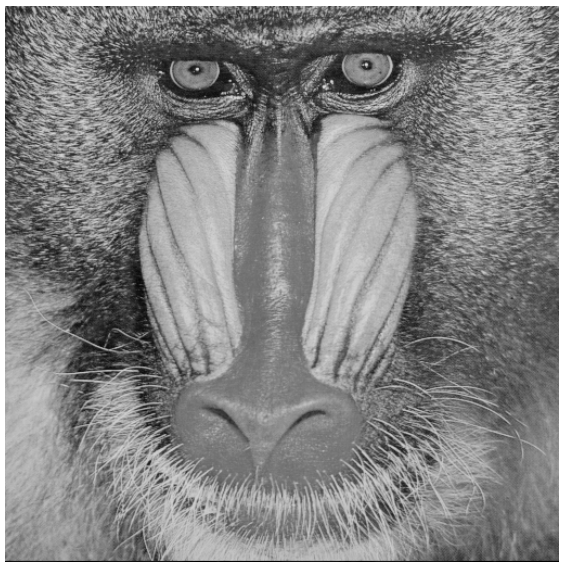
$$\|\phi(x_i) - c\|^2 \leq R^2 + \xi_i \quad \text{for } i = 1, \ldots, n.$$

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

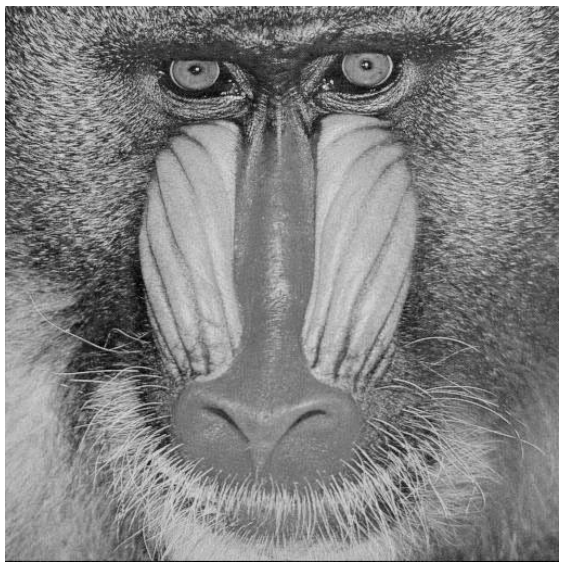
Support Vector Data Description

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]