Support Vector Machines

VL Algorithmisches Lernen, Teil 3b

Norman Hendrich & Jianwei Zhang

University of Hamburg
MIN Faculty, Dept. of Informatics
Vogt-Kölln-Str. 30, D-22527 Hamburg
hendrich@informatik.uni-hamburg.de

19/05/2010

Outline

Support Vector Machines

- a.k.a. maximum margin classifiers

- a family of related
- supervised
- learning methods
- for classification and regression

- try to minimize the classification error
- while maximizing the geometric margin

Support Vector Machine

- based on the linear classifier

Four new main concepts:

- Maximum margin classification
- Soft-margin classification for noisy data
- Introduce non-linearity via feature maps
- Kernel trick: implicit calculation of feature maps

- use Quadratic Programming for training
- polynomial or gaussian kernels often work well

Overall concept and architecture

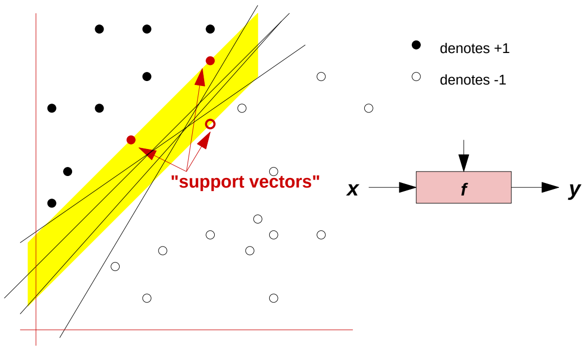
- select a feature space $\mathcal{H}$ and a mapping function $\Phi : x \mapsto \Phi(x)$
- select a classification (output) function σ

$$y(x) = \sigma(\sum_i \vartheta_i \langle \Phi(x), \Phi(x_i) \rangle)$$

- during training, find the *support-vectors* $x_1 \ldots x_n$
- and weights ϑ which minimize the classification error

- map test input x to $\Phi(x)$
- calculate dot-products $\langle \Phi(x) \Phi(x_i) \rangle$
- feed linear combination of the dot-products into σ
- get the classification result

Maximum margin and support vectors



- the (linear) classifier with the largest margin
- data points that limit the margin are called the *support vectors*

Soft-margin classification

- allow some patterns to violate the margin constraints
- find a compromise between large margins
- and the number of violations

Idea:

- introduce slack-variables $\xi = (\xi_i \dots \xi_n)$, $\xi_i \geq 0$
- which measure the margin violation (or classification error) on pattern x_i: $y(x_i)(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i$
- introduce one global parameter C which controls the compromise between large margins and the number of violations

Soft-margin classification

- introduce slack-variables ξ_i
- and global control parameter C

$$\max_{w,b,\xi} \mathcal{P}(w,b,\xi) = \frac{1}{2}w^2 + C\sum_{i=1}^{n} \xi_i$$

subject to:

$$\forall i : \quad y(x_i)(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i$$
$$\forall i : \quad \xi_i \geq 0$$

- problem is now very similar to the hard-margin case
- again, the dual representation is often easier to solve

Nonlinearity through feature maps

General idea:

- introduce a function Φ which maps the input data into a higher dimensional *feature space*

$$\Phi : x \in X \mapsto \Phi(x) \in \mathcal{H}$$

- similar to hidden layers of multi-layer ANNs
- explicit mappings can be expensive in terms of CPU and/or memory (especially in high dimensions)
- "*Kernel functions*" achieve this mapping *implicitly*
- often, very good performance

Common SVM feature maps

kernels

- $z_k = ($ polynomial terms of x_k of degree 1 to $q)$
- $z_k = ($ radial basis functions of $x_k)$
- $z_k = ($ sigmoid functions of $x_k)$
- ...
- combinations of the above, e.g.
- $K(x,z) = K_1(x,z) + K_2(x,z)$;
- $K(x,z) = K_1(x,z) \cdot K_2(x,z)$;

Note:

- feature map Φ only used in inner products
- for training, information on pairwise inner products is sufficient

Quadratic polynomial map: scalar product

- Calculating $\langle \Phi(x), \Phi(y) \rangle$ is $O(m^2)$
- For comparison, calculate $(x \cdot y + 1)^2$:

- $(x \cdot y + 1)^2 = ((\sum_{i=1}^m x_i \cdot y_i) + 1)^2$
  $= \left(\sum_{i=1}^m x_i y_i\right)^2 + 2\left(\sum_{i=1}^m x_i y_i\right) + 1$
  $= \sum_{i=1}^m \sum_{j=1}^m x_i y_i x_j y_j + 2\sum_{i=1}^m x_i y_i + 1$
  $= \sum_{i=1}^m (x_i y_i)^2 + 2\sum_{i=1}^m \sum_{j=1}^m x_i y_i x_j y_j + 2\sum_{i=1}^m x_i y_i + 1$
  $= \Phi(x) \cdot \Phi(y)$

- We can replace $\langle \Phi(x), \Phi(y) \rangle$ with $(x \cdot y + 1)^2$, which is $O(m)$

References: web resources

- see full references in part one (AL 3a)

- L. Bottou, O. Chapelle, D. DeCoste, J. Weste (Eds), *Large-Scale Kernel Machines*, MIT Press, 2007
- C. J. C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery 2, 121–167 (1998)
- A. Ben-Hur, D. Horn, H. T. Siegelmann, V. Vapnik, *Support Vector Clustering*, Journal of Machine Learning Research 2, 125–137 (2001)

Applications of SVM

- data clustering
- multi-class classification
- visual pattern (object) recognition
- text classification: string kernels
- DNA sequence classification
- function approximation
- . . .

- of course, streamlined kernels for each domain
- let's take a look at a few examples. . .

SVC - support vector clustering

Ben-Hur, Horn, Siegelmann, Vapnik (2001)

- map data points to high-dimensional feature space
- using the Gaussian kernel
- look for the smallest sphere that encloses the data

- map back to data space
- to get the set of contours which enclose the cluster(s)

- identifies valleys in the data probability distribution
- use soft-margin SVM to handle outliers

SVC: Example data set and results

Gaussian kernel $K(x,z) = e^{-q\,(x-z)^2}$, radius q

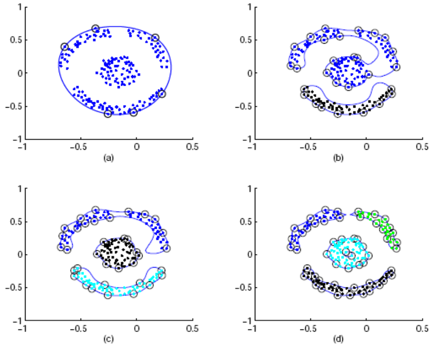


Figure 1: Clustering of a data set containing 183 points using SVC with $C = 1$. Support vectors are designated by small circles, and cluster assignments are represented by different grey scales of the data points. (a): $q = 1$ (b): $q = 20$ (c): $q = 24$ (d): $q = 48$.

SVC: Example data set, number of support vectors

Gaussian kernel $K(x,z) = e^{-q\,(x-z)^2}$, radius q

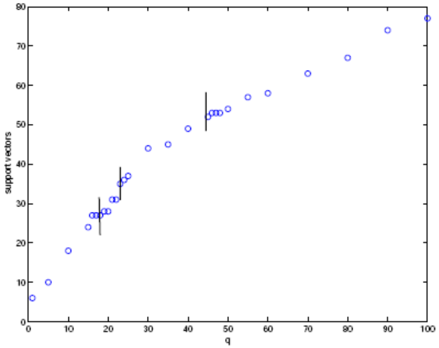


Figure 2: Number of SVs as a function of q for the data of Figure 1. Contour splitting points are denoted by vertical lines.

SVC: Noisy data

- Use soft-margin SVM learning algorithm
- with control parameter C
- and slack variables $\xi_i \geq 0$

- non-support vectors: inside the cluster
- support vectors: on the cluster boundary

- bounded support vectors: outside the boundary (violation)
- number of bounded support vectors is $n_{\mathrm{bsv}} < 1/C$
- fraction of outliers: $p = 1/NC$

SVC: Noisy data and bounded support vectors

Soft-margin SVM learning with control parameter C

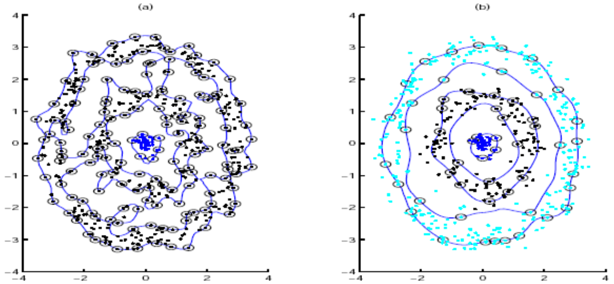


Figure 3: Clustering with and without BSVs. The inner cluster is composed of 50 points generated from a Gaussian distribution. The two concentric rings contain 150/300 points, generated from a uniform angular distribution and radial Gaussian distribution. (a) The rings cannot be distinguished when $C = 1$. Shown here is $q = 3.5$, the lowest q value that leads to separation of the inner cluster. (b) Outliers allow easy clustering. The parameters are $p = 0.3$ and $q = 1.0$.

SVC: Noisy data and bounded support vectors

Soft-margin SVM learning with control parameter C

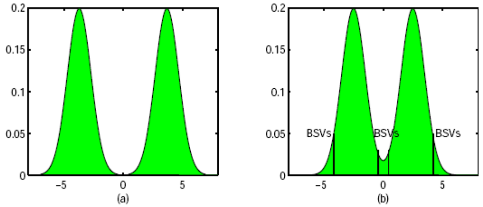


Figure 4: Clusters with overlapping density functions require the introduction of BSVs.

- Remember: larger C implies larger margin
- at the cost of more *bounded support vectors*
- with $y_i \langle w_i, x_i \rangle \geq 1 - \xi_i$

SVC: Strongly overlapping clusters

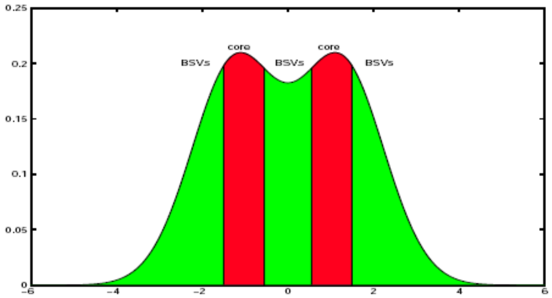


Figure 5: In the case of significant overlap between clusters the algorithm identifies clusters according to dense cores, or maxima of the underlying probability distribution.

SVC: Comparison with classical clustering

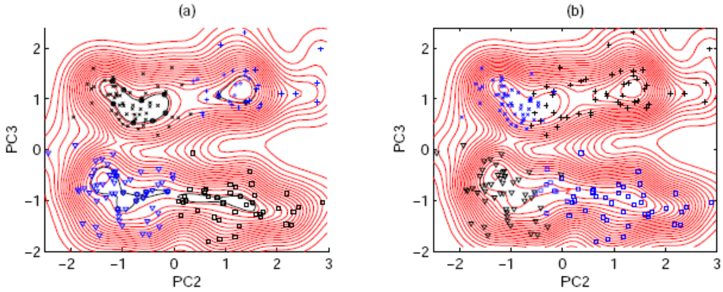


Figure 6: Ripley's crab data displayed on a plot of their 2nd and 3rd principal components: (a) Topographic map of $P_{svc}(\mathrm{x})$ and SVC cluster assignments. Cluster core boundaries are denoted by bold contours; parameters were $q = 4.8, p = 0.7$. (b) The Parzen window topographic map $P_w(\mathrm{x})$ for the same q value, and the data represented by the original classification given by Ripley (1996).

Selecting the control parameter

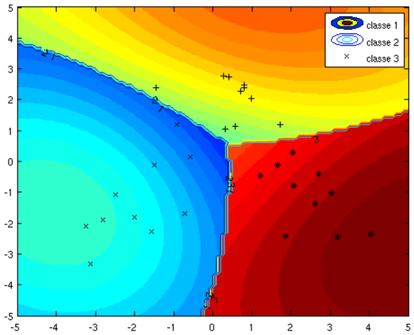
- training result depends on the specified control parameter C
- how to select the value of C?

- depends on the application and training data
- Numerical Recipes recommends the following
  - start with $C = 1$
  - then try to increase or decrease by powers of 10
  - until you find a broad plateau where the exact value of C doesn't matter much
  - a good SVM solution should classify most patterns correctly,
  - with many $\alpha_i = 0$ and many $\alpha_i = C$, but only a few in between

Multi-class classification

- many practical classification problems involve more than just two classes!
- for example, clustering (see above), object recognition, handwritten digit and character recognition, audio and natural speach recognition, etc.
- but standard SVM handles only exactly two classes
- "hard-coded" in the SVM algorithm
- What to do?

Example: multi-class classification



- exmulticlassall from SVM-KM toolbox (3 classes)
- demo

One versus the rest classification

To get an *M*-class classifier:

- construct a set of M binary classifiers $f^1, \dots, f^M$
- each trained to separate one class from the rest
- combine them according to the maximal individual output before applying the sgn-function

$$\arg \max_{j=1,\dots,M} g^j(x), \quad \text{where} \quad g^j(x) = \sum_{i=1}^{m} \left(y_i \alpha_i^j k(x, x_i) + b^j \right)$$

- the *winner-takes-all* approach

One versus the rest: winner takes all

- the above algorithm looks for $\arg\max g^j(x)$

- the M different classifiers have been trained
- on the same training data
- but with different binary classification problems

- unclear whether the $g^j(x)$ are on comparable scales
- a problem, when several (or none) classifiers claim the pattern
- try to balance/scale the $g^j(x)$

- all classifiers trained on very unsymmetrical problems
- many more negative than positive patterns
  - (e.g. digit-7 vs. all handwritten characters and digits)

One versus the rest: reject decisions

- The values of $g^j(x)$ can be used for reject decisions in the classification of x
- consider the difference between the two largest $g^j(x)$ as a measure of confidence

- if the measure falls short of a threshold θ, the classifier rejects the pattern
- can often lower the error-rate on other patterns
- can forward un-classified patterns to human experts

Pairwise classification

- train a classifier for each possible pair of classes
- for M classes, requires $M(M-1)/2$ binary classifiers
  - digit-0-vs-digit-1, digit-0-vs-digit-2, $\ldots$, digit-8-vs-digit-9

- (many) more classifiers than one-vs-the-rest for $M > 3$
- and probably, longer training times

- but each individual pairwise classifier is (usually) much simpler than each one-vs-the-rest classifier

Pairwise classification: tradeoff

- requires $(M-1)M/2$ classifiers vs M one-vs-the-rest

- each individual classifier much simpler
- smaller training sets (e.g. digit-7 vs. digit-8)
  - for super-linear learning complexity like $O(n^3)$, the shorter training times can outweigh the higher number of classifiers

- usually, fewer support vectors
  - training sets are smaller
  - classes have less overlap

Pairwise classification: tradeoff

- requires $(M-1)M/2$ classifiers vs M one-vs-the-rest
- but fewer support vectors per classifier

- if M is large, will be slower than M one-vs-the-rest
- example: digit-recognition task and the following scenario:
  - after evaluating the first few classifiers,
  - digit 7 and digit 8 seem unlikely ("lost" in the first rounds)
  - rather pointless to run the digit-7-vs-digit-8 classifier

- embed the pairwise classifiers into a directed acyclic graph
  - each classification run corresponds to a graph traversal
  - much faster than running all pairwise classifiers

Error-correcting output coding

- train a number of L binary classifiers $f^1, \ldots, f^L$
- on subproblems involving subsets of the M classes
  - e.g. separate digits 0..4 from 5..9, 0..2 from 3..9, etc.

- If the set of binary classifiers is chosen correctly,
- their responses $\{\pm 1\}^L$ determine the output class of a test pattern
- e.g., $\log_2(M)$ classifiers on binary-encoding ...

- use error-correcting codes to improve robustness against individual mis-classifications
  - note: newest schemes also use the margins of the individual classifiers for decoding

Error-correcting output coding

- example: Hamming (7,4) code
  - linear error-correcting block-code
  - 4 databits, 3 parity bits
  - detects and corrects 1-bit errors
  - generator matrix G and decoding matrix H

$$\mathbf{G}' := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\mathbf{H}' := \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

- more efficient codes
  - BCD (Bose, Chaudhuri, Hocquenghem)
    e.g. BCH(15,7,5) corrects 2-bit errors
  - RS (Reed-Solomon)
  - ...
  - large block-sizes required for low overhead

Multi-class objective functions
LWK p.213

- re-design the SVM algorithm to directly handle multi-classes
- $y_i \in \{1, \dots M\}$ the multi-class label of pattern x_i
- $m \in \{1, \dots, M\}$

$$\text{minimize}_{w_r \in \mathcal{H}, \xi_i^r \in \mathbb{R}^m, b_r \in \mathbb{R}} \frac{1}{2} \sum_{r=1}^{M} ||w_r||^2 + \frac{C}{m} \sum_{i=1}^{m} \sum_{r \neq y_i} \xi_i^r$$

subject to $\langle w_{y_i}, x_i \rangle + b_{y_i} \geq \langle w_r, x_i \rangle + b_r + 2 - \xi_i^r$, with $\xi_i^r \geq 0$.

- optimization problem has to deal with all SVMs at once
- large number of support vectors
- results comparable with one-vs-the-rest approach

Summary: multi-class classification

LWK p.214

- Basic SVM algorithm only supports binary classifications
- Several options for M-class classification

1. M one-versus-the-rest classifiers
2. $M(M-1)/2$ pairwise binary classifiers
3. suitably chosen subset classifiers (at least, $\log_2 M$), plus error-correcting codes for robustness
4. redesigned SVM with multi-class objective function

- no approach outperforms all others
- often, one-vs-the-rest produces acceptable results

SVM for visual pattern recognition

- one very popular application of SVMs
- can work on raw pixels
- or handcrafted feature maps

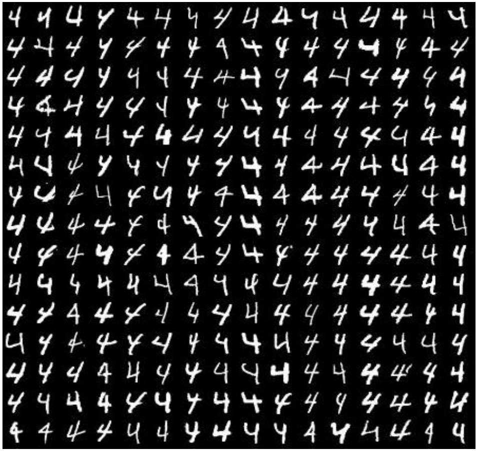
- MNIST handwritten digit recognition
- NORB object recognition
- histogram based classification

MNIST handwritten characters data set

- set of handwritten digits
- based on NIST database 1 and -3 (b&w)
- 20x20 pixel grayscale images (interpolated from 28x28 b&w)
- used as a benchmark for (multi-class) classifiers
- training set with 60.000 patterns
- test set with 10.000+ patterns

- http://yann.lecun.com/exdb/mnist/
- current best classifier achieves 0.38% error

MNIST data set: example '4'



http://www.cvl.isy.liu.se/ImageDB/images/external_images/MNIST_digits/mnist_train4.jpg

MNIST benchmark

- typical published results on raw MNIST
  - 0% percent errors on training set
  - about 3% errors on the test set

- apparently, training and test set don't match perfectly
- some test patterns quite different from training patterns
- difficult to achieve very good error rates

- several approaches based on extra training patterns

MNIST results overview

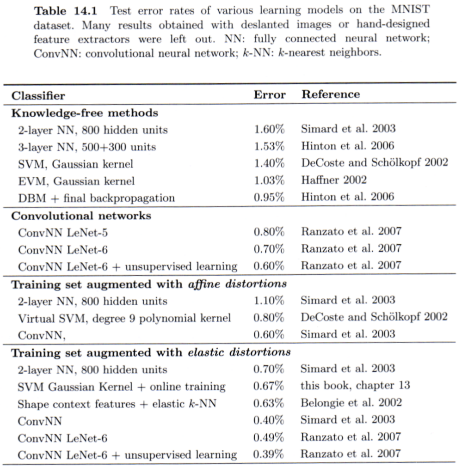


Table 14.1 Test error rates of various learning models on the MNIST dataset. Many results obtained with deslanted images or hand-designed feature extractors were left out. NN: fully connected neural network; ConvNN: convolutional neural network; k-NN: k-nearest neighbors.

Classifier	Error	Reference
Knowledge-free methods		
2-layer NN, 800 hidden units	1.60%	Simard et al. 2003
3-layer NN, 500+300 units	1.53%	Hinton et al. 2006
SVM, Gaussian kernel	1.40%	DeCoste and Schölkopf 2002
EVM, Gaussian kernel	1.03%	Haffner 2002
DBM + final backpropagation	0.95%	Hinton et al. 2006
Convolutional networks		
ConvNN LeNet-5	0.80%	Ranzato et al. 2007
ConvNN LeNet-6	0.70%	Ranzato et al. 2007
ConvNN LeNet-6 + unsupervised learning	0.60%	Ranzato et al. 2007
Training set augmented with *affine distortions*		
2-layer NN, 800 hidden units	1.10%	Simard et al. 2003
Virtual SVM, degree 9 polynomial kernel	0.80%	DeCoste and Schölkopf 2002
ConvNN,	0.60%	Simard et al. 2003
Training set augmented with *elastic distortions*		
2-layer NN, 800 hidden units	0.70%	Simard et al. 2003
SVM Gaussian Kernel + online training	0.67%	this book, chapter 13
Shape context features + elastic k-NN	0.63%	Belongie et al. 2002
ConvNN	0.40%	Simard et al. 2003
ConvNN LeNet-6	0.49%	Ranzato et al. 2007
ConvNN LeNet-6 + unsupervised learning	0.39%	Ranzato et al. 2007

MNIST experiments with SVMs

- G. Loosli, L. Bottou, S. Canu, *Training Invariant SVMs Using Selective Sampling*, in *Large-Scale Kernel Machines*, 2007

- an approach to improve the classification error rate
- by increasing the training set
- with automatically synthesized patterns
- derived from the original training patterns

- 100 random deformations of each original image
- 6 million training images. . .

Virtual training set

- synthesize new training patterns
- by applying deformations on each original pattern
  - affine transformations (sub-pixel accuracy)
    translations, rotations, scaling
  - deformation-fields (elastic transformation)
  - thickening
  - ...

Goal:

- a *transformation invariant* classifier
- more robust to slight variations in the test patterns
- but handling transformations can also increase the test set error
- of course, much higher training effort and time

MNIST modified training patterns

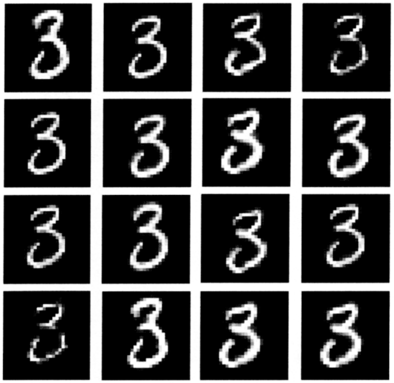


Figure 13.4 This figure shows 16 variations of a digit with all the transformations cited here.

MNIST effect of transformations

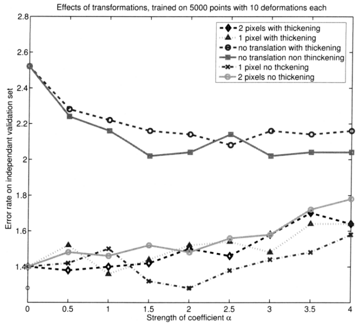
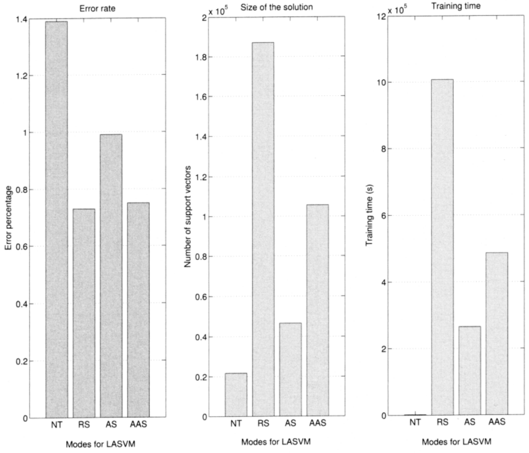


Figure 13.5 Effects of transformations on the performance. This graph is obtained on a validation set of 10,000 points, trained on 5000 points and 10 transformations for each (55,000 training points in total) with an RBF kernel with bandwidth $\gamma = 0.006$. The best configuration is elastic deformation without thickening, for $\eta = 2$ and one-pixel translations, which gives 1.28%. Note that $\eta = 0$ is equivalent to no elastic deformation. The baseline result for the validation set is thus 2.52%.

LASVM training algorithm

- due to the problem size, training is done using an iterative algorithm, one pattern a time.
- several choices to select the next training pattern:
  - random selection: picks a random unseen training example
  - gradient selection: pick the most poorly classified example (smallest value of $y_k f(x_k)$ among 50 randomly selected unseen training examples)
  - active selection: pick the training example that is closest to the decision boundary (smallest value of $|f(x_k)|$) among 50 randomly selected unseen training examples)
  - autoactive selection: randomly sample at most 100 unseen training examples, but stop as soon as 5 fall inside the margins (will become support vectors). Pick the one closest to the decision boundary.

MNIST benchmark results



MNIST benchmark results explanation

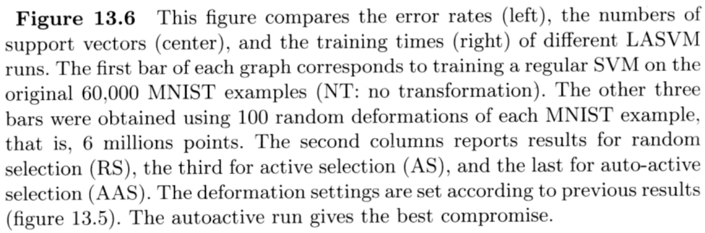


Figure 13.6 This figure compares the error rates (left), the numbers of support vectors (center), and the training times (right) of different LASVM runs. The first bar of each graph corresponds to training a regular SVM on the original 60,000 MNIST examples (NT: no transformation). The other three bars were obtained using 100 random deformations of each MNIST example, that is, 6 millions points. The second columns reports results for random selection (RS), the third for active selection (AS), and the last for auto-active selection (AAS). The deformation settings are set according to previous results (figure 13.5). The autoactive run gives the best compromise.

MNIST evolution of training and test errors

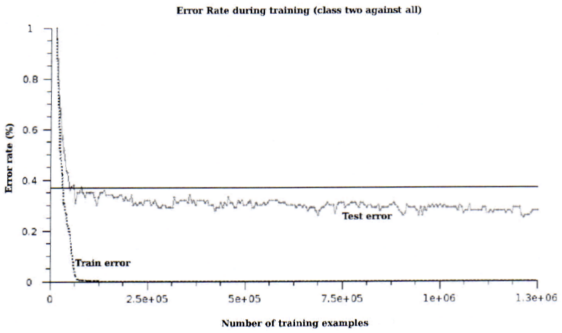


Figure 13.8 This figure shows the evolution of the training error on the 60,000 original points and the test error during training. The results are shown for class 2 against the others. We observe that training on deformed examples does not affect the performance on orginal examples but increases the performance on unseen data.

MNIST SVM experiment results

G. Loosli, L. Bottou, S. Canu, Training Invariant SVMs Using Selective Sampling

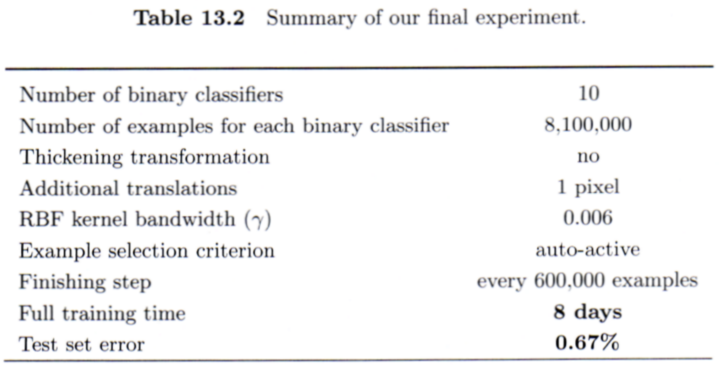


Table 13.2 Summary of our final experiment.

Number of binary classifiers	10
Number of examples for each binary classifier	8,100,000
Thickening transformation	no
Additional translations	1 pixel
RBF kernel bandwidth (γ)	0.006
Example selection criterion	auto-active
Finishing step	every 600,000 examples
Full training time	**8 days**
Test set error	**0.67%**

NORB

- Natural images of 3D-objects, 96x96 pixels
- 50 different toys in 5 categories
- 25 objects for training, 25 for testing
- each object captured in stereo from 162 viewpoints (9 elevations, 18 azimuths)

- objects in front of uniform background
- or in front of cluttered background (or missing object)

- http://www.cs.nyu.edu/~ylclab/data/norb-v1.0/

NORB normalized-uniform training set

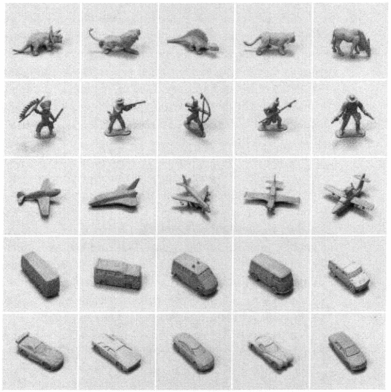


Figure 14.6 The 25 testing objects in the *normalized-uniform* NORB set. The testing objects are unseen by the trained system.

NORB jittered-cluttered training set

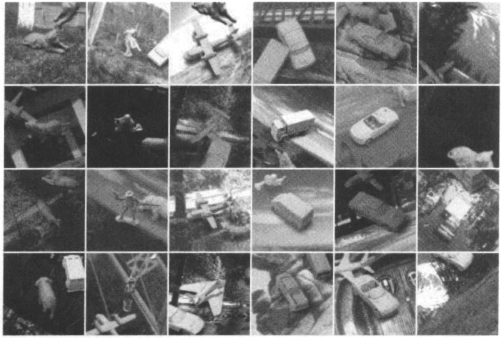


Figure 14.7 Some of the 291,600 examples from the *jittered-cluttered* training set (left camera images). Each column shows images from one category. A 6-th background category is added

NORB data set

- recognition basically can only rely on the shape of the object
- all other typical clues eliminated or unusable

- different orientations (viewing angles)
- different lighting conditions
- no color information (grayscale only)
- no object texture
- different backgrounds (cluttered set)

- no hidden regularities

Error rates: normalized uniform set

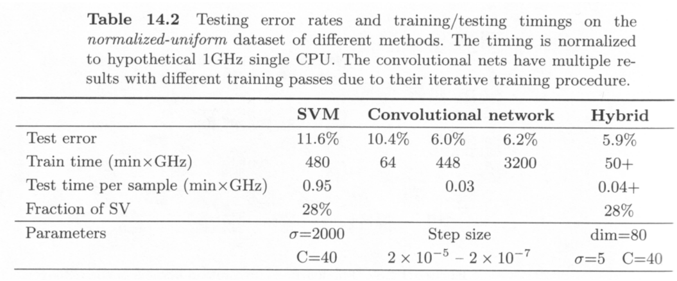


Table 14.2 Testing error rates and training/testing timings on the *normalized-uniform* dataset of different methods. The timing is normalized to hypothetical 1GHz single CPU. The convolutional nets have multiple results with different training passes due to their iterative training procedure.

	SVM	Convolutional network			Hybrid
Test error	11.6%	10.4%	6.0%	6.2%	5.9%
Train time (min×GHz)	480	64	448	3200	50+
Test time per sample (min×GHz)	0.95		0.03		0.04+
Fraction of SV	28%				28%
Parameters	σ=2000		Step size		dim=80
	C=40		$2 \times 10^{-5} - 2 \times 10^{-7}$		σ=5 C=40

- five binary SVMs, one for each class
- trained on the raw pixel images ($d = 96 \cdot 96 \cdot 2 = 18432$)
- convolutional network uses handcrafted feature map
- hybrid system trains SVMs on those features

Feature functions for the NORB set

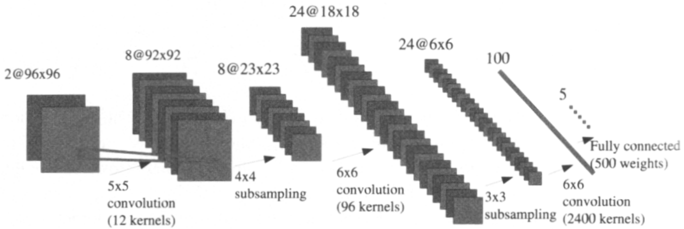

2@96x96
8@92x92
8@23x23
24@18x18
24@6x6
100
5
5x5 convolution (12 kernels)
4x4 subsampling
6x6 convolution (96 kernels)
3x3 subsampling
6x6 convolution (2400 kernels)
Fully connected (500 weights)

Figure 14.5 The architecture of the convolutional net used for the NORB experiments. The input is an image pair, the system extracts 8 feature maps of size 92×92, 8 maps of 23×23, 24 maps of 18×18, 24 maps of 6×6, and a 100-dimensional feature vector. The feature vector is then transformed into a 5-dimensional vector in the last layer to compute the distance with target vectors.

Some feature functions for the NORB set

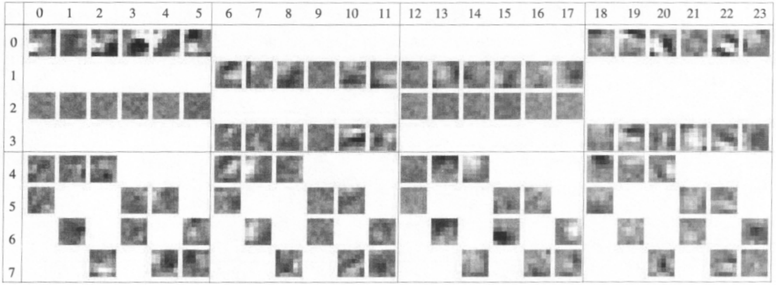


Figure 14.8 The learned convolution kernels of the C3 layer. The columns correspond to the 24 feature maps output by C3, and the rows correspond to the 8 feature maps output by the S2 layer. Each feature map draws from 2 monocular maps and 2 binocular maps of S2. Ninety-six convolution kernels are used in total.

Error rates: jittered cluttered set

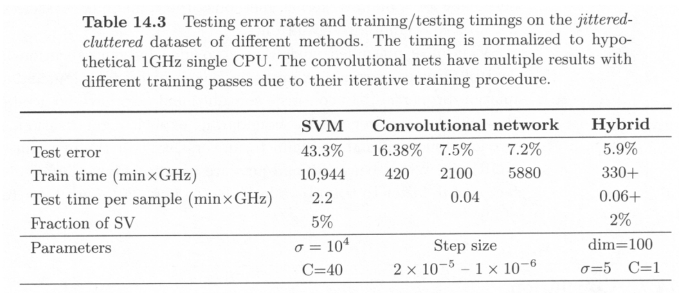


Table 14.3 Testing error rates and training/testing timings on the *jittered-cluttered* dataset of different methods. The timing is normalized to hypothetical 1GHz single CPU. The convolutional nets have multiple results with different training passes due to their iterative training procedure.

	SVM	Convolutional network			Hybrid
Test error	43.3%	16.38%	7.5%	7.2%	5.9%
Train time (min×GHz)	10,944	420	2100	5880	330+
Test time per sample (min×GHz)	2.2		0.04		0.06+
Fraction of SV	5%				2%
Parameters	$\sigma = 10^4$		Step size		dim=100
	C=40		$2 \times 10^{-5} - 1 \times 10^{-6}$		σ=5 C=1

- again, SVM trained on raw pixels
- convolutional network uses handcrafted feature maps
- hybrid system trains SVM on those feature maps

Error rates: SVM training and setup

- SVM with 43% error-rate?

- six one-vs-the-rest binary SVMs (one per catogory)
- training samples are raw 108x108 pixel images
  - again, use a virtual training set
  - ± 3 pixel translations
  - scaling from 80% to 110%
  - rotations $\pm 5^o$
  - changed brightness ± 20 and contrast
  - a total of 291.600 images
- overall, a 23 328-dimensional input vector
- only the Gaussian width σ as a free parameter

Poor performance of SVM on raw pixels?

- Gaussian kernel basically computers matching score (based on Euclidean distance) between training templates and the test pattern
- very sensitive to variations in registration, pose, illumination
- most of the pixels in NORB are background clutter
- hence, template matching dominated by background irregularities

- a general weakness of standard kernel methods: their inability to select relevant input features
- feature maps must be hand-crafted by experts

Example: Histogram correlation kernel

A. Barla et.al., *Image Kernels*, LNCS 2388, 83–96

- calculate image histograms
- use histogram intersection as the feature
- instead of in addition to raw pixels

- applied to indoor/outdoor image classification
- e.g. for improved color image printing
- test set with 300 indoor and outdoor test images each
- http://www.benchathlon.net/img/todo/

Histogram intersection kernel

- histograms A and B of images A_{im} and B_{im} with N pixels
- each histogram has m bins, A_i $(i = 1, \dots, m)$
- histogram intersection:

$$K_{\text{int}}(A, B) = \sum_{i=1}^{m} \min\{A_i, B_i\}$$

- note: this can be written as a kernel
- represent A as an $N \times m$ vector $\mathcal{A}$ defined as $\mathcal{A} :=$

$$(\overbrace{1, 1, \dots, 1}^{A_1}, \underbrace{0, 0, \dots, 0}_{N-A_1}, \overbrace{1, 1, \dots, 1}^{A_2}, \underbrace{0, 0, \dots, 0}_{N-A_2}, \dots, \overbrace{1, 1, \dots, 1}^{A_m}, \underbrace{0, 0, \dots, 0}_{N-A_m})$$

- Then $K_{\text{int}}(A, B) = \mathcal{A} \cdot \mathcal{B}$

Histogram kernel: recognition results

kernel	r.r.(%)
histogram intersection	93.1
linear	88.8
2-nd deg polynomial	89.2
3-rd deg polynomial	89.4
4-rd deg polynomial	88.1
Gaussian ($\sigma = 0.1$)	89.1
Gaussian ($\sigma = 0.3$)	86.5
Gaussian ($\sigma = 0.5$)	87.8

- 600 training images, 123 indoor and 260 outdoor test images
- histogram used $15 \times 15 \times 15$ bins in HSV colorspace
- other kernels trained on pixel data

Summary: visual pattern recognition

- SVM can be trained on and applied to raw pixel data
- use *virtual training set* for better generalization
- but no performance guarantees

- good results on MNIST
- but total breakdown on NORB

- must use appropriate feature maps
- or hybrid architectures

Text classification
LSKM p0.41

- another high dimensional problem...
- e.g. Reuters RCV1 text corpus
  - 810.000 news stories from 1996/1997
  - partitioned and indexed in 135 categories
  - http://trec.nist.gov/data/reuters/reuters.html

- represent word frequencies, e.g. *bag of words*
- or represent substring correlations
- train a SVM on the corpus
- classify the given input texts

Bag of words representation
LWK 13.2

Sparse vector kernel

- map the given text into a sparse vector
- where each component corresponds to a word
- and component is set to one when the word occurs
- dot products between such vectors are fast

- but ignores the ordering of the words
- no vicinity information (e.g. words in one sentence)
- only detects exact matches (e.g. mismatch on mathces)

String kernel

- efficient kernel that computes the dot product
- in the feature space spanned by *all* substrings of documents
- compuational complexity is linear in the document length and the length of the substring

- allows to classify texts on similarity of substrings
- the more substrings match, the higher the output value
- use for text classification, DNA sequence analysis, . . .

String kernel: definitions

- a finite alphabet Σ, so Σ^n the set of all string of length n
- $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ the set of all finite strings
- length of a string $s \in \Sigma^*$ is $|s|$
- string elements are $s(1) \ldots s(|s|)$
- $s\,t$ is the string concatenation of s and t

subsequences u of strings:

- index sequence $i := (i_1, \ldots, i_{|u|})$ with $1 \leq i_1 < dots < i_{|u|} \leq |s|$
- define $u := s(i) := s(i1) \ldots s(i_{|u|})$
- $l(i) := i_{|u|} - i_1 + 1$ the length of the subsequence in s

String kernel: feature space

- feature space $\mathcal{H} := \mathbb{R}^{(\Sigma^n)}$ built from strings of length n
- one dimension (coordinate) for each element of Σ^n

- feature map

$$[\Phi_n(s)]_u := \sum_{i:s(i)=u} \lambda^{l(i)}$$

- with *decay parameter* λ, $0 < \lambda < 1$
- the larger the length of the subsequence in s, the smaller its contribution to $[\Phi_n(s)]_u$.
- sum over all subsequences of s which equal u

String kernel: the actual kernel

- consider the dimension of $\mathcal{H}$ for the string asd
- $[\Phi_n(\texttt{Nasdaq})]_{\texttt{asd}} = \lambda^3$ (one exact match of length 3)
- $[\Phi_n(\texttt{lass das})]_{\texttt{asd}} = 2\lambda^5$ (two matches of length 5: ⊔as⊔⊔d⊔⊔ and ⊔a⊔s⊔d⊔⊔)

- kernel corresponding to the map $\Phi(n)$ is:

$$k_n(s,t) = \sum_{u \in \Sigma^n} [\Phi_n(s)]_u [\Phi_n(t)]_u = \sum_{u \in \Sigma^n} \sum_{(i,j):s(i)=t(j)=u} \lambda^{l(i)} \lambda^{l(j)}$$

- normalize: use $k(s,t)/\sqrt{k(s,s)k(t,t)}$

DNA sequence classification
LWK table 13.2 p.417

- DNA sequence contains *coding sequences* which encode proteins, but also untranslated sequences
- find the start points of proteins (TIS: translation initiation sites)
- out of $\{A, C, G, T\}$, typically an ATG triplet

- certain local correlations are typical
- $\mathrm{match}_{p+j}(x, x')$ is 1 for matching nucleotides at position $p + j$, 0 otherwise
- construct kernel that rewards nearby matches
  $\mathrm{win}_p(x, x') = \left(\sum_{j=-l}^{+l} \nu_j \mathrm{match}_{p+j}(x, x')\right)^{d_1}$
  $k(x, x') = \left(\sum_{p=1}^{l} \mathrm{win}_p(x, x')\right)^{d_2}$

WK kernel with shifts

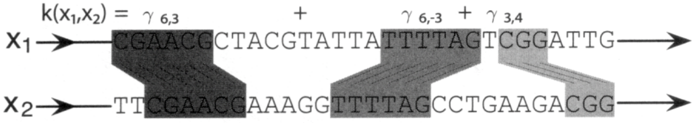

k(x1,x2) = γ 6,3 + γ 6,-3 + γ 3,4
x1 CGAACGCTACGTATTATTTTAGTCGGATTG
x2 TTCGAACGAAAGGTTTTAGCCTGAAGACGG

Figure 4.2 Given two sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ of equal length, the WD kernel with shifts consists of a weighted sum to which each match in the sequences makes a contribution $\gamma_{k,p}$ depending on its length k and relative position p, where long matches at the same position contribute most significantly. The γ's can be computed from the β's and δ's in (4.4). The spectrum kernel is based on a similar idea, but it only considers substrings of a fixed length and the contributions are independent of the relative positions of the matches to each other.

SVM-based regression

- use SVMs for function approximation?
- especially, for high-dimensional functions?

basic idea is very similar to classification:

- estimate linear functions $f(x) = \langle w, x \rangle + b$
- based on $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{H} \times \mathbb{R}$
- use a $||w||^2$ regularizer ("maximum margin")
- use optimization algorithm similar to SVM training
- use feature-maps to generalize to the non-linear case

ϵ-insensitive loss function
Vapnik 1995

- need a suitable cost-function
- define the following ϵ-*insensitive loss function*:

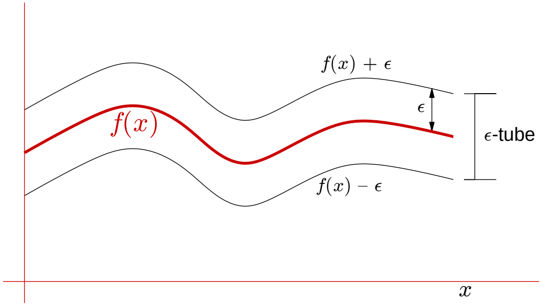
$$|y - f(x)|_\epsilon = \max\{0, |y - f(x)| - \epsilon\}$$

- threshold $\epsilon \geq 0$ is chosen a-priori
- small ϵ implies high approximation accuracy
- no penalty, when error below some threshold

- similar to classification loss-function: no penalty for correctly-classified training patterns

The ϵ-tube

around the ϵ-insensitive loss function



- geometrical interpretation: allow a tube
- of width ϵ around the given function values

Goal of the SVR learning

Given:

- a dot product space $\mathcal{H}$
- (mapped) input patterns $(x_1, y_1), \ldots, (x_m, y_m) \in \mathcal{H} \times \mathbb{R}$

Goal:

- find a function f with a small *risk* (or test error),
  $R[f] = \int c(f, x, y) dP(x, y)$
- where P is the probability measure for the observations
- and c is a loss function, e.g. $c(f, x, y) = (f(x) - y)^2$
- loss function can be chosen depending on the application

Regularized risk functional

- find a function f with a small *risk* (or test error), $R[f] = \int c(f, x, y) dP(x, y)$
- where P is the probability measure for the observations
- and c is a loss function, e.g. $c(f, x, y) = (f(x) - y)^2$
- cannot minimize c directly, because P is not known

- Instead, minimize the *regularized risk functional*

$$\frac{1}{2}||w||^2 + C \cdot R^{\epsilon}_{\text{emp}}, \qquad \text{where} \quad R^{\epsilon}_{\text{emp}} := \frac{1}{m}\sum_{i=1}^{m} |y_i - f(x_i)|_{\epsilon}$$

- $R^{\epsilon}_{\text{emp}}$ measures the ϵ-insensitive training error
- C controls the trade-off between margin and training error

Main idea

- minimize the *regularized risk functional*

$$\frac{1}{2}||w||^2 + C \cdot R_{\text{emp}}^{\epsilon}$$

- where $R_{\text{emp}}^{\epsilon} := \frac{1}{m}\sum_{i=1}^{m} |y_i - f(x_i)|_{\epsilon}$ measures the training error
- constant C determines the trade-off

To obtain a small risk

- control both the training error ($R_{\text{emp}}^{\epsilon}$)
- and the model complixity ($||w||^2$)
- in short, "explain the data with a simple model"

ϵ-SVR objective function

- again, rewrite as (soft-margin) optimization problem:

$$\text{minimize}_{w_r \in \mathcal{H}, \xi^{(*)} \in \mathbb{R}^m, b \in \mathbb{R}} \frac{1}{2} ||w||^2 + \frac{C}{m} \sum_{i=1}^{m} (\xi_i + \xi^*)$$

subject to

- $\big(\langle x, x_i \rangle + b\big) - y_i \leq \epsilon + \xi_i$
- $y_i - \big(\langle w, x_i \rangle + b\big) \leq \epsilon + \xi_i^*$
- $\xi_i^{(*)} \geq 0$
- where $(*)$ means both the variables with and without asterisks.

ϵ-SVR dual problem

- introduce two sets of Lagrange multipliers $\alpha_i^{(*)}$ and $\eta_i^{(*)}$
- and minimize

$$\begin{aligned} L \quad := \quad & \frac{1}{2}||w||^2 + \frac{C}{m}\sum_{i=1}^{m}(\xi_i + \xi^*) - \sum_{i=1}^{m}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{m}\alpha_i\left(\epsilon + \xi_i + y_i - \langle w, x_i\rangle - b\right) \\ & - \sum_{i=1}^{m}\alpha_i^*\left(\epsilon + \xi_i^* + y_i - \langle w, x_i\rangle - b\right) \end{aligned}$$

- subject to $\alpha_i^{(*)}, \eta_i^{(*)} \geq 0$.

SV expansion

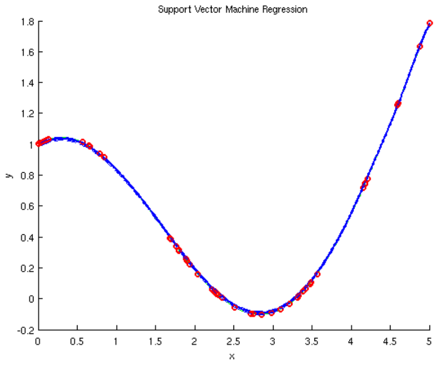
For the whole details, see LWK 9.2 (p. 254ff)

- solution of the optimization problem results in the SV expansion

$$f(x) = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i)\langle x_i, x\rangle + b$$

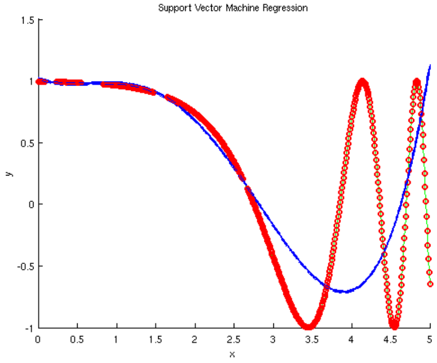
- w can be written as a linear combination
- of a subset of the training patterns x_i
- algorithm can be desribed in terms of dot products between the data
- when evaluating $f(x)$, we need not to compute w explicitly

Example: function approximation



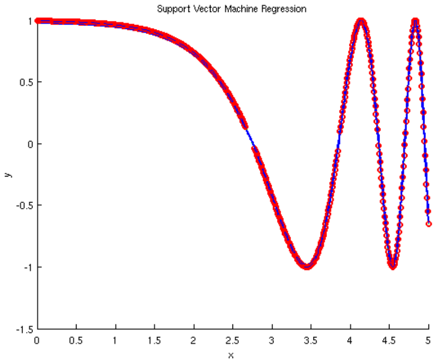
- exreg1dls from SVM-KM toolbox (gaussian basis function)
- only a few support-vectors

Example: function approximation



- exreg1dls from SVM-KM toolbox (4th-order polynom)
- fails to approximate the target function sin(exp(x))

Example: function approximation



- exreg1dls from SVM-KM toolbox (ht radial basis function)
- very many support-vectors, but good approximation

SVM complexity analysis

what is the complexity of SVM classification?

- n examples (training patterns)
- soft-margin control parameter C
- S support vectors, R free support vectors

two intuitive bounds:

- $O(nS)$ when few support vectors (C small)
- $O(R^3)$ if many support vectors (C large)
- so, both quadratic and cubic terms

Number of support vectors

assume an oracle which tells us

- the patterns that are not support vectors ($\alpha_i = 0$)
- and the bounded support vectors ($\alpha_i = C$)

- remaining R free support vectors are determined by R linear equations (representing the derivatives of the object function)
- solving this system takes $O(R^3)$ operations

Verifying a solution

- n examples
- S support vectors

- verification that vector α is a solution
- compute the gradient g of the dual
- and check the optimality conditions
- requires $O(n \cdot S)$ operations

SVM training time: example

LSKM: SVM training with primal and dual represenations

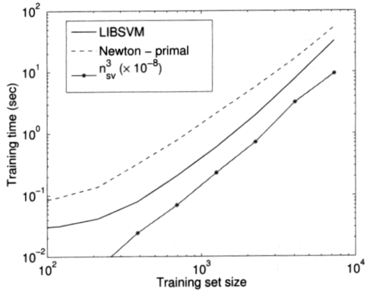


Figure 2.8 Time comparison between LIBSVM and Newton optimization. Here n_{sv} has been computed from LIBSVM (note that the solutions are not exactly the same). For this plot, $h = 2^{-5}$.

Computation of Kernel values?

- computing kernels can be expensive,
  e.g. consider images with thousands of pixels
  e.g. consider documents with thousands of words

- computing the full kernel matrix is wasteful
- gradient computation only depend on K_{ij} which invole support vectors (otherwise multiplied by zero)
- the kernel matrix does not fit in memory
- use special caching, or re-compute on-the-fly

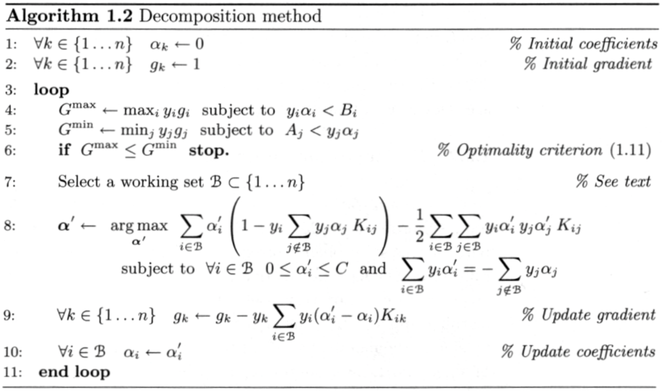
Efficient algorithms?

See LSKM p.12ff

- use traditional QP solver: works, but
  - kernel-matrix is seldom sparse (only the SVs are)
  - kernel-matrix rarely fits in memory
  - high-accuracy solution not required
- use "chunking": guess the support-vectors
  - select a "working set" and solve
  - usually works well, due to generalization
  - misclassified patterns are SV candidates
- use efficient direction search to improve the α_i
  - to speed up the training

Decomposition method

LSKM p. 17

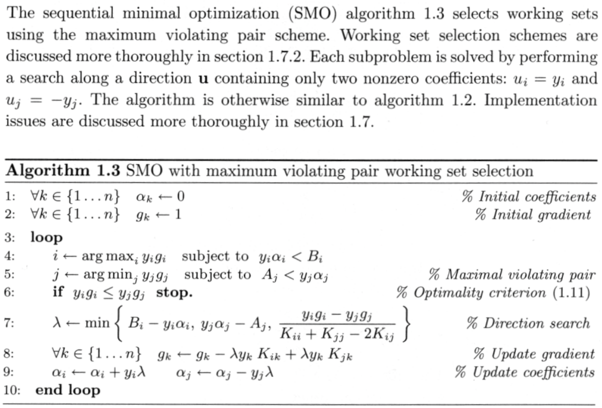


Algorithm 1.2 Decomposition method

```
1:  $\forall k \in \{1 \dots n\} \quad \alpha_k \leftarrow 0$        % Initial coefficients
2:  $\forall k \in \{1 \dots n\} \quad g_k \leftarrow 1$        % Initial gradient
3:  loop
4:      $G^{\max} \leftarrow \max_i y_i g_i$ subject to $y_i \alpha_i < B_i$
5:      $G^{\min} \leftarrow \min_j y_j g_j$ subject to $A_j < y_j \alpha_j$
6:      if $G^{\max} \leq G^{\min}$ stop.        % Optimality criterion (1.11)
7:      Select a working set $\mathcal{B} \subset \{1 \dots n\}$        % See text
8:      $\alpha' \leftarrow \underset{\alpha'}{\arg\max} \sum_{i \in \mathcal{B}} \alpha'_i \left(1 - y_i \sum_{j \notin \mathcal{B}} y_j \alpha_j K_{ij}\right) - \frac{1}{2} \sum_{i \in \mathcal{B}} \sum_{j \in \mathcal{B}} y_i \alpha'_i y_j \alpha'_j K_{ij}$
            subject to $\forall i \in \mathcal{B} \quad 0 \leq \alpha'_i \leq C$ and $\sum_{i \in \mathcal{B}} y_i \alpha'_i = - \sum_{j \notin \mathcal{B}} y_j \alpha_j$
9:      $\forall k \in \{1 \dots n\} \quad g_k \leftarrow g_k - y_k \sum_{i \in \mathcal{B}} y_i (\alpha'_i - \alpha_i) K_{ik}$        % Update gradient
10:     $\forall i \in \mathcal{B} \quad \alpha_i \leftarrow \alpha'_i$        % Update coefficients
11: end loop
```

SMO with maximum violating pair

LSKM p. 19



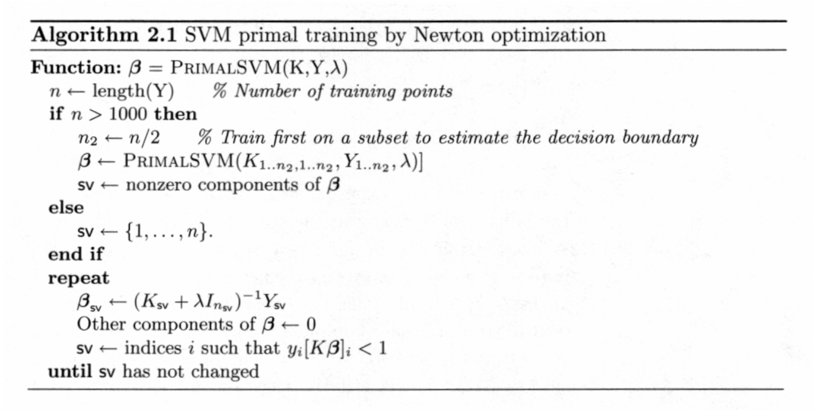
The sequential minimal optimization (SMO) algorithm 1.3 selects working sets using the maximum violating pair scheme. Working set selection schemes are discussed more thoroughly in section 1.7.2. Each subproblem is solved by performing a search along a direction $\mathbf{u}$ containing only two nonzero coefficients: $u_i = y_i$ and $u_j = -y_j$. The algorithm is otherwise similar to algorithm 1.2. Implementation issues are discussed more thoroughly in section 1.7.

Algorithm 1.3 SMO with maximum violating pair working set selection

```
1:  $\forall k \in \{1 \dots n\} \quad \alpha_k \leftarrow 0$                                  % Initial coefficients
2:  $\forall k \in \{1 \dots n\} \quad g_k \leftarrow 1$                                      % Initial gradient
3:  loop
4:      $i \leftarrow \arg\max_i y_i g_i$  subject to  $y_i \alpha_i < B_i$
5:      $j \leftarrow \arg\min_j y_j g_j$  subject to  $A_j < y_j \alpha_j$                  % Maximal violating pair
6:      if $y_i g_i \leq y_j g_j$ stop.                                                      % Optimality criterion (1.11)
7:      $\lambda \leftarrow \min \left\{ B_i - y_i\alpha_i,\ y_j\alpha_j - A_j,\ \frac{y_i g_i - y_j g_j}{K_{ii} + K_{jj} - 2K_{ij}} \right\}$      % Direction search
8:      $\forall k \in \{1 \dots n\} \quad g_k \leftarrow g_k - \lambda y_k K_{ik} + \lambda y_k K_{jk}$     % Update gradient
9:      $\alpha_i \leftarrow \alpha_i + y_i\lambda \quad \alpha_j \leftarrow \alpha_j - y_j\lambda$         % Update coefficients
10: end loop
```

SVM primal training

LSKM p. 41



Algorithm 2.1 SVM primal training by Newton optimization

```
Function: $\beta$ = PrimalSVM(K,Y,$\lambda$)
  $n \leftarrow$ length(Y)      % *Number of training points*
  if $n > 1000$ then
     $n_2 \leftarrow n/2$      % *Train first on a subset to estimate the decision boundary*
     $\beta \leftarrow$ PrimalSVM($K_{1..n_2,1..n_2}, Y_{1..n_2}, \lambda$)]
     sv $\leftarrow$ nonzero components of $\beta$
  else
     sv $\leftarrow \{1, \ldots, n\}$.
  end if
  repeat
     $\beta_{\mathsf{sv}} \leftarrow (K_{\mathsf{sv}} + \lambda I_{n_{\mathsf{sv}}})^{-1} Y_{\mathsf{sv}}$
     Other components of $\beta \leftarrow 0$
     sv $\leftarrow$ indices $i$ such that $y_i[K\beta]_i < 1$
  until sv has not changed
```

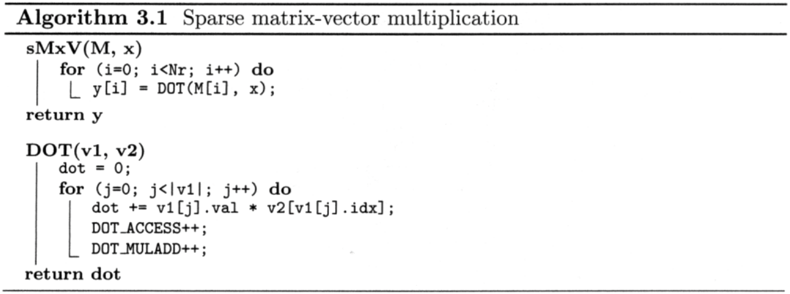
Sparse vector and matrix representation
LSKM chapter 3, p. 51ff

- basic SVM training and classification algorithms:
  - loops over weighted scalar products $\alpha_i \langle x_i, x \rangle$
  - for all training patterns x_i

- but after training, most weights $\alpha_i = 0$
- only the support vectors have $\alpha_{i,\mathrm{SV}} \neq 0$

- study *sparse* representations of vectors and matrices
- to reduce the storage requirements
- to improve performance

Sparse matrix by vector multiplication



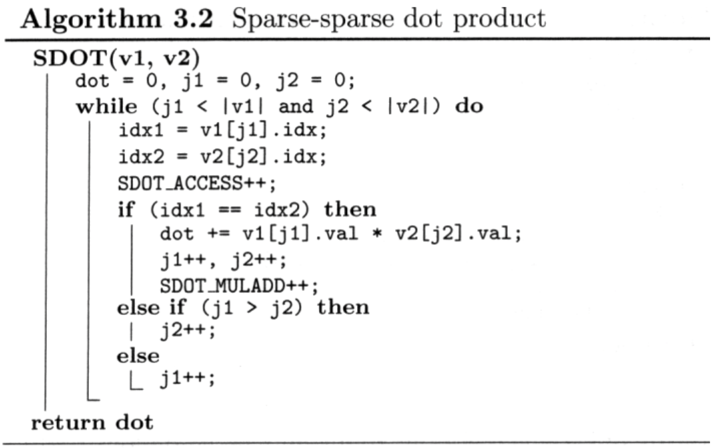
Algorithm 3.1 Sparse matrix-vector multiplication

```
sMxV(M, x)
    for (i=0; i<Nr; i++) do
        y[i] = DOT(M[i], x);
return y

DOT(v1, v2)
    dot = 0;
    for (j=0; j<|v1|; j++) do
        dot += v1[j].val * v2[v1[j].idx];
        DOT_ACCESS++;
        DOT_MULADD++;
return dot
```

- vector v_1 stored as sorted table of (index,value) pairs
- complexity of DOT is $O(|v1|)$
- memory accesses more expensive than the multiply-add

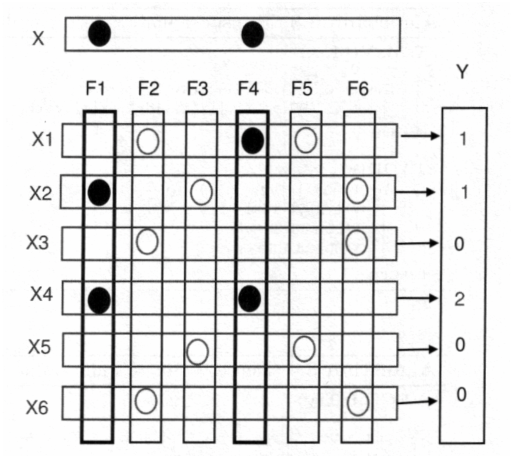
Sparse sparse dot product



Algorithm 3.2 Sparse-sparse dot product

```
SDOT(v1, v2)
    dot = 0, j1 = 0, j2 = 0;
    while (j1 < |v1| and j2 < |v2|) do
        idx1 = v1[j1].idx;
        idx2 = v2[j2].idx;
        SDOT_ACCESS++;
        if (idx1 == idx2) then
            dot += v1[j1].val * v2[j2].val;
            j1++, j2++;
            SDOT_MULADD++;
        else if (j1 > j2) then
            j2++;
        else
            j1++;
return dot
```

Sparse matrix vector product



Sparse matrix vector product

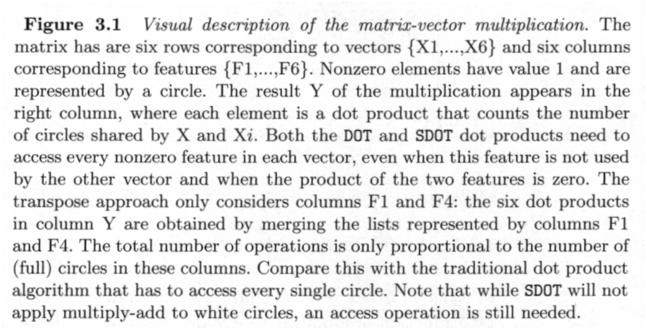
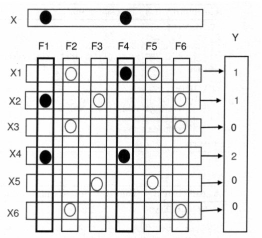
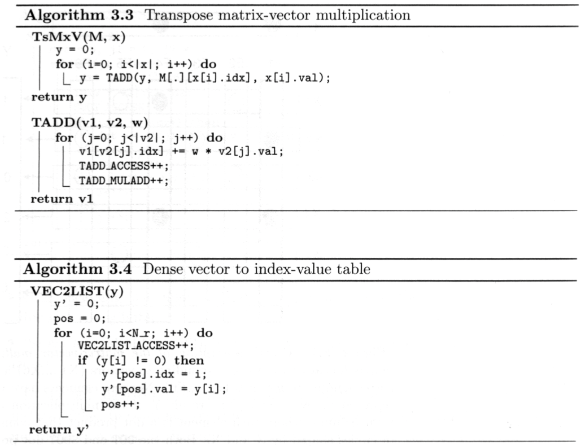


Figure 3.1 *Visual description of the matrix-vector multiplication.* The matrix has are six rows corresponding to vectors {X1,...,X6} and six columns corresponding to features {F1,...,F6}. Nonzero elements have value 1 and are represented by a circle. The result Y of the multiplication appears in the right column, where each element is a dot product that counts the number of circles shared by X and X*i*. Both the DOT and SDOT dot products need to access every nonzero feature in each vector, even when this feature is not used by the other vector and when the product of the two features is zero. The transpose approach only considers columns F1 and F4: the six dot products in column Y are obtained by merging the lists represented by columns F1 and F4. The total number of operations is only proportional to the number of (full) circles in these columns. Compare this with the traditional dot product algorithm that has to access every single circle. Note that while SDOT will not apply multiply-add to white circles, an access operation is still needed.



Transpose matrix vector product



Algorithm 3.3 Transpose matrix-vector multiplication

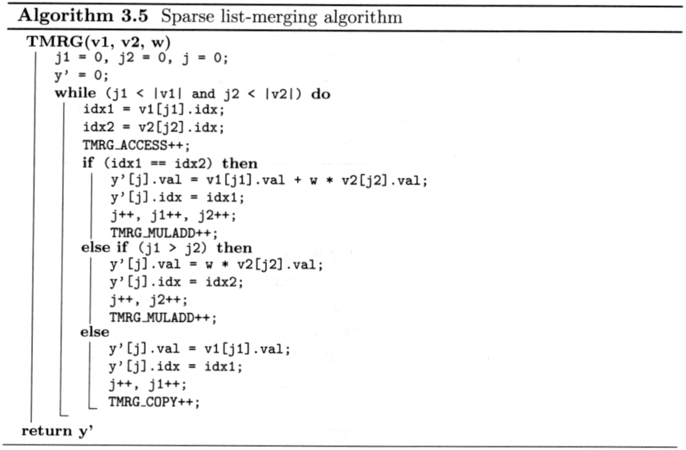
```
TsMxV(M, x)
    y = 0;
    for (i=0; i<|x|; i++) do
        y = TADD(y, M[.][x[i].idx], x[i].val);
return y

TADD(v1, v2, w)
    for (j=0; j<|v2|; j++) do
        v1[v2[j].idx] += w * v2[j].val;
        TADD_ACCESS++;
        TADD_MULADD++;
return v1
```

Algorithm 3.4 Dense vector to index-value table

```
VEC2LIST(y)
    y' = 0;
    pos = 0;
    for (i=0; i<N_r; i++) do
        VEC2LIST_ACCESS++;
        if (y[i] != 0) then
            y'[pos].idx = i;
            y'[pos].val = y[i];
            pos++;
return y'
```

Sparse list merging



Algorithm 3.5 Sparse list-merging algorithm

```
TMRG(v1, v2, w)
    j1 = 0, j2 = 0, j = 0;
    y' = 0;
    while (j1 < |v1| and j2 < |v2|) do
        idx1 = v1[j1].idx;
        idx2 = v2[j2].idx;
        TMRG_ACCESS++;
        if (idx1 == idx2) then
            y'[j].val = v1[j1].val + w * v2[j2].val;
            y'[j].idx = idx1;
            j++, j1++, j2++;
            TMRG_MULADD++;
        else if (j1 > j2) then
            y'[j].val = w * v2[j2].val;
            y'[j].idx = idx2;
            j++, j2++;
            TMRG_MULADD++;
        else
            y'[j].val = v1[j1].val;
            y'[j].idx = idx1;
            j++, j1++;
            TMRG_COPY++;
return y'
```

Training time on large data set

LSKM p0.41

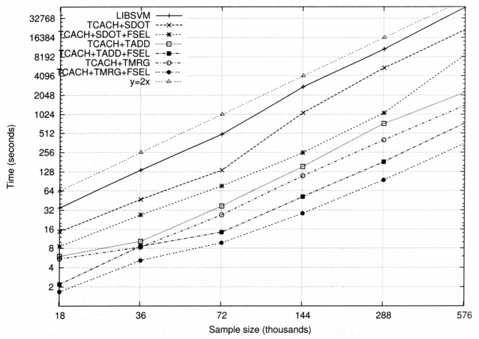


Figure 3.3 Evolution of learning time with the training set size on the VoiceTone data. TCACH indicates LLAMA with the transpose cache algorithm. The matrix multiplication in LLAMA is performed using the SDOT, TADD, or TMRG algorithms. FSEL indicates the optional feature selection algorithm.

VC dimension

Vapnik-Chervonenkis dimension

- a measure of the capacity
- of a statistical classification algorithm

- defined as the cardinality of the largest set of points
- that the algorithm can *shatter*

- Informally, the capacity of a classification model is related to how complicated it can be.
- higher capacity: can handle complicated situations,
- but might overfit

VC dimension: example

LWK p.20

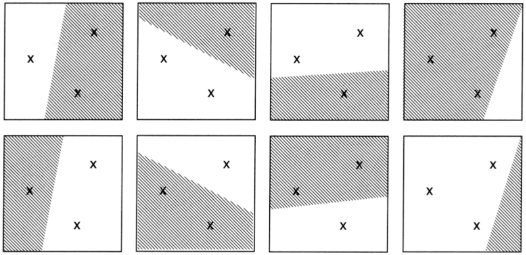


Figure 1.4 A simple VC dimension example. There are $2^3 = 8$ ways of assigning 3 points to two classes. For the displayed points in $\mathbb{R}^2$, all 8 possibilities can be realized using separating hyperplanes, in other words, the function class can shatter 3 points. This would not work if we were given 4 points, no matter how we placed them. Therefore, the VC dimension of the class of separating hyperplanes in $\mathbb{R}^2$ is 3.

VC dimension vs. test errors

LWK p.20

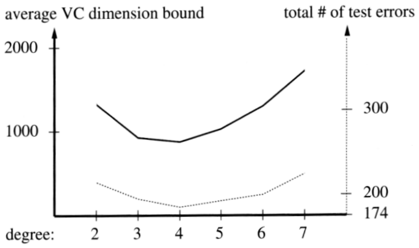
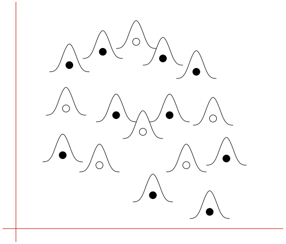


Figure 5.5 Average VC dimension (solid), and total number of test errors, of ten two-class-classifiers (dotted) with polynomial degrees 2 through 7, trained on the USPS set of handwritten digits. The baseline 174 on the error scale, corresponds to the total number of test errors of the ten *best* binary classifiers, chosen from degrees 2 through 7. The graph shows that for this problem, which can essentially be solved with zero training error for all degrees greater than 1, the VC dimension allows us to predict that degree 4 yields the best overall performance of the two-class-classifier on the test set (from [470, 467]).

VC dimension: Gaussian kernels



- SVM with Gaussian kernel can classify every input function
- if the Gaussian kernels are "narrow" enough
- Gaussian-kernel SVM has infinite VC dimension

Summary

- Support Vector Machine
  - maximum-margin linear classifier
  - concept of support vectors
  - soft-margin classifier
  - feature-maps and kernels to handle non-linearity
  - training via Quadratic Programming algorithms

- (multi-class) classification and clustering
- pattern and object recognition
- regression and function approximation
- algorithms and complexity estimates

- still an active research topic

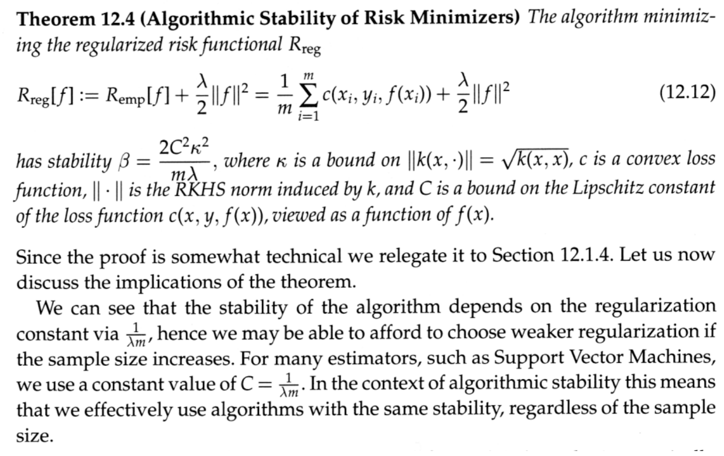
Thanks for your attention!

- Questions?

- bug-reports and feedback:
  hendrich@informatik.uni-hamburg.de
  zhang@informatik.uni-hamburg.de

A bit of fun: from Learning with Kernels

LWK p.363. . . the whole book is like this :-)



Theorem 12.4 (Algorithmic Stability of Risk Minimizers) *The algorithm minimizing the regularized risk functional* R_{reg}

$$R_{\text{reg}}[f] := R_{\text{emp}}[f] + \frac{\lambda}{2}\|f\|^2 = \frac{1}{m}\sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \frac{\lambda}{2}\|f\|^2 \qquad (12.12)$$

has stability $\beta = \frac{2C^2\kappa^2}{m\lambda}$, *where* κ *is a bound on* $\|k(x,\cdot)\| = \sqrt{k(x,x)}$, c *is a convex loss function,* $\|\cdot\|$ *is the RKHS norm induced by* k, *and* C *is a bound on the Lipschitz constant of the loss function* $c(x, y, f(x))$, *viewed as a function of* $f(x)$.

Since the proof is somewhat technical we relegate it to Section 12.1.4. Let us now discuss the implications of the theorem.

We can see that the stability of the algorithm depends on the regularization constant via $\frac{1}{\lambda m}$, hence we may be able to afford to choose weaker regularization if the sample size increases. For many estimators, such as Support Vector Machines, we use a constant value of $C = \frac{1}{\lambda m}$. In the context of algorithmic stability this means that we effectively use algorithms with the same stability, regardless of the sample size.

Software: libsvm

- C.-C. Chang & C.-J. Lin, *libsvm*
  http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  Bindings to C/C++, Java, . . .
- Alain Rakotomamonjy, Stephane Canu, *SVM and Kernel Methods Matlab Toolbox*, http://asi.insa-rouen.fr/enseignants/~arakotom/toolbox/index.html
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes – The Art of Scientific Computing*, Cambridge University Press, 2007 (all algorithms on CD-ROM)
- several other software packages (Matlab, C/C++, . . .)

Example datasets

- the libsvm page links to several training datasets
  http://www.csie.ntu.edu.tw/˜cjlin/libsvm/
- MNIST handwritten digits
  http://yann.lecun.com/exdb/mnist/
- NORB object recognition datasets
  http://www.cs.nyu.edu/˜ylclab/data/norb-v1.0/