A Simple Introduction to Support Vector Machines

Adapted from various authors

by Mario Martin

Outline

- Large-margin linear classifier
  - Linear separable
  - Nonlinear separable
- Creating nonlinear classifiers: kernel trick
- Transduction
- Discussion on SVM
- Conclusion

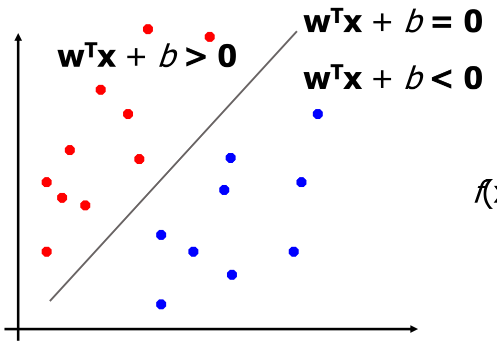
History of SVM

- SVM is related to statistical learning theory [3]
- Introduced by Vapnik
- SVM was first introduced in 1992
- SVM becomes popular because of its success a lot of classification problems

SVM: Large-margin linear classifier

Perceptron Revisited: Linear Separators

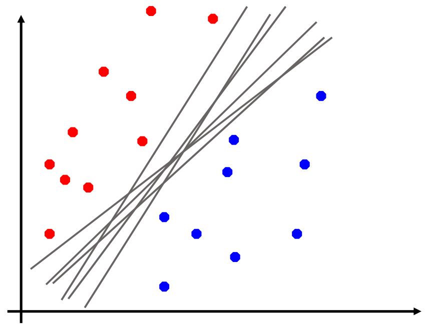
- Binary classification can be viewed as the task of separating classes in feature space:



$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

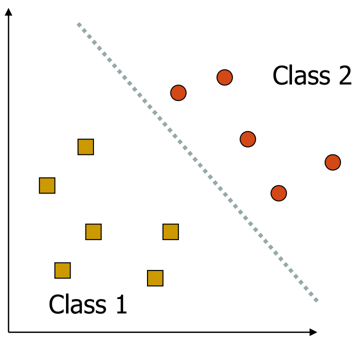
Linear Separators

- Which of the linear separators is optimal?

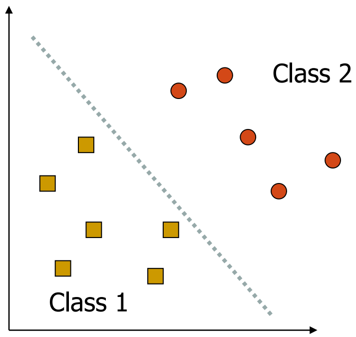


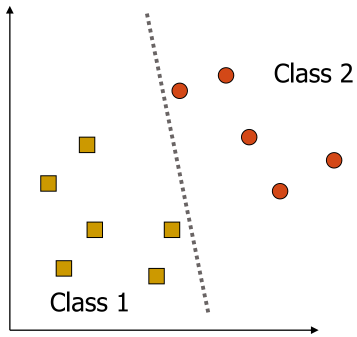
What is a good Decision Boundary?

- Consider a two-class, linearly separable classification problem
- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
  - Other different algorithms have been proposed
  - Are all decision boundaries equally good?



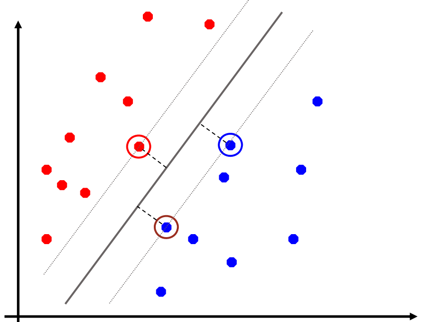
Examples of Bad Decision Boundaries





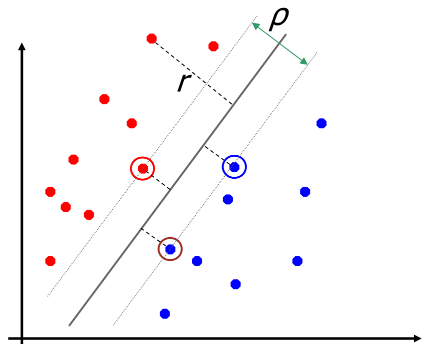
Maximum Margin Classification

- Maximizing the distance to examples is good according to intuition and PAC theory.
- Implies that only few vectors matter; other training examples are ignorable.



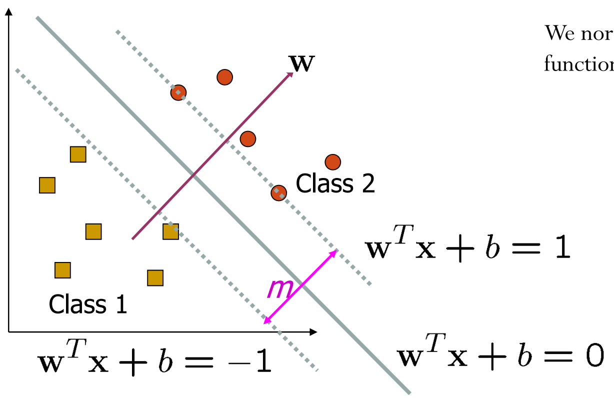
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible: We should maximize the margin, m



We normalize equations so function in supports is 1/-1.

$$r = \frac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$$

$$m = \frac{2}{||\mathbf{w}||}$$

Finding the Decision Boundary

- Let $\{x_1, \ldots, x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow$

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- Maximizing margin classifying all points correctly constraints is defined as follows:

Finding the Decision Boundary

- Primal formulation

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

- We can solve this problem using this formulation, or using the dual formulation…

[Recap of Constrained Optimization]

- Suppose we want to: minimize $f(\mathbf{x})$ subject to $g(\mathbf{x}) = 0$
- A necessary condition for $\mathbf{x}_0$ to be a solution:

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\left(f(\mathbf{x}) + \alpha g(\mathbf{x})\right)\Big|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{0} \\ g(\mathbf{x}) = 0 \end{cases}$$

- α: the Lagrange multiplier
- For multiple constraints $g_i(\mathbf{x}) = 0$, i=1, …, m, we need a Lagrange multiplier α_i for each of the constraints

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\left(f(\mathbf{x}) + \sum_{i=1}^n \alpha_i g_i(\mathbf{x})\right)\Big|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{0} \\ g_i(\mathbf{x}) = 0 \qquad \text{for } i = 1, \dots, m \end{cases}$$

[Recap of Constrained Optimization]

- The case for inequality constraint $g_i(\mathbf{x}) \leq 0$ is similar, except that the Lagrange multiplier α_i should be positive
- If $\mathbf{x}_0$ is a solution to the constrained optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \leq 0 \quad \text{for } i = 1, \dots, m$$

- There must exist $\alpha_i \geq 0$ for i=1, …, m such that $\mathbf{x}_0$ satisfy

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\left(f(\mathbf{x}) + \sum_i \alpha_i g_i(\mathbf{x})\right)\Big|_{\mathbf{x}=jx_0} = 0 \\ g_i(\mathbf{x}) \leq 0 \qquad \text{for } i = 1, \dots, m \end{cases}$$

- The function $f(\mathbf{x}) + \sum_i \alpha_i g_i(\mathbf{x})$ is also known as the Lagrangrian. We want to set its gradient to $\mathbf{0}$

Back to the Original Problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to } 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0 \qquad \text{for } i = 1, \dots, n$$

- The Lagrangian is

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^n \alpha_i \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right)$$

  - Note that $||\mathbf{w}||^2 = \mathbf{w}^T\mathbf{w}$
- Setting the gradient of $\mathcal{L}$ w.r.t. $\mathbf{w}$ and b to zero, we have

$$\mathbf{w} + \sum_{i=1}^n \alpha_i(-y_i)\mathbf{x}_i = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$
$$\sum_{i=1}^n \alpha_i y_i = 0$$

The Dual Formulation

- If we substitute $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ to $\mathcal{L}$, we have

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b)\right)$$

$$= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \alpha_i y_i \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i - b\sum_{i=1}^{n} \alpha_i y_i$$

$$= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i$$

- Remember that $\sum_{i=1}^{n} \alpha_i y_i = 0$
- **This is a function of α_i only**

The Dual formulation

- It is known as the dual problem (the original problem is known as the primal problem): if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The objective function of the dual problem needs to be maximized!
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

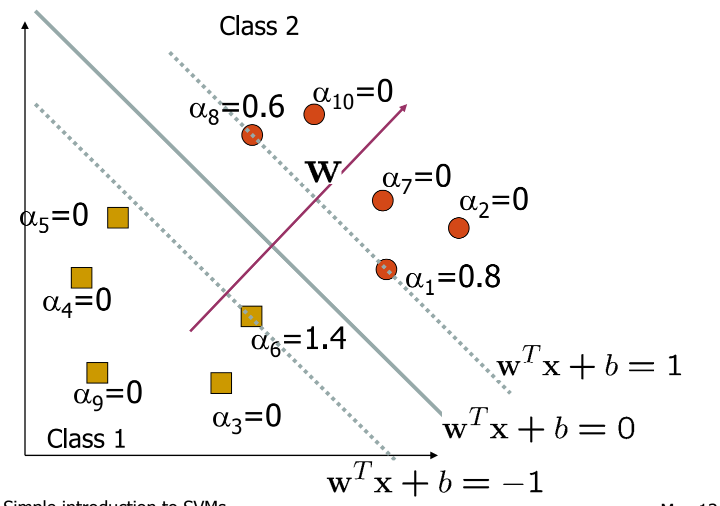
The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
- A global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

A Geometrical Interpretation



Characteristics of the Solution

- Many of the α_i are zero
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

Characteristics of the Solution

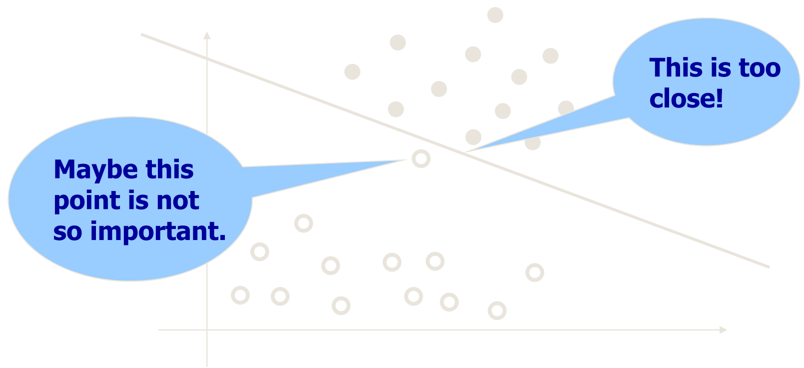
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$
  - classify **z** as class 1 if the sum is positive, and class 2 otherwise
- Note: **w** need not be formed explicitly

The Quadratic Programming Problem

- Many approaches have been proposed
  - Loqo, cplex, etc. (see http://www.numerical.rl.ac.uk/qp/qp.html)
- Most are "interior-point" methods
  - Start with an initial solution that can violate the constraints
  - Improve this solution by optimizing the objective function and/or reducing the amount of constraint violation
- For SVM, sequential minimal optimization (SMO) seems to be the most popular
  - A QP with two variables is trivial to solve
  - Each iteration of SMO picks a pair of (α_i,α_j) and solve the QP with these two variables; repeat until convergence
- In practice, we can just regard the QP solver as a "black-box" without bothering how it works

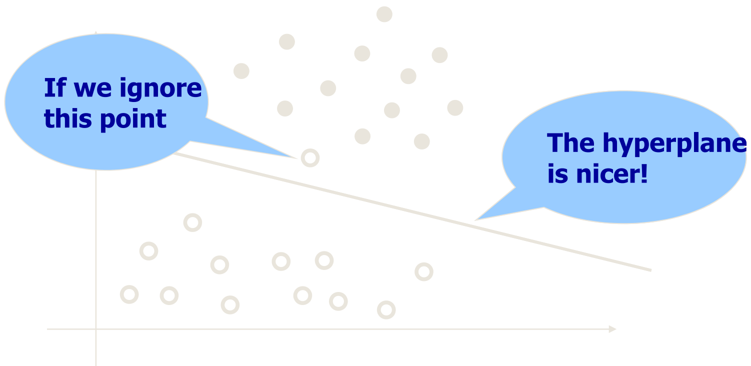
Non-Separable Sets

- Sometimes, we **do not** want to separate perfectly.



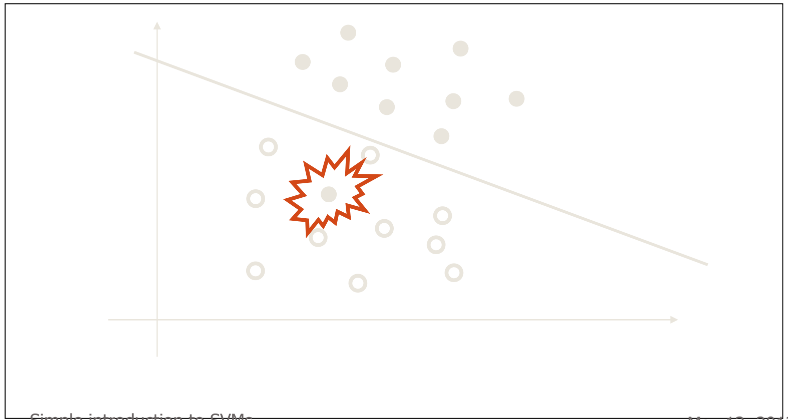
Non-Separable Sets

- Sometimes, we **do not** want to separate perfectly.



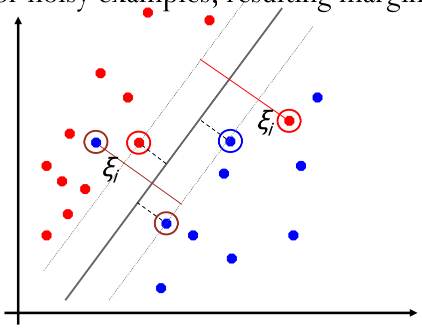
Non-Separable Sets

- Sometimes, data sets are not linearly separable.



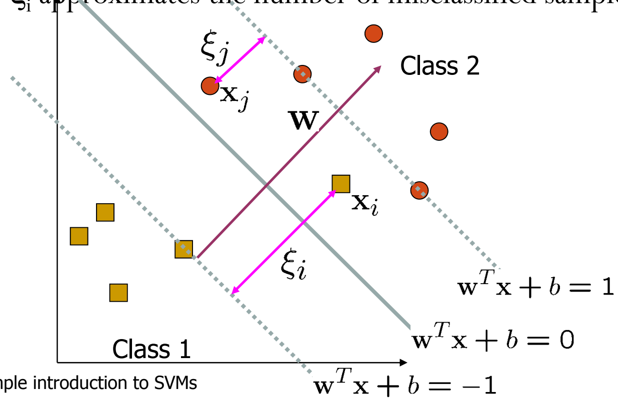
Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Non-linearly Separable Problems

- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $\mathbf{w}^T\mathbf{x}+b$
- ξ_i approximates the number of misclassified samples



Soft Margin Hyperplane

- If we minimize $\sum_i \xi_i$, ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

  - ξ_i are "slack variables" in optimization
  - Note that ξ_i=0 if there is no error for $\mathbf{x}_i$
  - Number of slacks + supports is an upper bound of the number of errors (Leave one out error)

Soft Margin Hyperplane

- We want to minimize

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

  - C : tradeoff parameter between error and margin

- The optimization problem becomes

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Optimization Problem

- The dual of this new constrained optimization problem is

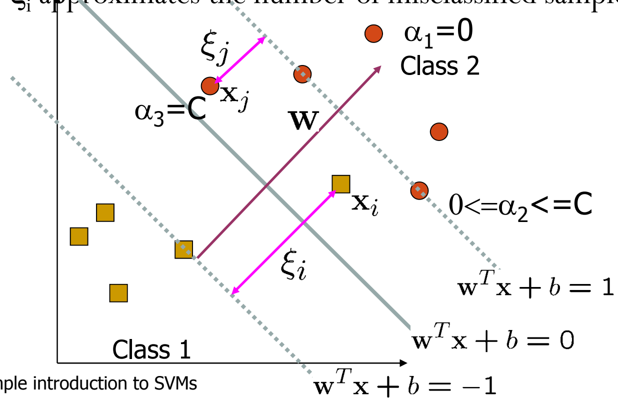
$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

- $\mathbf{w}$ is recovered as: $\mathbf{w} = \sum_{j=1}^{s}\alpha_{t_j} y_{t_j}\mathbf{x}_{t_j}$
- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound C on α_i now
- Once again, a QP solver can be used to find α_i

Non-linearly Separable Problems

- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $\mathbf{w}^T\mathbf{x}+b$
- ξ_i approximates the number of misclassified samples

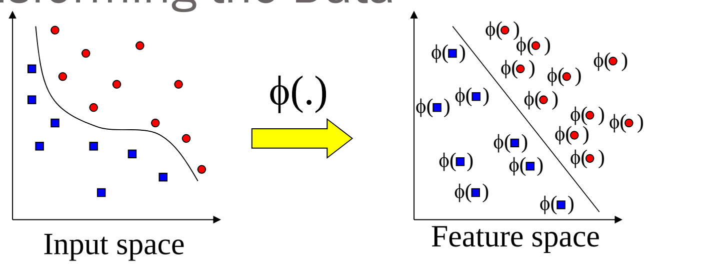


SVM with KERNELS: Large-margin NON-linear classifiers

Extension to Non-linear Decision Boundary

- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space the point $\mathbf{x}_i$ are located
  - Feature space: the space of $\phi(\mathbf{x}_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Classification can become easier with a proper transformation. In the XOR problem, for example, adding a new feature of x_1x_2 make the problem linearly separable

Transforming the Data

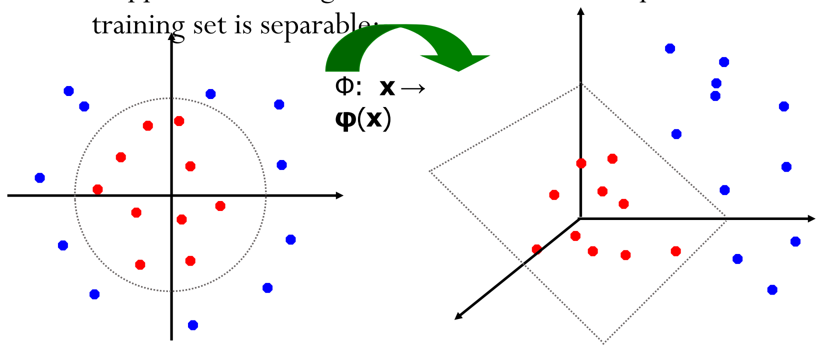


Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



The Kernel Trick

- Recall the SVM optimization problem

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \ge \alpha_i \ge 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function K by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

SVMs with kernels

- Training

$$\text{maximize}_{\alpha} \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to } \sum_{i=1}^{l} \alpha_i \cdot y_i = 0 \quad \text{and} \quad \forall i \ C \ge \alpha_i \ge 0$$

- Classification of **x**:

$$h(\mathbf{x}) = sign\left(\sum_{i=1}^{l} \alpha_i \cdot y_i \cdot K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

An Example for φ(.) and K(.,.)

- Suppose φ(.) is given as follows

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\left\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right)\right\rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out φ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out φ(.) explicitly is known as the kernel trick

Kernel Functions

- Kernel (Gram) matrix:

$$\begin{pmatrix} K(\mathbf{x}_1,\mathbf{x}_1) & K(\mathbf{x}_1,\mathbf{x}_2) & K(\mathbf{x}_1,\mathbf{x}_3) & \cdots & K(\mathbf{x}_1,\mathbf{x}_l) \\ K(\mathbf{x}_2,\mathbf{x}_1) & K(\mathbf{x}_2,\mathbf{x}_2) & K(\mathbf{x}_2,\mathbf{x}_3) & & K(\mathbf{x}_2,\mathbf{x}_l) \\ \cdots & & & \cdots & \\ \cdots & & & \cdots & \\ K(\mathbf{x}_l,\mathbf{x}_1) & K(\mathbf{x}_l,\mathbf{x}_2) & K(\mathbf{x}_l,\mathbf{x}_3) & \cdots & K(\mathbf{x}_l,\mathbf{x}_l) \end{pmatrix}$$

Matrix obtained from product:

$$K = \phi'\phi$$

Kernel Functions

- Any function $K(\mathbf{x},\mathbf{z})$ that creates a symmetric, positive definite matrix $K_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$ is a valid kernel (an inner product in some space)

- Why? Because any sdp matrix M can be decomposed as

  N'N = M

  so N can be seen as the projection to the feature space

Kernel Functions

- Another view: kernel function, being an inner product, is really a similarity measure between the objects
- Not all similarity measures are allowed – they must Mercer conditions
- Any distance measure can be translated to a kernel

Examples of Kernel Functions

- Polynomial kernel with degree d

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y}+1)^d$$

- Radial basis function kernel with width σ

$$K(\mathbf{x},\mathbf{y}) = \exp(-||\mathbf{x}-\mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional
- Sigmoid with parameter κ and θ

$$K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y}+\theta)$$

  - It does not satisfy the Mercer condition on all κ and θ

Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

With kernel function

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{n}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n}\alpha_i y_i = 0$$

Modification Due to Kernel Function

- For testing, the new data **z** is classified as class 1 if $f \geq 0$, and as class 2 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

With kernel function

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

More on Kernel Functions

- Since the training of SVM only requires the value of $K(\mathbf{x}_i, \mathbf{x}_j)$, there is no restriction of the form of $\mathbf{x}_i$ and $\mathbf{x}_j$
  - $\mathbf{x}_i$ can be a sequence or a tree, instead of a feature vector
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is just a similarity measure comparing $\mathbf{x}_i$ and $\mathbf{x}_j$
- For a test object **z**, the discriminant function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

$\mathcal{S}$: the set of support vectors

More on Kernel Functions

- Not all similarity measure can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer function, i.e., the function is "positive-definite"
  - This implies that the *n* by *n* kernel matrix, in which the (i,j)-th entry is the $K(\mathbf{x}_i, \mathbf{x}_j)$, is always positive definite
  - This also means that the QP is convex and can be solved in polynomial time

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarizes all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, …)
- There is even research to estimate the kernel matrix from available information

- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try
- Note that SVM with RBF kernel is closely related to RBF neural networks, with the centers of the radial basis functions automatically chosen for SVM

Other Aspects of SVM

- How to use SVM for multi-class classification?
  - One can change the QP formulation to become multi-class
  - More often, multiple binary classifiers are combined
  - One can train multiple one-versus-all classifiers, or combine multiple pairwise classifiers "intelligently"
- How to interpret the SVM discriminant function value as probability?
  - By performing logistic regression on the SVM output of a set of data (validation set) that is not used for training
- Some SVM software (like libsvm) have these features built-in

Software

- A list of SVM implementation can be found at http://www.kernel-machines.org/software.html
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Classification

- Prepare the pattern matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Strengths and Weaknesses of SVM

- Strengths
  - Training is relatively easy
    - No local optimal, unlike in neural networks
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
  - Need to choose a "good" kernel function.

Other Types of Kernel Methods

- A lesson learnt in SVM: a linear algorithm in the feature space is equivalent to a non-linear algorithm in the input space
- Standard linear algorithms can be generalized to its non-linear version by going to the feature space
  - Kernel principal component analysis, kernel independent component analysis, kernel canonical correlation analysis, kernel k-means, 1-class SVM are some examples

Conclusion

- SVM is a useful alternative to neural networks
- Two key concepts of SVM: maximize the margin and the kernel trick
- Many SVM implementations are available on the web for you to try on your data set!

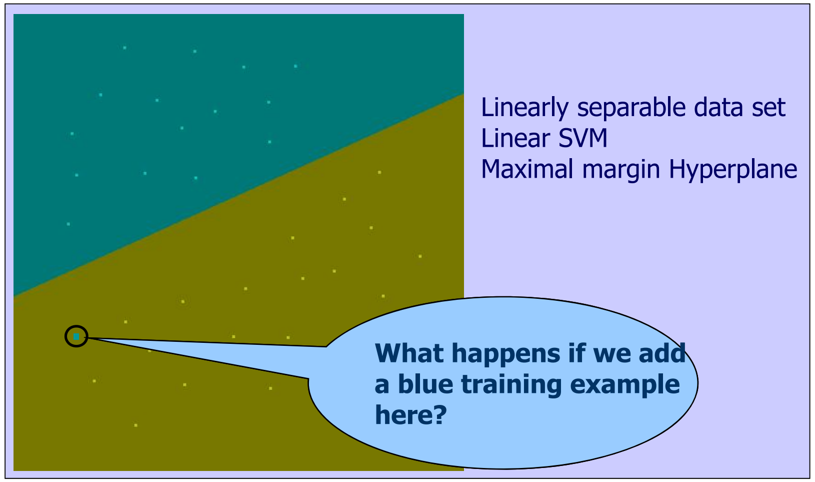
Toy Examples

- All examples have been run with the 2D graphic interface of SVMLIB (Chang and Lin, National University of Taiwan)

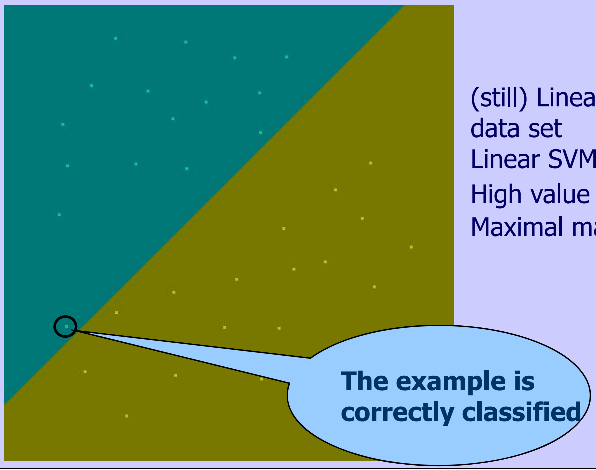
  "**LIBSVM** is an integrated software for support vector classification, (C-SVC, nu-SVC), regression (epsilon-SVR, un-SVR) and distribution estimation (one-class SVM). It supports multi-class classification. The basic algorithm is a simplification of both SMO by Platt and SVMLight by Joachims. It is also a simplification of the modification 2 of SMO by Keerthy et al. Our goal is to help users from other fields to easily use SVM as a tool. **LIBSVM** provides a simple interface where users can easily link it with their own programs…"

- Available from: www.csie.ntu.edu.tw/~cjlin/libsvm (it icludes a Web integrated demo tool)

Toy Examples (I)

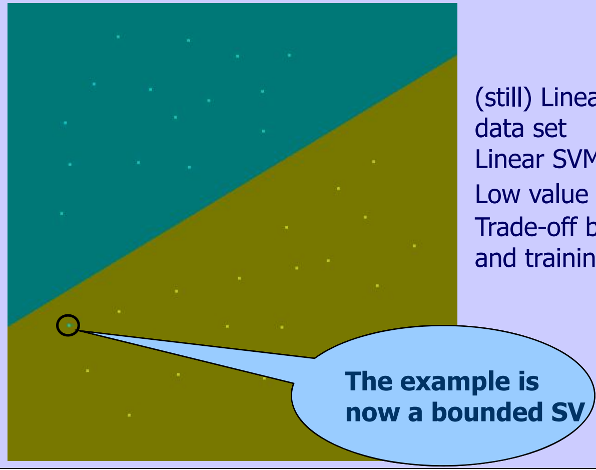


Toy Examples (I)



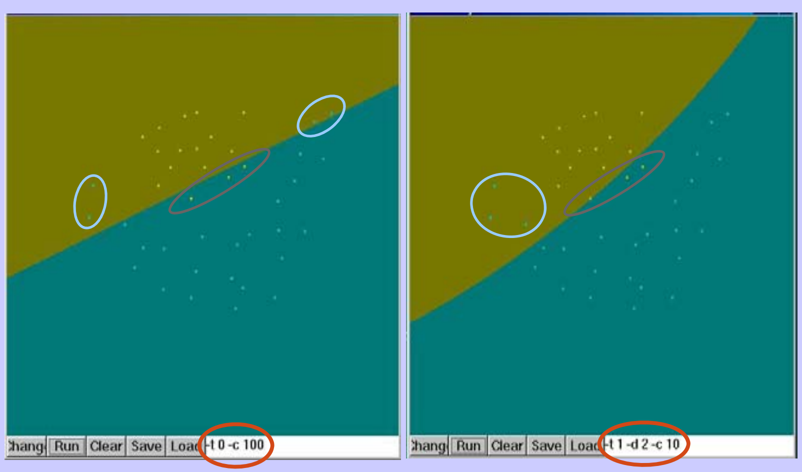
(still) Linearly separable data set
Linear SVM
High value of C parameter
Maximal margin Hyperplane

Toy Examples (I)

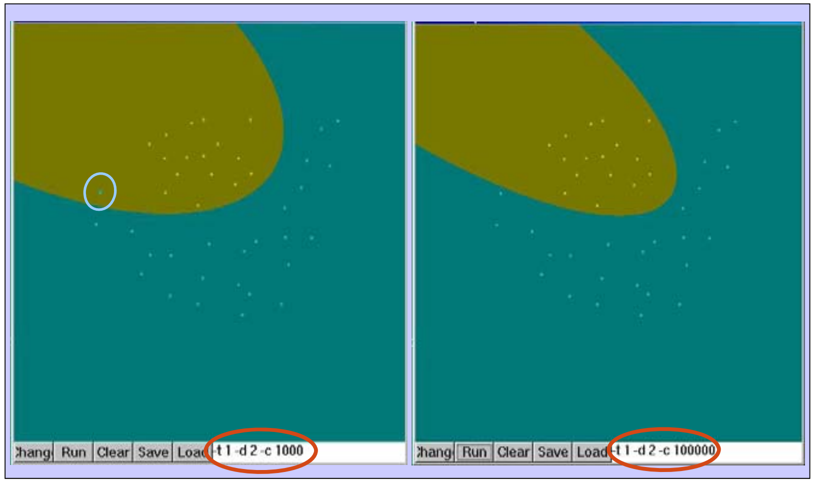


(still) Linearly separable data set
Linear SVM
Low value of C parameter
Trade-off between: margin and training error

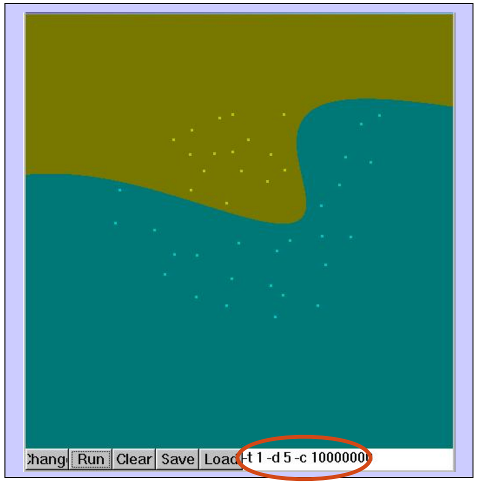
Toy Examples (I)



Toy Examples (I)



Toy Examples (I)



Examples

Toy Examples (I)

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html

Transduction with SVMs

The learning problem

- Transduction:

  We consider a phenomenon f that maps inputs (instances) x to outputs (labels) y = f(x) (y {−1, 1})
  - Given a set of labeled examples {(xi, yi) : i = 1, …, n},
  - and a set of unlabeled examples x'1, …, x'm
  - the goal is to find the labels y'1 , …, y'm
- No need to construct a function f, the output of the transduction algorithm is a vector of labels.

Transduction based on margin size

- Binary classification, linear parameterization, joint set of (training + working) samples
- **Two objectives of transductive learning:**

  (TL1) separate labeled training data using a large-margin hyperplane (as in standard inductive SVM)

  (TL2) separating (explain) working data set using a large-margin hyperplane.

Transductive SVMs

- **Transductive** instead of inductive (Vapnik 98)
- TSVMs take into account a particular test set and try to minimize misclassifications of just those particular examples
- Formal setting:

$S_{train} = \{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$

$S_{test} = \{\mathbf{x}^*_1,\ \mathbf{x}^*_2,\ \ldots,\ \mathbf{x}^*_k\}$ (normally $k >> n$)

Goal of the transductive learner L:

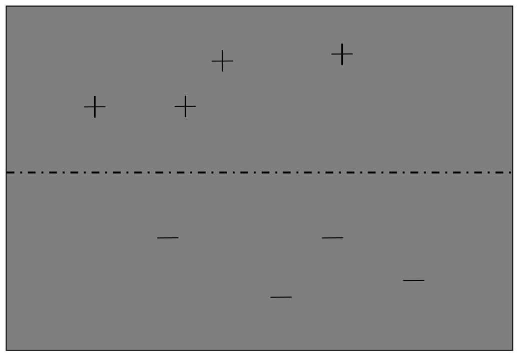
find a function $h_L = L(S_{train}, S_{test})$ so that the expected number of erroneous predictions on the test examples is minimized

(Joachims, 1999)

TextCat

Transductive SVMs

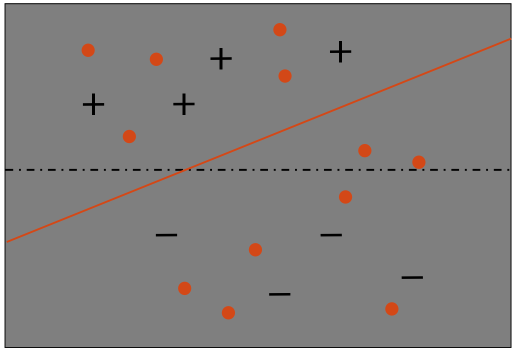
How does the transductive approach work?



(Joachims, 1999)

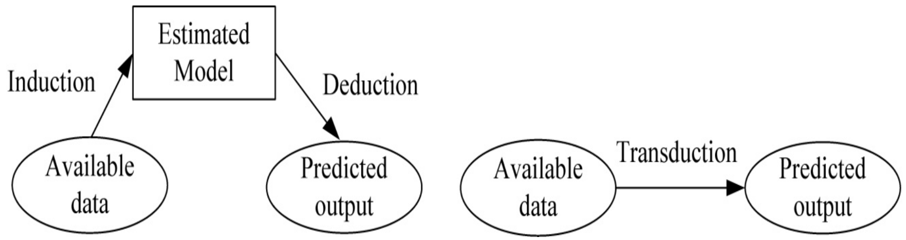
Transductive SVMs

How does the transductive approach work?



(Joachims, 1999)

Induction vs Transduction



Optimization formulation for SVM transduction

- Given: joint set of (training + working) samples
- Denote slack variables ξ_i for training, ξ_j^* for working
- Minimize $R(\mathbf{w},b)=\frac{1}{2}(\mathbf{w}\cdot\mathbf{w})+C\sum_{i=1}^{n}\xi_i+C^*\sum_{j=1}^{m}\xi_j^*$

  subject to
  $$\begin{cases} y_i[(\mathbf{w}\cdot\mathbf{x}_i)+b]\ge 1-\xi_i \\ y_j^*[(\mathbf{w}\cdot\mathbf{x}_j)+b]\ge 1-\xi_j^* \\ \xi_i,\xi_j^*\ge 0, i=1,\dots,n,\ j=1,\dots,m \end{cases}$$

  where $y_j^*=sign(\mathbf{w}\cdot\mathbf{x}_j+b),\ j=1,\dots,m$

  → Solution (~ decision boundary) $D(\mathbf{x})=(\mathbf{w}^*\cdot\mathbf{x})+b^*$
- Unbalanced situation (small training/ large test)

  → all unlabeled samples assigned to one class
- Additional constraint: $\frac{1}{n}\sum_{i=1}^{n}y_i=\frac{1}{m}\sum_{j=1}^{m}[(\mathbf{w}\cdot\mathbf{x}_i)+b]$

Optimization formulation (cont'd)

- Hyperparameters C and C^* control the trade-off between explanation and margin size
- Soft-margin inductive SVM is a special case of soft-margin transduction with zero slacks $\xi_j^*=0$
- Dual + kernel version of SVM transduction
- Transductive SVM optimization is **not convex**

  (~ non-convexity of the loss for unlabeled data) –

  → different opt. heuristics ~ different solutions
- Exact solution (via exhaustive search) possible for small number of test samples (m)

Many applications for transduction

- Text categorization: classify word documents into a number of predetermined categories
- Email classification: Spam vs non-spam
- Web page classification
- Image database classification
- All these applications:
  - high-dimensional data
  - small labeled training set (human-labeled)
  - large unlabeled test set

Example application

- Prediction of molecular bioactivity for drug discovery
- Training data~1,909; test~634 samples
- Input space ~ 139,351-dimensional
- Prediction accuracy:

SVM induction ~74.5%; transduction ~ 82.3%

Ref: J. Weston et al, KDD cup 2001 data analysis: prediction of molecular bioactivity for drug design – binding to thrombin, *Bioinformatics 2003*