PhD in Computer Science and Engineering
Bologna, April 2016

Machine Learning

Marco Lippi

marco.lippi3@unibo.it

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

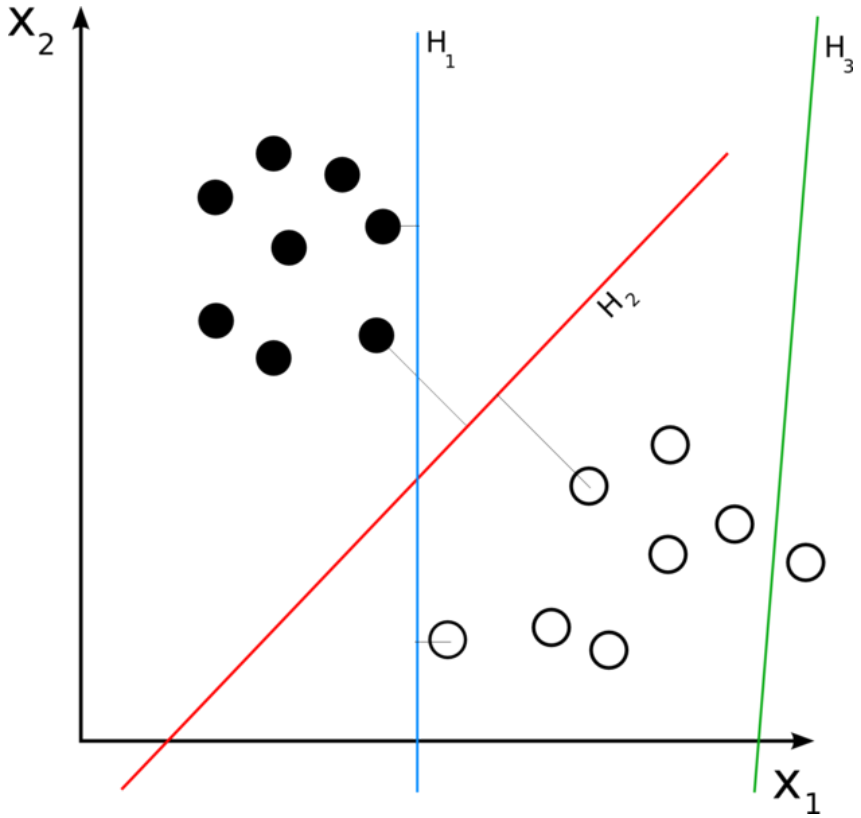
Kernel machines

History

- First algorithm on **Support Vector Machines** in 1963 (Vapnik & Chervonenkis)
- Extension to **non-linear classifiers** (Kernels) in 1992 (Boser, Guyon & Vapnik)
- Current formalization in 1995 (Cortes & Vapnik)
- Extremely popular at the end of the 90s...
  ...Now we have deep learning, yet...

Maximum separating hyperplane

A common binary classification problem

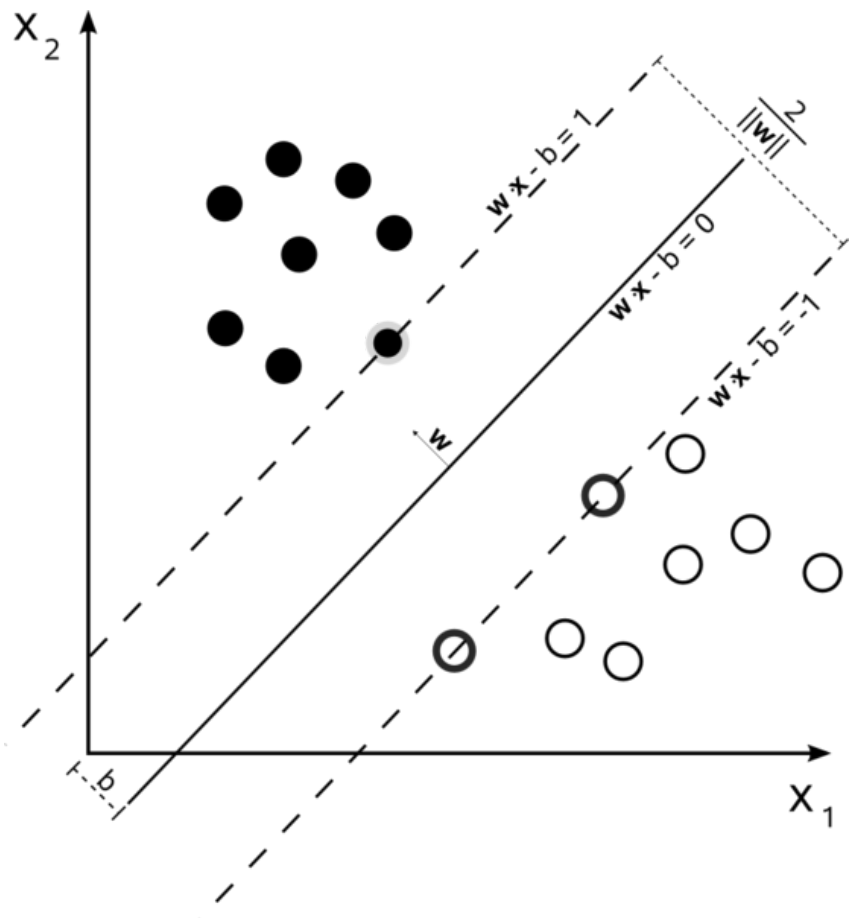


[Figure from Wikipedia]

Which hypothesis is to be preferred ?

Maximum separating hyperplane

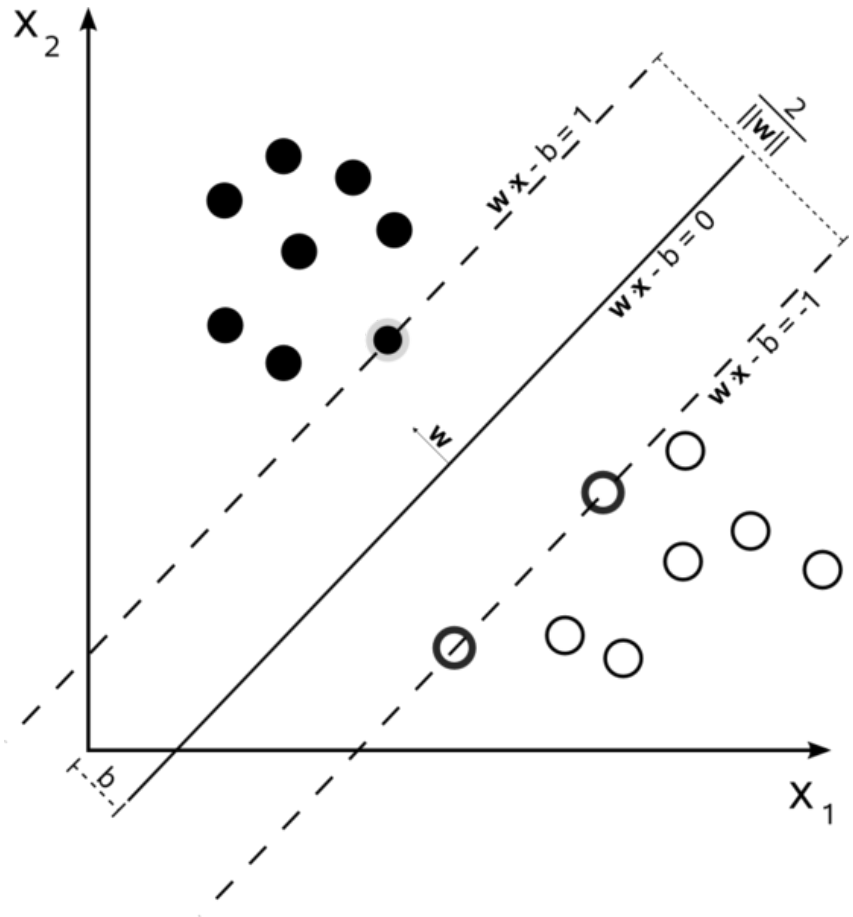
We aim at finding the hyperplane with **maximum margin**



[Figure from Wikipedia]

Maximum separating hyperplane

The solution depends only on a (small ?) **subset** of the examples



[Figure from Wikipedia]

Hard margin SVM

Learning problem:

$$\min_{w} \frac{1}{2}\|w\|^2$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 \quad \forall (x_i, y_i) \in \mathcal{D}$$

The constraints guarantee correct classification of points in $\mathcal{D}$

It is a **quadratic** optimization problem ☺

To include constraints we employ the **Lagrangian**:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i (y_i (w^T x_i + b) - 1)$$

The solution is to **minimize** wrt w and **maximize** wrt α_i

Vanishing derivatives wrt primal variables w, b:

$$\frac{\partial}{\partial b} L(w, b, \alpha) = 0 \quad \Rightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\frac{\partial}{\partial w} L(w, b, \alpha) = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i = 0$$

By substituting back in the Lagrangian we get the **dual problem**:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

which is to be maximized wrt **dual variables** α

Dual formulation:

$$\max_{\alpha \in \Re^N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$s.t. \qquad \alpha_i \geq 0 \qquad \forall i = 1, \ldots, m$$

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

Hard margin SVM

Primal vs. dual problem:

- **primal** problem has $d + 1$ variables (# features + 1)
- **dual** problem has N variables (# examples)
- we can solve either
- the constraints of the dual are **easier** (box constraints)

We know (from the derivatives) that $w = \sum_{i=1}^{N} \alpha_i y_i x_i$

The **decision function** then becomes:

$$f(z) = w^T z + b = \sum_{i=1}^{N} \alpha_i y_i x_i^T z + b$$

It is a linear combination of **dot products** with training examples !

Karush-Kuhn-Tacker conditions

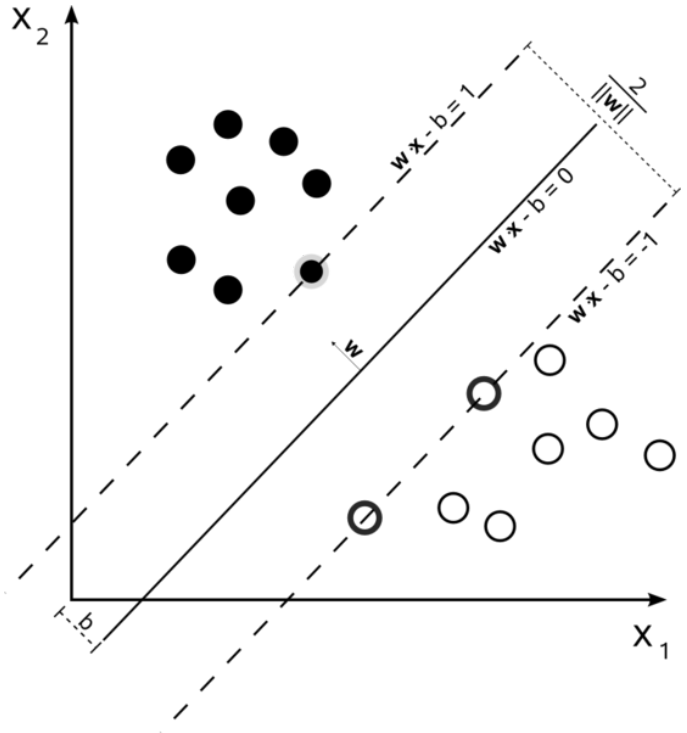
The solution of the optimization problem is such that:

$$\alpha_i(y_i(w^T x_i + b) - 1) = 0$$

- **either** a point has associated $\alpha_i = 0$
- **or** it stays on the boundary: $y_i(w^T x_i + b) = 1$

Maximum separating hyperplane

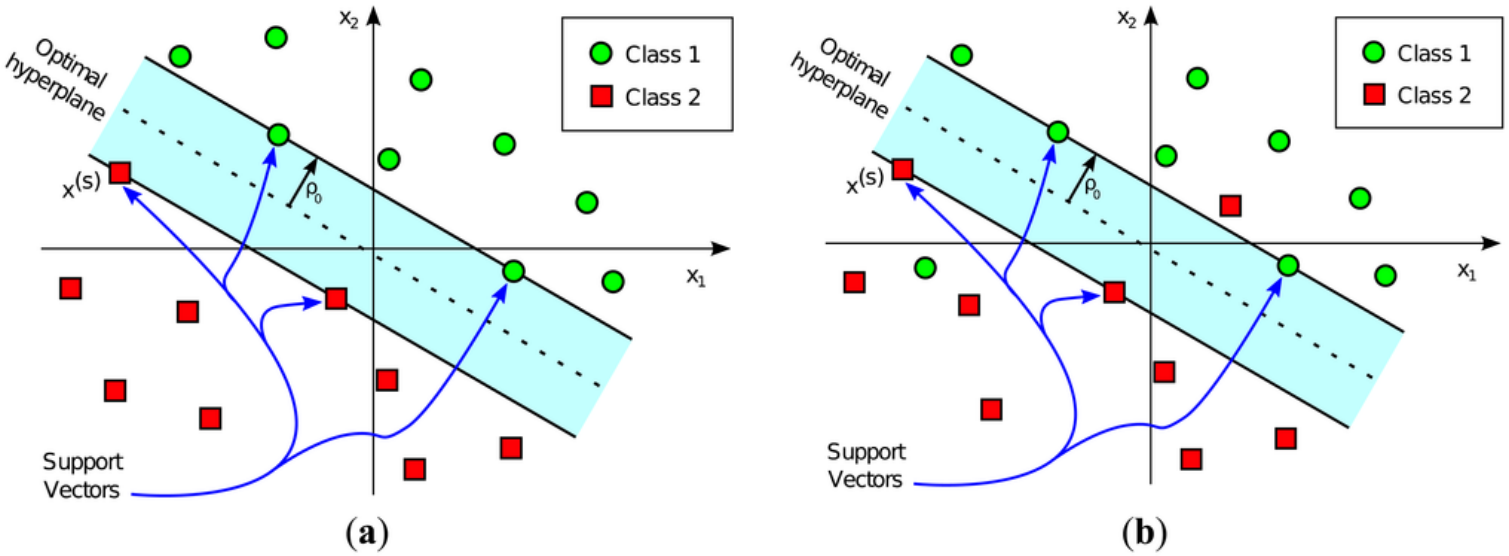
Such points are named **support vectors** !
The other points **do not contribute** to the solution !



[Figure from Wikipedia]

Soft margin SVM

In most cases, **data will not be perfectly separable**



[Figure from Ruiz-Gonzalez et al., 2014]

Soft margin SVM

Learning problem:

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

$$s.t. \qquad y_i(w^T x + b) \geq 1 - \xi_i \qquad \forall (x_i, y_i) \in \mathcal{D}$$

$$\xi_i \geq 0 \qquad \forall i = 1, \ldots, N$$

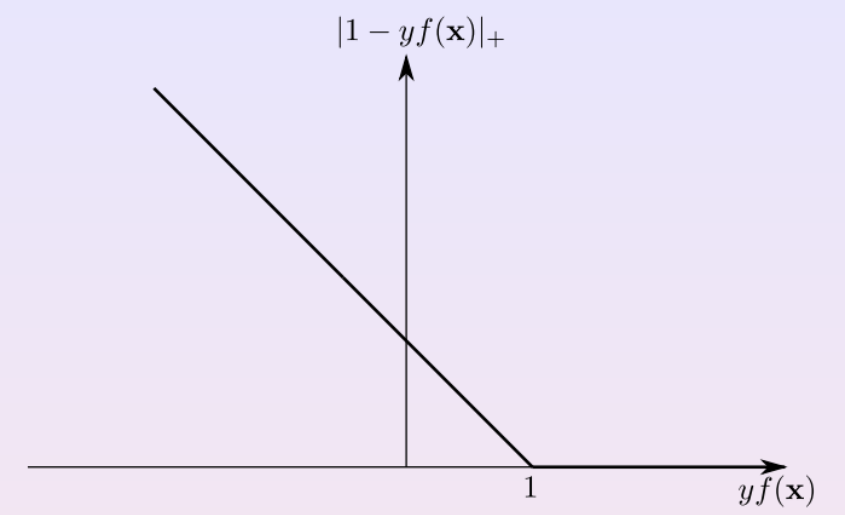
Soft margin SVM

$$\min_w \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \ell(y_i, f(x_i))$$

The second term is about **error minimization** (loss function)
The first term is about **regularization** (smooth solutions)

$$\ell(y_i, f(x_i)) = |1 - y_i f(x_i)|_+$$

Hinge loss



[Figure by A. Passerini]

$$\ell(y_i, f(x_i)) = |1 - y_i f(x_i)|_+$$

$$|z|_+ = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

The examples not violating the margin constraint have **zero loss**

Soft margin SVM

Again we can exploit the Lagrangian approach:

$$L(w, b, \alpha, \beta) = \frac{1}{2} - \sum_{i=1}^{N} \alpha_i (y_i (w^T x_i + b) - 1) - \sum_{i=1}^{N} \beta_i \xi_i$$

from which we obtain the following dual formulation:

$$\max_{\alpha \in \Re^N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j {x_i}^T x_j$$

$$s.t. \qquad 0 \leq \alpha_i \leq C \qquad \forall i = 1, \ldots, m$$

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

Soft margin SVM

This time the support vectors are those examples for which:

$$y_i(w^T x_i + b) \leq 1$$

- if $\alpha_i < C$ we have $\xi_i = 0$ (**unbound SV**)
- if $\alpha_i = C$ we have $\xi_i > 0$ (**bound SV**) $\rightarrow$ margin errors

Primal vs. dual problem **at prediction time**:

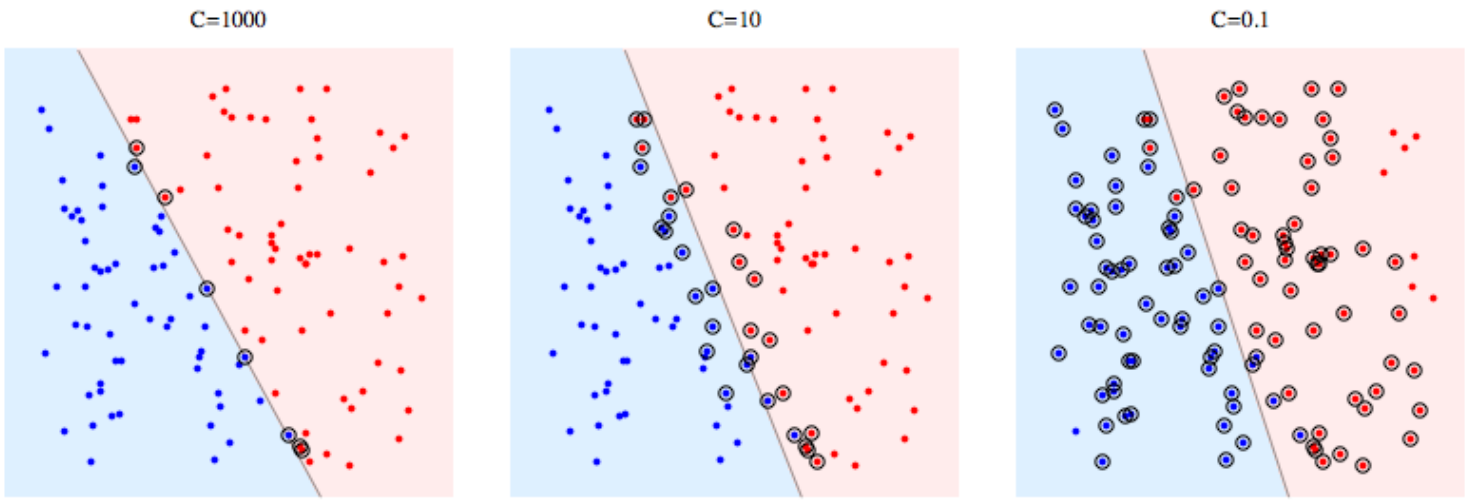
- the solution of the dual problem depends on the **number of support vectors** $\rightarrow$ it could be quite costly !
- the solution of the primal problem is **independent** of the number of support vectors

Regularization

The effect of the C parameter is **crucial**:

- large $C \rightarrow$ avoid mis-classifying training examples $\rightarrow$
  $\rightarrow$ narrow margin $\rightarrow$ overfitting (?)
- small $C \rightarrow$ penalize non-smooth functions $\rightarrow$
  $\rightarrow$ larger error on training set $\rightarrow$ generalization (?)

Regularization



[Figure from Stack Overflow]

Multi-class Support Vector Machines

We have considered a binary classification problem.
What if we have to deal with **M classes** ?

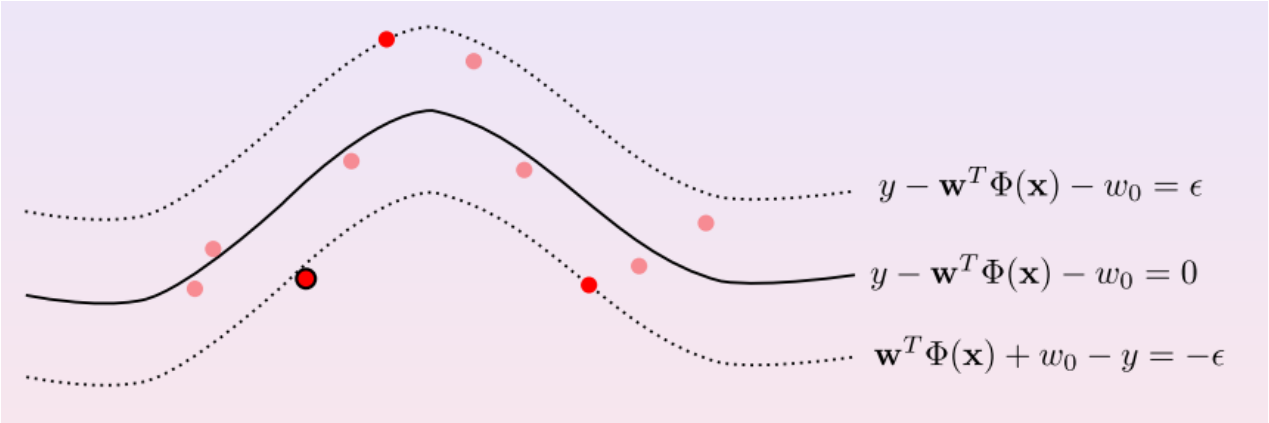
Typically reduced to many binary classification problems:

- **one-vs-all**: build M binary classifiers $\rightarrow$ in turn each class is the positive class and **all the other classes** are the negative class $\rightarrow$ the winner is the class with **the largest score**
- **one-vs-one**: build $M(M-1)/2$ binary classifiers for each pair of classes $\rightarrow$ count **the number of wins**

Support Vector Regression

Also regression problems can be easily defined in the SVM setting

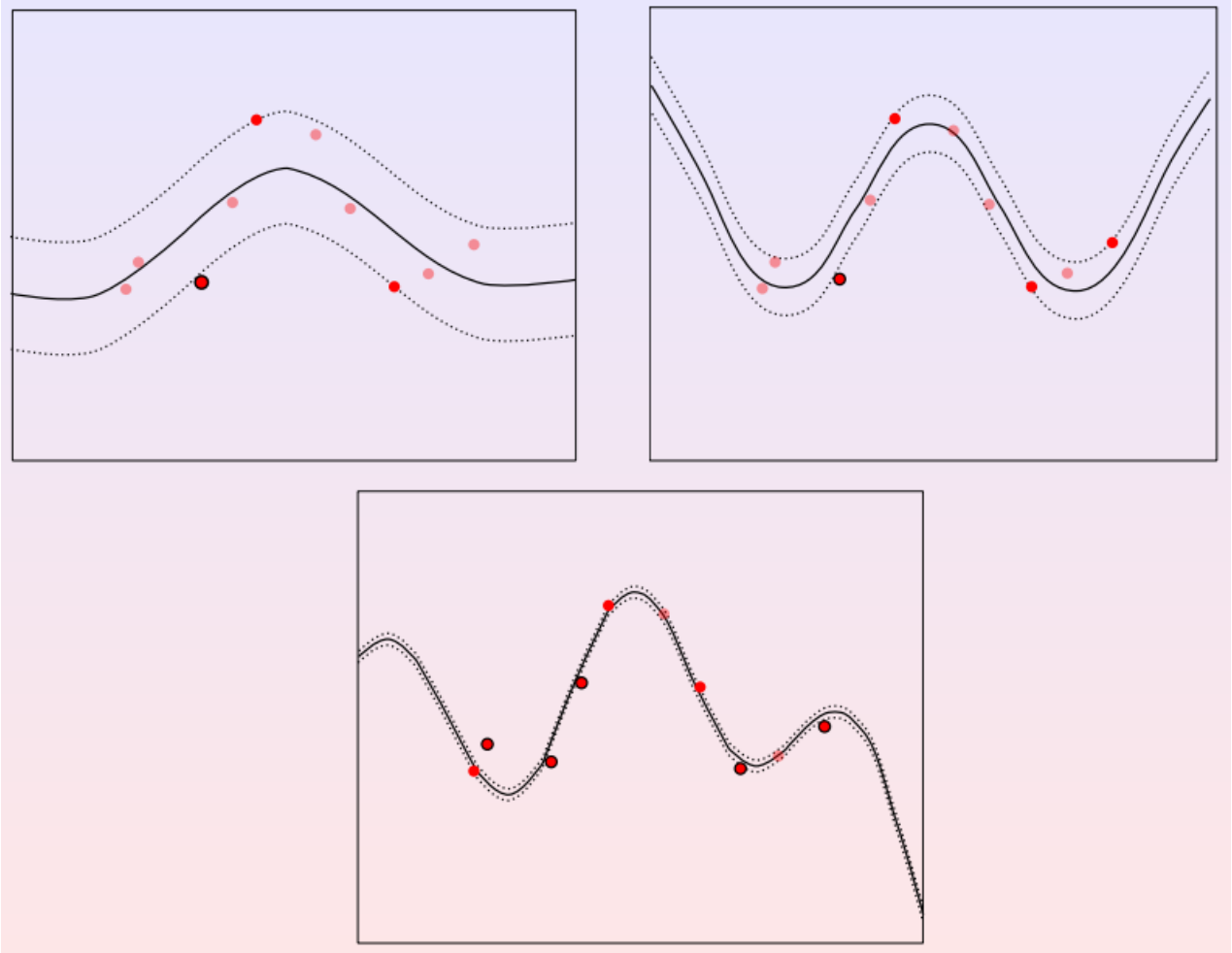
Define an ϵ-tube to tolerate errors up to a certain value ϵ



[Figure from A. Passerini]

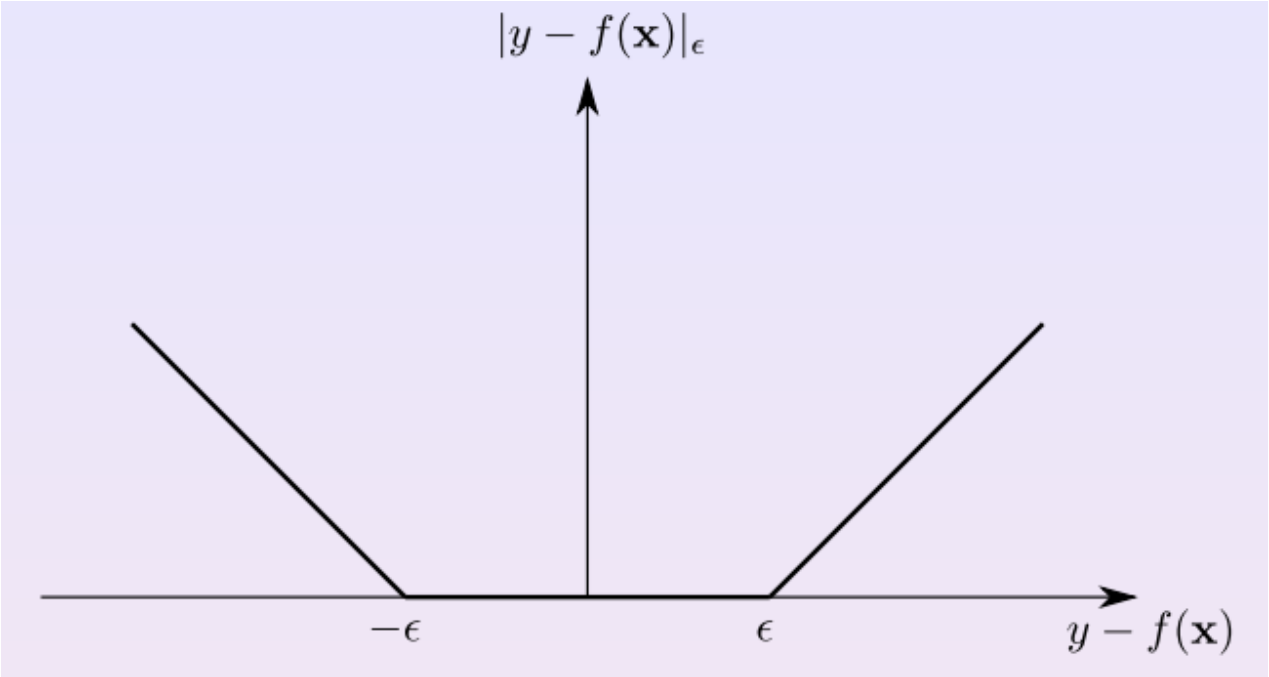
Support Vector Regression

The choice of ϵ clearly induces **different solutions**



[Figure from A. Passerini]

We define an ϵ-**insensitive** loss, that penalizes only errors ¿ ϵ



[Figure from A. Passerini]

$$\ell(f(x), y) = |y - f(x)|_\epsilon = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{otherwise} \end{cases}$$

Learning problem:

$$\min_{w,\xi,\xi*} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi*_i)$$

$$s.t. \quad w^T x_i + b - y_i \leq \epsilon + \xi_i$$

$$y_i - (w^T x_i + b) \leq \epsilon + \xi*_i$$

$$\xi_i, \xi*_i \geq 0$$

Two constraints for **upper** and **lower** sides of the ϵ-tube

Support Vector Ranking

In some applications, one would like just to have a **ranking** between examples: information retrieval, recommendation, ...

- Given **pairwise comparisons** $x_i \prec x_j$ in the training set
- Constrain function f to score x_i higher than x_j
- It can be formalized as a **classification task** !

Support Vector Ranking

Learning problem:

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

$$s.t. \qquad w^T\Phi(x_i) - w^T\Phi(x_j) \geq 1 - \epsilon_{ij} \quad \text{if} \quad x_i \prec x_j$$

$$\xi_{ij} \geq 0 \qquad \forall i,j = 1,\ldots,N$$

$w^T\Phi(x_i) - w^T\Phi(x_j)$ can be written as $w^T\Phi(x_i - x_j)$

The decision function becomes: $f(x) = w^T\Phi(x)$

One-class SVM

In some cases, one may be given **only positive examples**, or one would like to predict deviation from a certain **model** class

$\rightarrow$ **novelty/anomaly detection**

It is possible to **learn an SVM from positive examples only**

Key idea:

- find the **smallest hypersphere** that encloses the examples
- **penalty** for leaving outliers out of the hypersphere

Optimization problem:

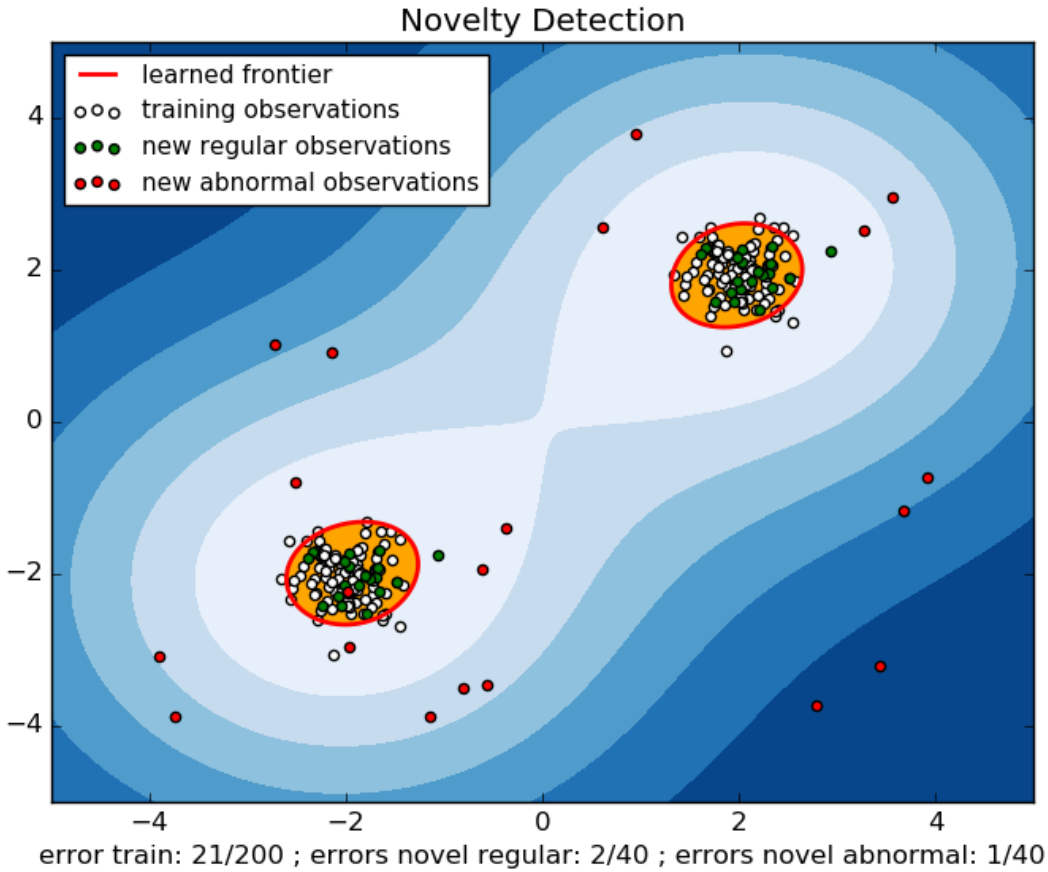
$$\min_{R,O,\xi} R^2 + C \sum_{i=1}^{N} \xi_i$$

$$s.t. \qquad \|\Phi(x_i) - O\|^2 \leq R^2 + \xi_i \qquad \forall i = 1, \ldots, N$$

$$\xi_i \geq 0, \qquad i = 1, \ldots, N$$

The hypersphere has center in O and radius R (to be minimized)

One-class SVM



[Figure from http://scikit-learn.org]

Soft margin SVM

Still not enough...

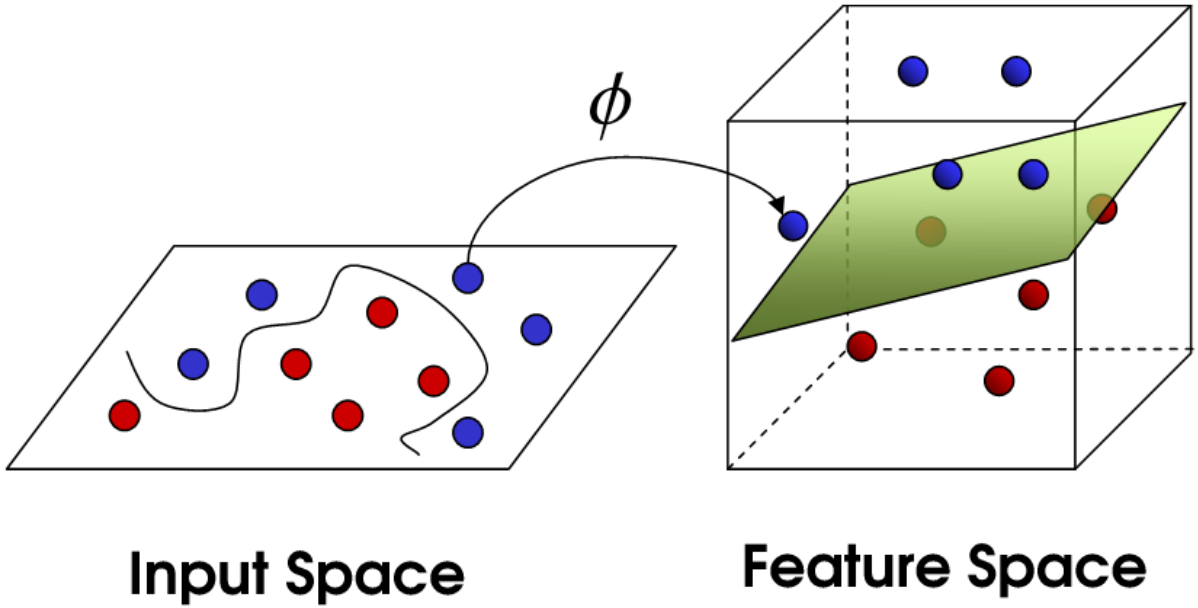
We can solve linearly separable problems (**hard case**)...
...Possibly with outliers (**soft case**)

Since the linear classifier is **very efficient**...

- map the examples into a **higher dimensional space**
- perform **linear classification** in that space

Feature spaces

Find a mapping $\Phi : \mathcal{X} \to \mathcal{H}$



[Figure from Stack Overflow]

A non-linear separation in $\mathcal{X}$ becomes linear in $\mathcal{H}$

Feature spaces

Now replace x with $\Phi(x)$ in the SVM algorithm:

$$f(x) = w^T \Phi(x) + b$$

Example 1:
$\Phi(x_1, x_2) = (x_1^2, x_1 x_2, x_2^2)$

Example 2:
$\Phi(x_1, x_2) = (x_1, x_2, x_1 x_2)$
Consider this case for the XOR function...

Kernel trick

Using function Φ to map examples to high-dimensional spaces:

- could be computationally much expensive
- also remember that Φ appears only in **dot products** !

Kernel trick

Replace dot product with an equivalent **kernel function**

$$k(x, z) = \Phi(x)^T \Phi(z)$$

Kernel properties

The SVM problem formulation (and solution) **remain the same** !

What are valid kernel functions ?

- A similarity function $k : \mathcal{X} \times \mathcal{X} \rightarrow \Re$
- A generalization of a dot product in arbitrary spaces

Kernel properties

The **Gram matrix** is defined as the matrix of pairwise kernels between examples:

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

A kernel is valid if one of the following holds:

- matrix K is positive definite
- all eigenvalues are non-negative
- there exists a matrix A such that $K = A^T A$

Kernel properties

Positive definiteness is **necessary and sufficient condition** for a kernel to correspond to a **dot product of some feature map** Φ

How to verify kernel validity:

- **prove** positive definiteness
- find out **explicit** feature map
- **combine** kernels

Kernel properties

Combine simple kernels to obtain more complex ones

Kernel Sum

$$(k_1 + k_2)(x, z) = k_1(x, z) + k_2(x, z) = \\ = \Phi_1(x)^T \Phi_1(x) + \Phi_2(x)^T \Phi_2(x)$$

The two kernels can be defined on **completely different spaces**

Kernel properties

Combine simple kernels to obtain more complex ones

Kernel Product

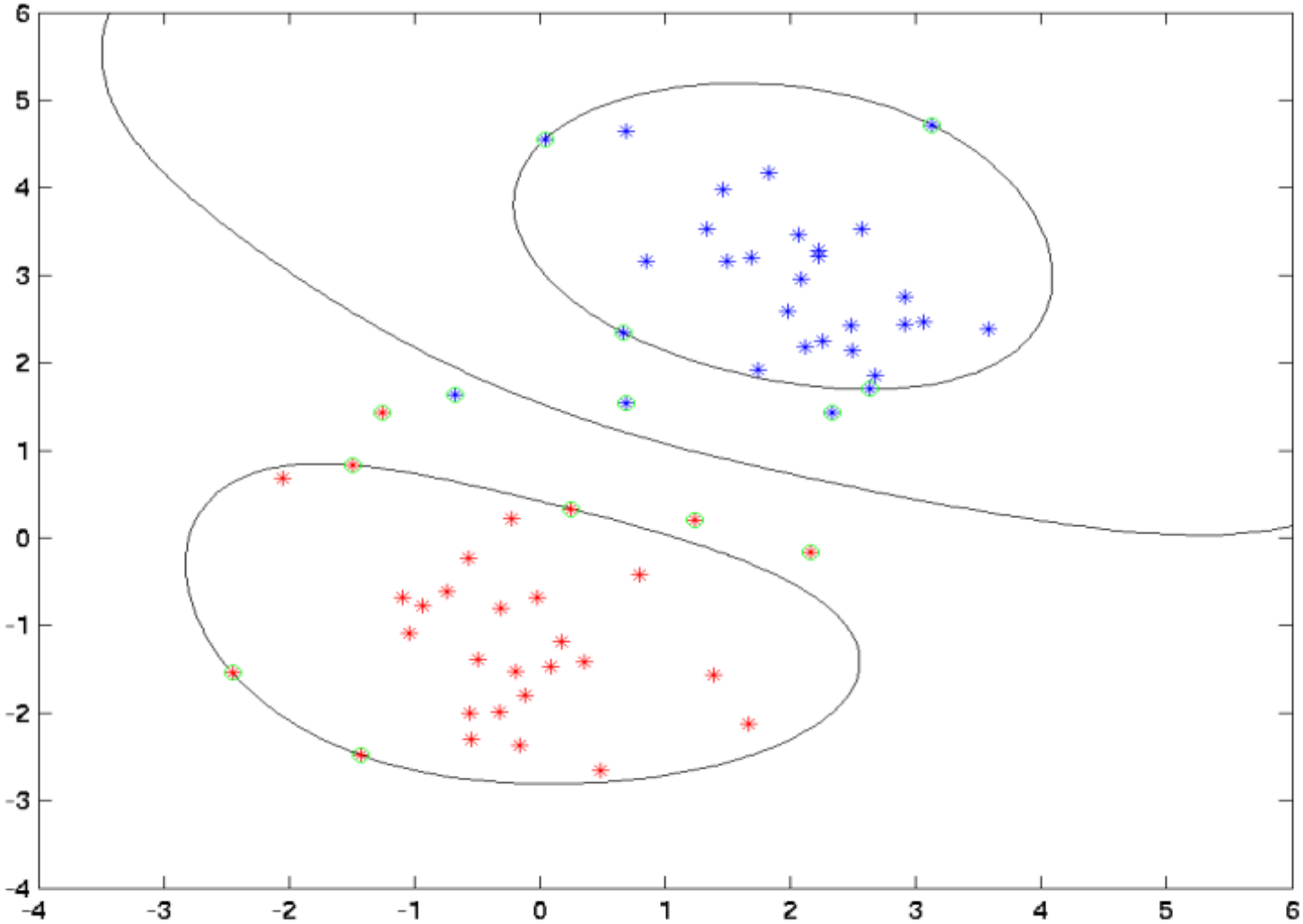
$$(k_1 \cdot k_2)(x, z) = k_1(x, z) \cdot k_2(x, z) = \\ = \sum_{i=1}^{n} \Phi_{1i}(x)\Phi_{1i}(z) \sum_{i=1}^{m} \Phi_{2i}(x)\Phi_{2i}(z)$$

It corresponds to the **Cartesian product of the features**

Kernel examples

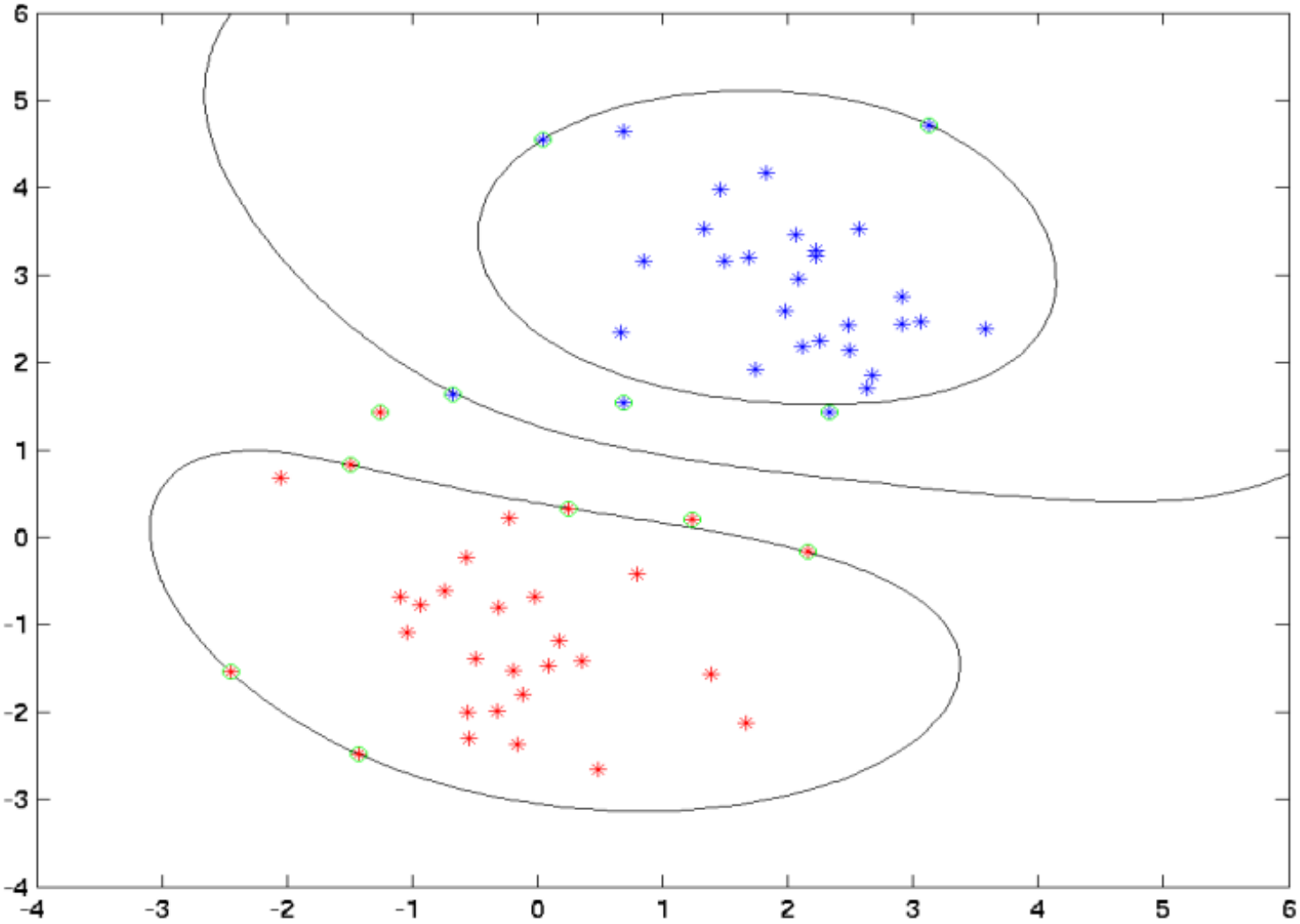
- Linear kernel
  $k(x, z) = x^T z$
- Polynomial kernel
  $k(x, z) = (x^T z + c)^d$
- Gaussian (RBF) kernel
  $k(x, z) = \exp(-\frac{\|x-z\|^2}{2\sigma^2})$

Regularization with RBF kernel



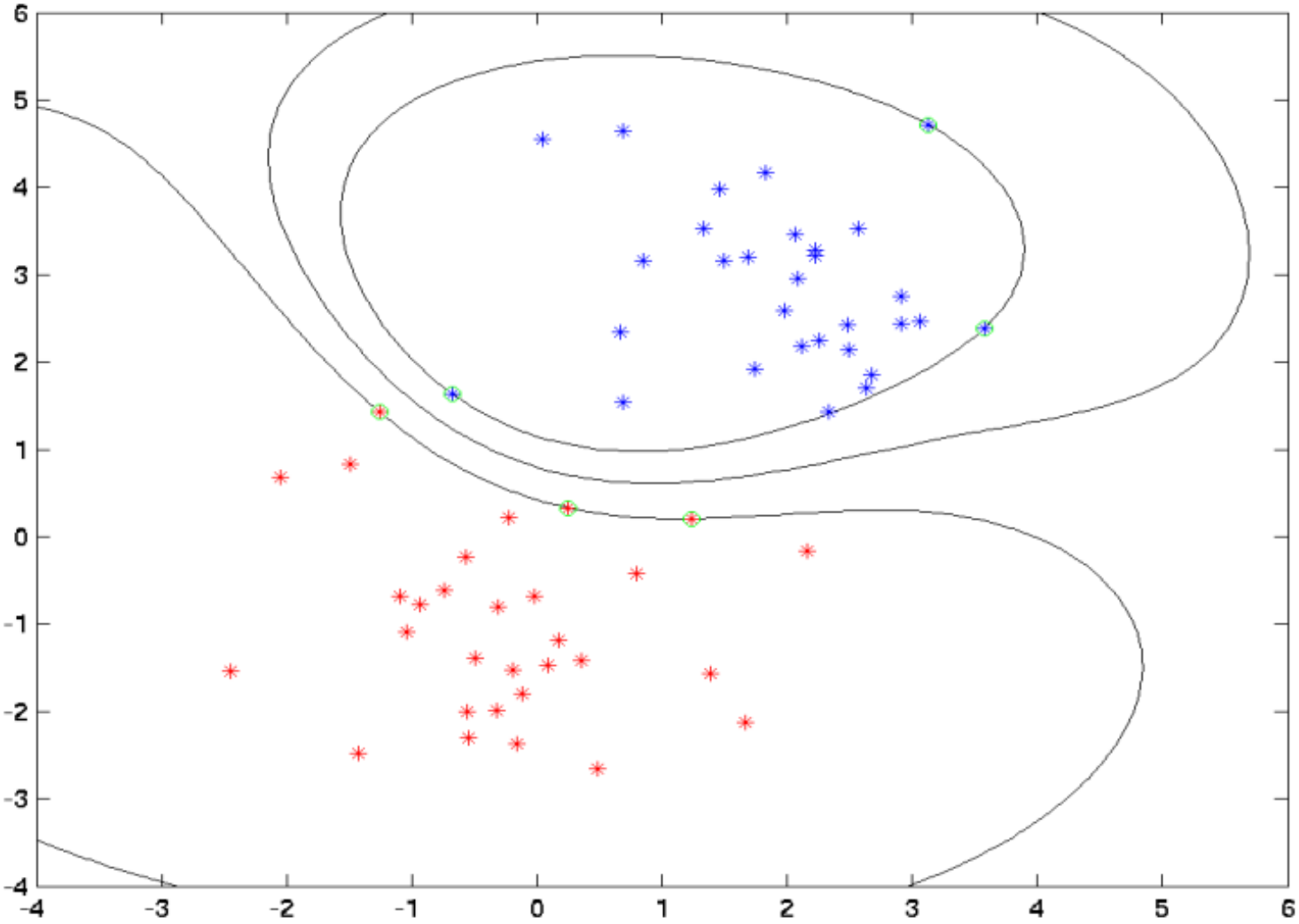
[Figure from A. Smola]

Regularization with RBF kernel



[Figure from A. Smola]

Regularization with RBF kernel



[Figure from A. Smola]

Kernel learning

The **linear combination** of kernels is still a kernel

$$k(x, z) = \sum_{p=1}^{P} w_p k_p(x, z)$$

The weights w_p of each kernel can be **jointly learned**

Structured data

In many real-world applications one has to deal with **structured data**, e.g., in the form of trees, graphs, sequences, strings, ...

Convolution kernel

Given the **decomposition** of a structure into its sub-parts:

$$R(X) = (x_1, \ldots, x_D)$$

the **convolution kernel** is defined as the convolution of its parts:

$$k(x, z) = \sum_{r \in R(x)} \sum_{s \in R(z)} \prod_{d=1}^{D} k_d(x_d, z_d)$$

computed over **all possible decompositions** of x and z

Kernels on sequences

Sequences are found in many application domains:

- bioinformatics – DNA, RNA, proteins
- time series
- text documents (word sequences)

Spectrum kernel

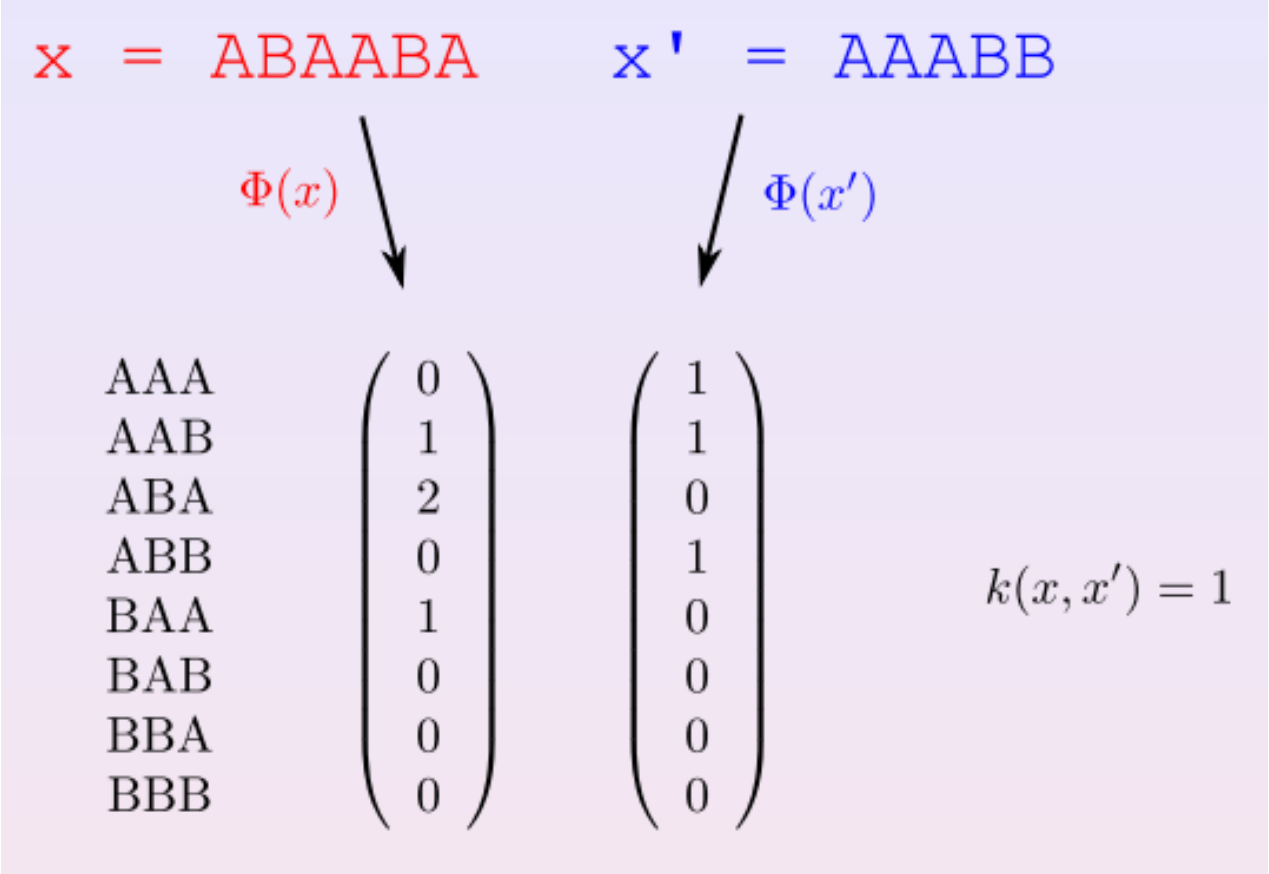
The k-spectrum of a sequence is the set of all k-length subsequences (named k-**mers**) that it contains

$$\text{ACGTGGCA} \rightarrow \text{ACG, CGT, GTG, TGG, GGC, GCA (if } k = 3)$$

- One feature for each existing k-mer
- Feature space dimension $= |\Sigma|^k$
- Count k-mer occurrences
- Dot product in this new feature space

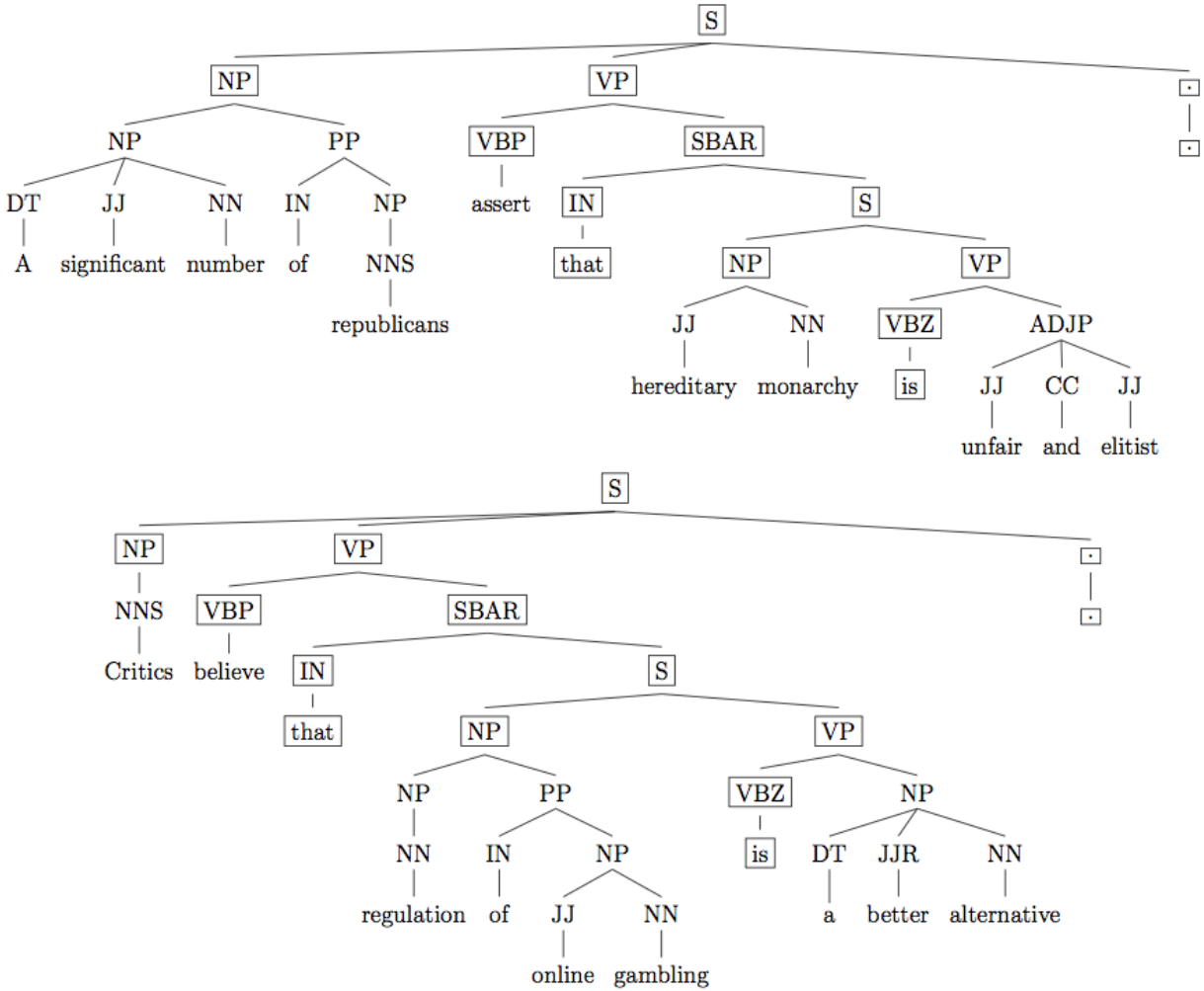
Spectrum kernel

$$k_s(x, z) = \Phi(x)^T \Phi(z)$$



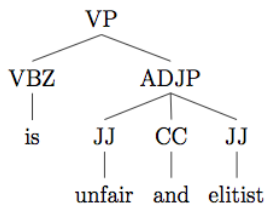
[Figure by A. Passerini]

Measuring similarity between two trees

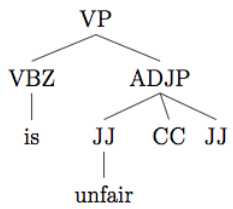


Tree Kernels

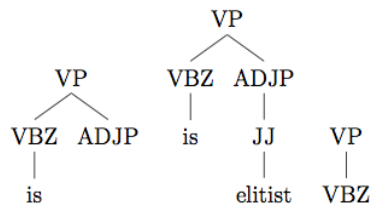
- Evaulating common substructures or **fragments**
- Different definitions of fragments $\rightarrow$ different kernels



(a) an STK fragment



(b) an SSTK fragment



(c) PTK fragments

Tree Kernels

Intuition:

- each possible **fragment** associated to a different **feature**
- feature space can become really **high-dimensional**

$$K(T_x, T_z) = \sum_{n_x \in N_{T_x}} \sum_{n_z \in N_{T_z}} \Delta(n_x, n_z)$$

- N_{T_x} and N_{T_z} are the set of nodes of the two trees
- $\Delta(\cdot, \cdot)$ measures the score between two nodes

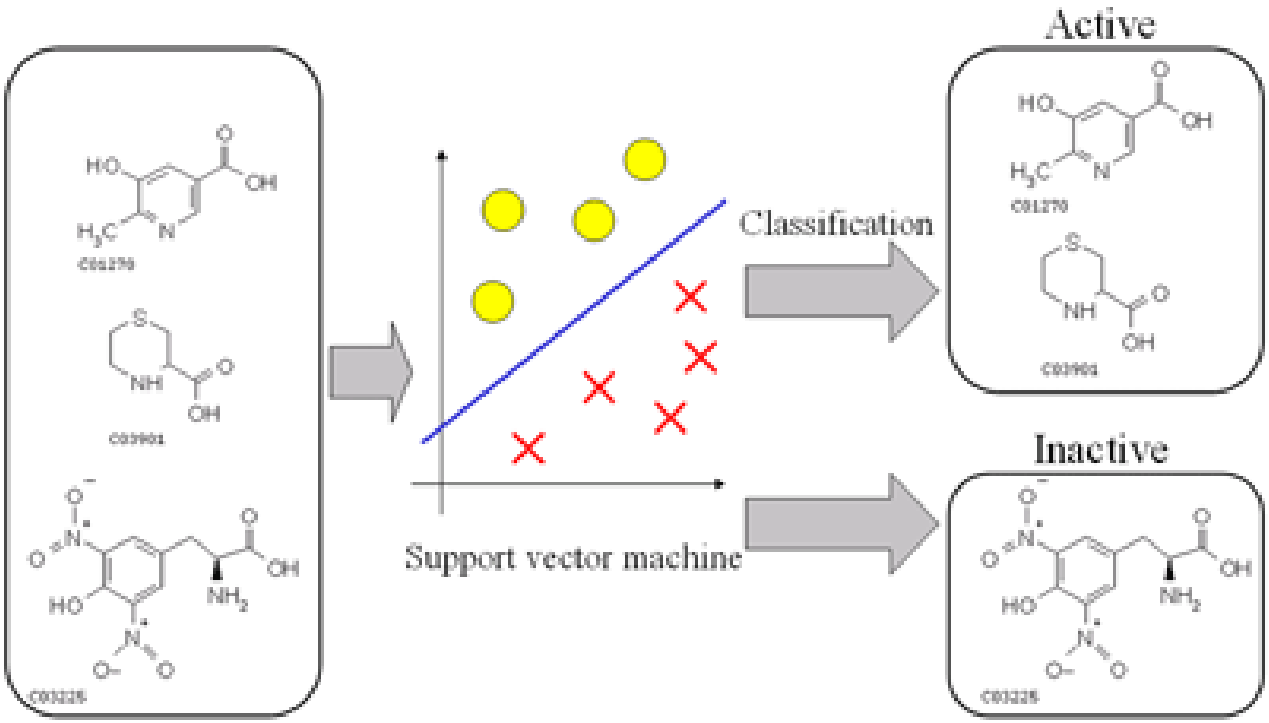
Tree Kernels

Rich feature space

Tree kernels can automatically generate a very rich feature set, capable of capturing structured representations **without the need of a costly feature engineering process**

Software: Alessandro Moschitti, KeLP

Measure similarity between graphs

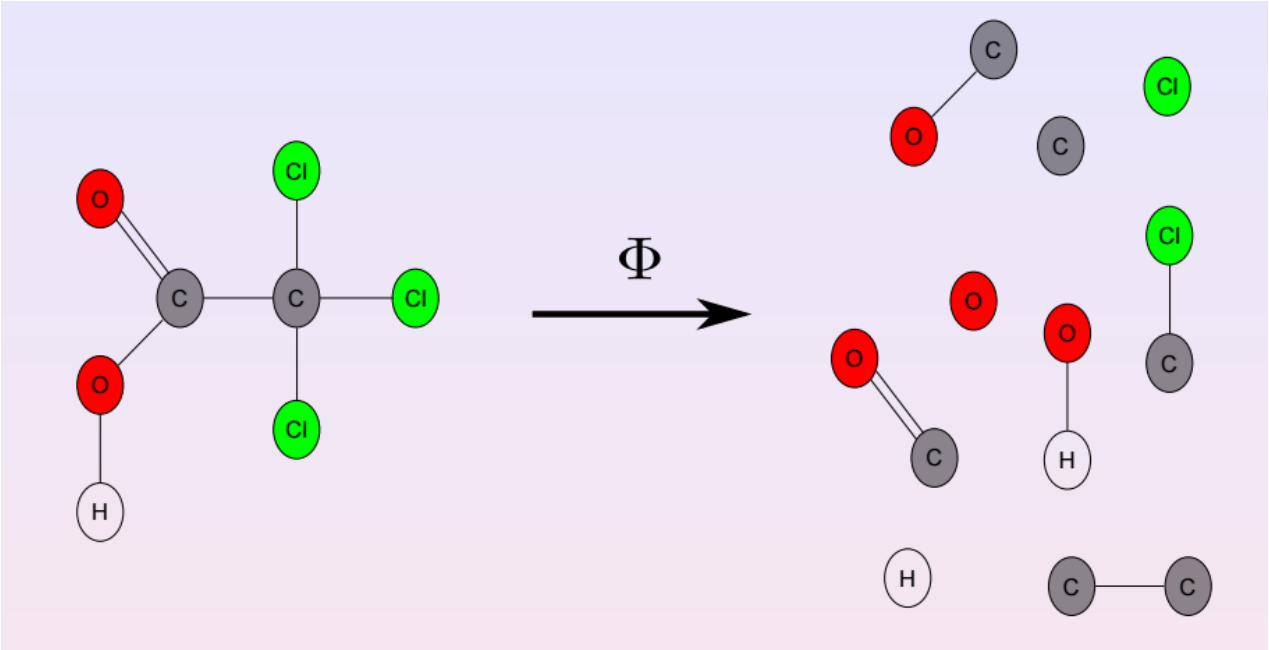


[Figure from http://www.bic.kyoto-u.ac.jp/]

Graph Kernels

Key ideas:

- one feature for each possible subgraph
- possibly limiting the size of considered subgraphs
- count number of subgraph occurrences



[Figure by A. Passerini]

Structured output

The SVM framework has been recenly [Tsochantaridis et al., 2005] extended towards **structured output predictions**, where the **output space** is a structure

- Hidden Markov Models
- Probabilistic Context Free Grammars
- Sequence Alignment
- Information Retrieval
- Activity Recognition

LIBSVM – A Library for Support Vector Machines

Download:
`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

Alternatively:
`http://svmlight.joachims.org/`

Data format

- Plain text file
- First column $\rightarrow$ prediction target
- Other columns $\rightarrow$ sparse vector describing the features

An example:

```
1    12:-0.87 23:0.11 25:0.05 51:-0.45
2    10:-0.18 23:0.01 77:0.98
2    21:0.13 40:0.58 71:0.91
...
```

Pay attention: feature indices **must be in increasing order**.

```
svm-train [options] training_set_file [model_file]
```

Choose the type of SVM

```
-s svm_type : set type of SVM (default 0)
        0 -- C-SVC
        1 -- nu-SVC
        2 -- one-class SVM
        3 -- epsilon-SVR
        4 -- nu-SVR
```

```
svm-train [options] training_set_file [model_file]
```

Choose the type of kernel

```
-t kernel_type : set type of kernel function (default 2)
        0 -- linear: u'*v
        1 -- polynomial: (g*u'*v + coef0)^d
        2 -- radial basis function: exp(-g*|u-v|^2)
        3 -- sigmoid: tanh(g*u'*v + coef0)
        4 -- precomputed kernel
```

```
svm-train [options] training_set_file [model_file]
```

Other options

```
-d degree : degree in kernel function (def. 3)
-g gamma : gamma in kernel function (def. 1/num_features)
-r coef0 : coef0 in kernel function (def. 0)
-c cost : parameter C of C-SVC, epsilon-SVR, and nu-SVR (def. 1)
-p epsilon : epsilon in loss function of epsilon-SVR (def. 0.1)
-e epsilon : tolerance of termination criterion (def. 0.001)
-b probability_estimates : whether to train a SVC or SVR model
                           for probability estimates, 0 or 1 (def. 0)
-wi weight : set the parameter C of class i
             to weight*C, for C-SVC (def. 1)
-v n: n-fold cross validation mode
```