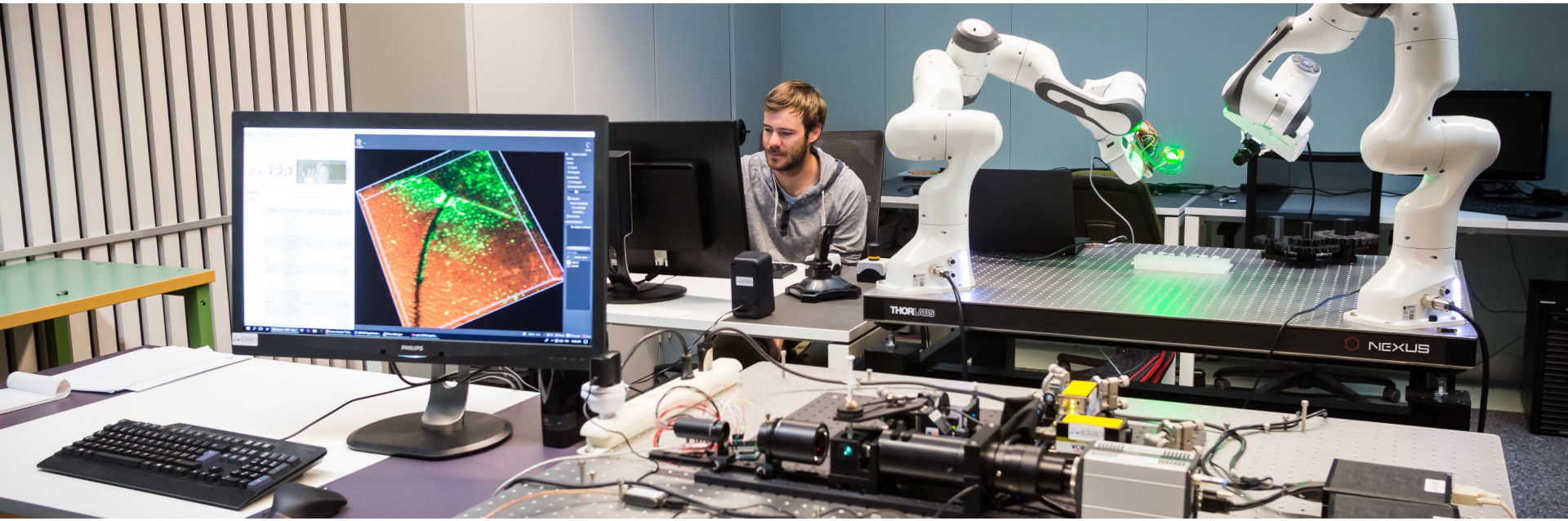
Machine Learning for Engineers

Kernel methods - Support Vector Machines

Jean-Marc Odobez



overview

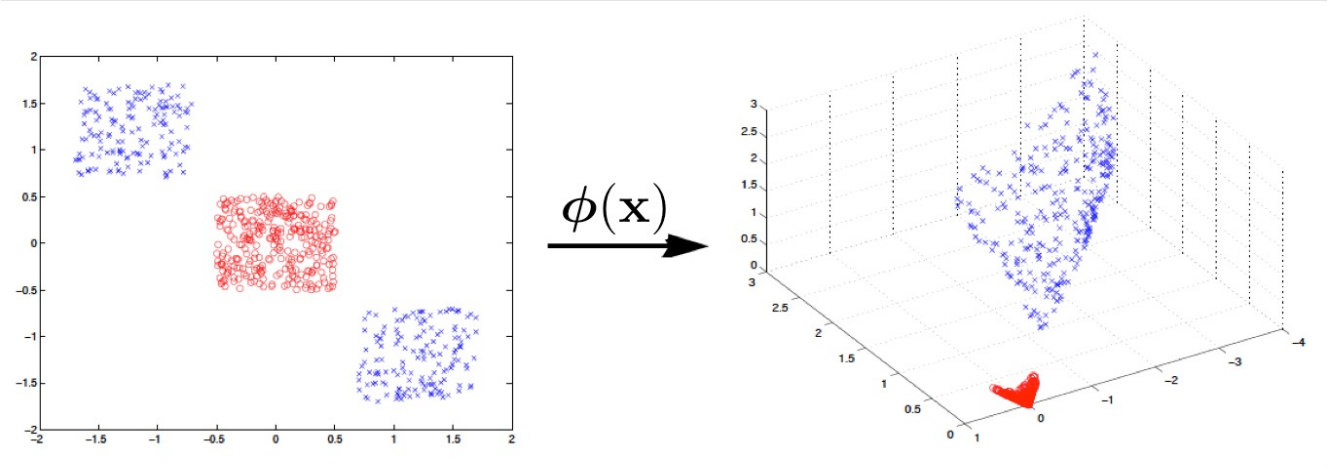
- Kernel methods
  - introduction and main elements
  - defining kernels
  - Kernelization of k-NN, K-Means, PCA
- Support Vector Machines (SVMs)
  - classification
  - regression

Kernel methods

introduction and main elements/considerations

(1) high dimensional spaces

- Data points in high dimensional spaces can be better separated
- Exemple: linear classifier (e.g. perceptron)
  - linear decision function => map feature in high dimensional space



  - here: polynomial kernel $\phi(\mathbf{x}) = \phi(x_1, x_2) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$

- Questions:
  - how to map data efficiently in high dimension (potentially infinite) spaces?
  - how does such mapping affect existing methods/classifiers?

(2) comparing samples

- We would like similar samples to be classified in the same way => distance
- We often think of distances in (euclidian) metric spaces
  - distance <-> scalar product

$$\|\mathbf{x}-\mathbf{x}'\|^2 = (\mathbf{x}-\mathbf{x}')\cdot(\mathbf{x}-\mathbf{x}') = \mathbf{x}\cdot\mathbf{x} - 2\mathbf{x}\cdot\mathbf{x}' + \mathbf{x}'\cdot\mathbf{x}'$$

$$\mathbf{x}\cdot\mathbf{x}' = \frac{1}{2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - \|\mathbf{x}-\mathbf{x}'\|^2\right)$$

- Might not always be easy or relevant
  - how to compare
    2 strings, 2 text paragraphs, 2 sequences, 2 images.....
- However: often we can define some similarity measures between elements
  - e.g. for strings: Sim(s1,s2) = EditDistance(s1,s2)
  - note: often triangular inequality not respected
- How can we exploit such measures in classification algorithms? which properties of these measures are useful?

(3) classifiers

Two types of classifiers

- model-based (classification, regression)

$$h(\mathbf{x}) = \begin{cases} +1 \text{ if } \mathbf{w}\cdot\mathbf{x} + b > 0 \\ -1 \text{ otherwise} \end{cases}$$

  - eg. linear classifier
  - data used to learn the model parameters, and then removed

- non-parametric approach
  - training data points are kept in classifier definition
    - K-Nearest Neighbour (kNN)
    - Parzen windows density estimation

$$P(\mathbf{x}) = \frac{1}{n}\sum_i \frac{1}{h_n^d} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right)$$

  - memory-based methods (fast at training, slow at testing)

In practice: in many memory-based methods, **the solution** can be written as **a linear combination** of **kernel function at training data points** representing scalar product in high dimension

This linear combination is often referred to **as the 'dual' representation**

(3) illustration: perceptron algorithm (1)

Goal: training a linear classifier

$$h(\mathbf{x}) = \begin{cases} +1 \text{ if } \mathbf{w}\cdot\mathbf{x} + b > 0 \\ -1 \text{ otherwise} \end{cases}$$

- Note: bias *b* can be introduced as one of the *weight term* by adding a constant component to x equal to 1

$$\tilde{\mathbf{w}} = (\mathbf{w}, b) \text{ and } \tilde{\mathbf{x}} = (\mathbf{x}, 1) \text{ then we have } \tilde{\mathbf{w}}\cdot\tilde{\mathbf{x}} > 0$$

- Next slides: drop the tilde notation, and define classifier as

$$h(\mathbf{x}) = \begin{cases} +1 \text{ if } \mathbf{w}\cdot\mathbf{x} > 0 \\ -1 \text{ otherwise} \end{cases}$$

(3) illustration: perceptron algorithm (2)

Perceptron algorithm (Rosenblatt)

Simple method to train a linear classifier $h(\mathbf{x}) = \begin{cases} +1 \text{ if } \mathbf{w}\cdot\mathbf{x} > 0 \\ -1 \text{ otherwise} \end{cases}$

- Given a training set $(x_n, y_n) \in \mathbb{R}^D \times \{-1, 1\}, \quad n = 1, \ldots, N,$

- Algorithm proceeds as follows

  1. Start with $w_0 = 0$,
  2. while $\exists n_k$ s.t. $y_{n_k}(w_k \cdot x_{n_k}) \leq 0$, update $w_{k+1} = w_k + y_{n_k} x_{n_k}$.

Meaning: sample not well classified

Result: if data are separable, algorithm is converging to a valid solution

(3) illustration: perceptron algorithm (3)

- Update rule at iteration l

$$\mathbf{w}_{l+1} = \mathbf{w}_l + \begin{cases} \mathrm{y}_l \mathbf{x}_l & \text{if } \mathrm{y}_l \mathbf{w}_l \cdot \mathbf{x}_l \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

- In (high dimension) projection space $\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x})$

$$\mathbf{w}_{l+1} = \mathbf{w}_l + \begin{cases} \mathrm{y}_l \boldsymbol{\phi}(\mathbf{x}_l) & \text{if } \mathrm{y}_l (\mathbf{w}_l \cdot \boldsymbol{\phi}(\mathbf{x}_l)) \leq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Result: run until convergence, keeping only *l* indices with a non 0 update
  - weights are a linear combination of training data

$$\mathbf{w} = \sum_l \mathrm{y}_l \boldsymbol{\phi}(\mathbf{x}_l)$$

  - the decision function can be rewritten as

$$\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}) = \sum_l \mathrm{y}_l \ \boldsymbol{\phi}(\mathbf{x}_l) \cdot \boldsymbol{\phi}(\mathbf{x}) = \sum_l \mathrm{y}_l \ k(\mathbf{x}_l, \mathbf{x})$$

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x}')$$

  - Decision function h(x) : data is thus used only through dot products in projected space, and implicitly, through a Kernel *k*

Kernel methods

defining kernels

Kernels

- Kernel – we are given a projection operator $\mathbf{x} \rightarrow \phi(\mathbf{x})$

  => we can define a kernel as a dot product in that space $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$

- Alternatively – directly provide a kernel $k(x_1,x_2)$
  - Intuition – the kernel capture the similarity between x_1 and x_2
  - E.g. : perceptron example

$$h(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) = \sum_l \mathrm{y}_l \ \phi(\mathbf{x}_l) \cdot \phi(\mathbf{x}) = \sum_l \mathrm{y}_l \ k(\mathbf{x}_l, \mathbf{x})$$

    - Note: weighted sum of labels
    - kernel $k(x_l,x)$ is high => x will tend to be classified like x_l
    - kernel $k(x_l,x)$ is low => x_l will have no influence on output for x
  - In practice, the choice of kernel depends on application

Kernels

- Kernel – we are given a projection operator $\mathbf{x} \rightarrow \phi(\mathbf{x})$

  => we can define a kernel as a dot product in that space $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')$

- We are given a function k . Does it define a Kernel?
  - Valid kernels: Mercer Kernel
  - Consider a smooth symmetric function *k()* over a compact C $k : C \times C \rightarrow \mathbb{R}$
  - *k()* is a kernel if and only if it can be decomposed into

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \alpha_i \ \phi_i(\mathbf{x}) \cdot \phi_i(\mathbf{x}')$$

  - and if and only if
    - for all finite set $\{\mathbf{x}_1, \ldots, \mathbf{x}_p\} \subset C$
    - the matrix *K* defined by $K_{ij} = k(x_i, x_j)$ is semi-definite positive

Notable kernels

- Polynomial Kernels

$$k(\mathbf{x}, \mathbf{x}') = (u\ \mathbf{x}\cdot\mathbf{x}' + v)^p,\ \ u, v \geq 0, p \in \mathbb{N}$$

- Gaussian Kernels

$$k(\mathbf{x}, \mathbf{x}') = \exp^{-\gamma\|\mathbf{x}-\mathbf{x}'\|^2},\ \gamma > 0$$

  - note: not considered as a distribution here

    => no need for normalization constant
  - implicit projection: in an infinite dimension space

- String Kernel

$$k(\mathbf{x}, \mathbf{x}') = \sum_{s \in \mathcal{A}^\star} w_s \phi_s(\mathbf{x}) \phi_s(\mathbf{x}')$$

count number of times substring s occurs in **x**

- Fisher Kernel

Building Kernel

- Kernel can be constructed by combining kernels, e.g. like

  - $k(\mathbf{x}, \mathbf{x}') = c_1 k_1(\mathbf{x}, \mathbf{x}') + c_2 k_2(\mathbf{x}, \mathbf{x}')$
  - $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')$
  - $k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$
  - $k(\mathbf{x}, \mathbf{x}') = \exp(k_1(\mathbf{x}, \mathbf{x}'))$
  - $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$
  - $k(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}'))$
  - $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}'$
  - $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b)$
  - $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b)$

  where kernels on the right are valid kernels on their respective domains, c1>0 and c2>0, **A** is a symmetric semidefinitive positive matrix, f is any function, q is a polynomial of non-negative coefficients, and x_a and x_b are variables (not necessarily disjoint) with $x = (x_a, x_b)$

- Properties can be used to demonstrate whether a proposed kernel is a Mercer Kernel

Kernel methods

Kernelization of k-NN, K-Means, PCA

Kernelizing algorithms

- Many algorithms can be « Kernelized »
  - Straightforward for the perceptron
  - k-NN?
  - k-Mean?
  - PCA?

  - how?
    - write the algorithm using as data points the project data $\phi(\mathbf{x})$
    - express results on the form of dot product
    - use the kernel trick

Kernelizing k-NN

- k-NN algorithm
  - Training dataset: $D = \{(\mathbf{x}_i, y_i) | y_i \in \{1, \ldots, L\}, i = 1, ..., N\}$
  - Parameter : K
  - Classifying a new sample x
    - Find the set C of K samples from *D* closest to **x**

$$C(\mathbf{x}) = \{\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \ldots, \mathbf{x}_{i_K}\}$$

    - Assign to x the majority class in the associated set of labels

$$\{y_{i_1}, y_{i_2}, \ldots, y_{i_K}\}$$

- requires distances between two examples $\|\mathbf{x} - \mathbf{x}_i\|^2$

$$\begin{aligned}\|\boldsymbol{\phi}(\mathbf{x}) - \boldsymbol{\phi}(\mathbf{x}_i)\|^2 &= \boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{x}) - 2\boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{x}_i) + \boldsymbol{\phi}(\mathbf{x}_i)\cdot\boldsymbol{\phi}(\mathbf{x}_i) \\ &= k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}_i) + k(\mathbf{x}_i, \mathbf{x}_i)\end{aligned}$$

  - easy to kernelize....

Kernel K-Means

- Apply K-means in projected space ....
- Assumes μ_i denotes the means/centroids in this space
  - as the projected space can be infinite, we keep the means in their dual form

$$\{\alpha_1^i, \alpha_2^i, \ldots\}$$

i.e. as a weighted sum of the samples ...

$$\mu_i = \sum_j \alpha_j^i \boldsymbol{\phi}(\mathbf{x}_j) \qquad (\text{with } \alpha_j^i = \frac{1}{N_i} \text{ if } \mathbf{x}_j \text{ belongs to class } i\,, 0 \text{ otherwise})$$

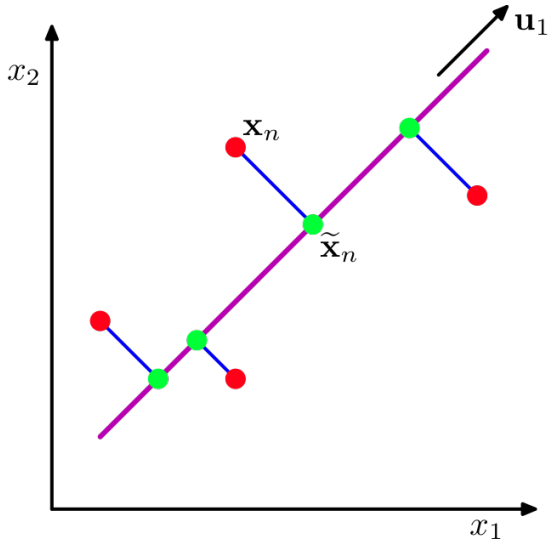
- Assignment step: for each data sample, we need to find the closest mean

$$\|\boldsymbol{\phi}(\mathbf{x}) - \mu_i\|^2 = \boldsymbol{\phi}(\mathbf{x})\cdot\boldsymbol{\phi}(\mathbf{x}) - 2\sum_j \alpha_j^i \boldsymbol{\phi}(\mathbf{x}_j)\cdot\boldsymbol{\phi}(\mathbf{x}) + \sum_{j,k} \alpha_j^i \alpha_k^i \boldsymbol{\phi}(\mathbf{x}_j)\cdot\boldsymbol{\phi}(\mathbf{x}_k)$$

$$\|\boldsymbol{\phi}(\mathbf{x}) - \mu_i\|^2 = k(\mathbf{x},\mathbf{x}) - 2\sum_j \alpha_j^i k(\mathbf{x}_j,\mathbf{x}) + \sum_{j,k} \alpha_j^i \alpha_k^i k(\mathbf{x}_j,\mathbf{x}_k)$$

- Mean computation: update the alpha accordingly

Kernel PCA

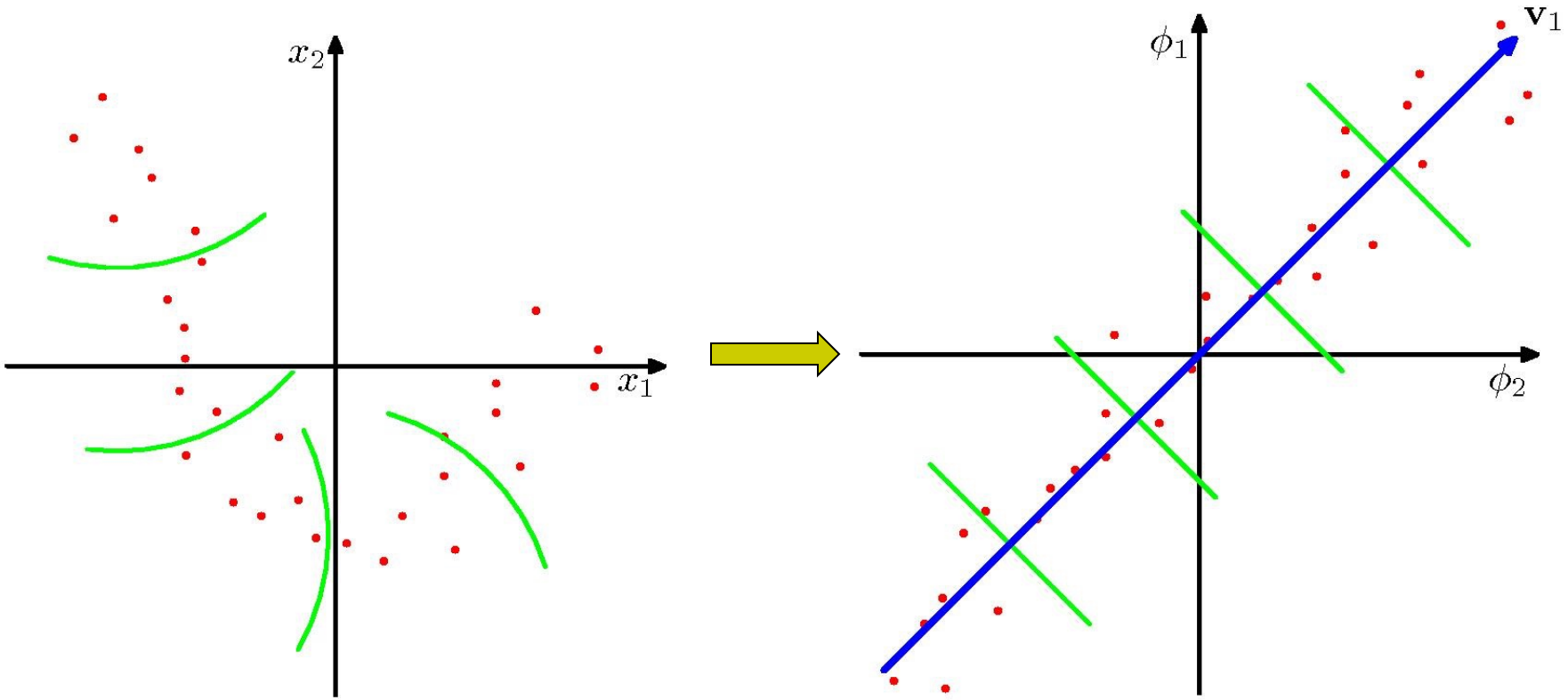


Standard PCA

- Way to remove correlation between points
  => reduce dimensions through linear projection

- Data driven: training samples $\{\mathbf{x}_1, ..., \mathbf{x}_N\}, \mathbf{x}_i \in R^D$
  - compute mean and covariance $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \qquad \mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$
  - find largest eigenvalues of covariance matrix
    => sort eigenvectors ui by decreasing order of eigenvalues
    => form matrix $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_M)$
  - lower dimensional representation of datapoints is given by $\mathbf{y}_n = \mathbf{U}^T(\mathbf{x}_n - \bar{\mathbf{x}})$
  - approximate reconstruction $\tilde{\mathbf{x}}_n \simeq \bar{\mathbf{x}} + \mathbf{U}\mathbf{y}_n$

Kernel PCA - intuition



- Apply normal PCA in high-dimensional projected space
- (straight) lines of constant projections in projected space correspond to nonlinear projections in original space

Kernel PCA

- Assume projected data are centered (have 0 mean)

$$\sum_i \phi(\mathbf{x}_i) = 0$$

- Covariance matrix in projected space

$$C = \frac{1}{N}\sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T = \frac{1}{N}\mathbf{X}\mathbf{X}^T \qquad \mathbf{X} = [\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N)]$$

where **X** is the design matrix, with column i defined by $\Phi(\mathbf{x}_i)$

- PCA computes the eigenvalues/eigenvector of C.
  How can we compute them (or involve) in terms of

$$K_{kl} = k(\mathbf{x}_k, \mathbf{x}_l) = \phi(\mathbf{x}_k)^T \phi(\mathbf{x}_l)$$

- Note that $\mathbf{K} = \mathbf{X}^T\mathbf{X}$

Kernel PCA

- By definition, we have $C\mathbf{v}_i = \lambda_i \mathbf{v}_i$

- Substituting the covariance definition leads to

$$\left(\frac{1}{N} \sum_{l=1}^{N} \boldsymbol{\phi}(\mathbf{x}_l)\ \boldsymbol{\phi}(\mathbf{x}_l)^T \right) \mathbf{v}_i = \frac{1}{N} \sum_{l=1}^{N} \boldsymbol{\phi}(\mathbf{x}_l)\ \left(\boldsymbol{\phi}(\mathbf{x}_l)^T \mathbf{v}_i \right) = \lambda_i \mathbf{v}_i$$

- Consequence: the eigenvector can be expressed as a linear combination of the projected samples (DUAL FORM)

$$\mathbf{v}_i = \sum_{l=1}^{N} \boldsymbol{\phi}(\mathbf{x}_l)\, a_{il}, \ \ (\text{with } a_{il} = \frac{1}{\lambda_i N} \boldsymbol{\phi}(\mathbf{x}_l)^T \mathbf{v}_i)$$

- Then, how can we actually determine the *a* coefficients?

  (and involve only the kernel function *k(. , .)*)

Kernel PCA

- In matrix form, eigengenvectors can thus be written as

$$\mathbf{v}_i = \mathbf{X}\mathbf{a}_i$$

- Eigenvalue problem

$$C\mathbf{v}_i = \lambda_i \mathbf{v}_i \qquad C = \frac{1}{N}\mathbf{X}\mathbf{X}^T \qquad \frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

- Introducing the decomposition into it leads to

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{a}_i = \lambda_i \mathbf{X}\mathbf{a}_i$$

$$\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{X}\mathbf{a}_i = N\lambda_i \mathbf{X}^T\mathbf{X}\mathbf{a}_i \qquad \text{i.e.} \qquad \mathbf{K}^2\mathbf{a}_i = \lambda_i N\mathbf{K}\mathbf{a}_i$$

- Thus, we can find solutions for $\mathbf{a_i}$ by solving the eigenvalue problem

$$\mathbf{K}\mathbf{a}_i = \lambda_i N\mathbf{a}_i$$

Kernel PCA

- We need to normalize the coefficient $\mathbf{a_i}$
  - impose that eigenvectors in projected space have norm 1

$$1 = \mathbf{v}_i^T \mathbf{v}_i = (\mathbf{X}\mathbf{a}_i)^T (\mathbf{X}\mathbf{a}_i) = \mathbf{a}_i^T \mathbf{X}^T \mathbf{X}\mathbf{a}_i = \mathbf{a}_i^T \mathbf{K}\mathbf{a}_i = \lambda_i N \mathbf{a}_i^T \mathbf{a}_i$$

- We need to center the data (in projected space)
  - we can not compute the mean (in projected space) as we want to avoid working directly in this projection space => we need to formulate the algorithm purely in term of the kernel function

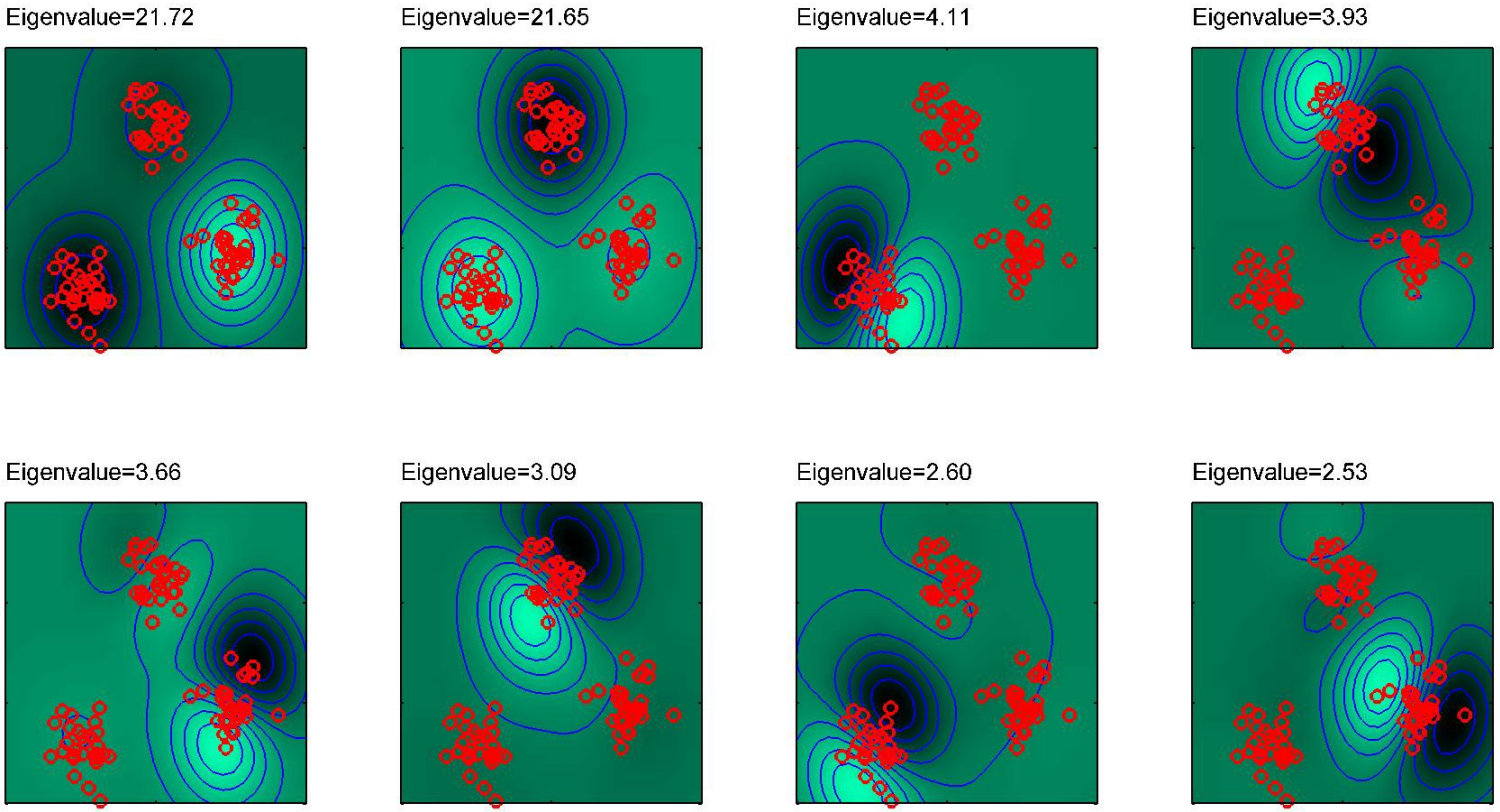
$$\tilde{\phi}(\mathbf{x}_j) = \phi(\mathbf{x}_j) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l) \qquad \Rightarrow \qquad \tilde{\mathbf{X}} = \mathbf{X} - \frac{1}{N}\mathbf{X}\mathbf{1}\mathbf{1}^T$$

$$\begin{aligned}\tilde{\mathbf{K}} = \tilde{\mathbf{X}}^T\tilde{\mathbf{X}} &= \mathbf{X}^T\mathbf{X} - \tfrac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{1}\mathbf{1}^T - \tfrac{1}{N}\mathbf{1}\mathbf{1}^T\mathbf{X}^T\mathbf{X} + \tfrac{1}{N^2}\mathbf{1}\mathbf{1}^T\mathbf{X}^T\mathbf{X}\mathbf{1}\mathbf{1}^T \\ &= \mathbf{K} - \tfrac{1}{N}\mathbf{K}\mathbf{1}\mathbf{1}^T - \tfrac{1}{N}\mathbf{1}\mathbf{1}^T\mathbf{K} + \tfrac{1}{N^2}\mathbf{1}\mathbf{1}^T\mathbf{K}\mathbf{1}\mathbf{1}^T\end{aligned}$$

- Projection (coordinate) of a point on eigenvector i

$$y_i(\mathbf{x}) = \mathbf{v}_i^T\phi(\mathbf{x}) = (\sum_{l=1}^{N} a_{il}\phi(\mathbf{x}_l))^T\phi(\mathbf{x}) = \sum_{l=1}^{N} a_{il}\phi(\mathbf{x}_l)^T\phi(\mathbf{x}) = \sum_{l=1}^{N} a_{il}k(\mathbf{x}, \mathbf{x}_l)$$

Kernel PCA - illustration



- *(Schölkopf et al1998)* – Kernel PCA with Gaussian kernel – first 8 eigenvalues
- contour lines = points with equal projection on corresponding eigenvector
- first two eigenvectors, separate the 3 main clusters
- following eigenvectors split cluster into halves; and further 3 as well (along orthogonal directions)

Kernel PCA - Summary

- Given a set of data points, stacked as X

$$\text{compute } \mathbf{K} \text{ and then } \tilde{\mathbf{K}}$$

$$\tilde{\mathbf{K}} = \mathbf{K} - \frac{1}{N}\mathbf{K}\mathbf{1}\mathbf{1}^T - \frac{1}{N}\mathbf{1}\mathbf{1}^T\mathbf{K} + \frac{1}{N^2}\mathbf{1}\mathbf{1}^T\mathbf{K}\mathbf{1}\mathbf{1}^T$$

- Compute the eigenvectors and eigenvalues $\tilde{\mathbf{K}}\mathbf{a}_i = \lambda_i \mathbf{a}_i$

- Normalize them properly $\lambda_i \mathbf{a}_i^T \mathbf{a}_i = 1$

- Projection of a new data point onto the principal components

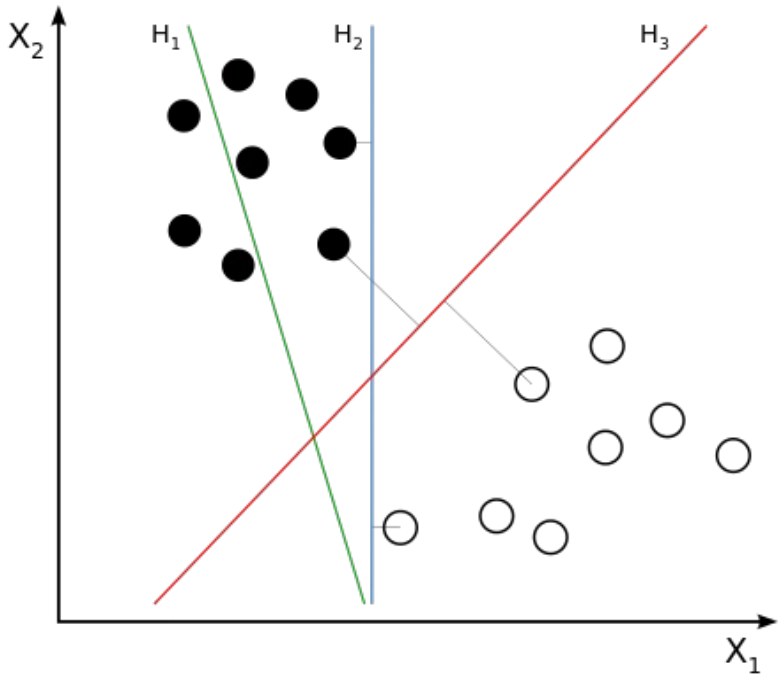
$$y_i(\mathbf{x}) = \sum_{l=1}^{N} a_{il}\, k(\mathbf{x}, \mathbf{x}_l)$$

overview

- Kernel methods
  - introduction and main elements
  - defining kernels
  - Kernelization of k-NN, K-Means, PCA

- **Support Vector Machines (SVMs)**
  - classification
  - regression

Support Vector Machines (SVM) - principle

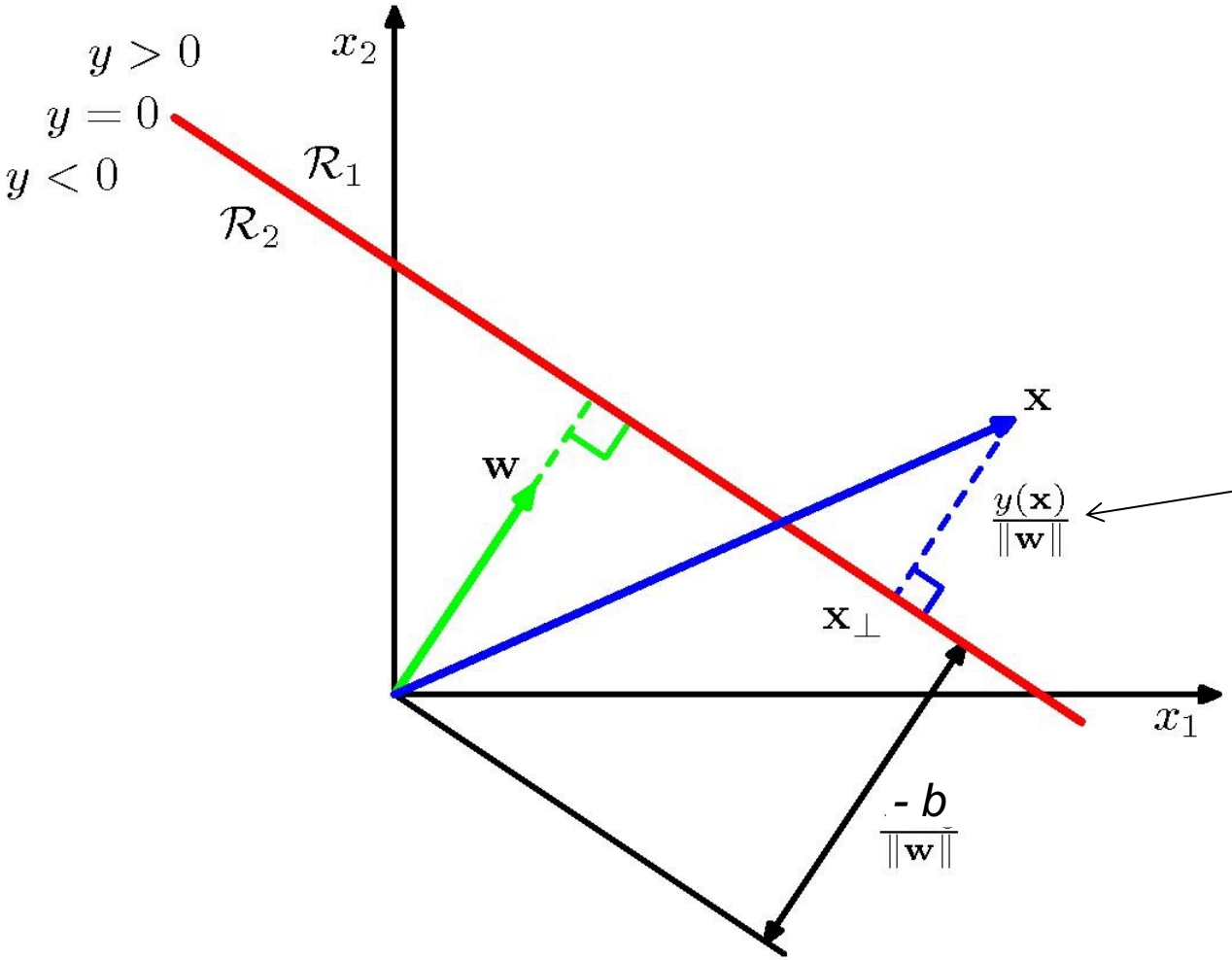


Main idea: look at the margin !

- H1: does not separate the classes
- H2: separate classes, but by a small margin
- H3: maximum margin

- separable data: several classifiers available. Which one is the best?
- perceptron: classifier depends on initialization, order of visit of datapoints
- margin
  - distance from the closest datapoint to the decision boundary
  - why do we want a large margin?
    - **classification more immune to small perturbation of the datapoints**

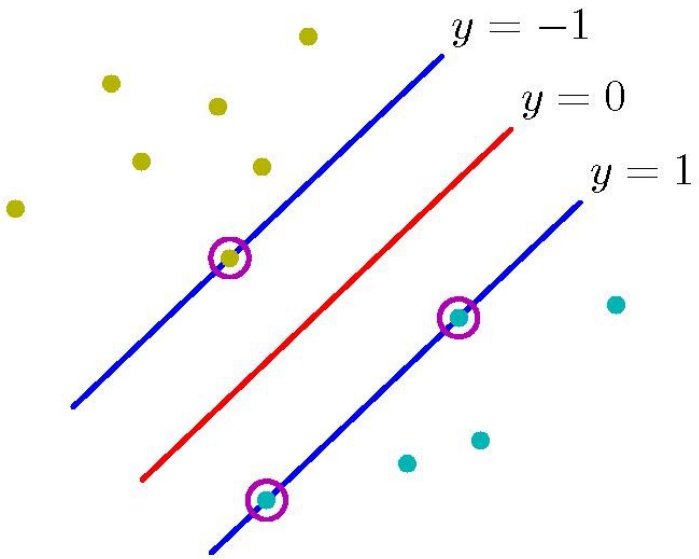
SVM – margin geometry



signed distance
to decision boundary

- linear decision function $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$

SVM – problem setting



- assume a linearly **separable dataset**
  (separable in a high dimensional space)

$$\mathcal{D} = \{(\mathbf{x}_i, t_i) | t_i \in \{-1, +1\}, i = 1, ..., N\}$$

- linear classifier

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b$$

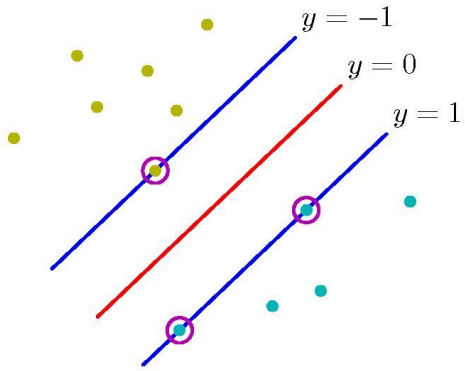
s.t. if y(**x**) > 0 then t = 1 otherwise t = -1

  - distance of point $\mathbf{x}_i$ to decision surface = $\dfrac{t_i y(\mathbf{x}_i)}{\|\mathbf{w}\|}$

- **Goal** : find parameters resulting in the maximum margin !
  (max of the minimum distance to the decision surface)

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i \left[t_i y(\mathbf{x}_i) \right] \right\} \qquad \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i \left[t_i (\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b) \right] \right\}$$

SVM – max margin



- Max-margin optimization

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_i \left[t_i(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b) \right] \right\}$$

- Note: rescaling **w** and b by s does not change the solution
  - use that to constrain the problem
    - set closest points (they exist) to the decision surface as $t_i(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b) = 1$
    - all other points are further away
    - note: margin (on one side) = $\frac{1}{\|\mathbf{w}\|}$

- Max-margin problem

$$\begin{cases} \arg\min \frac{1}{2}\|\mathbf{w}\|^2 & \text{subject to} \\ t_i(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_i) + b) \geq 1 & \forall i = 1, \ldots, N \end{cases}$$

  - quadratic programming (QP) problem:
    minimizing quadratic function subject to constraints
    => their exist QP solver libraries.

SVM – Lagrangian duality

- Primal optimization problem $\begin{cases} \min_w & f(w) \\ s.t. & g_i(w) \leq 0, i = 1, \ldots, k \\ & h_i(w) = 0, i = 1, \ldots, l \end{cases}$
  - introduce generalized Lagrangian $\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$
  - primal problem

$$\theta_{\mathcal{P}}(w) = \max_{\alpha,\beta|\alpha\geq 0} \mathcal{L}(w, \alpha, \beta) \qquad \min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha,\beta|\alpha\geq 0} \mathcal{L}(w, \alpha, \beta)$$

  - Note:
    - for w which do not verify the constraints, the primal is infinity;
    - otherwise it is equal to f(w)

=> primal might then be ill-defined in this case => consider the dual

SVM – Lagrangian duality

- Primal optimization problem

$$\begin{cases} \min_w & f(w) \\ s.t. & g_i(w) \leq 0, i = 1, \ldots, k \\ & h_i(w) = 0, i = 1, \ldots, l \end{cases}$$

  - introduce generalized Lagrangian $\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$
  - primal problem

$$\theta_{\mathcal{P}}(w) = \max_{\alpha,\beta|\alpha\geq 0} \mathcal{L}(w, \alpha, \beta) \qquad \min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha,\beta|\alpha\geq 0} \mathcal{L}(w, \alpha, \beta)$$

- Dual optimization problem

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta) \qquad \max_{\alpha,\beta|\alpha\geq 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha,\beta|\alpha\geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$$

- Under certain constraints (f and g_i convex, h_i affine; constraints are feasible)
  - dual problem leads to the same solution than the primal
  - solution satisfies (necessary and sufficient condition)
  - Karush-Kuhn-Tucker conditions

$$\begin{cases} \frac{\partial \mathcal{L}(w^\star, \alpha^\star, \beta^\star)}{\partial w_i} & = & 0\,, i = 1, \ldots n \\ \frac{\partial \mathcal{L}(w^\star, \alpha^\star, \beta^\star)}{\partial \beta_i} & = & 0\,, i = 1, \ldots l \\ \alpha_i^\star g_i(w^\star) & = & 0\,, i = 1, \ldots k \\ g_i(w^\star) & \leq & 0\,, i = 1, \ldots k \\ \alpha_i^\star & \geq & 0\,, i = 1, \ldots k \end{cases}$$

SVM – Dual form

$$\begin{cases} \arg\min \frac{1}{2}\|\mathbf{w}\|^2 & \text{subject to} \\ t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)+b) \geq 1 & \forall i = 1,\ldots,N \end{cases}$$

$$\begin{cases} \frac{\partial \mathcal{L}(w^\star,\alpha^\star,\beta^\star)}{\partial w_i} = 0\,, i=1,\ldots n \\ \frac{\partial \mathcal{L}(w^\star,\alpha^\star,\beta^\star)}{\partial \beta_i} = 0\,, i=1,\ldots l \\ \alpha_i^\star g_i(w^\star) = 0\,, i=1,\ldots k \\ g_i(w^\star) \leq 0\,, i=1,\ldots k \\ \alpha_i^\star \geq 0\,, i=1,\ldots k \end{cases}$$

- Primal problem
  - note: constraint is positive
  - Lagrangian $$\mathcal{L}(\mathbf{w},b;\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{1=1}^{N} a_i\left\{t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)+b)-1\right\}$$
- Dual problem: given **a**, minimize w.r.t. the weights and bias => derivatives

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}}(\mathbf{w},b;\mathbf{a}) = \mathbf{w} - \sum_i a_i t_i \boldsymbol{\phi}(\mathbf{x}_i) = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

$$\frac{\partial \mathcal{L}}{\partial b}(\mathbf{w},b;\mathbf{a}) = \sum_{i=1}^{N} a_i t_i = 0$$

  - Weights = linear combination of the projected datapoints
- We can substitute w in the lagrangian

SVM – Dual form

$$\begin{cases} \frac{\partial \mathcal{L}(w^\star,\alpha^\star,\beta^\star)}{\partial w_i} &= 0\,,\ i=1,\dots n \\ \frac{\partial \mathcal{L}(w^\star,\alpha^\star,\beta^\star)}{\partial \beta_i} &= 0\,,\ i=1,\dots l \\ \alpha_i^\star g_i(w^\star) &= 0\,,\ i=1,\dots k \\ g_i(w^\star) &\le 0\,,\ i=1,\dots k \\ \alpha_i^\star &\ge 0\,,\ i=1,\dots k \end{cases}$$

- Primal problem
  - Lagrangian $\mathcal{L}(\mathbf{w},b;\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{1=1}^{N} a_i \left\{t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)+b)-1\right\}$

$$\mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

- Substitution of W in the lagrangian => following problem

$$\begin{cases} \max_{\mathbf{a}} \tilde{\mathcal{L}}(\mathbf{a}) &= \sum_{i=1}^N a_i - \frac{1}{2}\sum_{i=1}^N\sum_{l=1}^N a_i a_l t_i t_l k(\mathbf{x}_i,\mathbf{x}_l) \quad \text{subject to} \\ \sum_{i=1}^N a_i t_i &= 0 \\ a_i &\ge 0, i=1,\dots,N \\ t_i y(\mathbf{x}_i) - 1 &\ge 0, i=1,\dots,N \\ a_i(t_i y(\mathbf{x}_i)-1) &= 0, i=1,\dots,N \end{cases}$$

  - last inequality:
    - either **point is on the margin** (constraint satisfied with an equality)

      then a_i will be non 0, we have a support vector
    - or the **point is not on the margin** (the point is beyond the margin)

      then the only way to satisfy the **constraint is to have $a_i = 0$**

      => it does not participate in defining the weights

SVM - discussion

$$\mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)$$

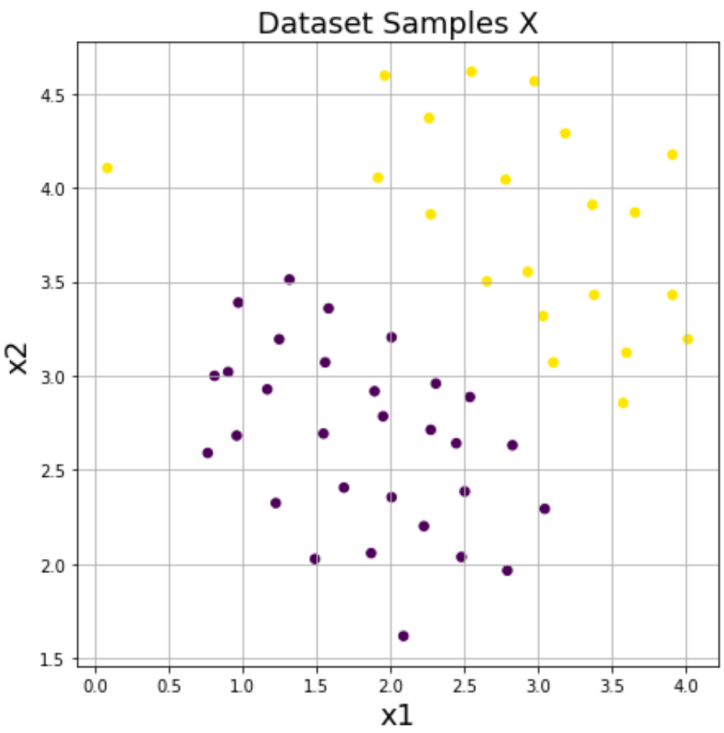
- Interest of using the Dual form
  - allows to introduce the kernel
  - **unique solution – quadratic optimization = no dependency on initialization**
- Computation of a new score (and classification)
  - **weights as linear combination of projected data point**
  - **score only expressed through the kernel**
  - the sum needs to run only on the set of Support Vectors

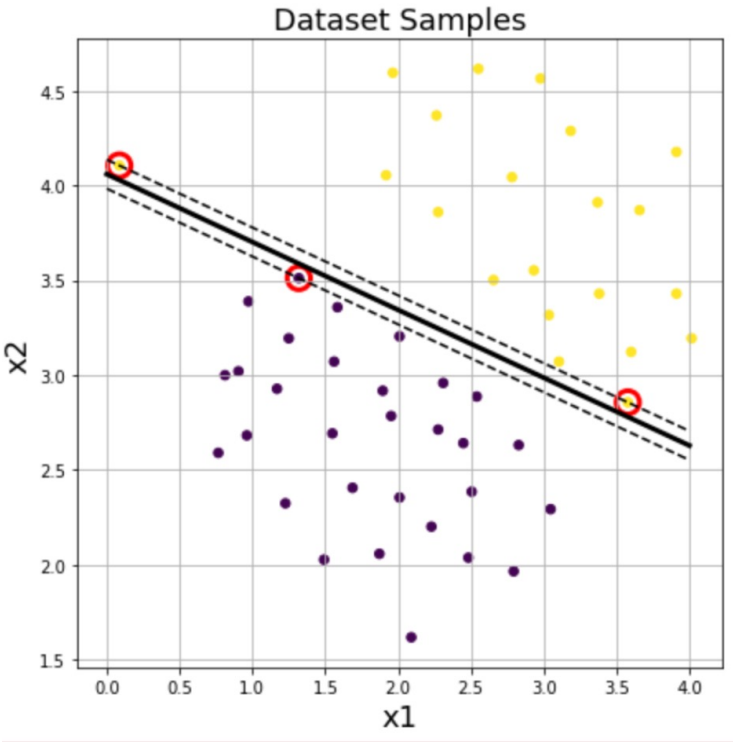
$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}) + b = \sum_{i \in \mathcal{S}} a_i t_i k(\mathbf{x}_i, \mathbf{x}) + b$$

- Bias computation
  - can be computed from any satisfied constraint, ie on support vectors
  - average on all support vectors

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{i \in \mathcal{S}} \left(t_i - \sum_{l \in \mathcal{S}} a_l t_l k(\mathbf{x}_l, \mathbf{x}_i) \right)$$

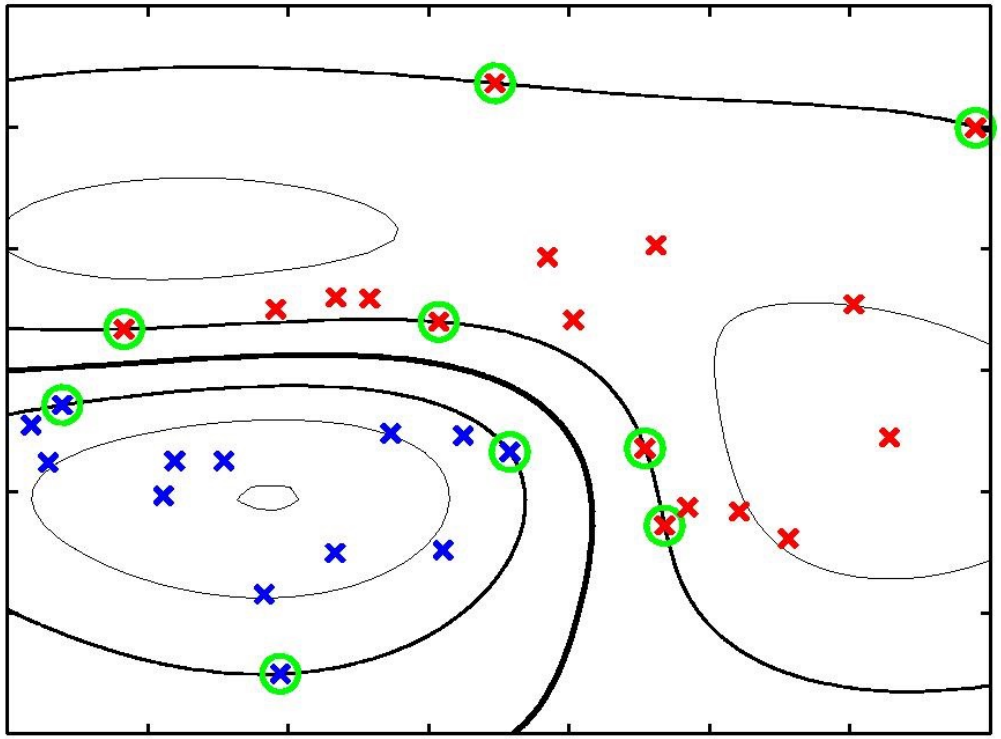
SVM - illustration





- Illustration with standard dot product as kernel
- Shown: decision boundary, plus margins
- Support Vectors (with non-zero weights) are on margin curves

SVM - illustration

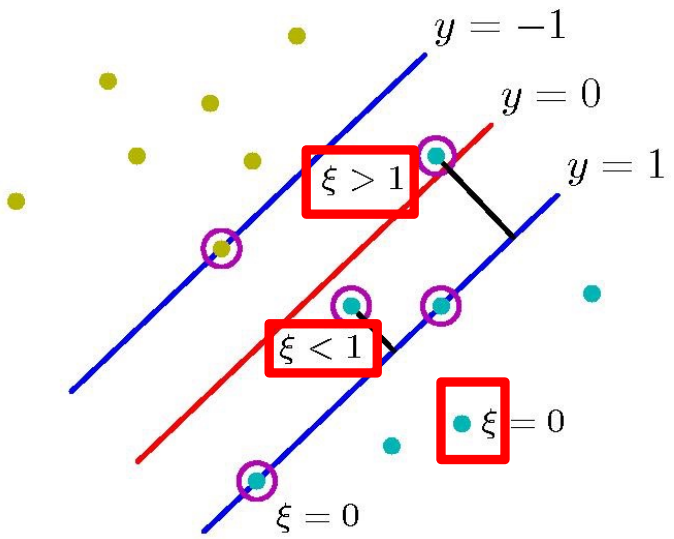


- Illustration with Radial Basis Function
- Shown: decision boundary, plus margins
- Support Vectors (with non-zero weights) are on margin curves

SVM

- So far, we assumed that the data was separable
  - not always possible
  - not always desirable
- Can the model extend to the non-separable case?
  - how to keep the notion of margin?

SVM – non separable case

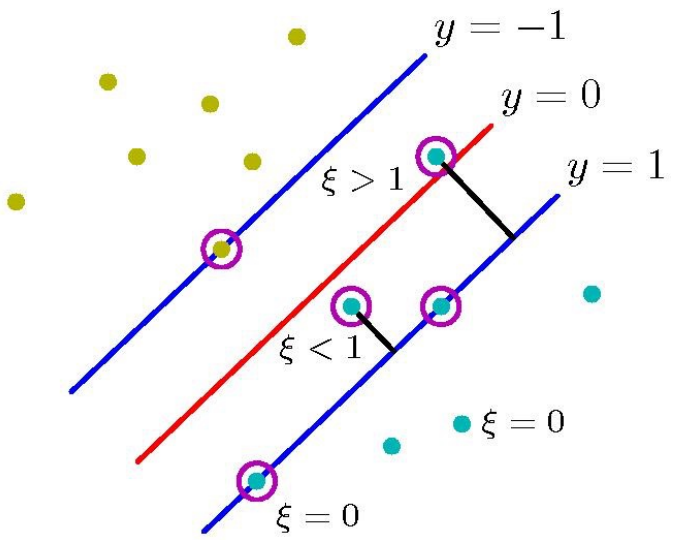


- We need to take into account the errors
- Use a soft margin instead of a hard margin
- How to measure errors and deviations?
  - add new variables – called slack variables
- Look at the points
  - points beyond the margin: no error
  - points within the margins : we want to penalize it, even if this is not an error

SVM – non separable case – formulation

- Separable case

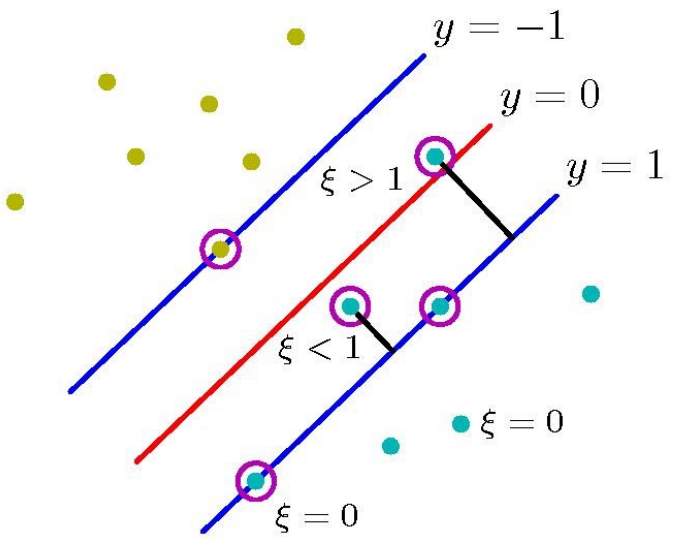
$$\begin{cases} \arg\min \frac{1}{2}\|\mathbf{w}\|^2 & \text{subject to} \\ t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)+b) \geq 1 & \forall i = 1,\ldots,N \end{cases}$$



SVM – non separable case – formulation

- Primal problem

$$\begin{cases} \arg\min_{\mathbf{w},b} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \xi_i \right) & \text{subject to} \\ t_i y(\mathbf{x}_i) \geq 1 - \xi_i \qquad \forall i = 1, \ldots, N \\ \xi_i \geq 0 \end{cases}$$



$$\arg\min_{\mathbf{w},b} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max(0, 1 - t_i y(\mathbf{x}_i)) \right)$$

- introduced variables are ***slack*** variables – their sum provides an upper bound on the error
- framework is sensitive to outliers : errors grows linearly with distance
- C is analagous to (the inverse of) a regularisation coefficient. It controls the trade-off between model complexity (the margin) and training errors
- when C ➔ ∞, we recover the separable case

SVM – non separable case – dual form

- Lagrangian $\mathcal{L}(\mathbf{w}, b, \xi; \mathbf{a}, \mathbf{r}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i \left\{t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) + b) - 1 + \xi_i\right\} - \sum_{i=1}^{N} r_i\xi_i$

- Derivating w.r.t. weights, bias, and slack variables

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i) \qquad \frac{\partial\mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} a_i t_i = 0 \qquad \frac{\partial\mathcal{L}}{\partial\xi_i} = 0 \Rightarrow a_i = C - r_i$$

- Note: in lagrangian, slack variables only appear in linear form

$$\mathcal{L}(\mathbf{w}, b, \xi; \mathbf{a}, \mathbf{r}) = \ldots + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i\xi_i - \sum_{i=1}^{N} r_i\xi_i = \ldots + \sum_{i=1}^{N}(C - a_i - r_i)\xi_i$$

  - so taking into account the constraint linked to setting the derivative to 0, the slack variables will vanish from the lagrangian at the optimum

SVM – non-separable case – dual form

- Lagrangian $\mathcal{L}(\mathbf{w}, b, \xi; \mathbf{a}, \mathbf{r}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N} a_i \left\{t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) + b) - 1 + \xi_i\right\} - \sum_{i=1}^{N} r_i\xi_i$

- Derivating w.r.t. weights, bias, and slack variables

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} a_i t_i \boldsymbol{\phi}(\mathbf{x}_i) \qquad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} a_i t_i = 0 \qquad \frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \Rightarrow a_i = C - r_i$$

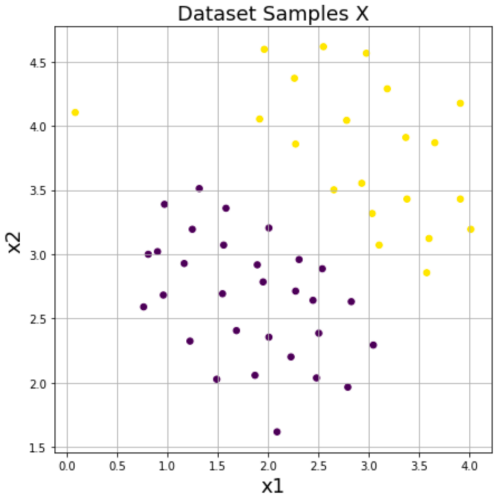
- We end with the dual problem, very similar to the separable case

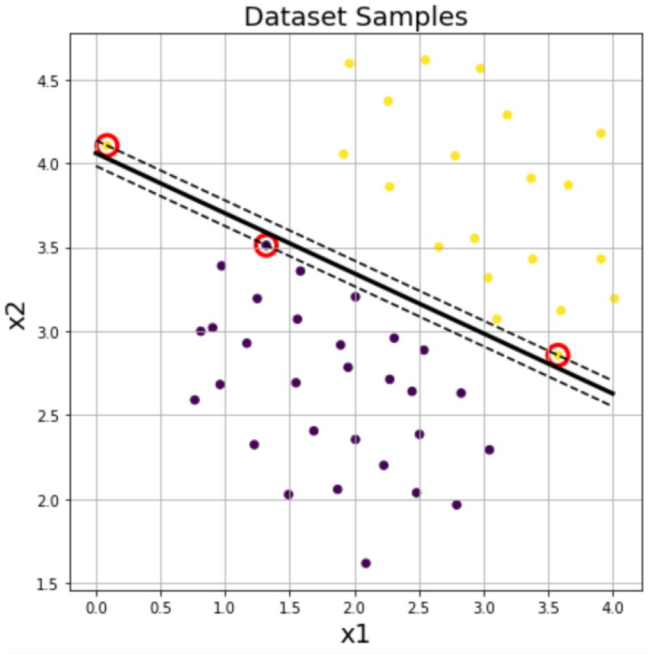
$$\begin{cases} \max_{\mathbf{a}} \tilde{\mathcal{L}}(\mathbf{a}) = \sum_{i=1}^{N} a_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{l=1}^{N} a_i a_l t_i t_l k(\mathbf{x}_i, \mathbf{x}_l) \quad \text{subject to} \\ \sum_{i=1}^{N} a_i t_i = 0 \\ 0 \leq a_i \leq C, i = 1, \ldots, N \\ a_i(t_i y(\mathbf{x}_i) - 1 + \xi_i) = 0 \text{ and } \xi_i r_i = 0, \ i = 1, \ldots, N \end{cases}$$

$$\begin{cases} \frac{\partial \mathcal{L}(w^\star, \alpha^\star, \beta^\star)}{\partial w_i} = 0, i = 1, \ldots n \\ \frac{\partial \mathcal{L}(w^\star, \alpha^\star, \beta^\star)}{\partial \beta_i} = 0, i = 1, \ldots l \\ \alpha_i^\star g_i(w^\star) = 0, i = 1, \ldots k \\ g_i(w^\star) \leq 0, i = 1, \ldots k \\ \alpha_i^\star \geq 0, i = 1, \ldots k \end{cases}$$

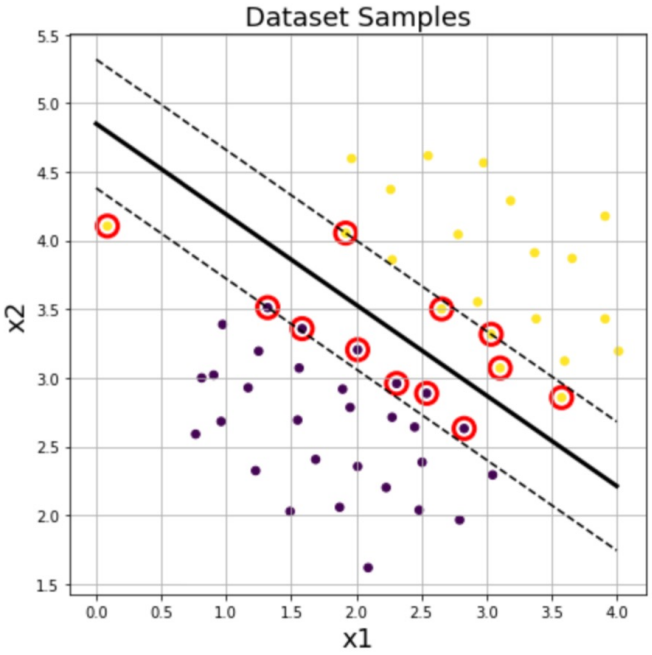
  - prediction formula is the same than in the separable case
  - some a_i will be 0 and will not contribute to the prediction; the rest will be Support Vectors
  - if $a_i < C$, then $r_i > 0$ and thus the slack variable $\xi_i = 0$ => the data are on the margin
  - $a_i = C$,then $r_i = 0$: point will lie within the margin (well classified or not) or on the opposite side

SVM - illustration





Hard margin



Soft margin

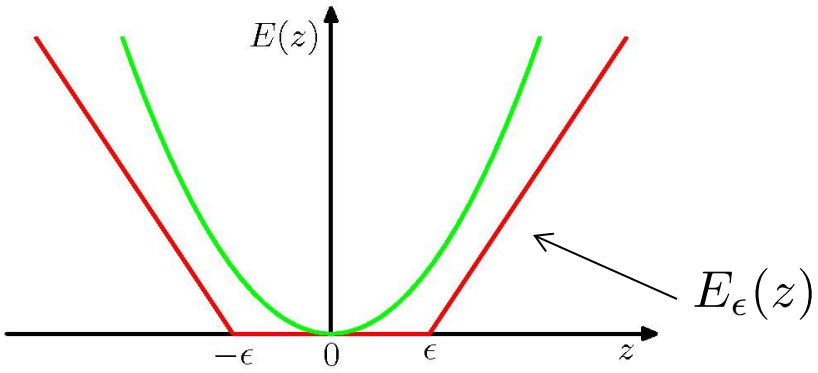
- Illustration with standard dot product as kernel
- Shown: decision boundary, plus margins
- Support Vectors (with non-zero weights) are on margin curves

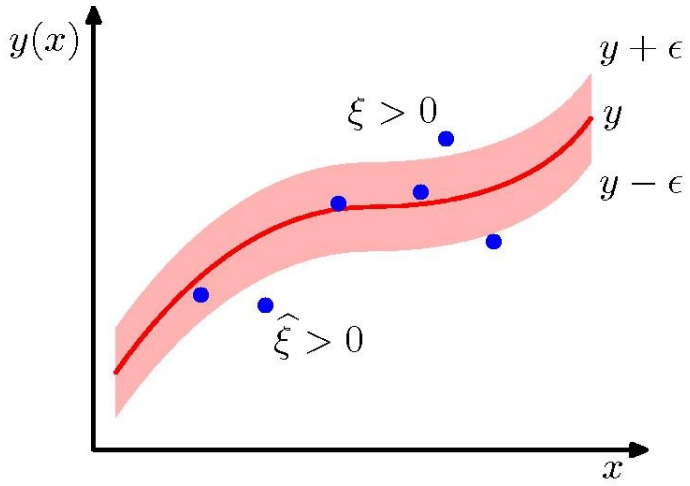
More in the laboratory !

Support Vector Machines (SVM)

The regression case

SVM – the regression case





- Idea: fit the training data using an ε-insensitive error function

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} E_\epsilon(y(\mathbf{x}_i) - t_i)$$

- As before, introduce relaxed constraints, resulting in primal:

$$\begin{cases} \arg\min_{\mathbf{w},b} & \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \hat{\xi}_i)\right) & \text{subject to} \\ & t_i \le y(\mathbf{x}_i) + \epsilon + \xi_i, & \xi_i \ge 0 & \forall i = 1,\ldots,N \\ & t_i \ge y(\mathbf{x}_i) - \epsilon - \hat{\xi}_i, & \hat{\xi}_i \ge 0 & \forall i = 1,\ldots,N \end{cases}$$

SVM – regression case – dual form

- Introducing Lagrangian variables, we end up maximizing

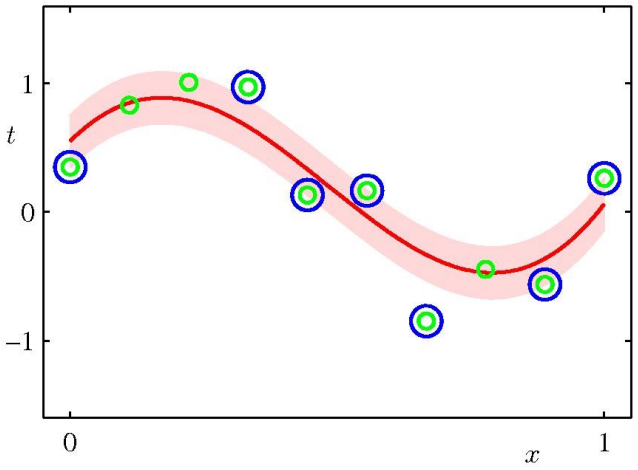
$$\begin{cases} \max_{\mathbf{a},\hat{\mathbf{a}}} \tilde{\mathcal{L}}(\mathbf{a},\hat{\mathbf{a}}) = \sum_{i=1}^{N}(a_i-\hat{a}_i)t_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{l=1}^{N}(a_i-\hat{a}_i)\sum_{i=1}^{N}(a_l-\hat{a}_l)k(\mathbf{x}_i,\mathbf{x}_l) - \epsilon\sum_{i=1}^{N}(a_i+\hat{a}_i) \quad \text{subject} \\ \sum_{i=1}^{N}(a_i-\hat{a}_i) = 0 \\ 0 \leq a_i, \hat{a}_i \leq C \end{cases}$$

- Weights are still obtained as linear combination:

$$\mathbf{w} = \sum_{i=1}^{N}(a_i-\hat{a}_i)\boldsymbol{\phi}(\mathbf{x}_i)$$

- Score of a new observation

$$y(\mathbf{x}) = \sum_{i=1}^{N}(a_i-\hat{a}_i)k(\mathbf{x},\mathbf{x}_i) + b$$



Support Vector Machines (SVM)

Optimization and note

SVM - optimization

- Both the classification and regression can be viewed as a minimization of the form

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^T Q \mathbf{a} - \beta^T \mathbf{a}$$

under the constraints

$$\mathbf{a}^T \gamma = 0$$
$$C_{min} \leq \mathbf{a} \leq C_{max}$$

- This problem is quadratic, convex, and in $0(N^3)$

SVM – optimization

Sequential Minimum Optimization (SMO) algorithm

- Can we do coordinate descent with one variable?
  - no: first constraint imposes that when N-1 parameters are known/fixed, the last one can only be set to a single value to satisfy the constraint
- idea: **optimize with respect to two variables** a_i and a_j (other are fixed) – constraints are reduced to

$$a_i\gamma_i + a_j\gamma_j = c_{ij}$$

$$C_{min} \leq a_i, a_j \leq C_{max}$$

  - and the optimization problem can be solved analytically
- Choosing pairs of a_i and a_j
  - consider the strongest gradient $g_i = [Qa-\beta]_i$
  - make sure going towards these gradient directions will not hit the bounds

$$C_{min} \leq \mathbf{a}_i - \lambda g_i, \mathbf{a}_j - \lambda g_j \leq C_{max}$$

  - $g_i\,\gamma_i$ and $g_j\,\gamma_j$ must point to opposite directions
- Cost about $0(N^2)$

Kernel Machines & sparsity

- Other existing Kernel Machines

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i k(\mathbf{x}, \mathbf{x}_i) + b \qquad p(t|\mathbf{x}, \theta) = \mathcal{N}(t; y(\mathbf{x}), \sigma^2)$$

$$L(\mathbf{w}, b) = L_{nll}(\mathcal{D}) + \lambda \|\mathbf{w}\|_1$$

  - directly express output as linear combination and estimate the weights
  - fit a probabilistic model (e.g. as in logistic regression)
  - use negative log-likelihood measure
  - optimize penalized loss using explicit sparsity
  - advantage: no need for Mercer Kernel, explicit sparsity, probabilistic interpretation, better extension to Multiple classes

- Relevance Vector Machines (RVMs)
  - other penalization function

Support Vector Machines (SVM)

Summary

SVM - Summary

- Classification SVM – why do we like them
  - finds **the largest margin** separating hyperplane
  - uses the **soft margin** trick
  - there is a **unique solution (no initialization issue)**
  - project data **in a high dimensional space for non-linear relation**
  - **a kernel** simplifies computation
  - indirectly **induces sparsity** of support vectors
- Can work with fairly large datasets (few ten of thousands)

- Drawback:
  - Complexity/computational can be high

SVM - Summary

- It leads to a **quadratic** (convex) minimization problem
- The **capacity (to fit) can be controlled** in several ways
  - C : controls the trade-off classification error/margin
  - Kernel choice
  - Kernel parameters, if any
- The idea can be **generalized to regression**
- **Other sparser methods**
  - Relevance Vector Machines
  - L1 regularization kernel machines

Thank you for your attention!