Data Mining

Predictive Modelling

Support Vector Machines

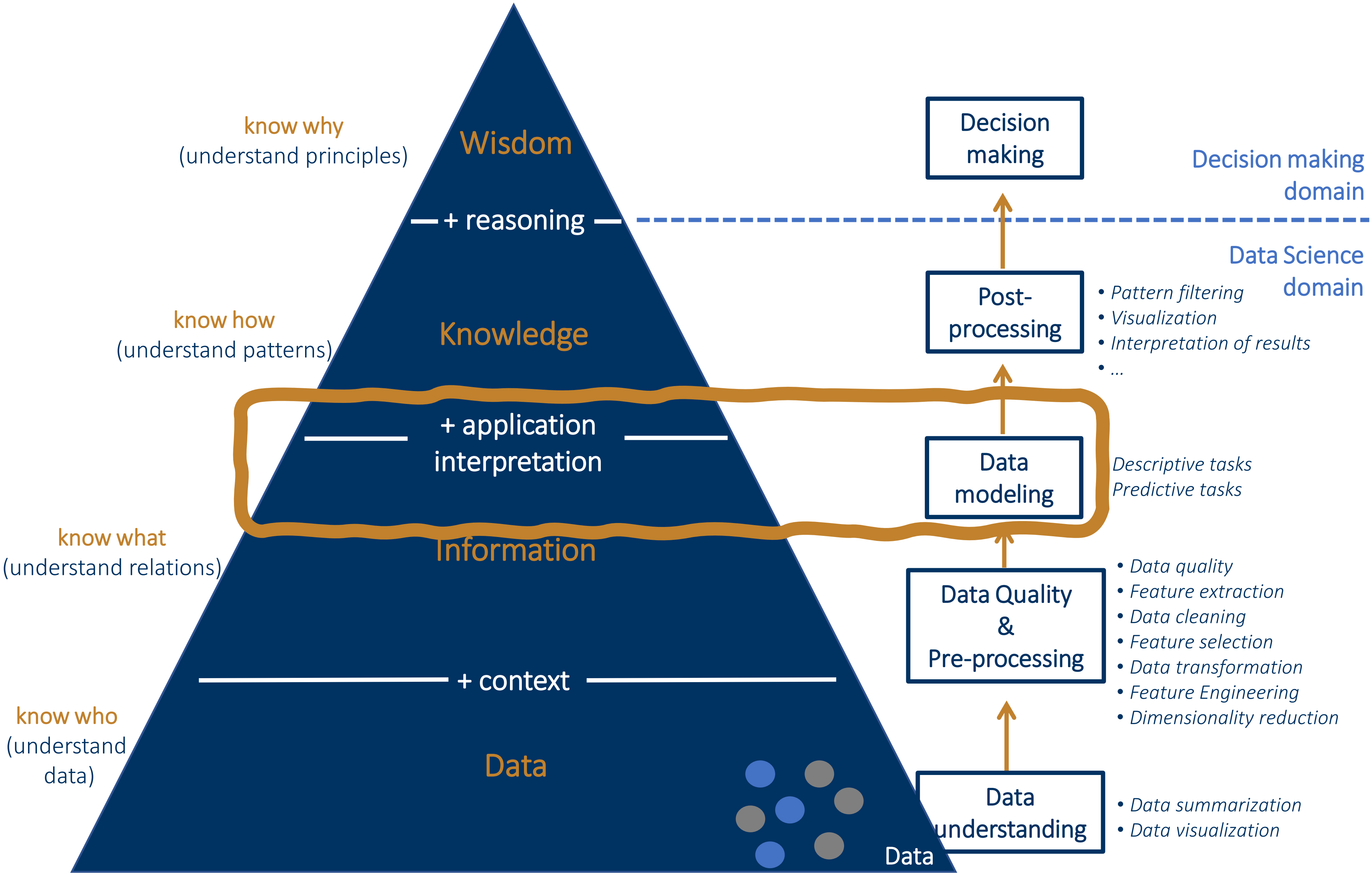
Raquel Sebastião

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

raquel.sebastiao@ua.pt

2022/2023



know why
(understand principles)
Wisdom
+ reasoning
know how
(understand patterns)
Knowledge
+ application
interpretation
know what
(understand relations)
Information
+ context
know who
(understand
data)
Data
Data
Decision making
Decision making domain
Data Science domain
Post-processing
• Pattern filtering
• Visualization
• Interpretation of results
• ...
Data modeling
Descriptive tasks
Predictive tasks
Data Quality & Pre-processing
• Data quality
• Feature extraction
• Data cleaning
• Feature selection
• Data transformation
• Feature Engineering
• Dimensionality reduction
Data understanding
• Data summarization
• Data visualization

Contents

Prediction Models – approaches

Geometric approaches

- **Distance-based:** kNN
- **Linear models:** Fisher's linear discriminant, perceptron, logistic regression, SVM (w. linear kernel)

Probabilistic approaches

- naive Bayes, logistic regression

Logical approaches

- classification or regression trees, rules

Optimization approaches

- neural networks, SVM

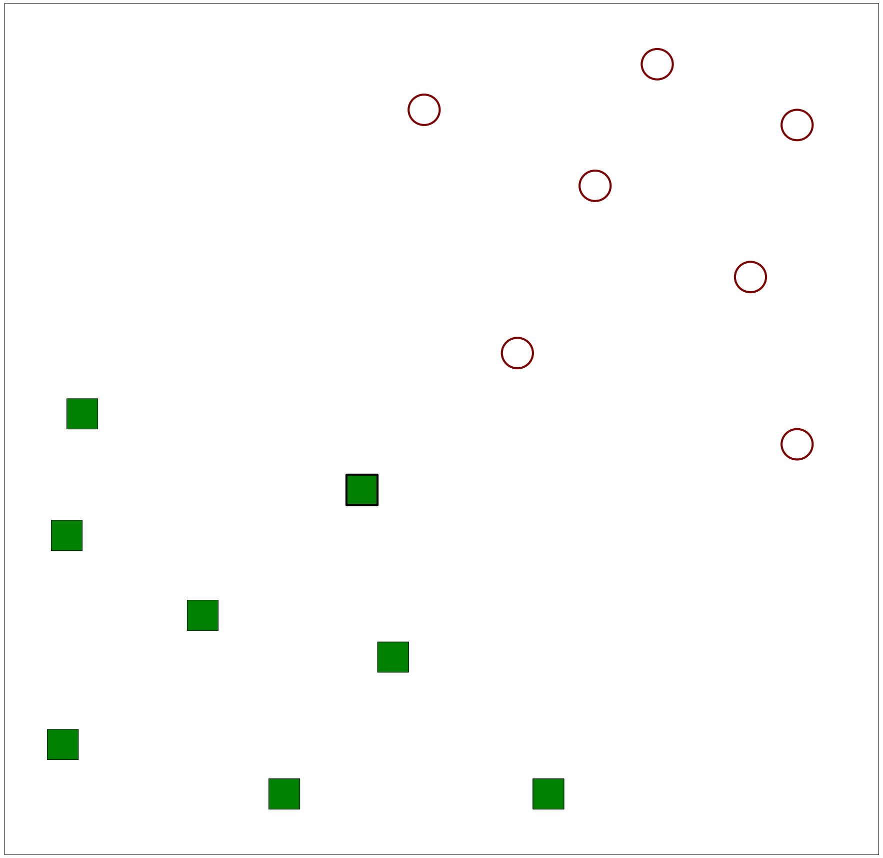
Sets of models (ensembles)

- random forests, adaBoost

Support Vector Machines

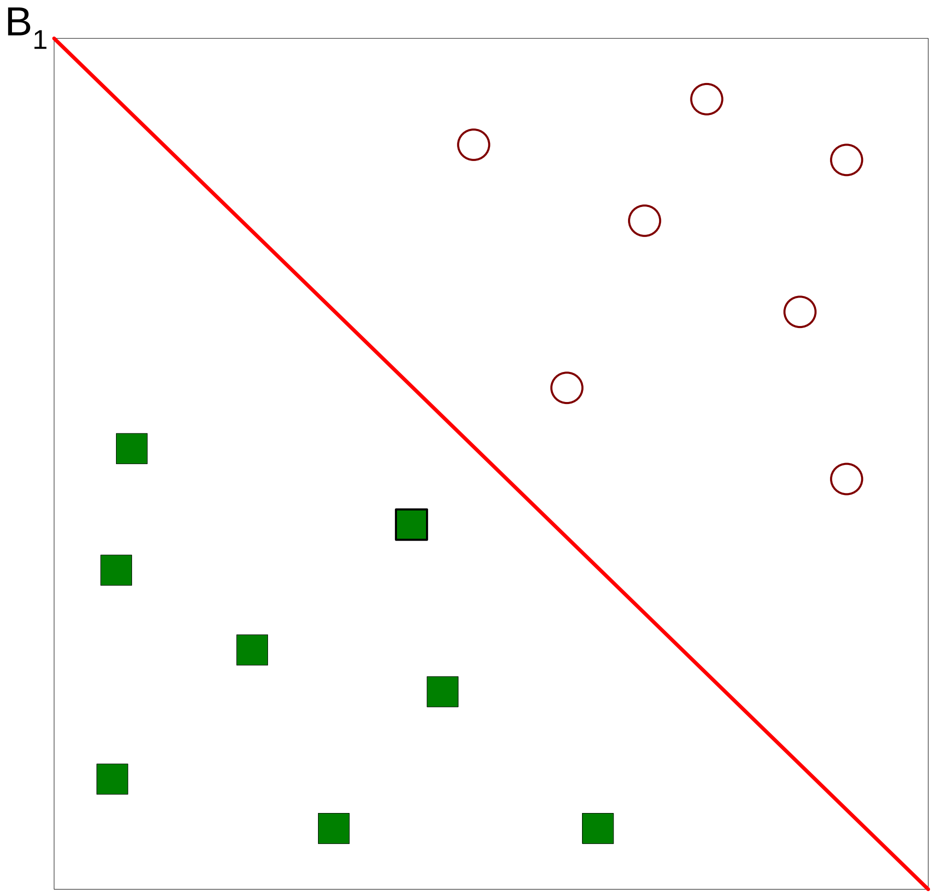
- "Relative recent"
  - introduced in 1992 (@COLT-92 conf)
- Gave origin to a new class of algorithms named kernel machines
- SVMs have been applied with success in a wide range of areas:
  - Bioinformatics
  - text mining
  - hand-written character recognition
  - Biometric recognition

Support Vector Machines



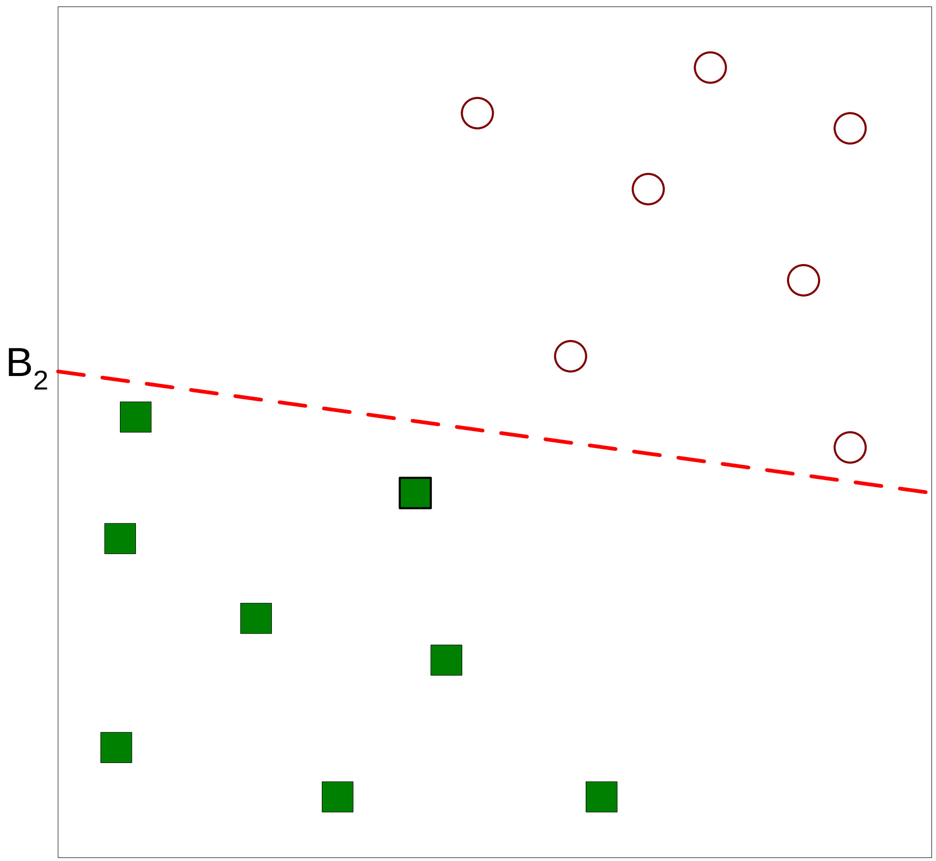
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



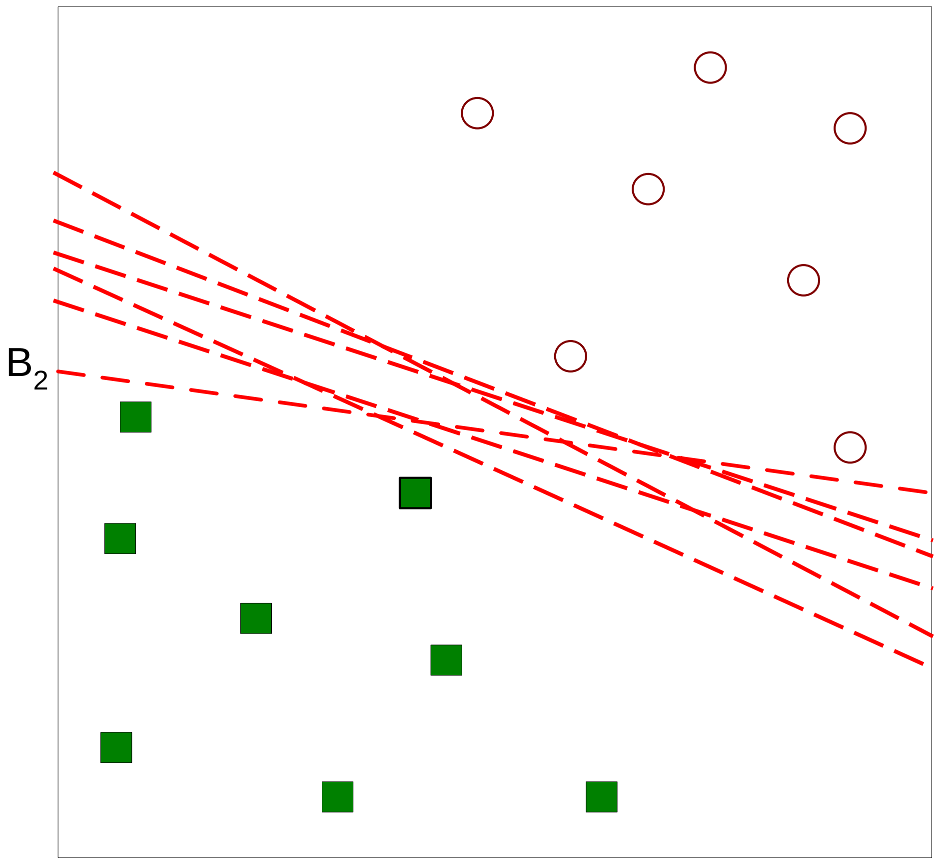
- One Possible Solution

Support Vector Machines



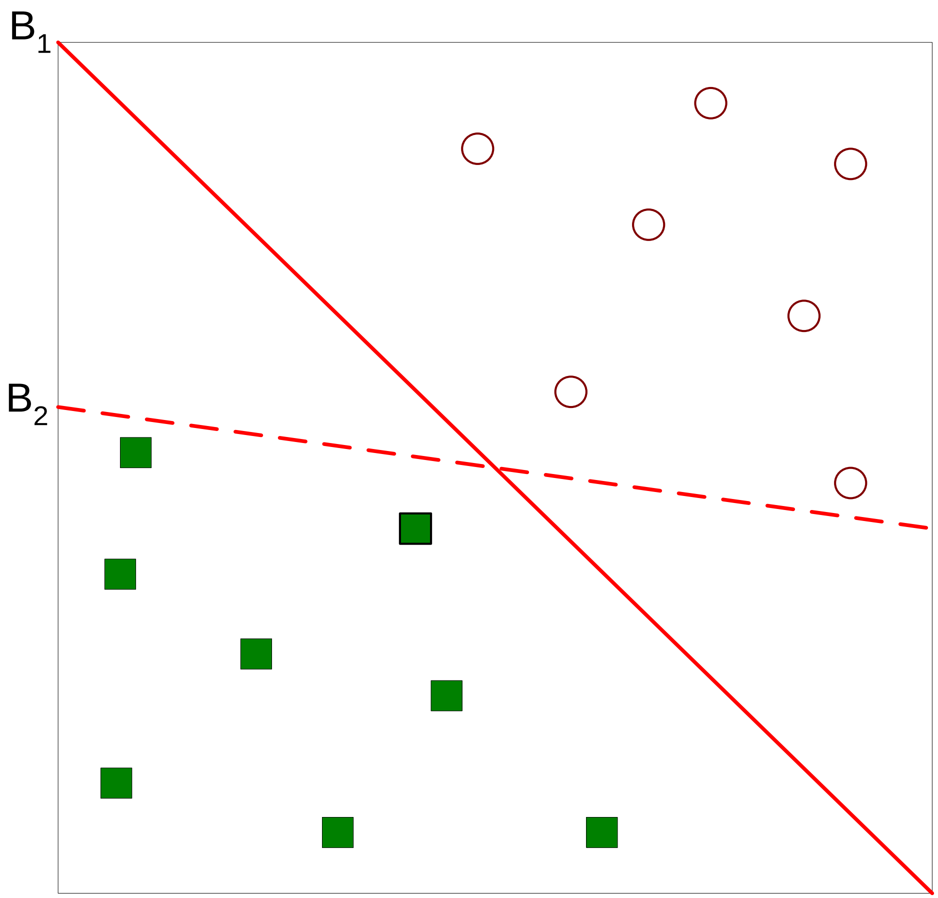
- Another possible solution

Support Vector Machines



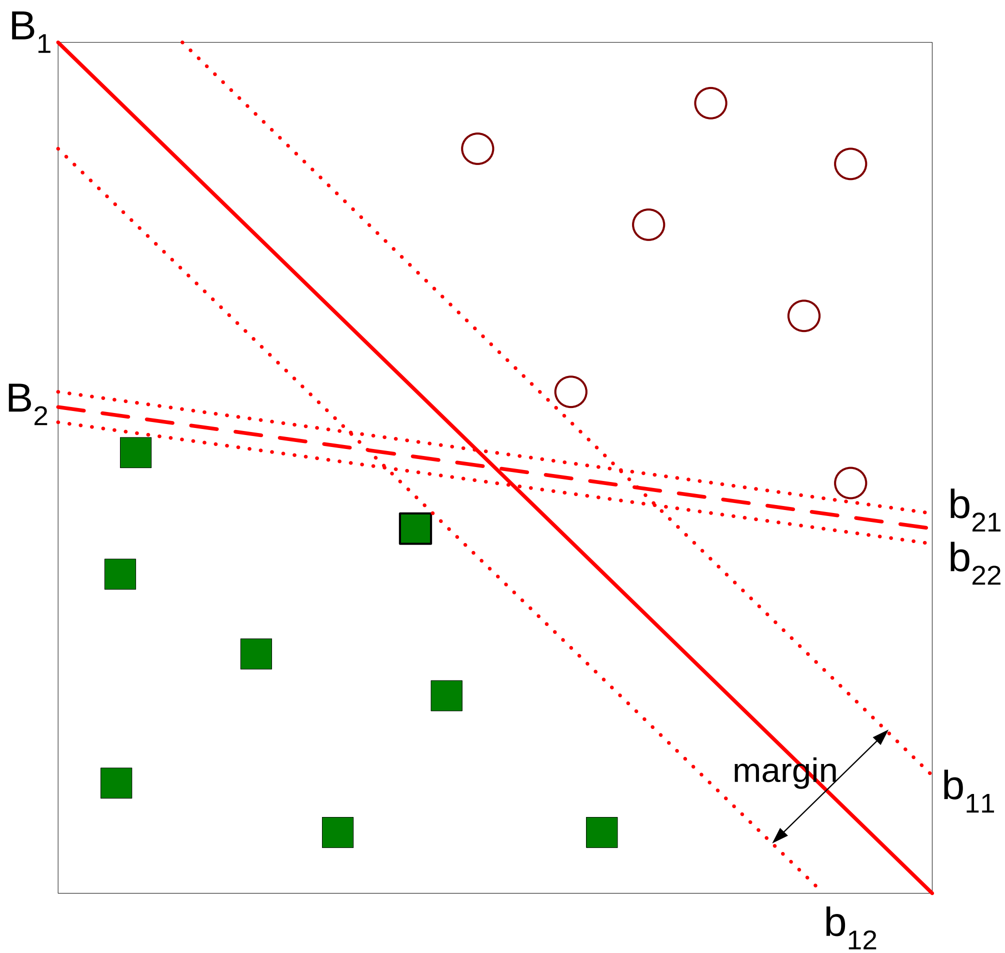
- Other possible solutions

Support Vector Machines



- Which one is **better**? B1 or B2?
- **How to define better?**

Support Vector Machines

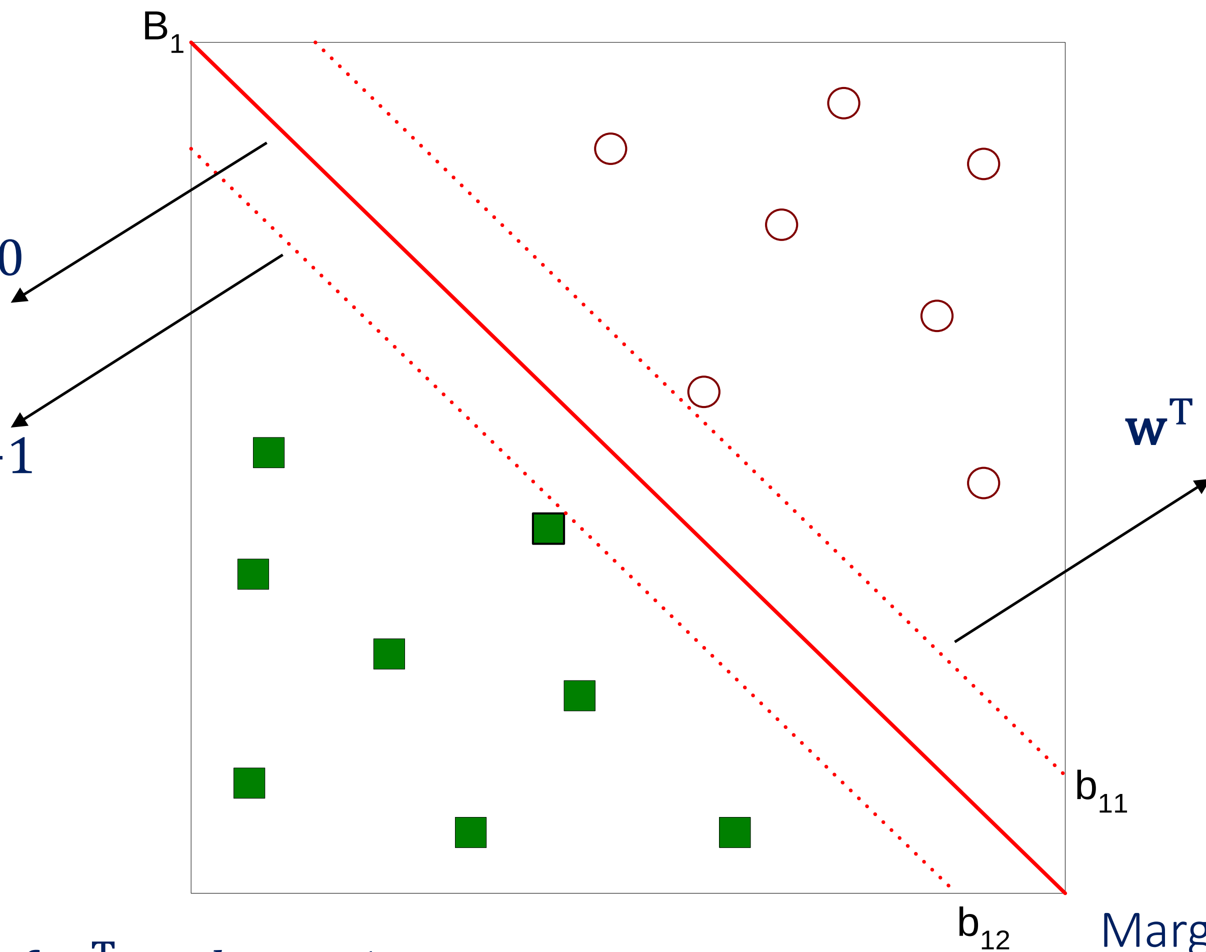


- Find hyperplane **maximizes** the margin => **B1 is better than B2**

Support Vector Machines

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = -1$$



$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = +1$$

$$\text{Margin} = \frac{2}{||\mathbf{w}||}$$

$$g(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \geq 1 \\ -1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \leq -1 \end{cases}$$

Contents

Linear Support Vector Machines

- Linear model:

$$g(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \geq \ \ 1 \\ -1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \leq -1 \end{cases}$$

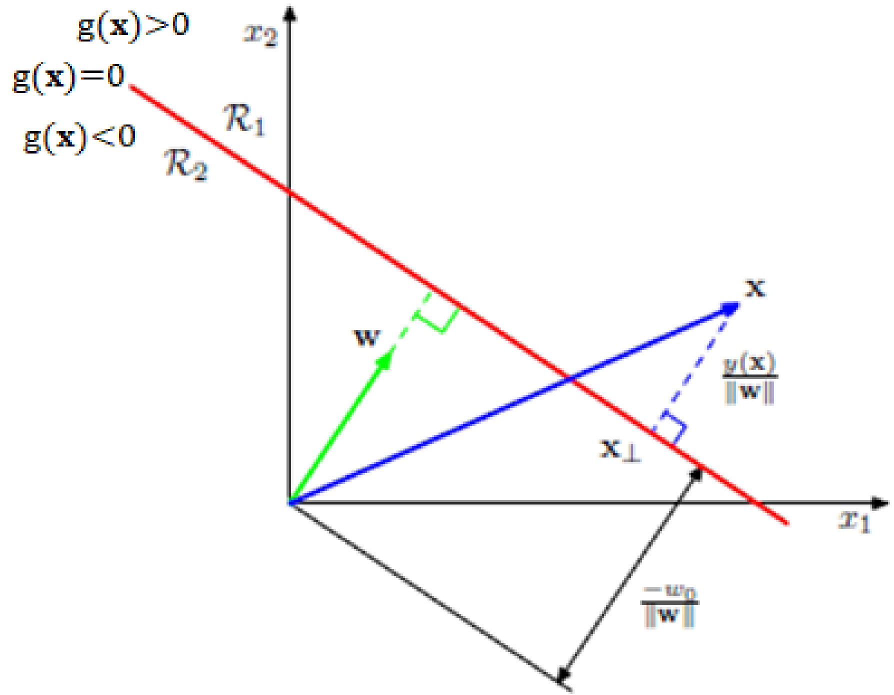
  - Target labels = {-1,1}

- Learning the model is equivalent to determining the values of $\mathbf{w}$ and b
  - How to find $\mathbf{w}$ and b from training data?

Linear Support Vector Machines

The decision rule is a linear discriminant function

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$



How to find $\mathbf{w}$ and b from training data?

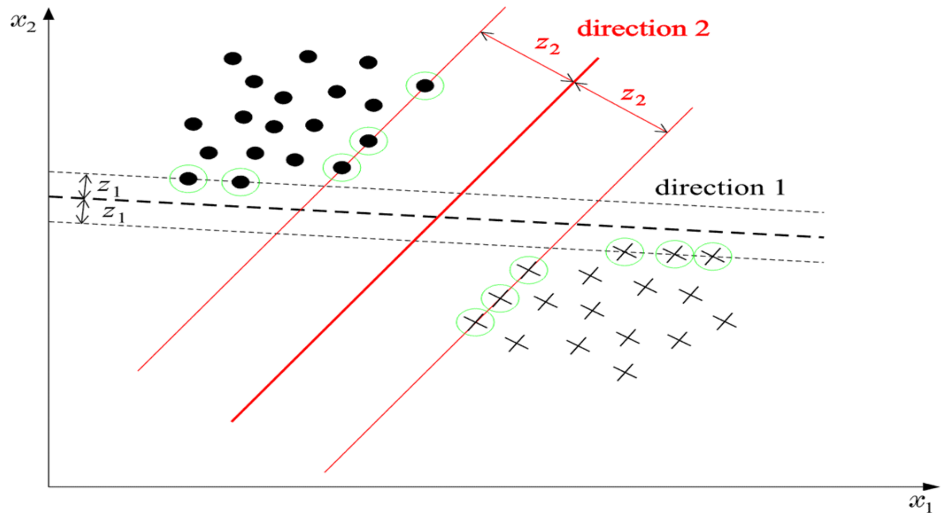
Separating hyperplane

- Direction in space: $\mathbf{w}$
- Position in space: b
- Distance of an object $\mathbf{x}$ (with label y) to the hyperplane: $\frac{y\, g(\mathbf{X})}{\|\mathbf{W}\|}$

Linear Support Vector Machines

Optimal hyperplane: $g(\mathbf{x}) = 0$

- for **EACH** possible direction $\mathbf{w}$**:**
  - choose the hyperplane that leaves the **SAME distance** from the nearest points from each class
  - The margin **is twice this distance**



- $g(\mathbf{x}) = 1$ and $g(\mathbf{x}) = -1$ define **two parallel hyperplane** to the optimal hyperplane $g(\mathbf{x}) = 0$
  - Cases that fall on the hyperplanes are the support vectors ($\mathbf{w}^{\mathrm{T}}\,\mathbf{x} + b = \pm 1$)
  - **Removing all other cases would not change the solution!**
- The optimal hyperplane classifier of a support vector machine is **unique**

Contents

Learning linear SVM

Given a **training data set** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where each **object** is represented by a **D+1**–tuple (D–dim) feature vector $\mathbf{x}_i \in \mathbb{R}^D$ and the corresponding **label** $\mathrm{y}_i \in \boldsymbol{Y}$

- **Goal**: maximize Margin $= \frac{2}{||\mathbf{w}||}$ (Largest margin → better generalization)
    - Which is equivalent to minimizing: $L(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}$
- subject to: $\mathbf{w}$ and b such that:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1 \\ -1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq -1 \end{cases}$$

- Which is **equivalent to:** $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1, \forall\{(\mathbf{x}_i, y_i)\}$

Contents

Learning linear SVM: Optimization problem

Maximum Margin Hyperplane

- The solution is achieved with
  - **minimizing** : $L(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2}$
    - subject to: $y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1, \forall\{(\mathbf{x}_i, y_{\boldsymbol{i}})\}$

- This is a constrained optimization problem
  - Solve it using Lagrange multiplier method

Learning linear SVM: Optimization problem

Maximum Margin Hyperplane

- **Minimization** is achieved by writing the Lagrangian primal problem

$$L = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{N} \lambda_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1)$$

  - Calculating

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \boxed{\mathbf{w} = \sum_{i=1}^{N} \lambda_i \ y_i \mathbf{x}_i}$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^{N} \lambda_i \ y_i = 0$$

and substituting in L the **dual optimization problem** $L_d(\lambda)$ is obtained

Learning linear SVM: Dual optimization problem

Dual optimization problem: data manipulations are dot products

- Training by **maximizing:**

$$L_d(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j=1}^{N} \lambda_i \lambda_j \, y_i \, y_j \left((\mathbf{x_i})^T \mathbf{x_j} \right)$$

  - Subjected to
    - Lagrangians $\lambda_i \geq 0$
    - $\sum_{i=1}^{N} \lambda_i \, y_i = 0$

- **outputs**
  - The **Lagrangians** λ_i computed for all the examples in the training data set
    - If $\lambda_i = 0$ the i –th example $\mathbf{x_i}$ is not relevant
    - If $\lambda_i \neq 0$ the i-th example $\mathbf{x_i}$ is a support vector

Learning linear SVM: Dual optimization problem

After learning the Lagrangians:

- Compute $\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \, \mathbf{x}_i$*
  - when $\lambda_i = 0$ the i –th example $\mathbf{x}_i$ is not relevant (it does not contribute to the sum)
  - when $\lambda_i \neq 0$ the corresponding i-th example $\mathbf{x}_i$ is a support vector
    - lies along the hyperplanes parallel to the decision hyperplane (linearly separable problem): $\mathbf{w}^{\mathrm{T}} \mathbf{x} + b = \pm 1$
- b is computed using **support vectors**

Decision boundary depends only on **support vectors**

$$g(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$$

* Note that only the support vectors contribute to compute $\mathbf{w}$

Learning linear SVM: Dual optimization problem

Testing

Applying **dual form of linear classifier,** the label of object $\boldsymbol{z}$ is

$$g(\mathbf{z}) = \sum_{i,j=1}^{K_s} \lambda_i \, y_i((\mathbf{x}_i)^T \mathbf{z}) + b \Rightarrow \begin{array}{l} g(\mathbf{z}) > 0, \mathbf{z} \in \omega_1 \\ g(\mathbf{z}) < 0, \mathbf{z} \in \omega_2 \end{array}$$

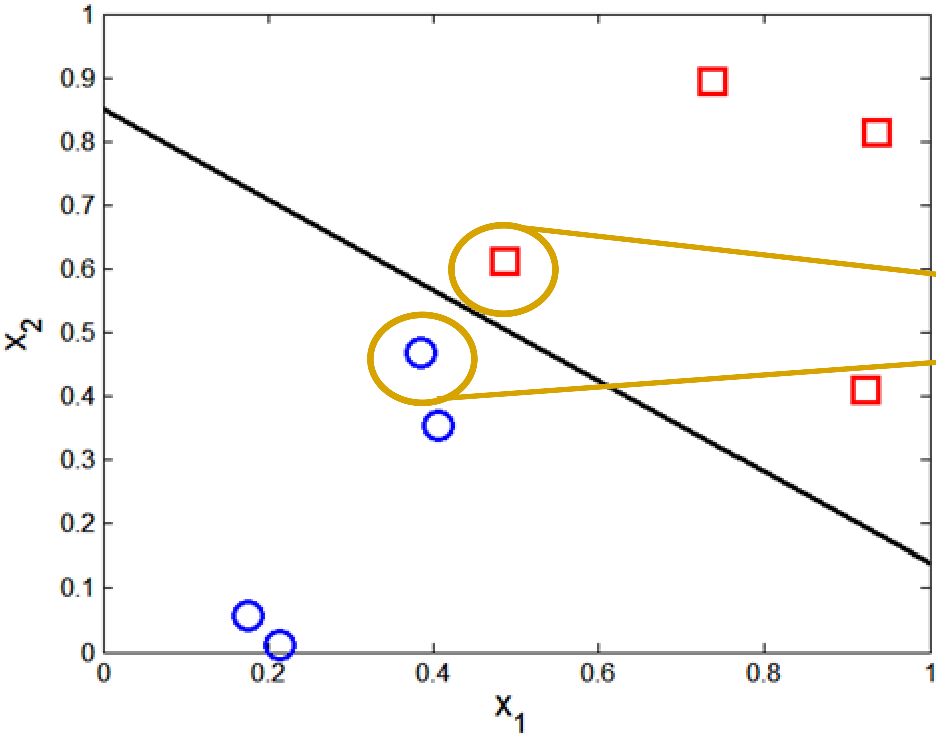
- K_s : the number **support vectors**

To apply **dual form** during testing phase it is needed:

- the support vectors to perform $(\mathbf{x}_i)^T\mathbf{z}$ and corresponding labels y_i
- the complexity of testing phase is dependent on the number of support vectors

Contents

Example of linear SVM

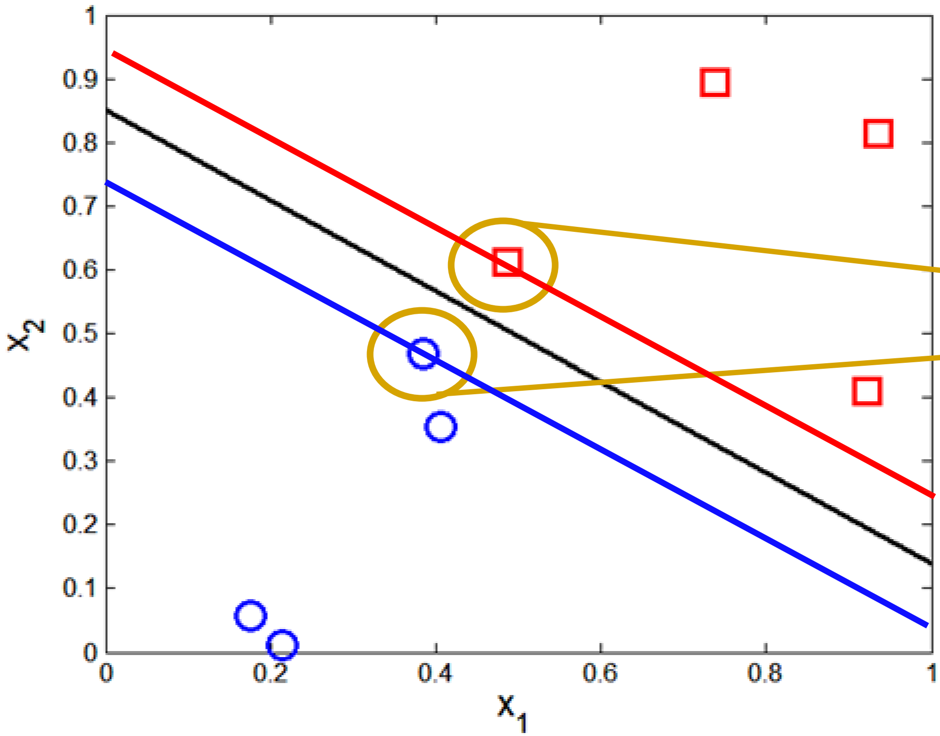


Support vectors

x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

How to find $\mathbf{w}$ and b from training data?

Example of linear SVM



Support vectors

x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

How to find $\mathbf{w}$ and b from training data?

Example of linear SVM

Support vectors

How to find **w** and b from training data?

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \mathbf{x}_i$$

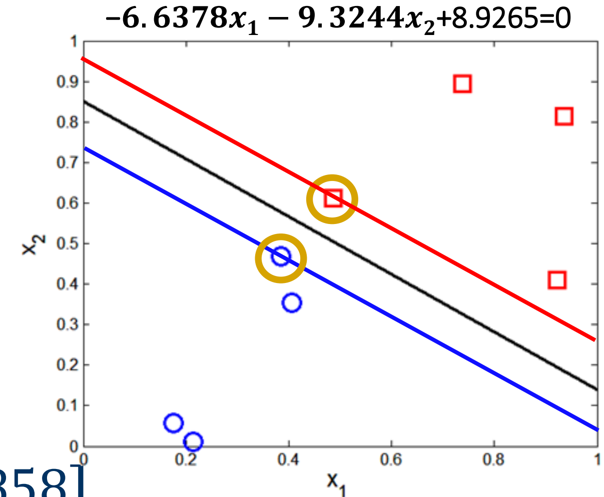
x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

$$\mathbf{w} = \sum_{i=1}^{K_s} \lambda_i y_i \mathbf{x}_i$$

$$\mathbf{w} = 65.5261 \times 1 \times [0.3858\ 0.4687] + 65.5261 \times (-1) \times [0.4871\ 0.611] =$$

$$\mathbf{w} = [-6.6378\ \ -9.3244]$$

Example of linear SVM



How to find $\mathbf{w}$ and b from training data?

At the margins: $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \pm 1$

Red hyperplane ($\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = +1$)

$$b = +1 - \mathbf{w}^{\mathrm{T}}\mathbf{x} = +1 - [-6.6378 \ \ -9.3244]\begin{bmatrix}0.3858\\0.4687\end{bmatrix} = 7.9226$$

Blue hyperplane ($\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = -1$)

$$b = -1 - \mathbf{w}^{\mathrm{T}}\mathbf{x} = -1 - [-6.6378 \ \ -9.3244]\begin{bmatrix}0.4871\\0.611\end{bmatrix} = 7.9305$$

$$b = \frac{(7.9226 + 7.9305)}{2} = 7.9265$$

x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

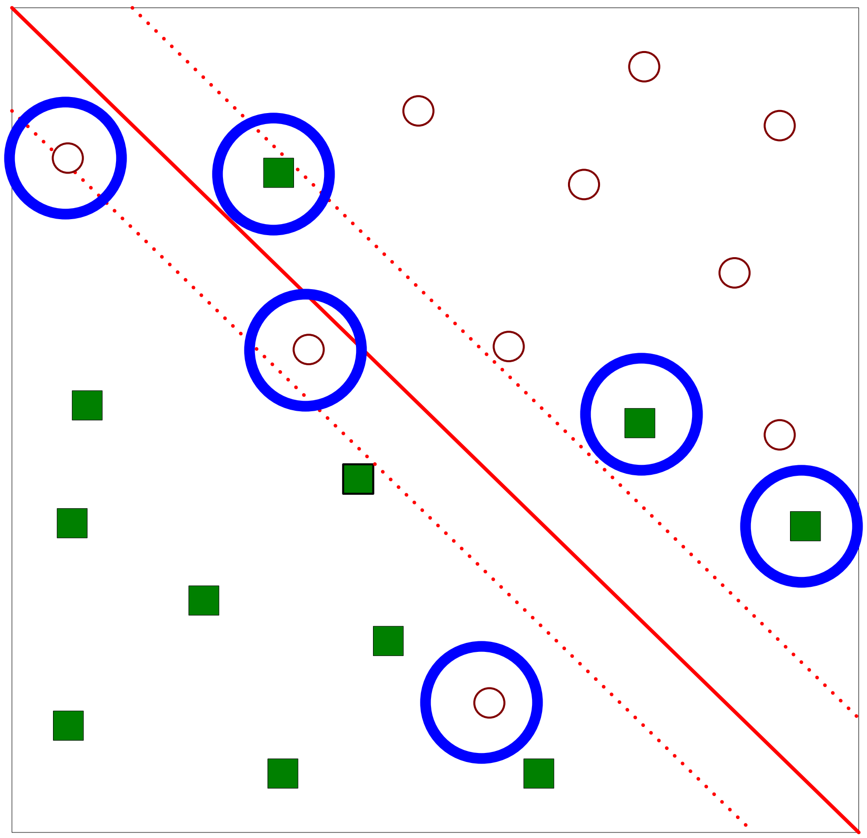
(it is numerically safer to take the mean value of b resulting from all support vectors)

Contents

Learning linear SVM
not linearly separable data

- What if the problem is **not linearly separable?**

Soft-margin SVM



Learning linear SVM
not linearly separable data

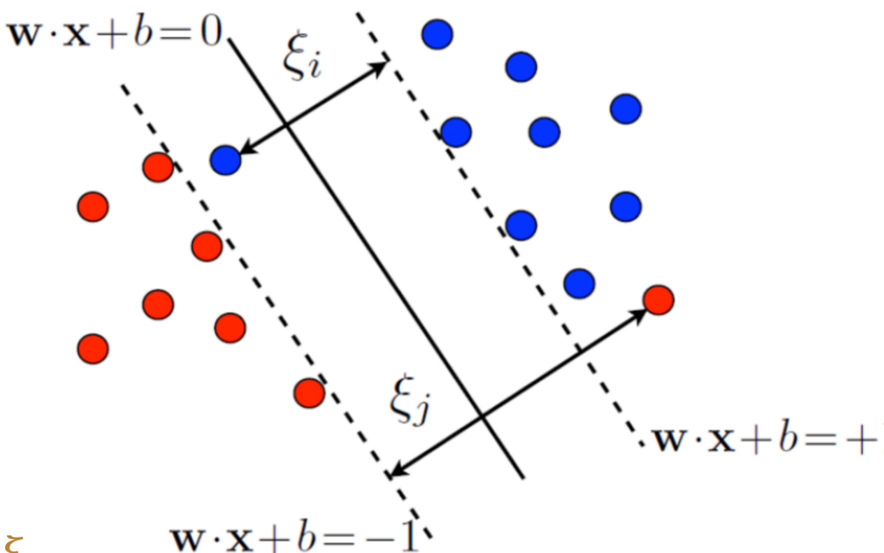
- What if the problem is **not linearly separable?**

Soft-margin SVM

  - Introduce slack variables to **tolerate some misclassification errors** controlled by the parameter **C** (regularization term)

    - **Objective**: minimize

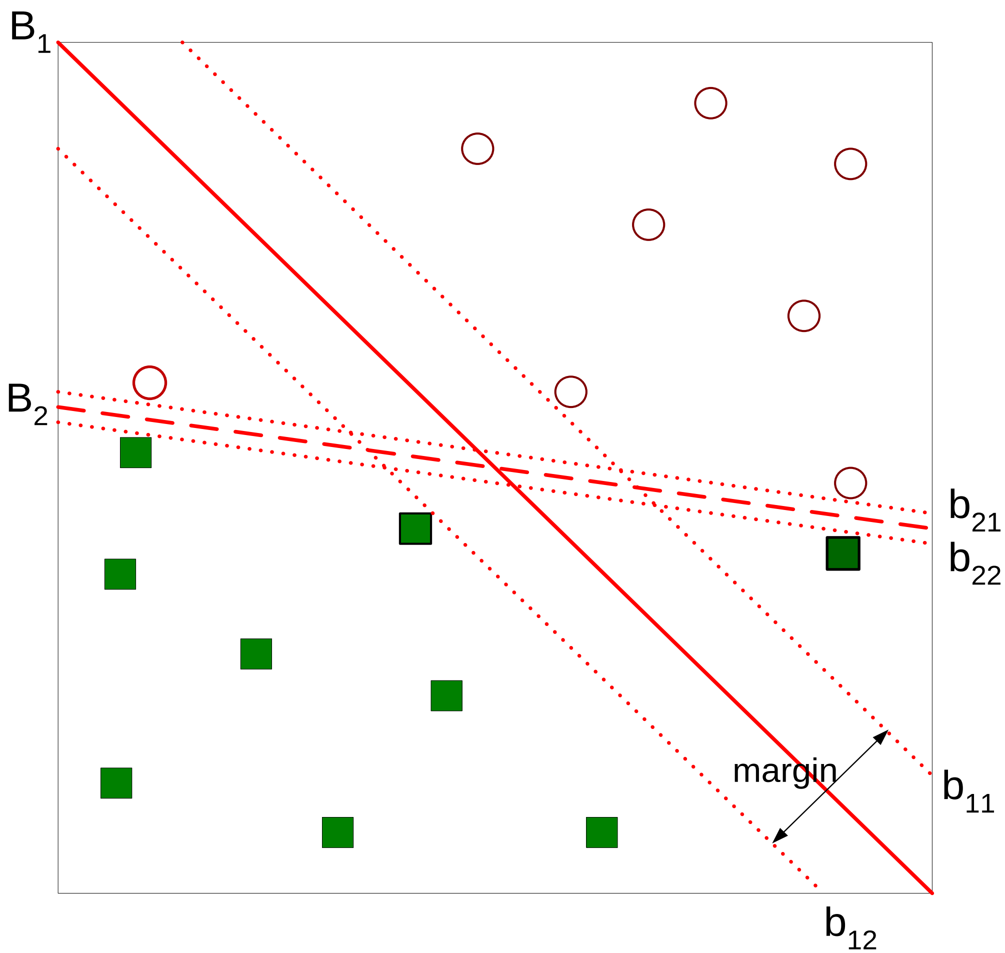
$$L(w) = \frac{||w||^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$$



    - Goal: find $\mathbf{w}$, b and ξ, given that:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1 - \xi_i \\ -1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq -1 + \xi_i \end{cases}$$

Learning linear SVM
not linearly separable data



- Find the hyperplane that optimizes both margins (B_1 and B_2)

Linear Support Vector Machines

Hard Margin SVMs: Linearly separable data

- works well when data is linearly separable
- on real-world data this is hardly the case
- does not take into account presence of noise

Soft Margin SVMs: Not linearly separable data

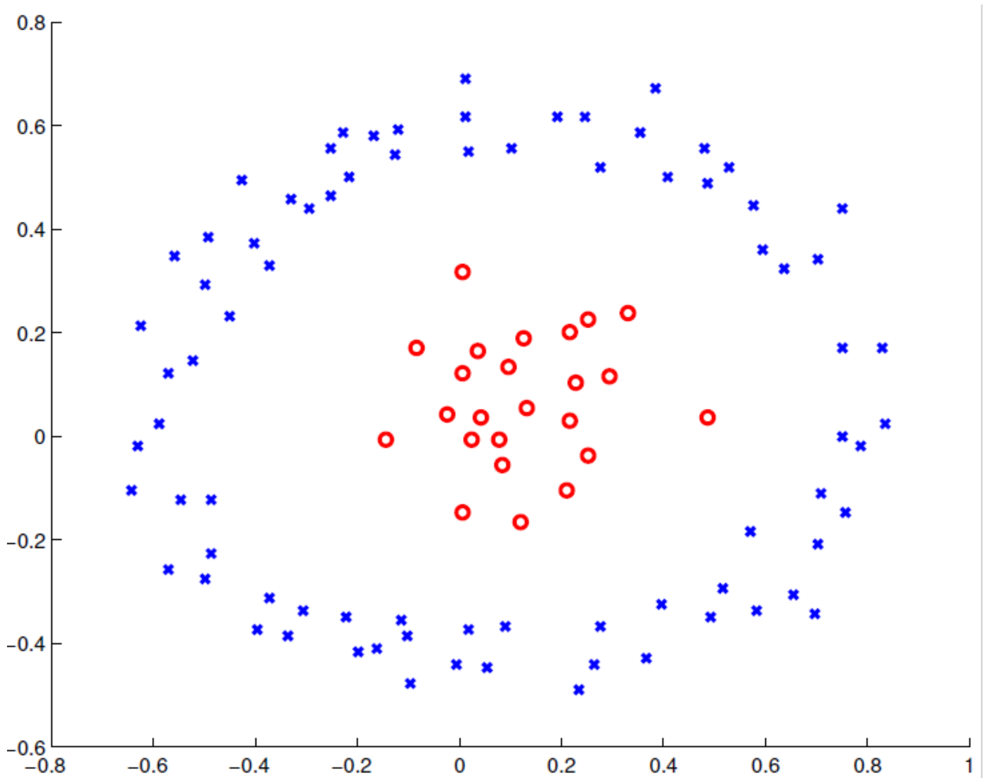
- it tolerates some misclassification
- errors controlled by a parameter ***C*** (regularization term)
- introduces slack variables for each example

Contents

- Support Vector Machines (SVM)
  - Support vectors
- Linear support vector machines
- **Nonlinear support vector machines**
  - Learning nonlinear SVM and the Kernel trick
  - Example
- Multi-class SVM
- Summary

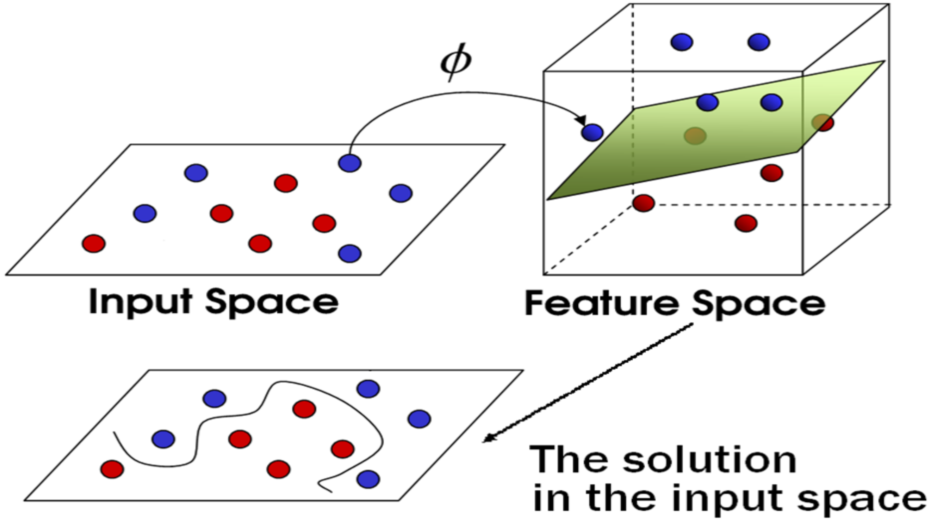
Nonlinear Support Vector Machines

- What if the decision boundary is **not linear?**



Nonlinear Support Vector Machines

- Most real world problems have **inherent nonlinearity**
- SVMs solve this by “moving” into an **extended input space** where classes are already linearly separable
- This means the **maximum margin hyperplane** needs to be found on this **new very high dimension space**



Contents

Learning nonlinear SVM

- Transform data into **higher dimensional space with $\Phi(\mathbf{x})$**
  - Such that classes are linearly separable
- Same optimization problem $\min_{\mathbf{w}} \frac{||\mathrm{W}||^2}{2}$, but involving $\Phi(\mathbf{x})$ instead of $\mathbf{x}$
- Computations involve dot product $\Phi(\mathbf{x}_i)\ \Phi(\mathbf{x}_j)$
  - Solution to the optimization equation involves dot products between feature vectors, $\mathbf{x}_i$ and $\mathbf{x}_j$, that are computationally heavy on high-dimensional spaces
  - Calculate the image of $\Phi(\mathbf{x})$ of each input vector $\mathbf{x}$ and then do the dot product can be quite expensive
- The kernel function defined as $\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\ \Phi(\mathbf{x}_j)$ performs simultaneously the mapping and dot product

→ Kernel Trick

Learning nonlinear SVM

Kernel Trick

- The result of complex calculations in higher dimensional space is equivalent to the result of applying certain functions (**kernel functions**) in the space of the original variables
    - replace the complex dot products by these simpler and efficient calculations
- The kernel function takes as inputs vectors in the original space and **returns the dot product of the vectors in the higher dimensional space**
    - perform operations in the **original space** (without a feature transformation!)
- Using kernels, we do not need to embed the data into the higher dimensional space **explicitly**!
- instead of calculating the dot products in a high dimensional space, take advantage of $K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)\,\Phi(\mathbf{x}_j)$
- use a linear optimization solution to solve a non-linear problem

Learning nonlinear SVM

Dual optimization problem: the same set of equations (but involve $\Phi(\mathbf{x})$ instead of $\mathbf{x}$)

$$L_d(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j \ y_i \ y_j \left((\Phi(\mathbf{x}_i))^T \Phi(\mathbf{x}_j) \right)$$

$$L_d(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j=1}^{N} \lambda_i \lambda_j \ y_i \ y_j \left(\mathrm{K}(\mathbf{x}_i, \mathbf{x}_j) \right)$$

Kernel Trick

- Subjected to

$$\mathbf{w} = \sum_{i=1}^{N} \lambda_i y_i \, \Phi(\mathbf{x}_i)$$

- SVM **dual form of decision rule**

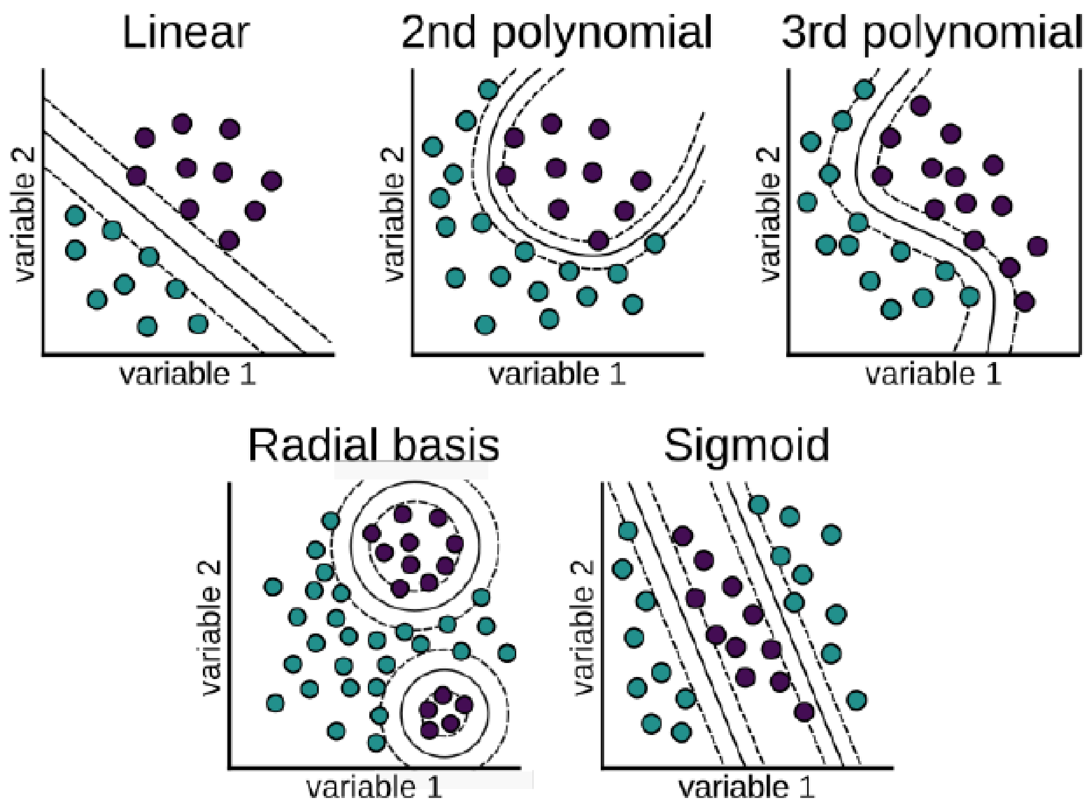
$$g(\mathbf{z}) = \mathbf{w}^{\mathrm{T}}\mathbf{z} + b = \sum_{i,j=1}^{K_s} \lambda_i \ y_i (\Phi(\mathbf{x}_i))^T \Phi(\mathbf{z}) + b \Rightarrow \begin{array}{l} g(\mathbf{z}) > 0, \mathbf{z} \ \in \omega_1 \\ g(\mathbf{z}) < 0, \mathbf{z} \ \in \omega_2 \end{array}$$

Learning nonlinear SVM

- Advantages of using kernel:
  - Don't have to know the mapping function Φ
  - Computing dot product $\Phi(\mathbf{x}_i)\ \Phi(\mathbf{x}_j)$ in the original space avoids curse of dimensionality
- Not all functions can be kernels
  - Must make sure there is a corresponding Φ in some high-dimensional space

Learning nonlinear SVM

Examples of different kernel functions:



Contents

Nonlinear Support Vector Machines: example

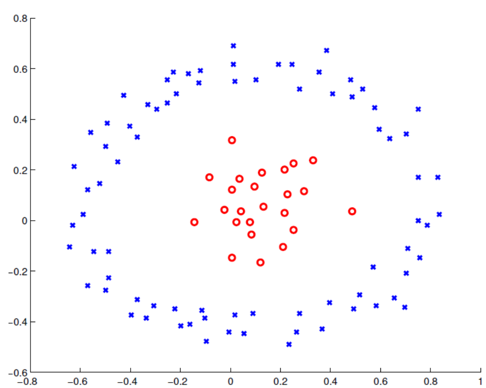
Transform data into **higher dimensional space (**from 2D to 3D**)**

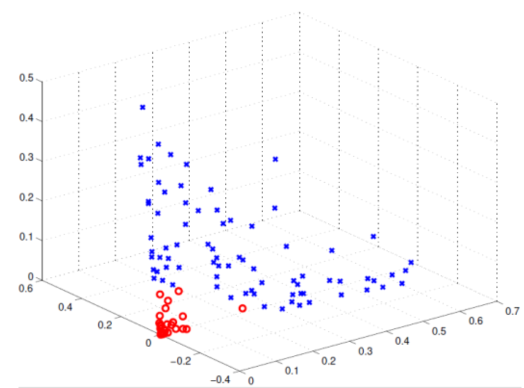
$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2] \quad \rightarrow \quad \Phi(\mathbf{x}) = [\mathbf{x}_1^2 \quad \sqrt{2}\mathbf{x}_1\mathbf{x}_2 \quad \mathbf{x}_2^2]$$

- **linear decision boundary** with SVM (for example..)
  - Linear hyperplane can be used to **separate** the instances in the **transformed space**
- The learned hyperplane can then be **projected back** to the **original** attribute space
  - **nonlinear decision boundary**

- **The dot product in the 3D space using the polynomial kernel**

$$\Phi(\mathbf{x}_1)^{\mathrm{T}}\Phi(\mathbf{x}_2) = K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T\mathbf{x}_2)^2$$





Nonlinear Support Vector Machines: example

Transform data into **higher dimensional space** (from 2D to 3D)

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2] \;\rightarrow\; \Phi(\mathbf{x}) = [\mathbf{x}_1^2 \quad \sqrt{2}\mathbf{x}_1\mathbf{x}_2 \quad \mathbf{x}_2^2]$$

Data set: 2 exemples (2D)

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

Mapped data set: 2 examples (3D)

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 & \sqrt{2} & 1 \\ 1 & \sqrt{2} & 1 \end{pmatrix}$$

Dot product in the 3D space using kernel

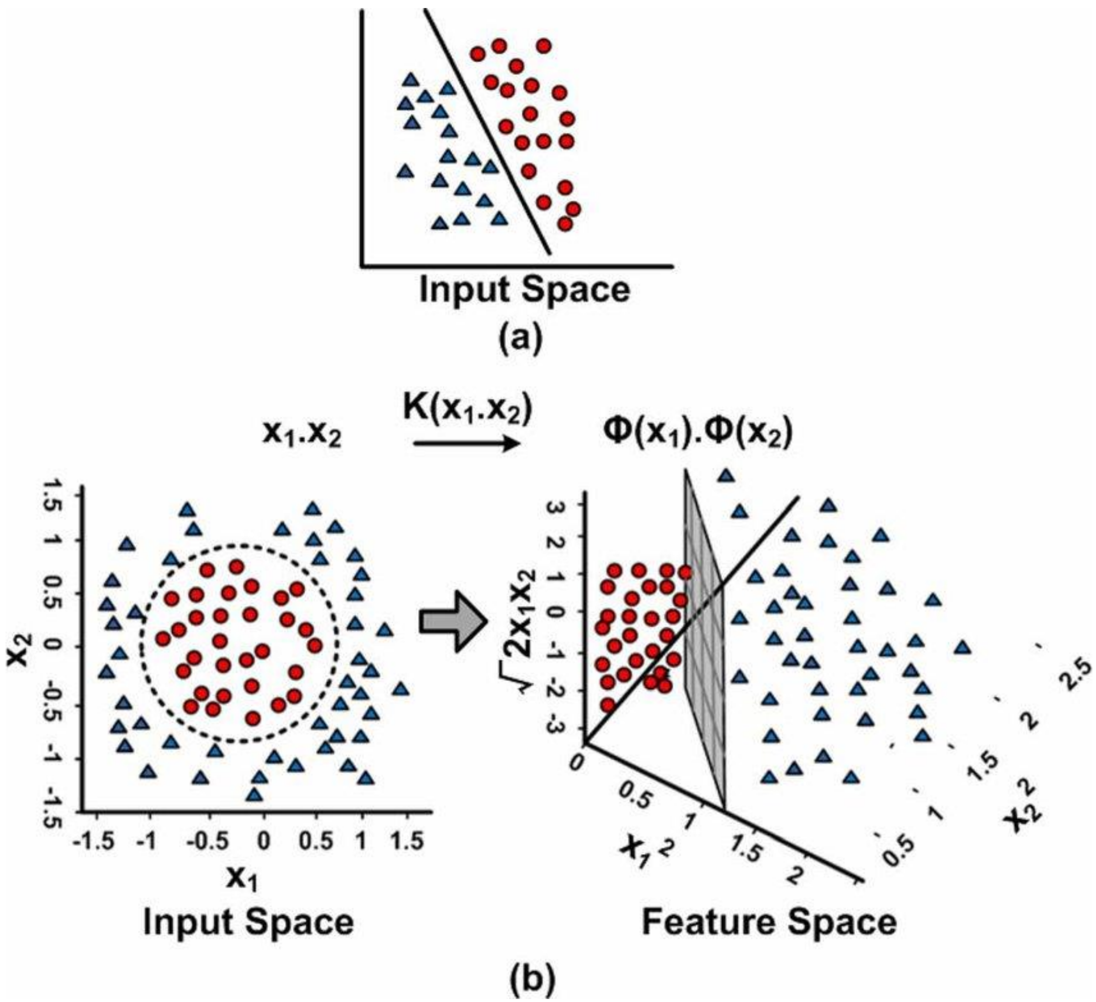
$$\left(\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \end{pmatrix} \right)^2 = 4$$

Dot product with the 3D data

$$\begin{pmatrix} 1 & \sqrt{2} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \sqrt{2} \\ 1 \end{pmatrix} = 4$$

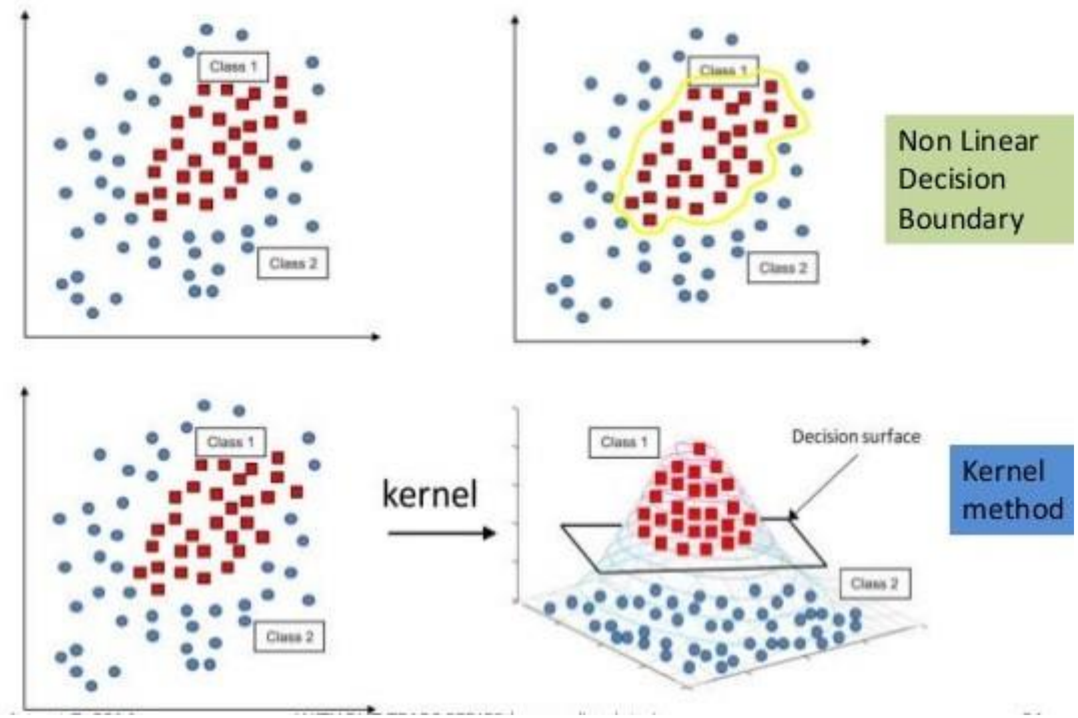
ADVANTAGE: The dot product in the 3D space with the data of the 2D space

Nonlinear Support Vector Machines: example



SVM with polynomial degree 2 kernel

Nonlinear Support Vector Machines: example



SVM with Gaussian RBF kernel

Contents

Support Vector Machines: multi-class

How to handle more than 2 classes?

- Solve several binary classification tasks
- Essentially, find the support vectors that separate each class from all others

Support Vector Machines: multi-class

Two strategies of training multiple binary classifiers. Considering **C** classes

- one-against-all (one-against-the rest): **C** binary classifiers
  - Training set: **Positive class** (objects of C_i), **Negative class** (objects of the rest C_j , $j \neq i$)
  - Testing a new object: winner-takes-all strategy, binary classifier with the highest (largest) output function assigns the class
- one-against-one: c(c -1)/2 binary classifiers
  - Training set: **Positive class** (objects of C_i), **Negative class** (examples of the other C_j , $j \neq i$)
  - Testing a new object: max-wins voting strategy, in which every classifier assigns the instance to one of the two classes, the class with most votes determines the instance classification

Contents

SVM: summary

The **outputs of the training algorithm** are

- the values of **Lagrangian** λ_i: $0 \leq \lambda_i < C$
- the parameter **b**
- the support vectors $(\mathbf{x}_i, y_i)$ a subset of training set
- Linear SVM: the $\mathbf{w}$ can be estimated
- Non-Linear SVM: dual form is a must
- Data set with same support vectors → decision boundary remains

SVM: training and testing

Training SVM

- Choose appropriate kernel function. This implicitly assumes a mapping to a higher dimensional (yet, not known) space
- Assign the parameters of the kernel (e.g., σ if RBF kernel)
- The margin control parameter C

The choice of parameters is usually experimentally driven: k-folder cross-validation

Testing SVM

- The support vectors need to be stored to apply the kernel function (if non-linear)
- The complexity depends on the number of support vectors
- Major limitations of (non-linear) SVM is the computational burden

SVM: characteristics

- The **learning problem** is formulated as a **convex** optimization problem
  - Efficient algorithms are available to find the global minima (e.g., SGD)
  - Many of the other methods use greedy approaches and find locally optimal solutions

- SVM is effective on large datasets
  - **Complexity** of trained classifier
    - characterized by the # of support vectors (rather than the size of data)
  - S**upport vectors**: essential or critical training examples
    - lie closest to the decision boundary

- SVM with a small number of support vectors can have good generalization, even on large datasets

- What about categorical variables?

SVM: advantages and disadvantages

Advantages

- Linear and non-Linear in the same algorithm.
- **Good generalization** (classification accuracy high)
  - Overfitting: handled by **maximizing the margin** of the decision boundary
- Robust to noise (works when training examples contain errors)
- Can handle irrelevant and redundant attributes better than many other techniques

Disadvantages

- In training: no criterium to choose of appropriate kernel function (and parameters)
- If number of support vectors is high: complexity (storage and computation) is high
- Not easy to interpret results
- Multiclass problems still need more improvement

Bibliography

Introduction to Data Mining, Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, Vipin Kumar, *Pearson*, 2019 (chap 6.9)

Data Mining, the Textbook, Charu C. Aggarwal, *Springer*, 2015 (chap 10.6)

Machine Learning: The Art and Science of Algorithms That Make Sense of Data, P. Flach, *Cambridge University Press*, 2012 (ch 7.3)