Introduction to Machine Learning
NPFL 054

http://ufal.mff.cuni.cz/course/npfl054

Barbora Hladká
hladka@ufal.mff.cuni.cz

Martin Holub
holub@ufal.mff.cuni.cz

Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics

Outline

- Support Vector Machines
- Naïve Bayes algorithm
- Bayesian networks

Support Vector Machines (SVM)

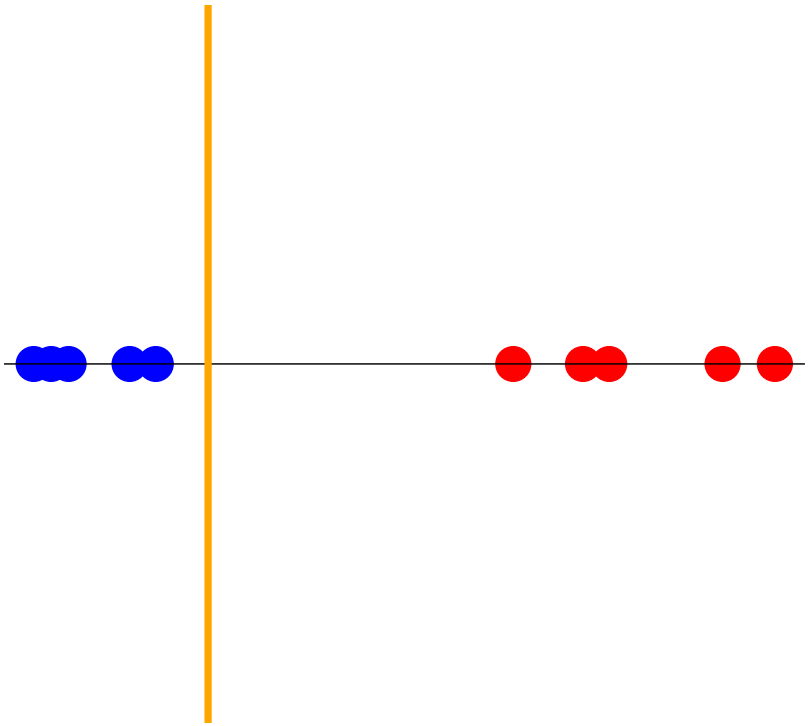
We focus on binary classification and separating the classes by a hyperplane

Key points

1. Hyperplane, Maximizing the margin
2. Quadratic programming, Duality optimization task, Dot product
3. Kernel trick

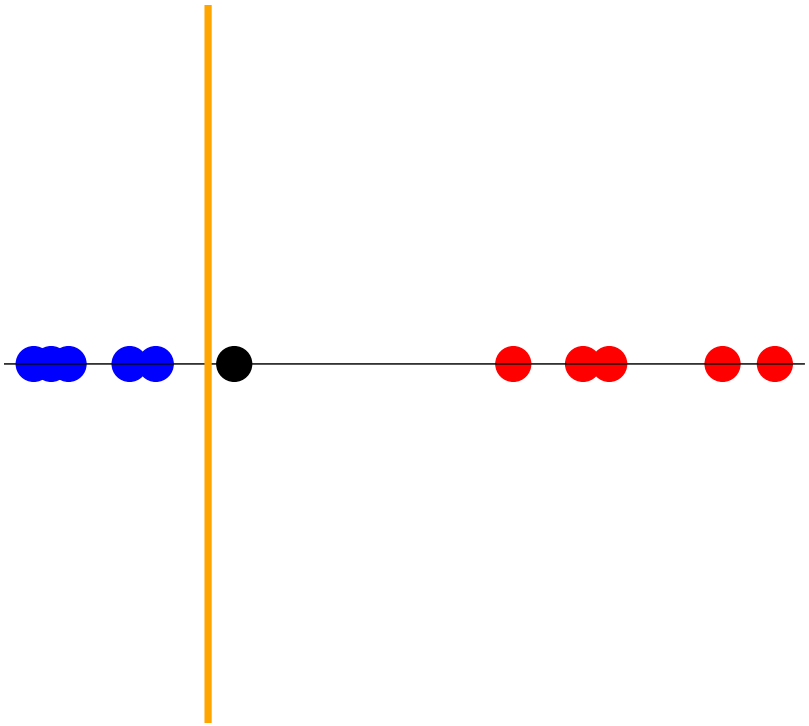
Basic idea

- Titanic data set
- Target class: Survived, Not Survived
- Feature: Fare
- Separating hyperplane (point in 1D)



Basic idea

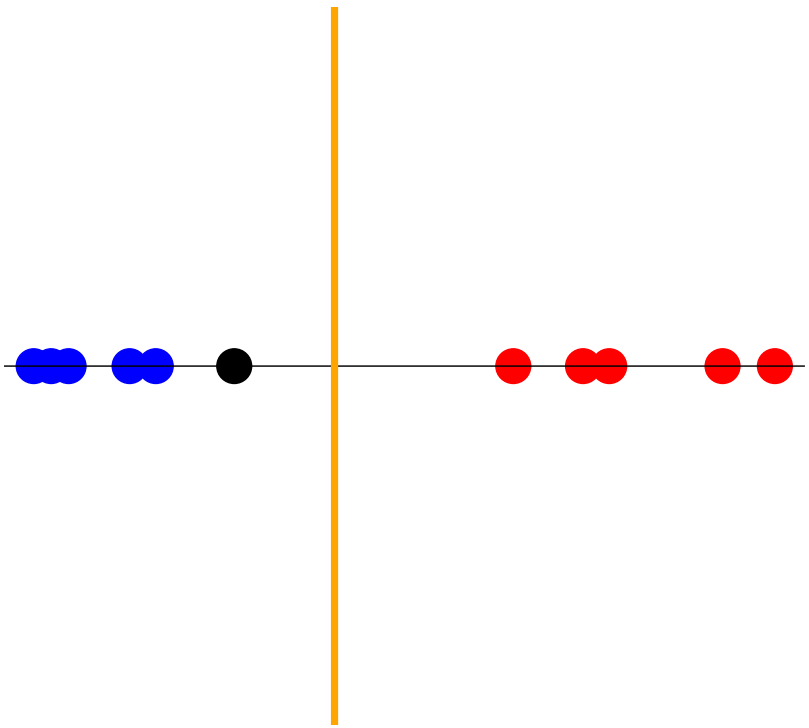
- New test example will by classified into Red. But it is closer to Blue



fare

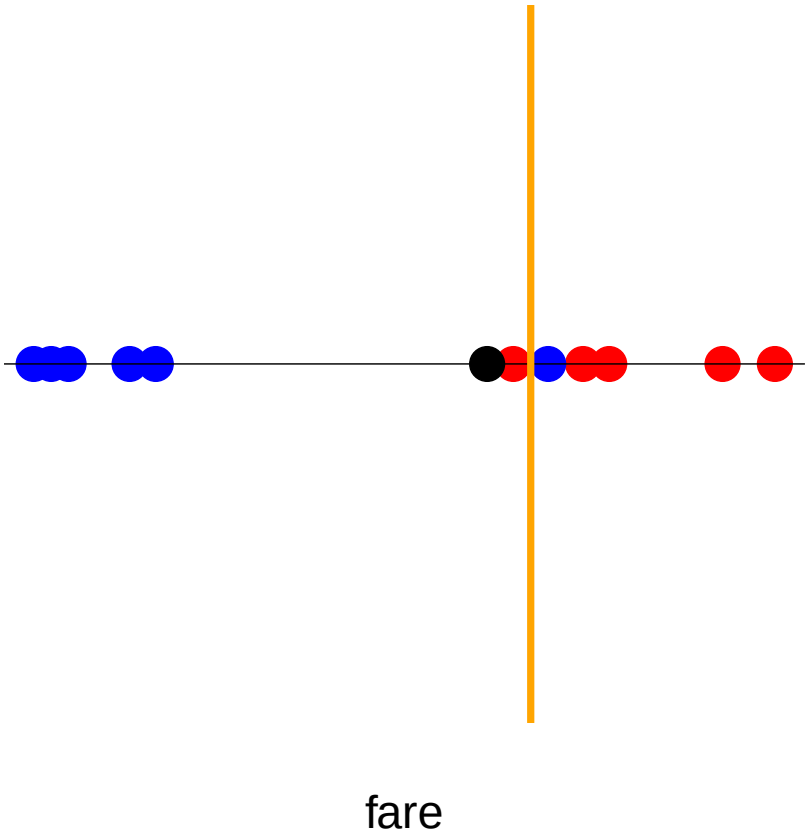
Basic idea

- Let separating hyperplane be the midpoint between the closest examples of the two clusters



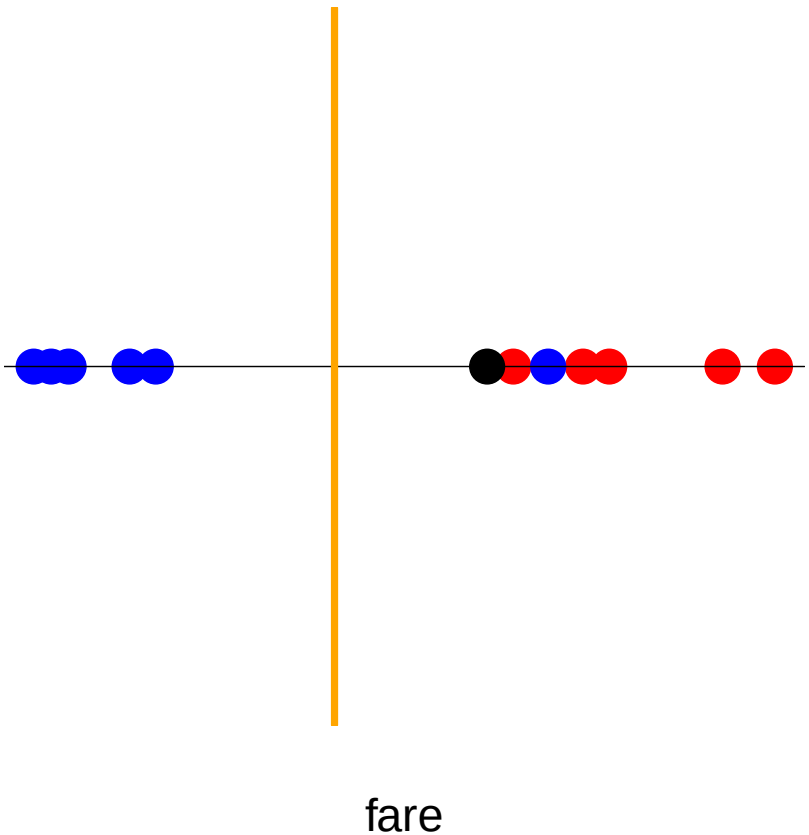
Basic idea

- But this idea is very sensitive to outliers



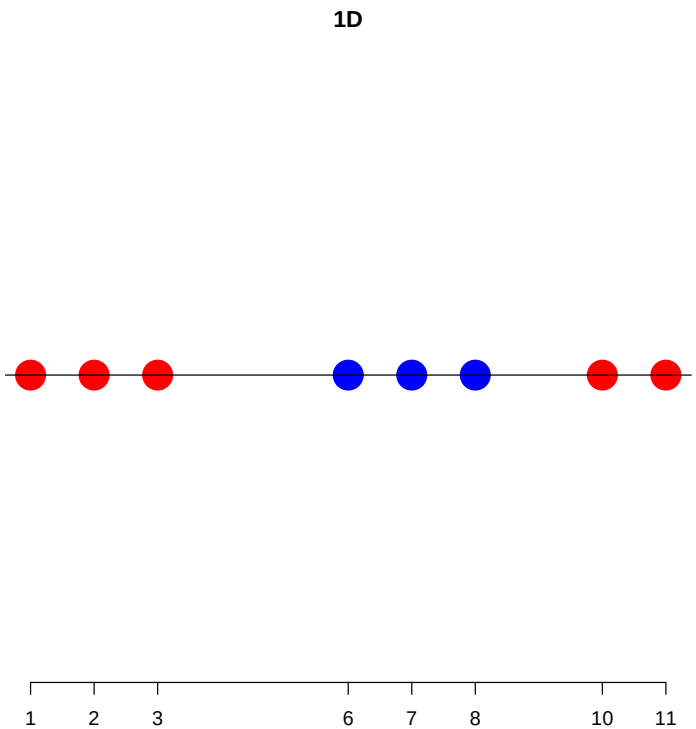
Basic idea

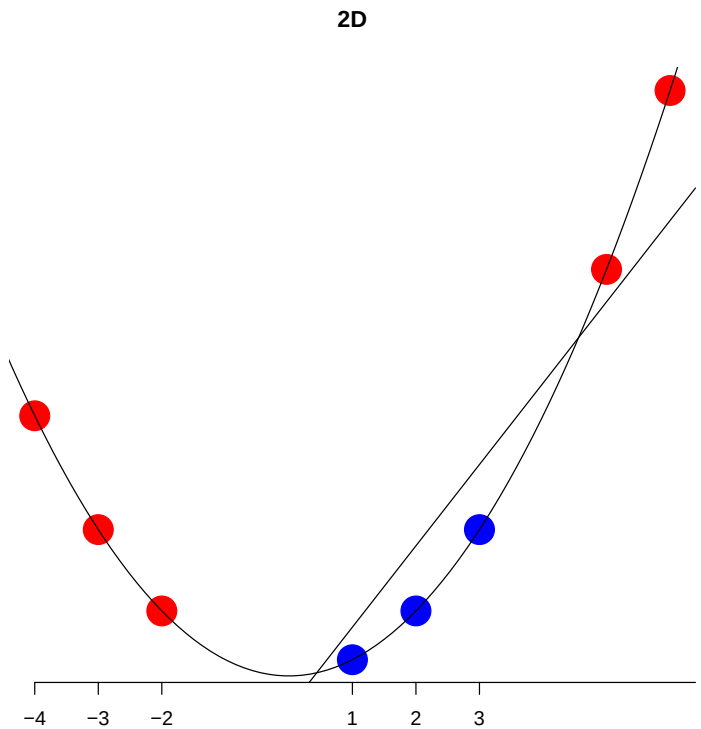
- Let's allow misclassification



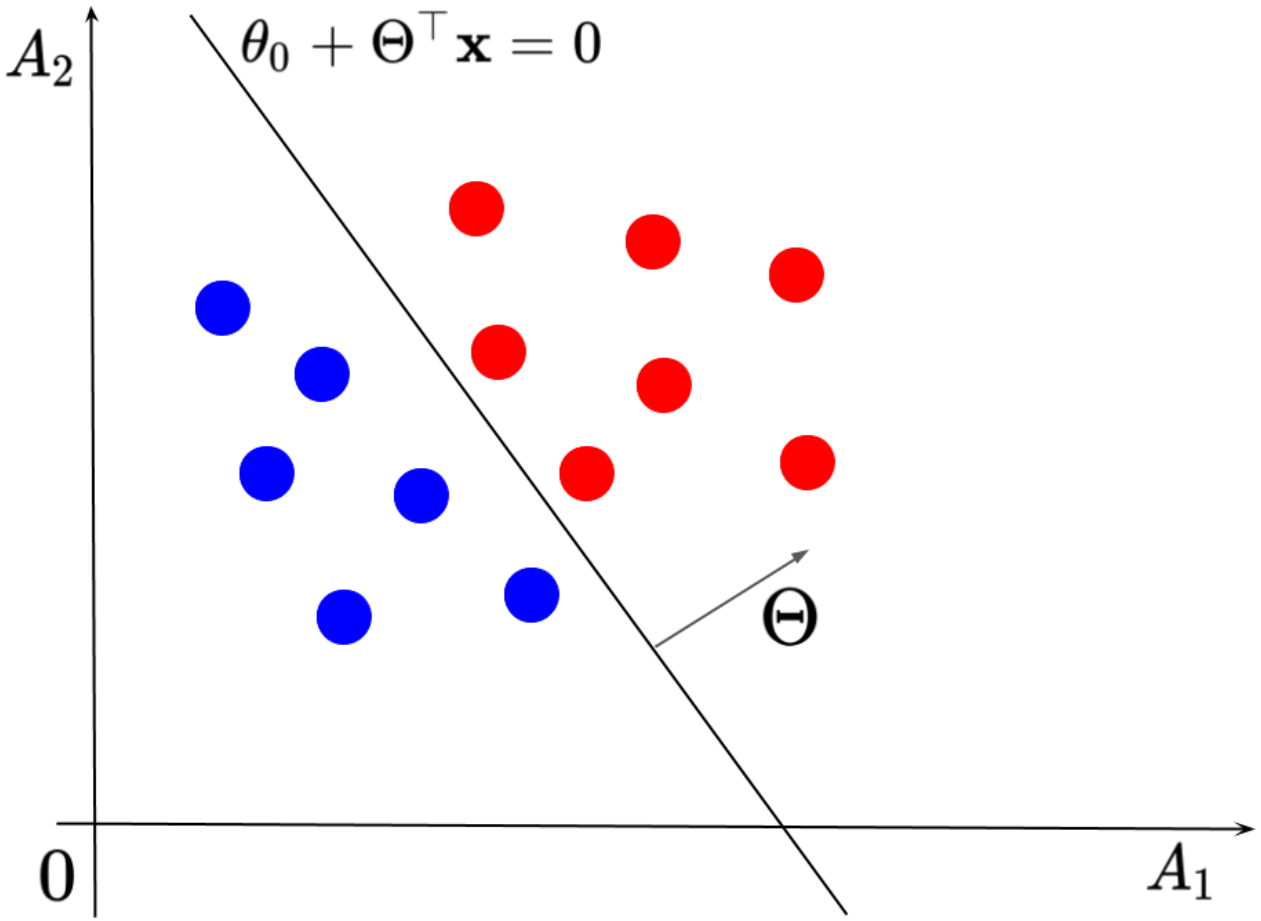
Basic idea

- Do separation with a hyperplane in a higher dimension space if not possible in an original space





Hyperplane



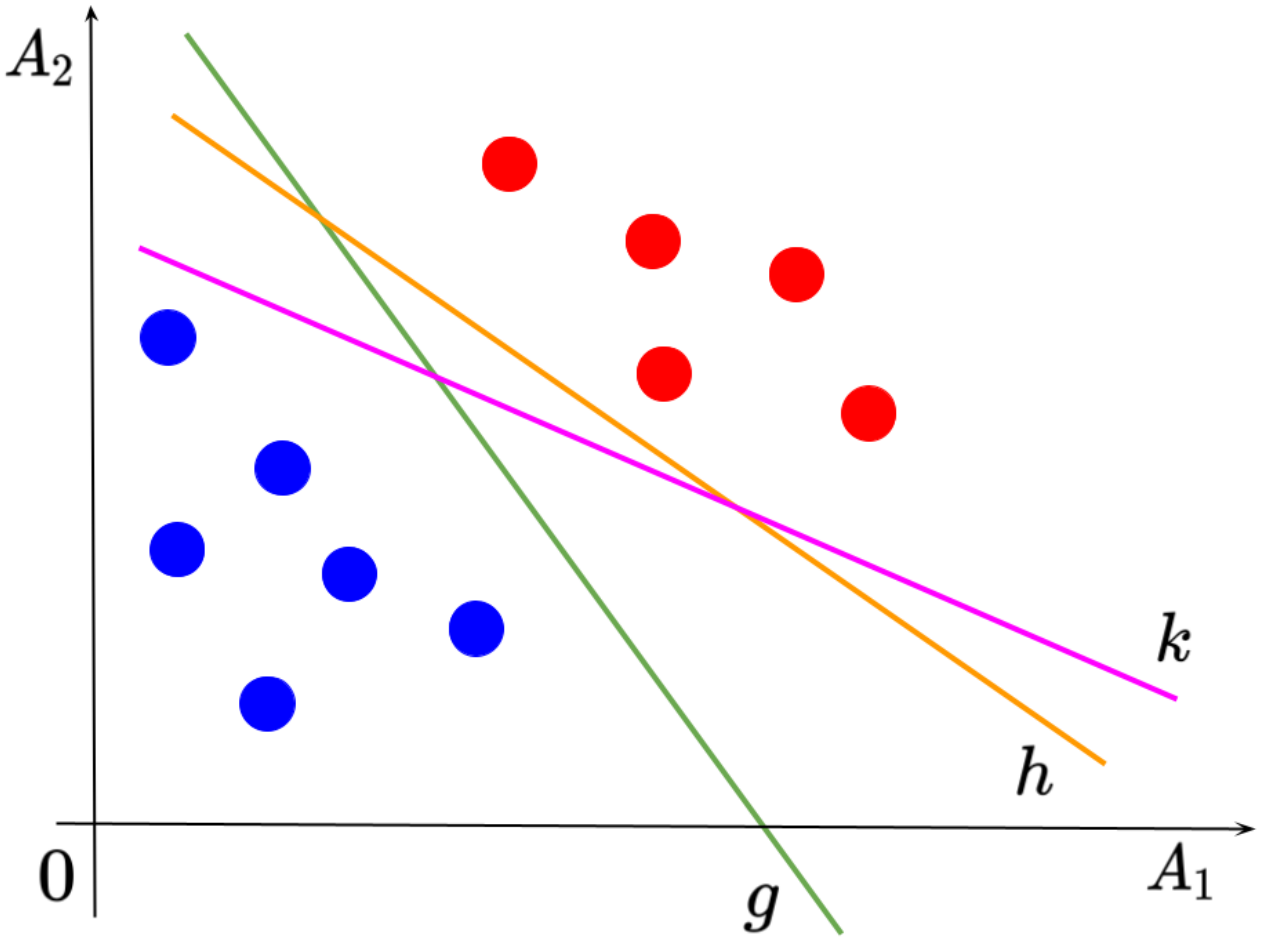
Classification using hyperplane

Recall the classification rule using $\theta_0 + \Theta^\top \mathbf{x} = 0$

$$f(\mathbf{x}) = \begin{cases} 1 \text{ if } \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m \geq 0 \\ 0 \text{ if } \theta_0 + \theta_1 x_1 + \cdots + \theta_m x_m < 0 \end{cases}$$

Classification using hyperplane

How to compare hyperplanes?



How to compare hyperplanes?

- training examples $D = \{\langle \mathbf{x}_i, y_i \rangle, \mathbf{x}_i \in X, y_i \in \{-1, +1\}\}$
- hyperplane g: $\theta_0 + \Theta^\top \mathbf{x} = 0$

1.

$$\overline{\rho}_g(D) = \min_{\langle \mathbf{x}, y \rangle \in D} (\theta_0 + \Theta^\top \mathbf{x})$$

But we work with negative values as well, therefore

2.

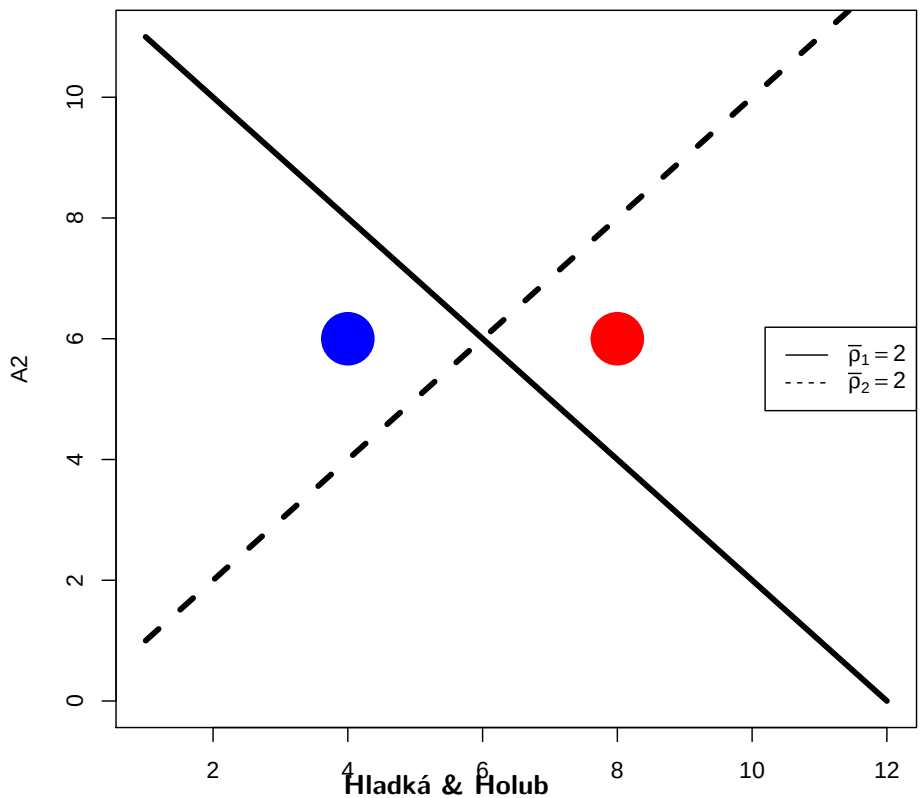
$$\overline{\rho}_g(D) = \min_{\langle \mathbf{x}, y \rangle \in D} |\theta_0 + \Theta^\top \mathbf{x}|$$

How to compare hyperplanes?

But

Example

- $g_1 : -12 + x_1 + x_2 = 0$
- $g_2 : -x_1 + x_2 = 0$, misclassification of the two examples



How to compare hyperplanes?

Therefore

3. **functional margin of D w.r.t. g**

$$\overline{\rho}_g(D) = \min_{\langle \mathbf{x}, y \rangle \in D} y \cdot (\theta_0 + \Theta^\top \mathbf{x})$$

But the functional margin is not scale invariant

How to compare hyperplanes?

Example

- g_1: $5 + 2x_1 + x_2 = 0$, $\Theta_1 = \langle 2, 1 \rangle$
- g_2: $50 + 20x_1 + 10x_2 = 0$, $\Theta_2 = \langle 20, 10 \rangle$
- Θ_1 and Θ_2 have the same unit vector $\langle \frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}} \rangle$, i.e. they represent the same hyperplane
- $\overline{\rho}_{g_2}(D) = 10 \cdot \overline{\rho}_{g_1}(D)$
- $||\Theta||$ does not matter
- Θ's direction matters, it's given by its unit vector

How to compare hyperplanes?

Therefore, instead of Θ we use its unit vector

4. **geometric margin of D w.r.t. g**

$$\rho_g(D) = \min_{\langle \mathbf{x}, y \rangle \in D} y \cdot \left(\frac{\theta_0}{||\Theta||} + \frac{\Theta^\top}{||\Theta||}\mathbf{x}\right) = \min_{\langle \mathbf{x}, y \rangle \in D} \frac{\overline{\rho}_g(D)}{||\Theta||}$$

Geometric margin of D

Example

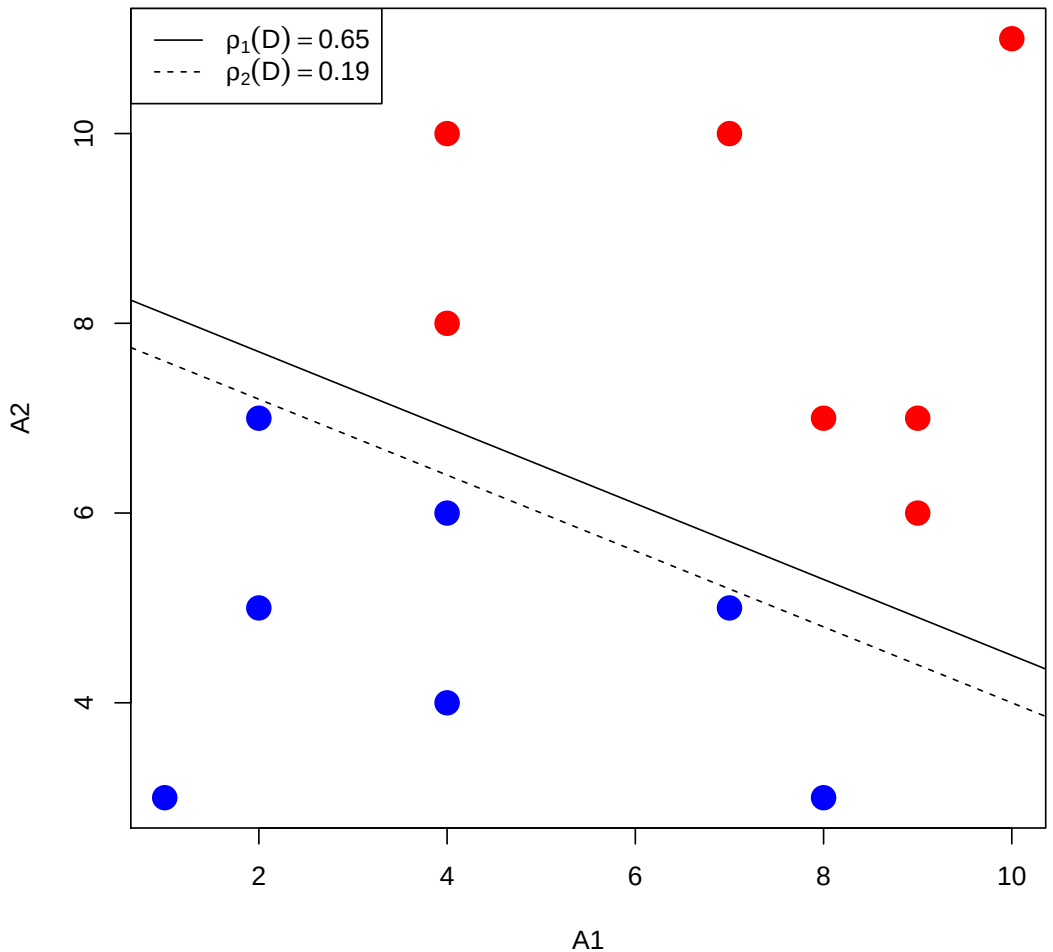
```
> A1 <- c(4,4,7,8,9,9,10, 1,2,2,4,4, 7,8)
> A2 <- c(8,10,10,7, 6,7,11,3,5,7,4,6,5,3)
> c <-  c(-1,-1,-1,-1,-1,-1,-1,1,1,1,1,1,1,1)
> d <- data.frame(A1,A2,c)

> t01 <- 8.5; t11 <- -0.4; t21 <- -1
> m.1 <- d[,3]*(t01 + t11*d[,1] + t21*d[,2])/sqrt(t11^2+t21^2)
> t02 <- 8; t12 <- -0.4; t22 <- -1
> m.2 <- d[,3]*(t02 + t12*d[,1] + t22*d[,2])/sqrt(t12^2+t22^2)

> min(m.1)
[1] 0.6499337
> min(m.2)
[1] 0.1856953
```

Geometric margin of D

Example (cntnd)



Margins of x

- **Margin of x** w.r.t. g is distance of $\mathbf{x}$ to g

$$\rho_g(\mathbf{x}) = \frac{|\theta_0 + \Theta^\top \mathbf{x}|}{||\Theta||}$$

- **Functional margin of** $\langle \mathbf{x}, y \rangle$ w.r.t. g is

$$\overline{\rho}_g(\mathbf{x}, y) = y \cdot (\Theta_0 + \Theta^\top \mathbf{x})$$

- **Geometric margin of** $\langle \mathbf{x}, y \rangle$ w.r.t. g is functional margin scaled by $||\Theta||$

$$\rho_g(\mathbf{x}, y) = \overline{\rho}_g(\mathbf{x}, y)/||\Theta||$$

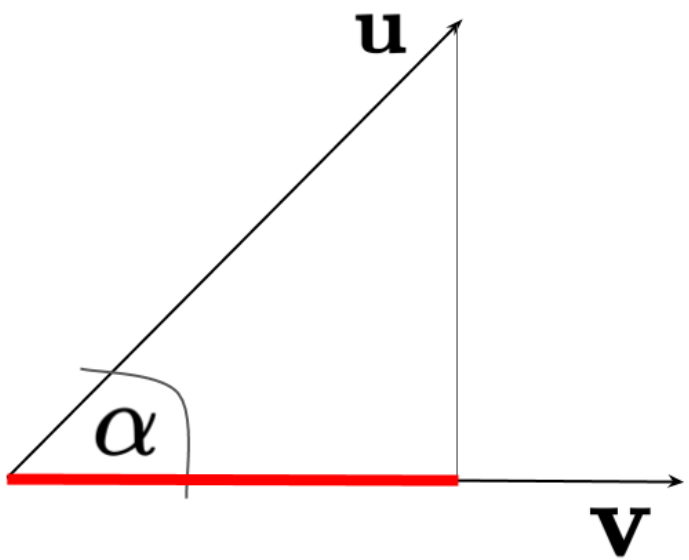
Linearly separable data

Data set $D = \{\langle \mathbf{x}_i, y_i \rangle, \mathbf{x}_i \in X, y_i \in \{-1, +1\}\}$ is **linearly separable** if there exists a hyperplane $g : \theta_0 + \Theta^\top \mathbf{x} = 0$ that separates the two classes completly, i.e.

$$\forall \langle \mathbf{x}, y \rangle \in D : \quad \overline{\rho_g}(\mathbf{x}, y) > 0$$

Dot product

- $\mathbf{u} = \langle u_1, \ldots, u_m \rangle$, $\mathbf{v} = \langle v_1, \ldots, v_m \rangle$
- algebraic definition $\mathbf{u} \cdot \mathbf{v} = u_1 v_1 + \cdots + u_m v_m$
- geometric definition $\mathbf{u} \cdot \mathbf{v} = ||\mathbf{u}|| \cdot ||\mathbf{v}|| \cdot \cos \alpha$



$$||\mathbf{v}|| = 1 \rightarrow \mathbf{u} \cdot \mathbf{v} = ||\mathbf{u}|| \cdot \cos \alpha$$

Quadratic programming

Quadratic programming

optimizes a quadratic objective function f

$$\min_{\mathbf{x}} / \max_{\mathbf{x}} f(\mathbf{x})$$

subject to constraints

$$g_i(\mathbf{x}) \leq 0, i = 1, \ldots, G$$

$$h_j(\mathbf{x}) = 0, j = 1, \ldots, H$$

Support Vector Machines
Binary classification task $Y = \{+1, -1\}$

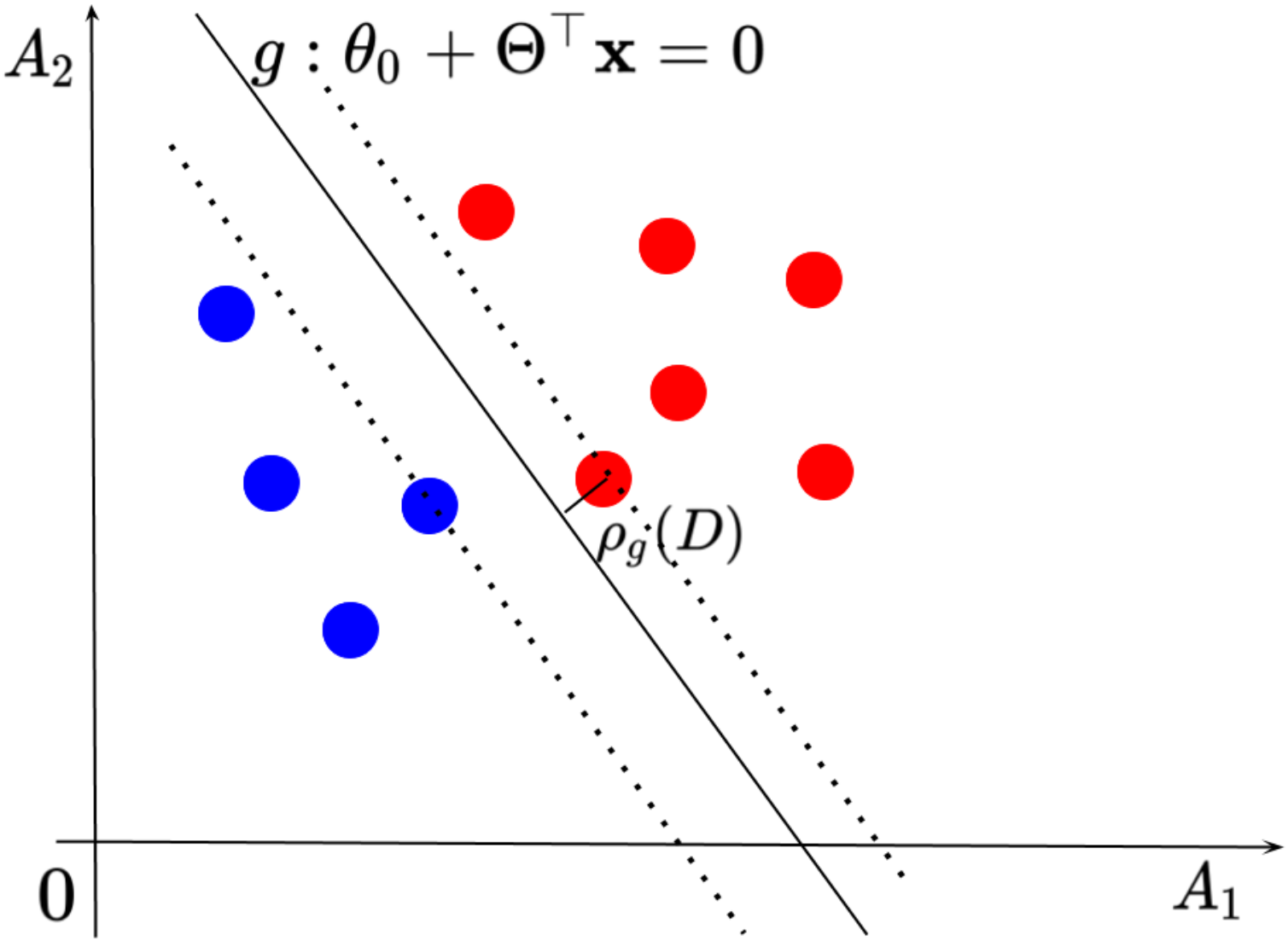
1. Large margin classifier (linear separability)
2. Soft margin classifier (not linear separability)
3. Kernels (non-linear class boundaries)

Large Margin Classifier
Training data is linearly separable

Optimization task

$$g^\star = \text{argmax}_g \rho_g(D)$$

Large Margin Classifier
Training data is linearly separable



Large Margin Classifier
Training data is linearly separable

$\Theta_0 + \Theta^\top \mathbf{x}$ and $k\Theta_0 + (k\Theta)^\top \mathbf{x}$ define the same hyperplane.

$$\frac{y_i(\Theta_0 + \Theta^\top \mathbf{x}_i)}{||\Theta||} = \frac{y_i(k\Theta_0 + (k\Theta)^\top \mathbf{x}_i)}{||k\Theta||}$$

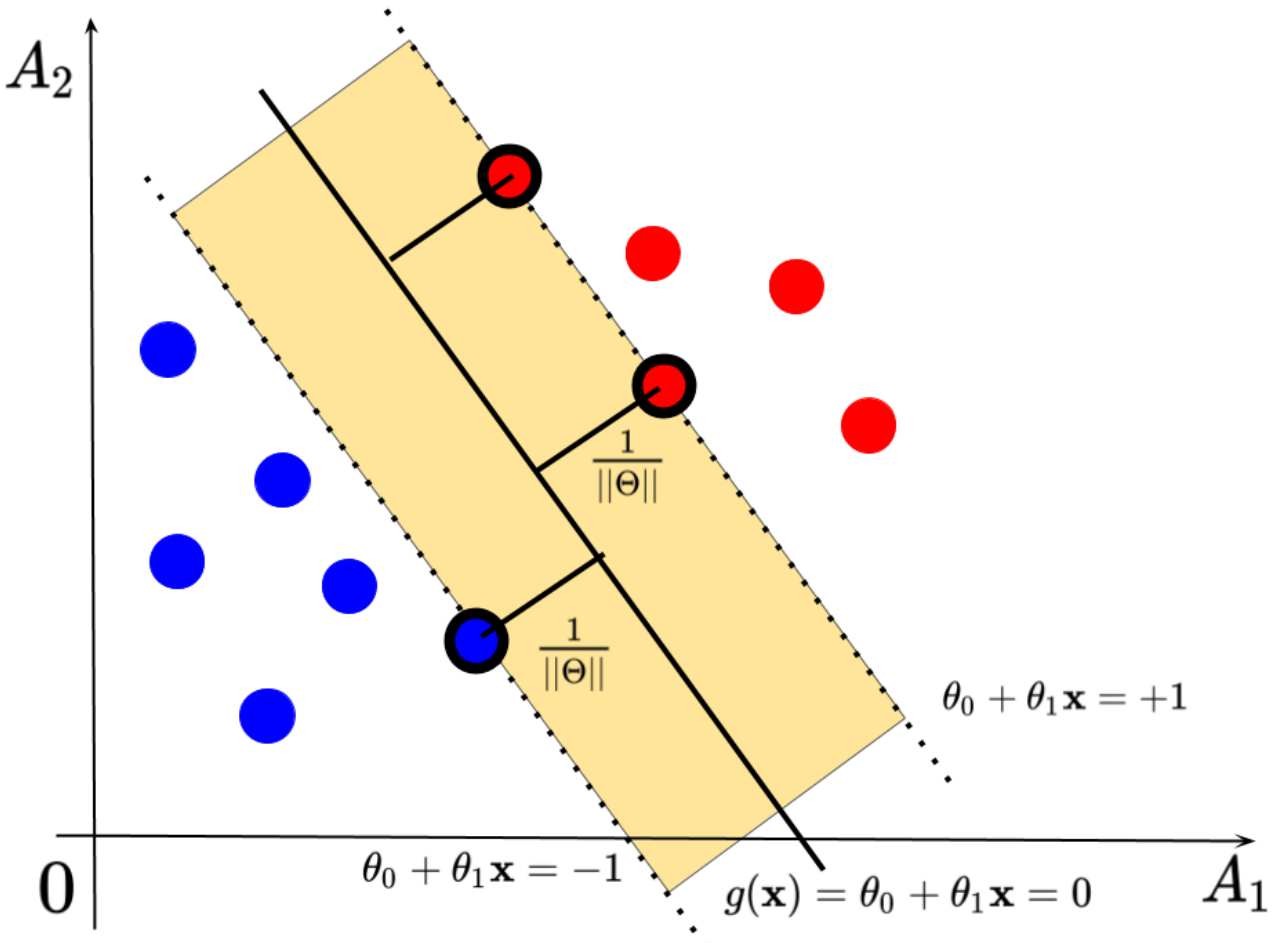
Therefore we can scale Θ so that $\overline{\rho}_g(D) = 1$.

Then

$$g^\star = \text{argmax}_g \rho_g(D) = \max_\Theta \frac{2}{||\Theta||}$$

Large Margin Classifier
Training data is linearly separable

Supporting hyperplanes $\theta_0 + \Theta^\top \mathbf{x} = -1$ and $\theta_0 + \Theta^\top \mathbf{x} = +1$.
Support vectors are the instances touching the supporting hyperplanes.



Large Margin Classifier
Training data is linearly separable

Primal problem

Minimize

$$\frac{1}{2}||\Theta||^2$$

subject to

$$y_i(\theta_0 + \Theta^\top \mathbf{x}_i) \geq 1, i = 1, \ldots, n$$

quadratic function + linear constraints $\rightarrow$ quadratic programming

Large Margin Classifier
Quadratic programming

For each constraint $y_i(\theta_0 + \Theta^\top \mathbf{x}_i) \geq 1$ introduce Lagrange multiplier $\alpha_i \geq 0$; let $\boldsymbol{\alpha} = \langle \alpha_1, \ldots, \alpha_n \rangle$.

Lagrangian function $\mathcal{L}_P(\Theta, \theta_0, \boldsymbol{\alpha})$

$$\mathcal{L}_P(\Theta, \theta_0, \boldsymbol{\alpha}) = \frac{1}{2}||\Theta||^2 + \sum_{i=1}^{n} \alpha_i (1 - y_i(\theta_0 + \Theta^\top \mathbf{x}_i)) \tag{1}$$

To get the solution of the primal problem we need to solve the **Lagrangian problem** $\rightarrow$

Large Margin Classifier
Quadratic programming

$$\min_{\Theta,\theta_0} \max_{\boldsymbol{\alpha}} \mathcal{L}_P(\Theta, \theta_0, \boldsymbol{\alpha}) \tag{2}$$

$$\max_{\boldsymbol{\alpha}} \min_{\Theta,\theta_0} \mathcal{L}_P(\Theta, \theta_0, \boldsymbol{\alpha}) \tag{3}$$

subject to

$$\alpha_i \geq 0, i = 1, \ldots n$$

1. Minimize $\mathcal{L}_P$ w.r.t. Θ
   Therefore differentiate $\mathcal{L}_P$ w.r.t. Θ and $\frac{\partial \mathcal{L}}{\partial \Theta} = 0$. It gives

$$\Theta = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{4}$$

2. Minimize $\mathcal{L}_P$ w.r.t. θ_0
   Therefore differentiate $\mathcal{L}_P$ w.r.t. θ_0 and $\frac{\partial L}{\partial \theta_0} = 0$. It gives

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{5}$$

Large Margin Classifier
Quadratic programming

3. Substitute (4) into the primal form (1) and solve **Wolfe dual optimization problem**

$$\max_{\boldsymbol{\alpha}} \mathcal{L}_D(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

subject to

$$\alpha_i \geq 0, i = 1 \ldots n$$

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

4. Get $\boldsymbol{\alpha}^\star$
5. Get $\Theta^\star = \sum_{i=1}^{n} \alpha_i^\star y_i \mathbf{x}_i$. Assume that $\mathbf{x}_i$ is a support vector. Then $1 = y_i(\theta_0^\star + {\Theta^\star}^\top \mathbf{x}_i) \rightarrow \theta_0^\star = y_i - {\Theta^\star}^\top \mathbf{x}_i$

Large Margin Classifier
Quadratic programming

- $\boldsymbol{\alpha}^\star$ is the solution to the dual problem
- $\Theta^\star$ is the solution to the primal problem
- the solutions $\boldsymbol{\alpha}^\star$ and $\Theta^\star$ must satisfy the Karush-Kuhn-Tucker conditions where one of them is *KKT dual complementarity*:

$$\alpha_i^\star \cdot (1 - y_i(\theta_0^\star + {\Theta^\star}^\top \mathbf{x}_i)) = 0$$

  - $y_i(\theta_0^\star + {\Theta^\star}^\top \mathbf{x}_i) \neq 1$, i.e., $\mathbf{x}_i$ is not support vector $\Rightarrow \alpha_i^\star = 0$
  - $\alpha_i^\star \neq 0 \Rightarrow y_i(\theta_0^\star + {\Theta^\star}^\top \mathbf{x}_i) = 1$, i.e., $\mathbf{x}_i$ is support vector

I.e., finding $\Theta^\star$ is equivalent to finding support vectors and their weights

Large Margin Classifier
Prediction

Prediction for a new instance $\mathbf{x}$

$$f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i^\star y_i \mathbf{x}_i \cdot \mathbf{x} + \theta_0^\star)$$

- similarity between $\mathbf{x}$ and support vector $\mathbf{x}_i$: a support vector that is more similar contributes more to the classification
- support vector that is more important, i.e. has larger α_i, contributes more to the classification
- if y_i is positive, than the contribution is positive, otherwise negative

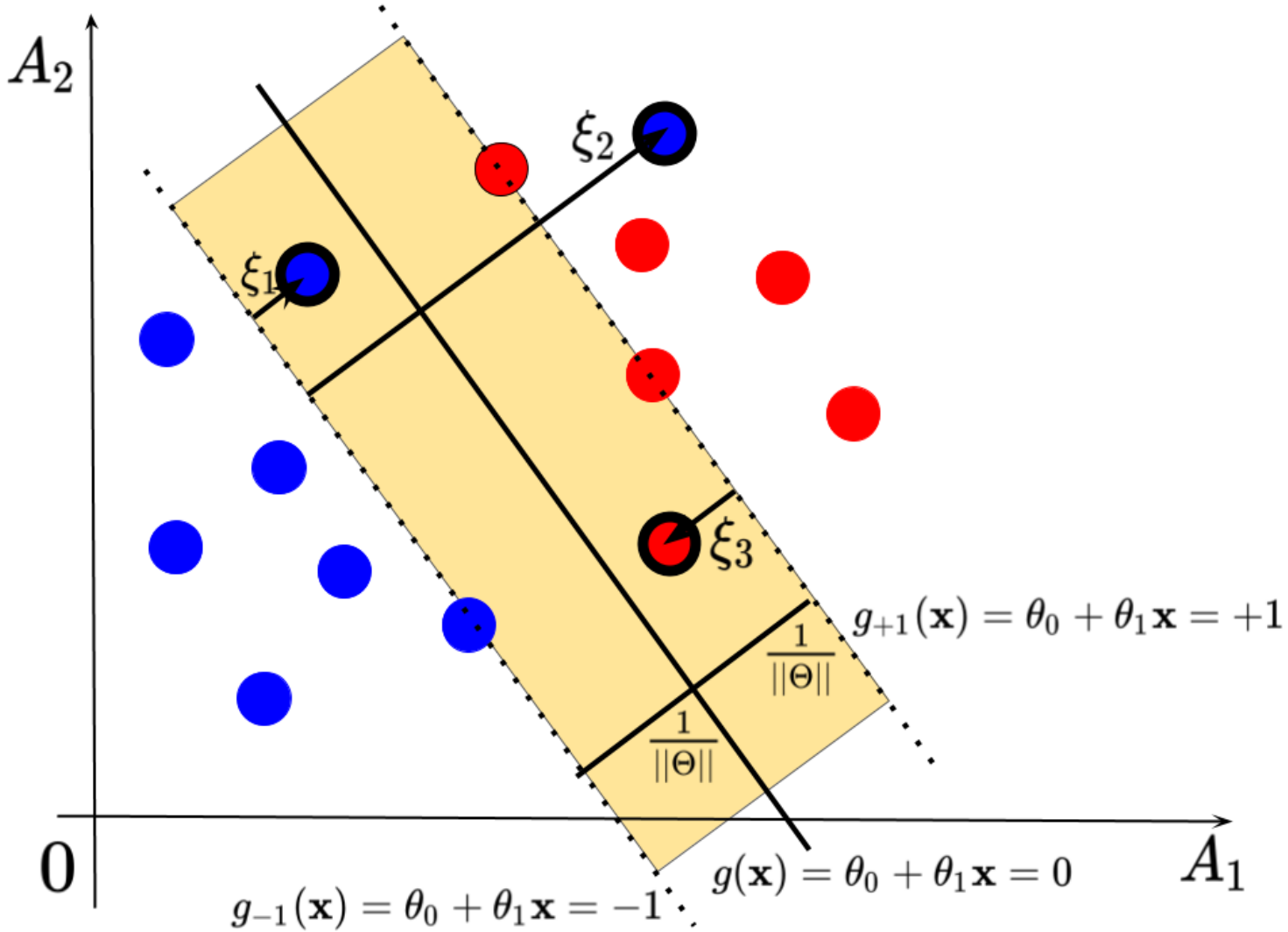
Soft Margin Classifier
Training data is not linearly separable

In a real problem it is unlikely that a hyperplane will exactly separate the data – even if a curved decision boundary is possible. So exactly separating the data is probably not desirable – if the data has noise and outliers, a smooth decision boundary that ignores a few data points is better than one that loops around the outliers.

Therefore

$$\text{minimize } ||\boldsymbol{\Theta}||^2 \textbf{ AND } \text{the number of training mistakes}$$

Soft Margin Classifier
Training data is not linearly separable



Soft Margin Classifier
Slack variables

$\xi_i \geq 0$

- $\xi_i = 0$ if $\mathbf{x}_i$ is correctly classified
- ξ_i is distance to y_i's supporting hyperplane" otherwise
  - margin violation – $0 < \xi_i \leq 1/||\Theta||$, see ξ_1, ξ_3 above
  - misclassification – $\xi_i > 1/||\Theta||$, see ξ_2 above

Soft Margin Classifier Optimization problem

Minimize

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

subject to

$$\xi_i \geq 0, y_i(\theta_0 + \Theta^\top \mathbf{x}_i) \geq 1 - \xi_i, i = 1, \ldots n$$

$C \geq 0$ trade-off parameter

- small $C \Rightarrow$ large margin
  relaxed model; misclassifications are not penalized
- large $C \Rightarrow$ narrow margin
  misclassifications are penalized strongly
  the model will not generalize much

Soft Margin Classifier
Quadratic programming

For each constraint $y_i(\theta_0 + \Theta^\top \mathbf{x}_i) \geq 1 - \xi_i$ introduce Lagrange multiplier $\alpha_i \geq 0$; let $\boldsymbol{\alpha} = \langle \alpha_1, ..., \alpha_n \rangle$.

Primal Lagrangian $\mathcal{L}_P(\Theta, \theta_0, \xi, \boldsymbol{\alpha})$ is given by

$$\mathcal{L}_P(\Theta, \theta_0, \xi, \boldsymbol{\alpha}) = \frac{1}{2}||\Theta||^2 + C\sum_{i=1}^{n} \xi_i + \sum_{i=1}^{n} \alpha_i(1 - \xi_i - y_i(\theta_0 + \Theta^\top \mathbf{x}_i)) \quad (6)$$

Soft Margin Classifier
Quadratic programming

Primal Lagrangian problem

$$\min_{\Theta,\theta_0,\xi} \max_{\boldsymbol{\alpha}} \mathcal{L}_P(\Theta, \theta_0, \xi, \boldsymbol{\alpha}) \tag{7}$$

$$\max_{\boldsymbol{\alpha}} \min_{\Theta,\theta_0,\xi} \mathcal{L}_P(\Theta, \theta_0, \xi, \boldsymbol{\alpha}) \tag{8}$$

subject to

$$\alpha_i \geq 0, \xi_i \geq 0, i = 1, \ldots, n$$

1. Minimize $\mathcal{L}_P$ w.r.t. Θ. Therefore $\frac{\partial \mathcal{L}}{\partial \Theta} = 0$. It gives

$$\Theta = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{9}$$

2. Minimize $\mathcal{L}_P$ w.r.t. θ_0. Therefore $\frac{\partial L}{\partial \theta_0} = 0$. It gives

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10}$$

Soft Margin Classifier
Quadratic programming

3. Minimize $\mathcal{L}_P$ w.r.t. ξ. Therefore $\frac{\partial L}{\partial \xi} = 0$. It gives

$$C\xi - \alpha = 0 \tag{11}$$

4. Substitute (9) into the primal form (6) and solve **Wolfe dual opt. problem**

$$\max_{\boldsymbol{\alpha}} \mathcal{L}_D(\boldsymbol{\alpha}) = \max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$

subject to

$$\alpha_i \geq 0, \quad i = 1, \ldots, n$$

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

5. Get $\boldsymbol{\alpha}^\star$
6. Get $\Theta^\star$

Soft Margin Classifier Prediction

Prediction for a new instance $\mathbf{x}$

$$f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i^\star y_i \mathbf{x}_i \cdot \mathbf{x} + \theta_0^\star)$$

Non-linear boundary

Recall polynomial regression

Polynomial regression is an extension of linear regression where the relationship between features and target value is modelled as a d-th order polynomial.

Simple regression
$y = \theta_0 + \theta_1 x_1$

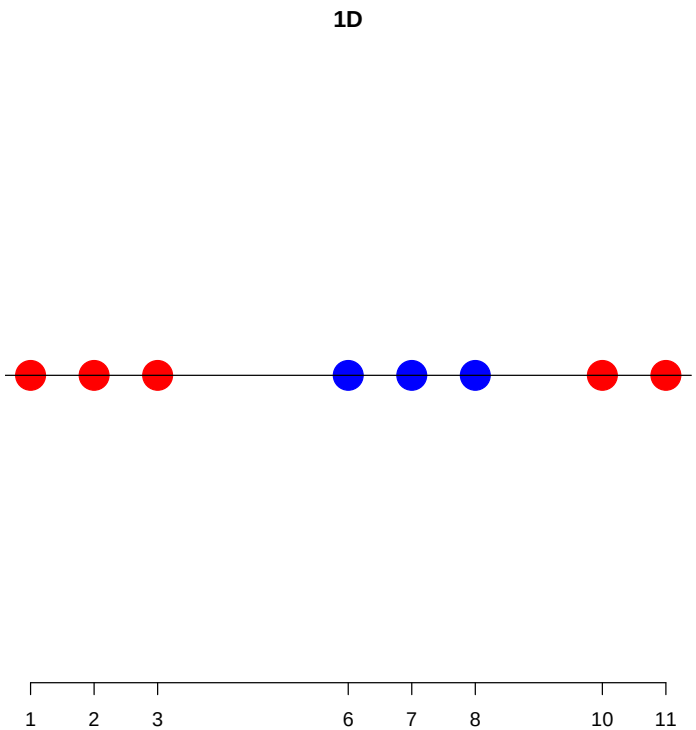
Polynomial regression
$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \ldots \theta_d x_1^d$

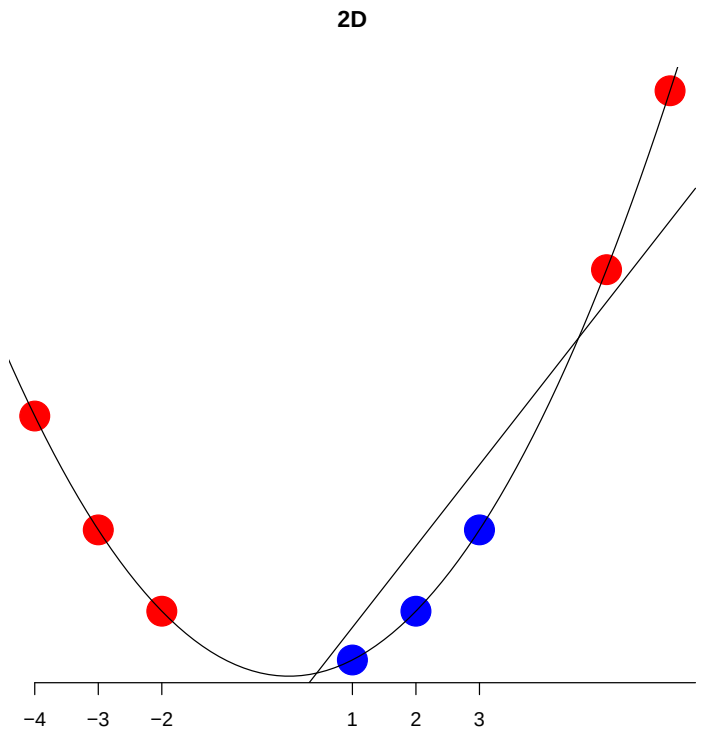
It is still a linear model with features $A_1, A_1^2, \ldots, A_1^d$.

This defines a feature mapping $\phi(x_1) = [x_1, x_1^2, \ldots, x_1^d]$

Non-linear boundary

- for example $\phi(x_1) = [x_1 - 5, (x_1 - 5)^2]$



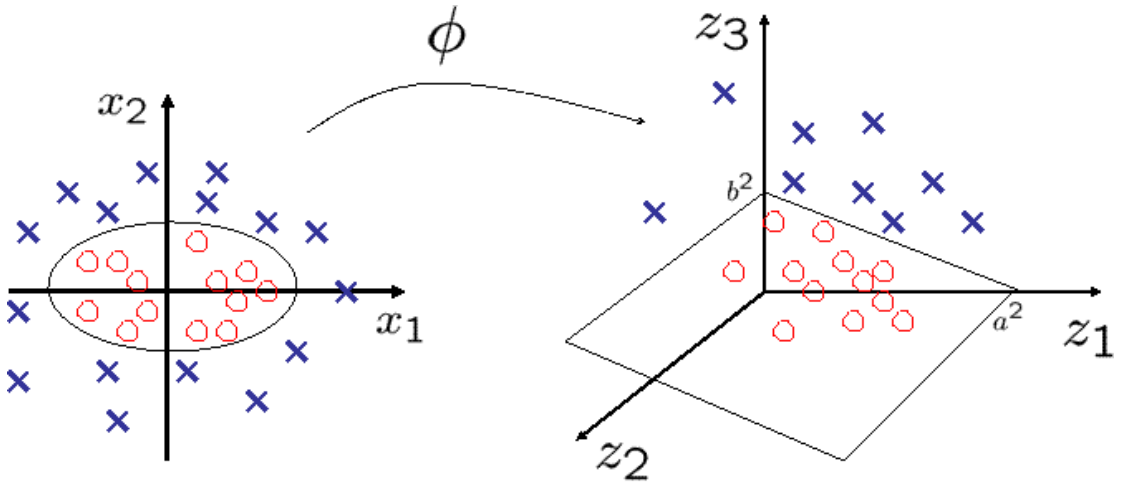


Kernels

Idea

- Apply Large/Soft margin classifier not to the orginal features but to the features obtained by the feature mapping $\phi(\mathbf{x}) : \mathcal{R}^m \to \mathcal{F}$
- Large/Soft margin classifier uses dot product $\mathbf{x}_i \cdot \mathbf{x}_j$.
  Replace it with $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$.

Kernels



$$\phi : (x_1, x_2) \longrightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\left(\frac{x_1}{a}\right)^2 + \left(\frac{x_2}{b}\right)^2 = 1 \longrightarrow \frac{z_1}{a^2} + \frac{z_3}{b^2} = 1$$

Source: `http://omega.albany.edu:8008/machine-learning-dir/notes-dir/ker1/ker1-l.html`

Kernels

However, finding ϕ could be expensive.

Kernel trick

- No need to know what ϕ is and what the feature space is, i.e. no need to explicitly map the data to the feature space
- Define a kernel function $K : \mathcal{R}^m \times \mathcal{R}^m \to \mathcal{R}$
- Replace the dot product $\mathbf{x}_i \cdot \mathbf{x}_j$ with a Kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$:

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

Kernels
Prediction

Prediction for a new instance $\mathbf{x}$

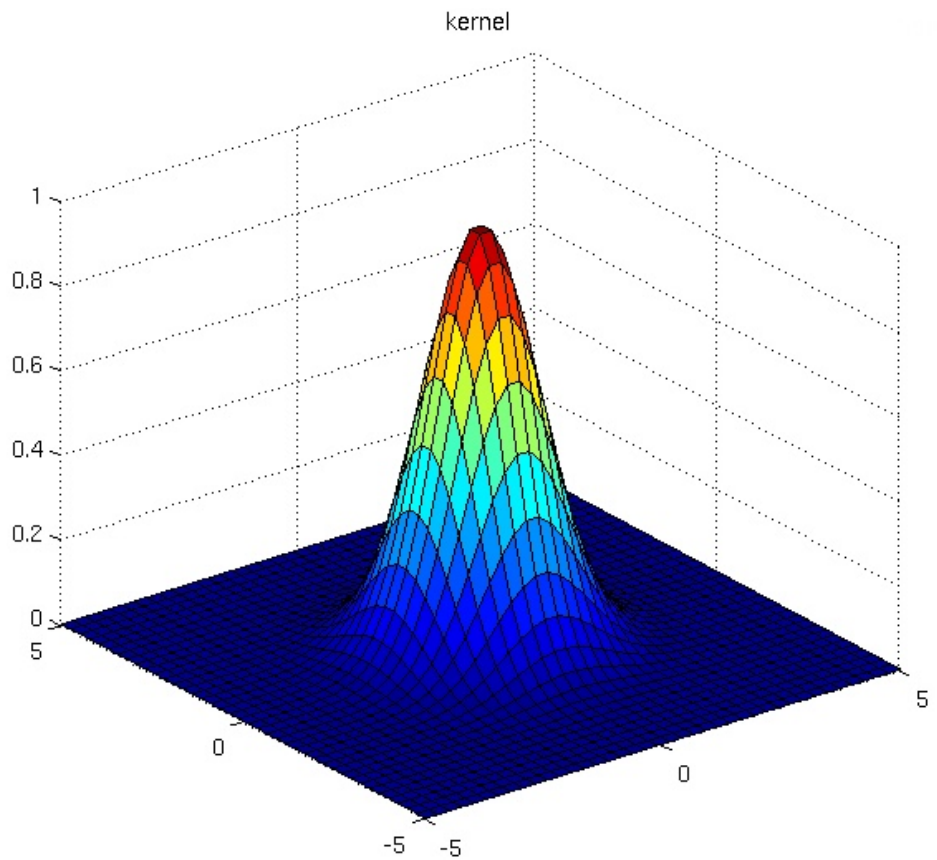
$$f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i^\star y_i K(\mathbf{x}_i, \mathbf{x}) + \theta_0^\star)$$

Common Kernel functions

- **Linear**
  $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$
- **Polynomial**
  $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)^d$
  – smaller degree can generalize better
  – higher degree can fit (only) training data better
- **Radial basis function**
  $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma(||\mathbf{x}_i - \mathbf{x}_j||^2))$
  – very robust
  – use it when polynomial kernel is weak to fit data
- **Sigmoid**
  $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i \cdot \mathbf{x}_j + c)$, where $\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$

Radial Basis Function Kernel

- $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2}$



Source: `http://www.cs.toronto.edu/~duvenaud/cookbook/index.html`

Bayes theorem

Probabilistic approach to classification $Y = \{y_{1,2}, \ldots, y_K\}$

$$y^\star = \text{argmax}_{y_k \in Y} \Pr(y_k | x_1, \ldots, x_m) \tag{12}$$

Bayes theorem

$$\text{posterior probability} = \frac{\text{prior probability} \times \text{likelihood}}{\text{marginal likelihood}} \tag{13}$$

$$\Pr(Y \mid A_1, \ldots, A_m) = \frac{\Pr(Y) \times \Pr(A_1, \ldots, A_m \mid Y)}{\Pr(A_1, \ldots, A_m)}$$

Then

$$y^\star = \text{argmax}_{y_k \in Y} \frac{\Pr(y_k) \times \Pr(x_1, \ldots, x_m | y_k)}{\Pr(x_1, \ldots, x_m)} \tag{14}$$

Conditional independence

Let X, Y and Z be three descrete random variables. We say that X is *conditionally independent* of Y given Z if

$\forall x_i, y_j, z_k, x_i \in \mathit{Values}(X), y_j \in \mathit{Values}(Y), z_k \in \mathit{Values}(Z)$:

$$\Pr(X = x_i | Y = y_j, Z = z_k) = \Pr(X = x_i | Z = z_k) \tag{15}$$

I.e., $\mathrm{P}(X|Y, Z) = \mathrm{P}(X|Z)$.

Conditional independence

Do we enjoy our favorite water sport on this day? (Credit: T. Mitchel, 1997)

Sky	AirTemp	Humidity	Wind	EnjoySport
sunny	warm	normal	strong	No
sunny	warm	high	strong	Yes
rainy	cold	high	strong	No
sunny	warm	high	strong	Yes

Conditional independence of features given *EnjoySport*: presence of one particular feature value does not affect the other features' values given *EnjoySport*, e.g., if the temperature is hot, it does not necessarily mean that the humidity is high and the features have an equal effect on the outcome

Conditional independence

If we work with two features A_1, A_2 and we assume that they are conditionally independent given the target class Y, then

$$\Pr(A_1, A_2|Y) \overset{\text{product rule}}{=} \Pr(A_1|A_2, Y) * \Pr(A_2|Y) \overset{\text{c. i. assumption}}{=} \Pr(A_1|Y) * \Pr(A_2|Y)$$

Note: Product rule (a.k.a. Chain rule)

$$\Pr(A_m, \ldots, A_1) = \Pr(A_m|A_{m-1}, \ldots, A_1) \cdot \Pr(A_{m-1}, \ldots, A_1)$$

Naïve Bayes classifier

$$y^\star = \text{argmax}_{y_k \in Y} \Pr(y_k|x_1, \ldots, x_m) = \text{argmax}_{y_k \in Y} \frac{\Pr(y_k) \Pr(x_1, \ldots, x_m|y_k)}{\Pr(x_1, \ldots, x_m)}$$

– Assume conditional independence of features $A_1, \ldots, A_m$ given Y. Then

$$\Pr(x_1, x_2, \ldots, x_m|y_k) \overset{\text{product rule}}{=} \prod_{j=1}^{m} \Pr(x_j|x_1, x_2, \ldots, x_{j-1}, y_k) \overset{\text{c. i. a.}}{=}$$

$$= \prod_{j=1}^{m} \Pr(x_j|y_k)$$

– $\Pr(x_1, \ldots, x_m)$ is constant. Then

$$y^\star = \text{argmax}_{y_k \in Y} \Pr(y_k) \prod_{j=1}^{m} \Pr(x_j|y_k) \tag{16}$$

Discriminative vs. generative classifiers

Computing $\Pr(y|\mathbf{x})$

- **discriminative classifier** does not care about how the data was generated. It directly discriminates the value of y for any $\mathbf{x}$.
- **generative classifier** models how the data was generated in order to classify an example.

Discriminative vs. generative classifiers

- Logistic regression classifier is a **discriminative classifier**

$$f(\mathbf{x}; \Theta) = p(y = 1|\mathbf{x}, \Theta)$$

- Naïve Bayes classifier is a **generative classifier**

1. Learn $\Pr(\mathbf{x}|y)$ and $\Pr(y)$
2. Apply Bayes rule to get

$$\Pr(y|\mathbf{x}) = \frac{\Pr(\mathbf{x}|y)\Pr(y)}{\Pr(\mathbf{x})} \sim \Pr(\mathbf{x}|y)\Pr(y)$$

3. Classify $\mathbf{x}$

$$y^\star = \operatorname{argmax}_y \Pr(y|\mathbf{x}) = \operatorname{argmax}_y \Pr(\mathbf{x}|y)\Pr(y)$$

Naïve Bayes classifier

Naive assumption of feature conditional independence given a target class is rarely true in real world applications (high bias). Nevertheless, Naïve Bayes classifier surprisingly often shows good performance in classification (low variance).

Bias-Variance Trade-off -> next lecture

Naïve Bayes Classifier is a linear classifier

NB classifier gives a method for predicting rather than for building an explicit classifier.

Let us focus on **binary classification** $Y = \{0, 1\}$ with binary features $A_1, \ldots, A_m$.

We predict 1 iff

$$\frac{\Pr(y = 1) \prod_{j=1}^{m} \Pr(x_j | y = 1)}{\Pr(y = 0) \prod_{j=1}^{m} \Pr(x_j | y = 0)} > 1$$

Naïve Bayes Classifier is a linear classifier

Denote $p_j = \Pr(x_j = 1|y = 1)$, $q_j = \Pr(x_j = 1|y = 0)$

Then

$$\frac{\Pr(y = 1) \prod_{j=1}^{m} p_j^{x_j} (1 - p_j)^{1-x_j}}{\Pr(y = 0) \prod_{j=1}^{m} q_j^{x_j} (1 - q_j)^{1-x_j}} > 1$$

$$\frac{\Pr(y = 1) \prod_{j=1}^{m} (1 - p_j)(\frac{p_j}{1-p_j})^{x_j}}{\Pr(y = 0) \prod_{j=1}^{m} (1 - q_j)(\frac{q_j}{1-q_j})^{x_j}} > 1$$

Naïve Bayes Classifier is a linear classifier

Take logarithm

$$\log \frac{\Pr(y=1)}{\Pr(y=0)} + \sum_{j=1}^{m} \log \frac{1-p_j}{1-q_j} + \sum_{j=1}^{m} (\log \frac{p_j}{1-p_j} - \log \frac{q_j}{1-q_j}) x_j > 0$$

NB classifier as a linear classifier where

$$\theta_0 = \log \frac{\Pr(y=1)}{\Pr(y=0)} + \sum_{j=1}^{m} \log \frac{1-p_j}{1-q_j}$$

$$\theta_j = \log \frac{p_j}{1-p_j} - \log \frac{q_j}{1-q_j}, \quad j = 1, \ldots, m$$

Bayesian belief networks (BBN)

- Naïve Bayes classifier assumes that ALL features are conditionally independent given a target attribute.
- A Bayesian network is a probabilistic graphical model that encodes probabilistic relationships among attributes of interest.
- BBNs allow stating conditional independence assumptions that apply to subsets of the attributes.
- Dependencies are modeled as graph where nodes correspond to attributes and edges to dependency between attributes.

Bayesian belief networks
Settings

Consider an arbitrary set of random variables $X_1, X_2, ..., X_m$. Each variable X_i can take on the set of possible values $Values(X_i)$.

We define the **joint space** of the variables $X_1, X_2, ..., X_m$ to be the cross product $Values(X_1) \times Values(X_2) \times Values(X_3) \times ... \times Values(X_m)$.

The probability distribution over the joint space is called the **joint probability distribution** $\Pr(x_1, x_2, ..., x_m)$ where
$x_1 \in Values(X_1), x_2 \in Values(X_2), ..., x_n \in Values(X_m)$.

BBN describes the joint probability distribution for a set of variables by specifying a set of conditional independence assumptions together with sets of local conditional probabilities.

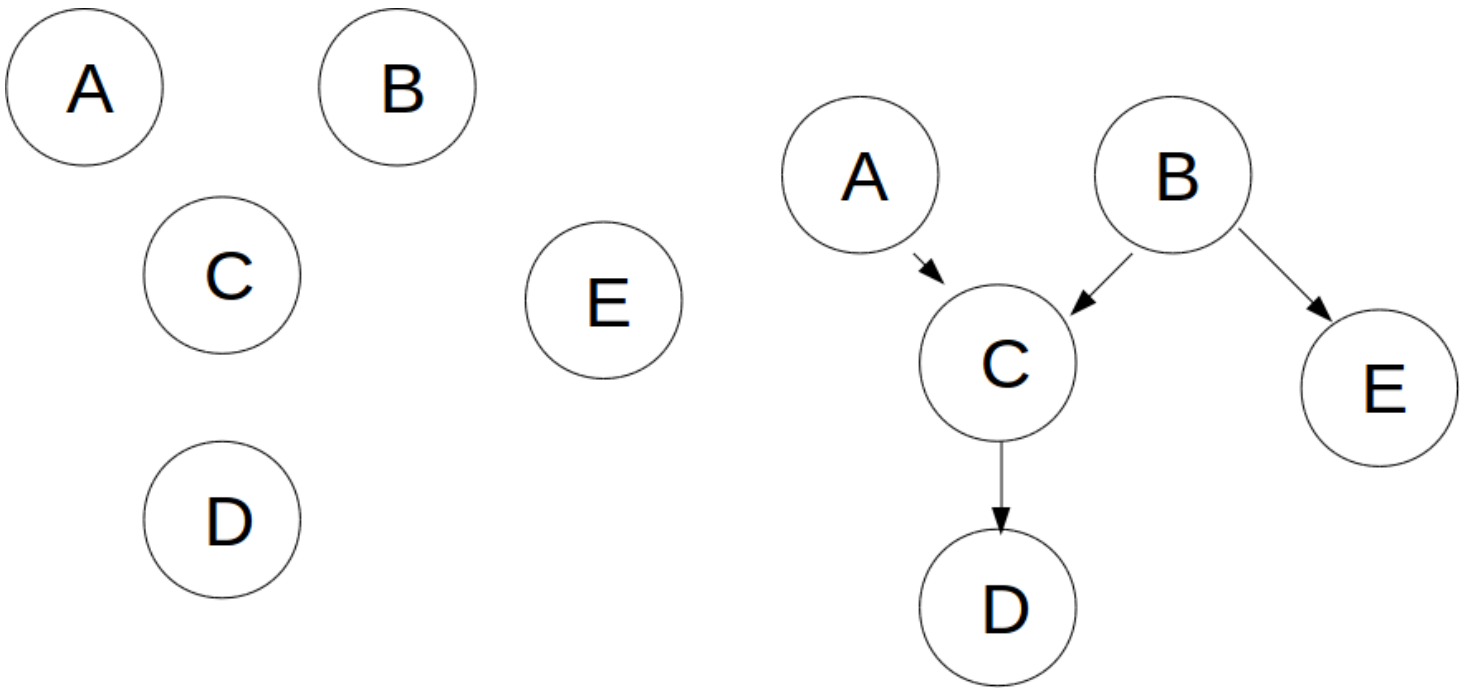
Bayesian belief networks

Representation

1. A directed acyclic graph $G = (V, E)$
   - nodes are random variables
   - arcs between nodes represent probabilistic dependencies
   - Y is a *descendant* of X if there is a directed path from X to Y
2. The network arcs represent the assertion that the variable X is conditionally independent of its nondescendants given its immediate predecessors $Parents(X)$; $\Pr(X|Parents(X))$
3. A set of tables for each node in the graph - a conditional probability table is given for each variable; it describes the probability distribution for that variable given the values of its immediate predecessors.

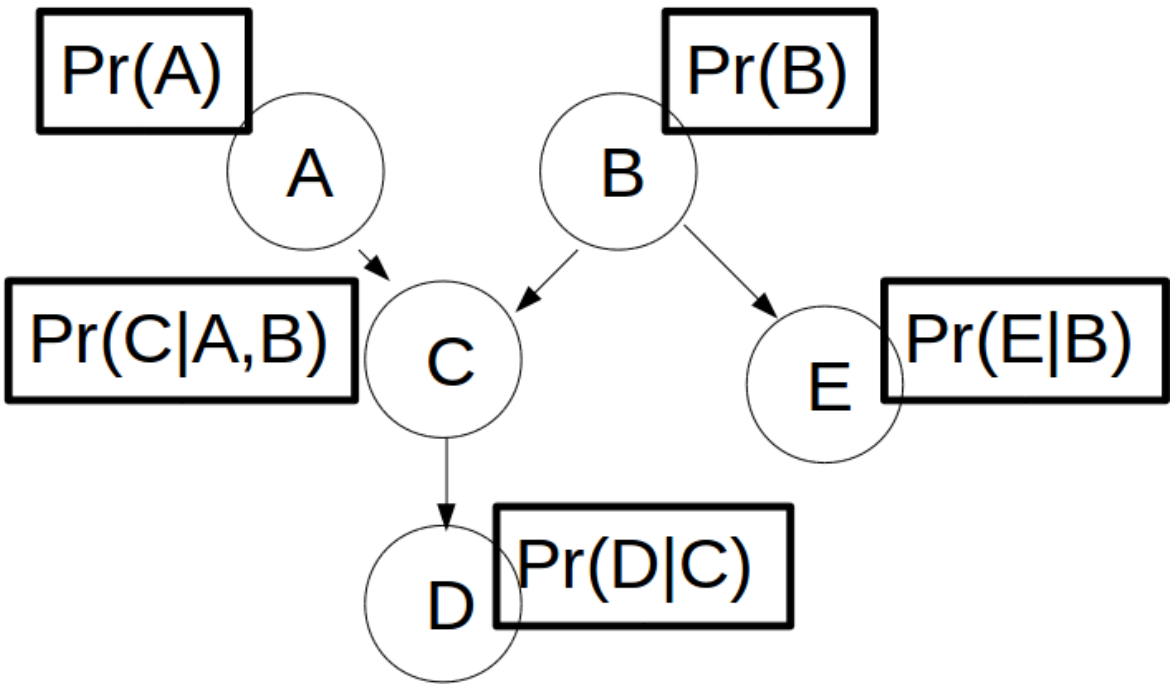
Building a Bayes net

1. Choose the variables to be included in the net: A, B, C, D, E
2. Add the links



Building a Bayes net

3. Add a probability table for each root node $\Pr(X)$ and nonroot node $\Pr(X|Parents(X))$



Once the net is built ...

The join probability of any assignment of values $x_1, x_2, ..., x_m$ to the tuple of network variables $X_1, X_2, ..., X_m$ can be computed by the formula

$$\Pr(x_1, x_2, ..., x_m) = \Pr(X_1 = x_1 \wedge X_2 = x_2 \wedge \cdots \wedge X_m = x_m) = \prod_{i=1}^{m} \Pr(x_i | Parents(X_i)) \tag{17}$$

Bayesian belief networks

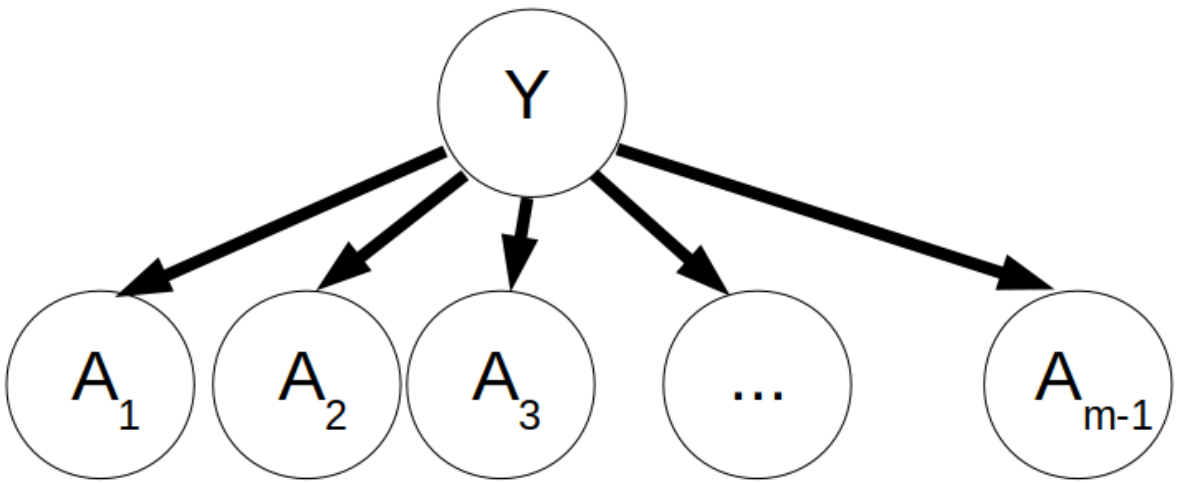
Two components

1. A function for evaluating a given network based on the data.
2. A method for searching through the space of possible networks.

Learning the network structure

- searching through the space of possible sets of edges
- estimating the conditional probability tables for each set
- computing the quality of the network

Bayesian belief networks
Naïve Bayes Classifier



K2 algorithm

This 'search and score' algorithm heuristically searches for the most probable belief-network structure given a training data.

It starts by assuming that a node has no parents, after which, in every step it adds incrementally the parent whose addition mostly increase the probability of the resulting structure. K2 stops adding parents to the nodes when the addition of a single parent cannot increase the probability of the network given the data.

Summary of Examination Requirements

- Hyperplane, margin, functional margin, geometric margin of example and data set
- Large margin classifier
  linearly separable data, supporting hyperplanes, support vectors, optimization task, prediction function
- Soft margin classifier
  not linearly separable data, supporting hyperplanes, support vectors, slack variables, optimization task, hyperparameter C, prediction function
- Kernel trick
  feature mapping, Kernel functions, prediction function
- Discriminative and generative classifiers
- Naïve Bayes Classifier
  conditional independence, linear decision boundary
- Bayesian networks
  structure, conditional probabilities