Machine Learning

Linear Models for Classification II

Manfred Jaeger

Aalborg University

Probabilistic Models

Classify data instance $\boldsymbol{x}$ to belong to the class k, for which probability

$$P(Y = k \mid \boldsymbol{x})$$

is maximal.

Classify data instance $\boldsymbol{x}$ to belong to the class k, for which probability

$$P(Y = k \mid \boldsymbol{x})$$

is maximal.

Generative Approach

- Define a **joint distribution** for the features $\boldsymbol{X} = X_1, \ldots, X_D$ and the class variable Y with values $1, \ldots, K$. Usually factored as $P(Y, \boldsymbol{X}) = P(Y)P(\boldsymbol{X} \mid Y)$.
- Compute

$$P(Y = k \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{P(Y = k, \boldsymbol{X} = \boldsymbol{x})}{P(\boldsymbol{X} = \boldsymbol{x})} = P(\boldsymbol{X} = \boldsymbol{x} \mid Y = k)\frac{P(Y = k)}{P(\boldsymbol{X} = \boldsymbol{x})}$$

Example: Naive Bayes, LDA

Classify data instance $\boldsymbol{x}$ to belong to the class k, for which probability

$$P(Y = k \mid \boldsymbol{x})$$

is maximal.

Generative Approach

- Define a **joint distribution** for the features $\boldsymbol{X} = X_1, \ldots, X_D$ and the class variable Y with values $1, \ldots, K$. Usually factored as $P(Y, \boldsymbol{X}) = P(Y)P(\boldsymbol{X} \mid Y)$.
- Compute

$$P(Y = k \mid \boldsymbol{X} = \boldsymbol{x}) = \frac{P(Y = k, \boldsymbol{X} = \boldsymbol{x})}{P(\boldsymbol{X} = \boldsymbol{x})} = P(\boldsymbol{X} = \boldsymbol{x} \mid Y = k)\frac{P(Y = k)}{P(\boldsymbol{X} = \boldsymbol{x})}$$

Example: Naive Bayes, LDA

Discriminative Approach

Directly learn the conditional distribution

$$P(Y \mid \boldsymbol{X})$$

Example: Logistic Regression, (Neural Networks)

Operations supported:

	Generative	Discriminative
Predicting Y given $\boldsymbol{X}$	yes	yes
Predicting any other variable X_i	yes (compute arbitrary conditional probabilities $P(X_i \mid \ldots)$)	no
Generating data points $(\boldsymbol{x}, y)$	yes	no

- Operations that are supported (in principle) by generative models are not always supported in a computationally efficient manner.
- For a fixed prediction task, discriminative models can be more efficient to learn (w.r.t the required data), and they can be more interpretable.

Notation: Upper case $X, \boldsymbol{X}$ for (tuples of) *variables*. Lower case $x, \boldsymbol{x}$ for (tuples of) specific values that the variables can have. **Example:** $X = \mathit{Temperature}, x = 23.5C$.

In all cases, the main criterion for learning probabilistic models is maximizing the likelihood:

Generative

$$\prod_{n=1}^{N} P(\boldsymbol{x}_n, y_n)$$

Discriminative

$$\prod_{n=1}^{N} P(y_n \mid \boldsymbol{x}_n)$$

Always required: a **parametric model** for the (conditional) distributions, i.e. specific functional forms

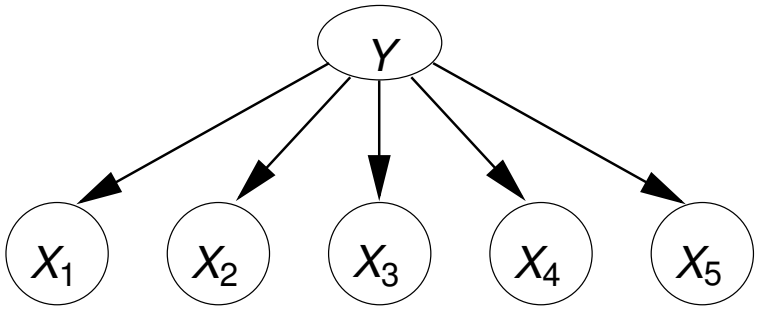
$$P(Y \mid \boldsymbol{w}),\ P(\boldsymbol{X} \mid Y, \boldsymbol{w}),\ P(Y \mid \boldsymbol{X}, \boldsymbol{w})$$

defined in terms of parameter vectors $\boldsymbol{w}$.

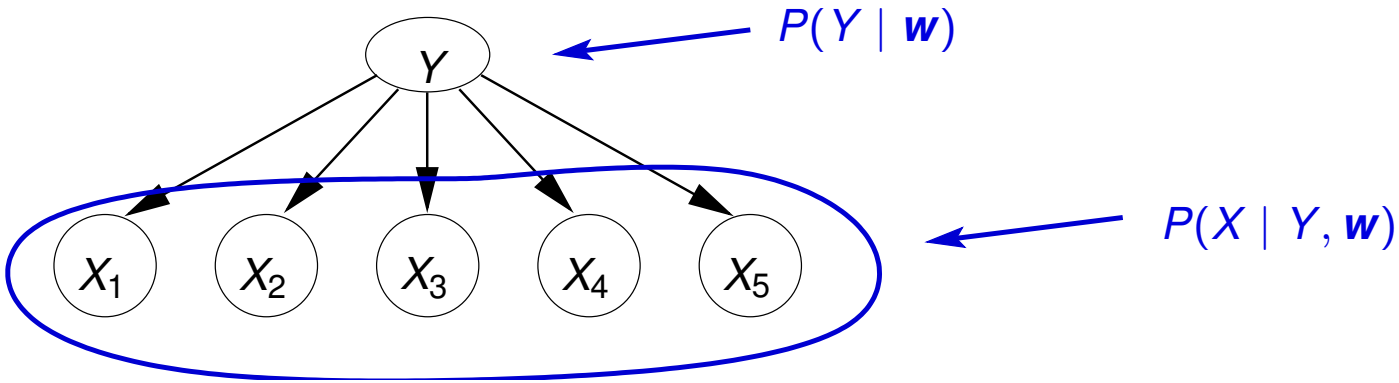
Model $P(Y \mid \boldsymbol{w})$ usually **multinomial ("multinoulli")**, i.e.

$$\boldsymbol{w} = (P(Y = 1), P(Y = 2), \ldots, P(Y = K - 1))$$

The naive Bayes model is a probabilistic generative model:



The naive Bayes model is a probabilistic generative model:



$P(Y \mid \boldsymbol{w})$: Conditional probability table for node Y; $\boldsymbol{w}$: entries in the conditional probability table

$P(X_i \mid Y, \boldsymbol{w})$:

- if X_i categorical (discrete): conditional probability tables with entries defined by $\boldsymbol{w}$
- if X_i continuous: e.g.: every distribution $P(X_i|Y, \boldsymbol{w})$ is a Gaussian distribution with parameters (mean,variance) defined by $\boldsymbol{w}$.

  ➥a simple Gaussian mixture model

Gaussian Mixtures

The Gaussian mixture model for a D-dimensional feature space, without the naive Bayes independence assumptions:

- $P(Y)$ multinomial
- $P(\boldsymbol{X} \mid Y)$ Gaussian, i.e. for all $k = 1, \ldots, K$:

$$P(\boldsymbol{X} = \boldsymbol{x} \mid Y = k) = \frac{1}{(2\pi)^{D/2} \mid \Sigma_k \mid^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \Sigma_k^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_k)}$$

  defined by parameters $\boldsymbol{\mu}_k$ (mean vectors) and Σ_k (co-variance matrices).

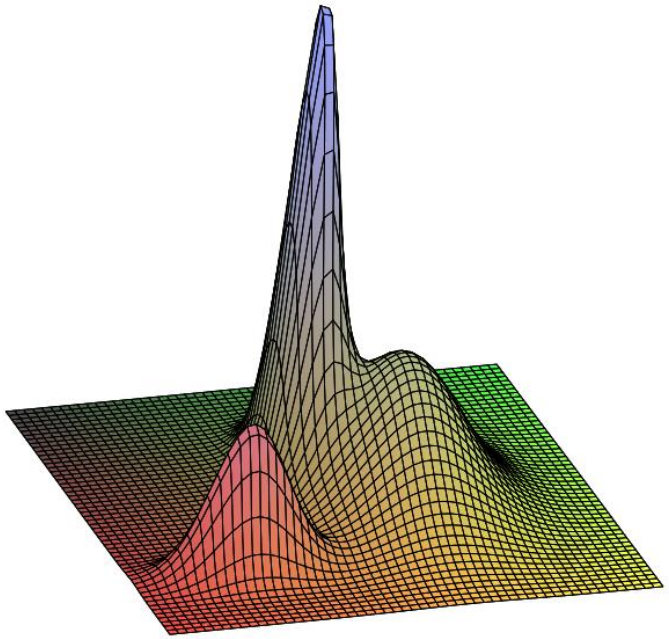
Example

- $Y = \mathit{Region} \in \{\mathit{Nordjylland}, \ldots, \mathit{Hovedstaden}\}$ ($K = 5$).
- $\boldsymbol{X} = (X_1, X_2)$ with X_1: annual income, X_2: annual housing expenditure (mortgage, rent).
- $\boldsymbol{\mu}_k = (\mu_{1,k}, \mu_{2,k})$: average income and housing costs in region k.
- $\Sigma_k = \begin{pmatrix} \sigma_{1,1,k} & \sigma_{1,2,k} \\ \sigma_{2,1,k} & \sigma_{2,2,k} \end{pmatrix}$

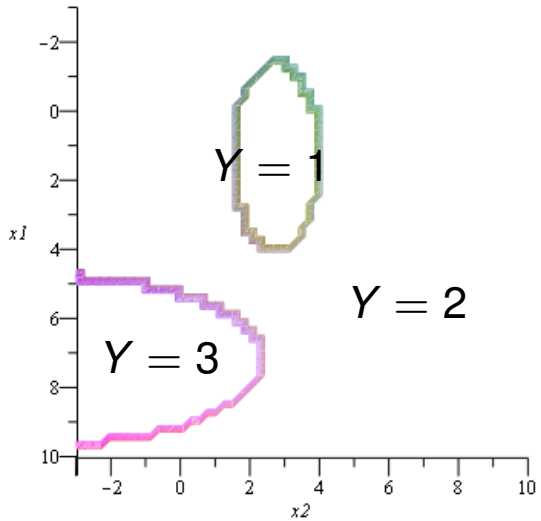
➥Presumably: X_1 and X_2 are positively correlated, i.e., $\sigma_{1,2,k} (= \sigma_{2,1,k}) > 0$.

Example with $D = 2$ and diagonal covariance matrices (= naive Bayes!):

$$P(Y = 1) = 0.3, \quad P(Y = 2) = 0.6, \quad P(Y = 3) = 0.1$$
$$\mu_1 = (2, 3), \ \mu_2 = (4, 5), \ \mu_3 = (7, 0)$$
$$\Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0.25 \end{pmatrix}, \ \Sigma_2 = \begin{pmatrix} 4 & 0 \\ 0 & 2.25 \end{pmatrix}, \ \Sigma_3 = \begin{pmatrix} 0.25 & 0 \\ 0 & 1 \end{pmatrix}$$



Distribution of $\boldsymbol{X}$

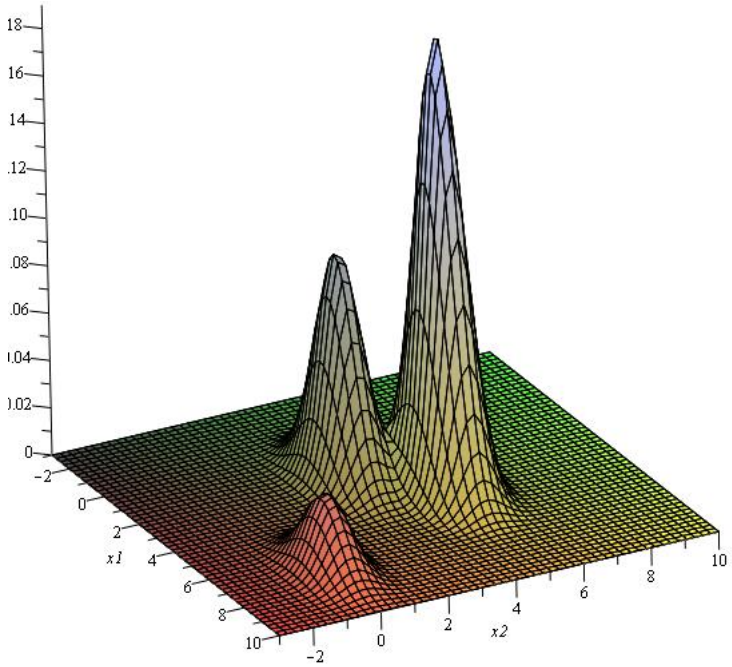


Decision boundaries

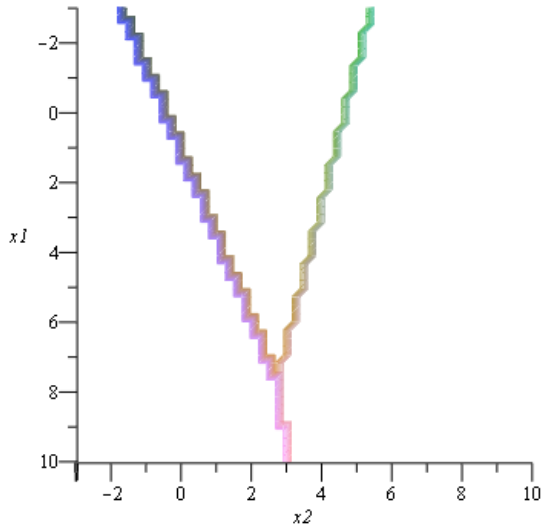
LDA: special case where all covariance matrices are identical. [Warning: not to be confused with *Fisher's linear discriminant* – Bishop Sec. 4.2.1 describes LDA, but does not use this name!]

Example:

$$P(Y=1)=0.3,\ P(Y=2)=0.6,\ P(Y=3)=0.1$$
$$\mu_1=(2,3),\ \mu_2=(4,5),\ \mu_3=(7,0)$$
$$\Sigma_1=\Sigma_2=\Sigma_3=\begin{pmatrix}1 & 0\\ 0 & 0.25\end{pmatrix}$$

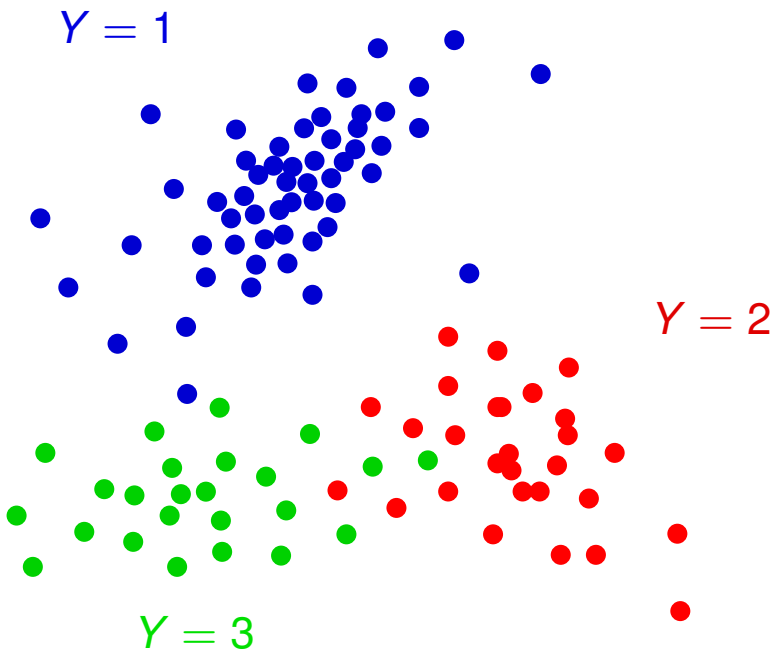


Distribution of $\boldsymbol{X}$



Decision boundaries linear!

Observed data:



Total of N data points $(\boldsymbol{x}_n, y_n)$;
N_k data points belonging to class k

Maximum likelihood parameters:

- $P(Y = k) := \frac{N_k}{N}$
- $\mu_k := \frac{1}{N_k} \sum_{n:y_n=k} \boldsymbol{x}_n$
- $\Sigma_k[i,j] := \frac{1}{N_k} \sum_{n:y_n=k} (\boldsymbol{x}_n[i] - \mu_k[i])(\boldsymbol{x}_n[j] - \mu_k[j])$ (unrestricted mixtures)
- $\Sigma[i,j] := \frac{1}{N} \sum_k \sum_{n:y_n=k} (\boldsymbol{x}_n[i] - \mu_k[i])(\boldsymbol{x}_n[j] - \mu_k[j])$ (LDA)

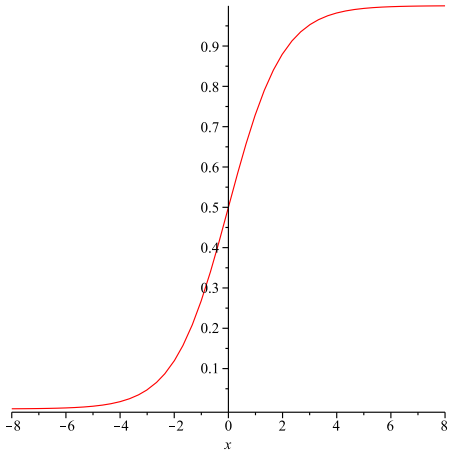
Logistic Regression

Assume continuous features $\boldsymbol{X}$, and two classes: $Y \in \{0, 1\}$.

Define

$$P(Y = 1 \mid \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x})$$

where $\boldsymbol{w}$ is a weight vector, and σ is the sigmoid function:



$$\sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$

Then

$$P(Y = 1 \mid \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{w}) \geq 0.5 \;\Leftrightarrow\; \boldsymbol{w} \cdot \boldsymbol{x} \geq 0$$

$\rightsquigarrow$ linear decision boundary!

➥Not a generative model: we cannot use model as a random generator of $(\boldsymbol{x}, y)$ data.

Categorical Features

- Categorical feature X with values $\{x_1, \ldots, x_h\}$ is coded by h "numerical" features via one-hot-encoding.

Categorical Features

- Categorical feature X with values $\{x_1, \ldots, x_h\}$ is coded by h "numerical" features via one-hot-encoding.

More than 2 classes

Alternative view of logistic regression formula:

$$\frac{P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w})}{P(Y=0 \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w})} = e^{\boldsymbol{w}\cdot\boldsymbol{x}} \qquad \text{or} \qquad \log\left(\frac{P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w})}{P(Y=0 \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w})}\right) = \boldsymbol{w}\cdot\boldsymbol{x}$$

➥The linear function $\boldsymbol{w}\cdot\boldsymbol{x}$ defines the *log-odds* of class 1 against class 0.

For multi-class: $Y \in \{1, \ldots, K\}$:

- Pick a *reference class*, e.g., $Y = K$
- Learn $K-1$ models (= parameters $\boldsymbol{w}_k$) for the (log) odds
  $odds(k, K, \boldsymbol{x}) = P(Y=k \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w}_k)/P(Y=K \mid \boldsymbol{X}=\boldsymbol{x}, \boldsymbol{w}_k)$ $(k=1,\ldots,K-1)$
- Classify point $\boldsymbol{x}$ as the k for which $odds(k, K, \boldsymbol{x})$ is maximal, if this is > 1, and as K otherwise.

➥K discriminant functions: $odds(k, K, \boldsymbol{x})$ $(k=1,\ldots,K-1)$ and $odds(K, K, \boldsymbol{x}) \equiv 1$.

Maximize likelihood

$$\prod_{n:y_n=1} P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}) \prod_{n:y_n=0} (1 - P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}))$$

Taking the log: maximize

$$\sum_{n:y_n=1} \log P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}) + \sum_{n:y_n=0} \log (1 - P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}))$$

Rewritten, using the $0, 1$ encoding of the two classes:

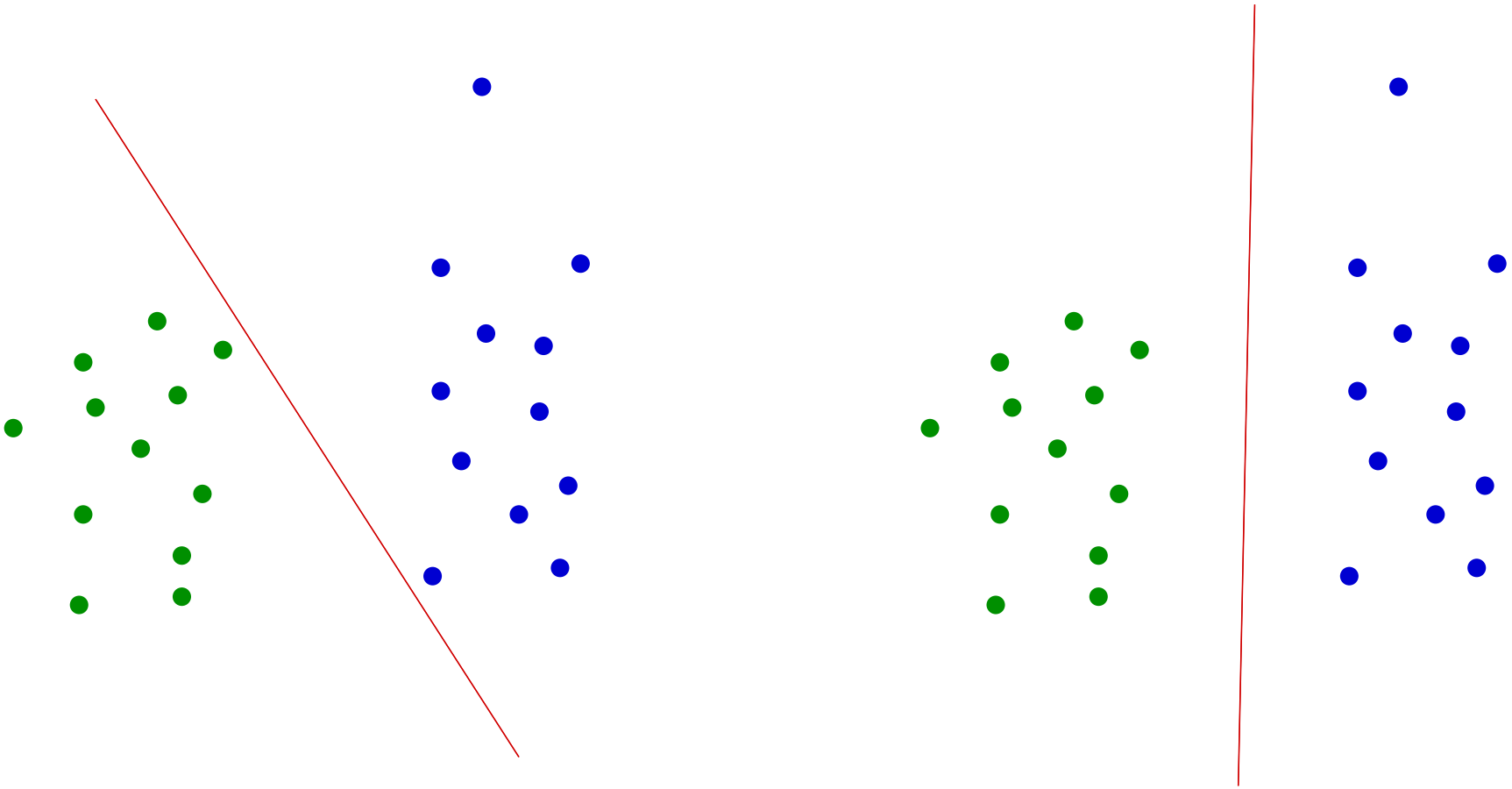
$$\sum_{n=1}^{N} y_n \log P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}) + (1 - y_n) \log (1 - P(Y=1 \mid \boldsymbol{X}=\boldsymbol{x}_n, \boldsymbol{w}))$$

- Set derivative to zero; find root of derivative using numerical methods, often Newton-Raphson

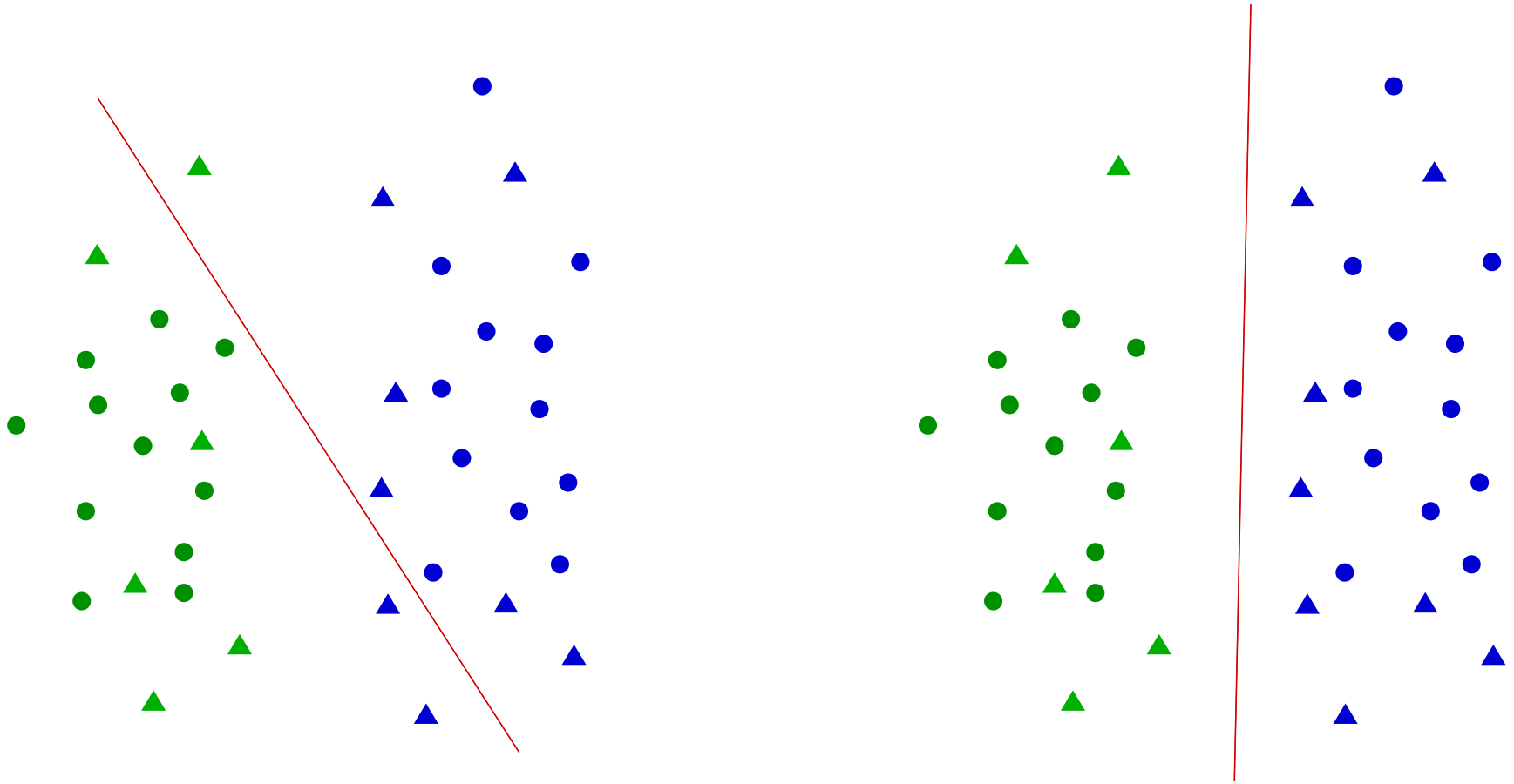
	Least Squares	Perceptron	LDA	Logistic Regresssion
Criterion	Approximate one-hot class encoding	Minimize error function	Maximize Likelihood	Maximize Likelihood
Multi-class	Yes	No, but extendible	yes	No, but extendible
Learning	Solving matrix equation	Iterative optimization	One-step optimization	Iterative optimization
Finds separating decision boundary if classes linearly separable ($K = 2$):	not always	yes	not always	not always
Works for not linearly separable data	yes	poorly (may not converge)	yes	yes

Maximum Margin Hyperplanes

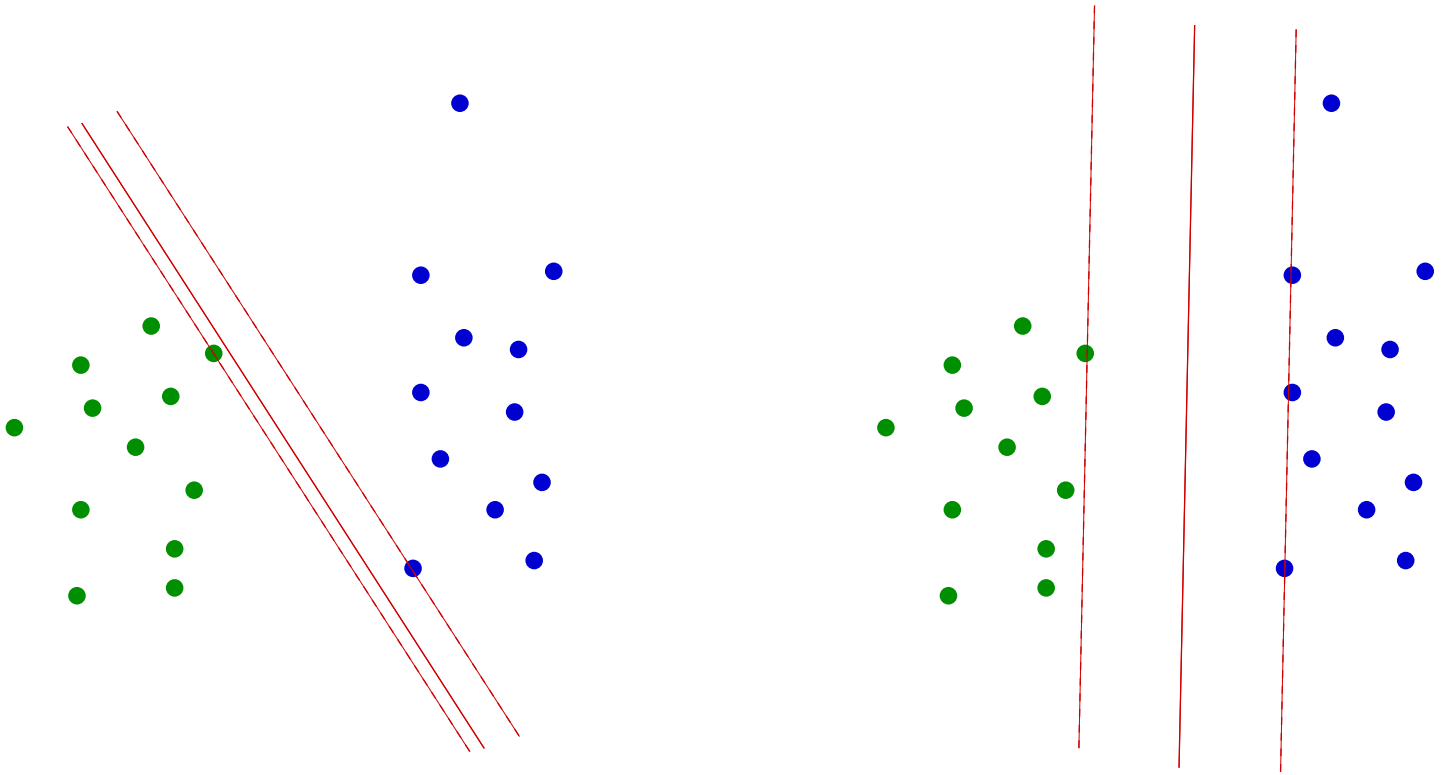
Two separating hyperplanes:



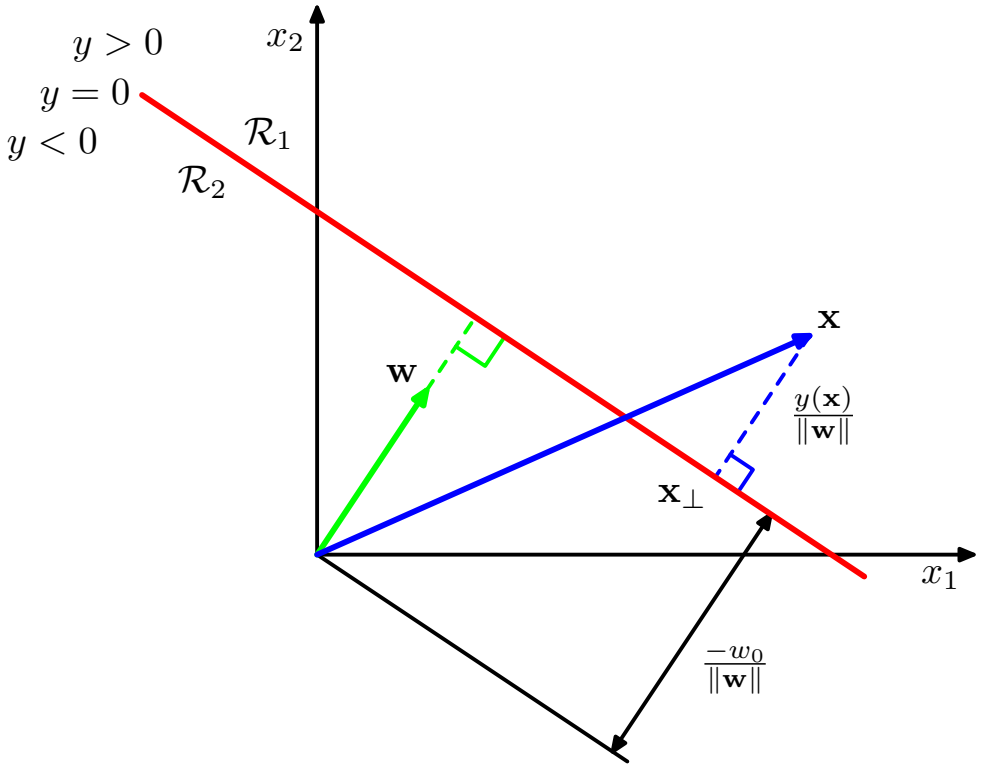
Two separating hyperplanes:



Hyperplane on the right expected to have lower error on new test data (triangles)



- Margin of a separating hyperplane: minimum distance of any datapoint to the hyperplane
- **Maximum-margin hyperplane**: hyperplane with maximum margin.
- For Max-margin hyperplane:
  - minimum distance of point from green class to hyperplane = minimum distance of point from blue class to hyperplane
  - there are datapoint in both classes whose distance to the hyperplane equals the margin. These are the **support vectors** of the hyperplane



$y(\boldsymbol{x}) = w_0 + \boldsymbol{w} \cdot \boldsymbol{x}$

Distance of $\boldsymbol{x}$ to hyperplane $y = 0$:

$$\frac{|y(\boldsymbol{x})|}{\| \boldsymbol{w} \|}$$

(in Figure: $y(\mathbf{0}) = w_0 < 0$ and $y(\boldsymbol{x}) > 0$, so $y(\boldsymbol{x}) = |y(\boldsymbol{x})|$)

[Bishop, Fig. 4.1]

With class encoding $Y \in \{-1, 1\}$:

distance of datapoint $\boldsymbol{x}_n$ with label y_n to hyperplane $y = 0$ (if datapoint lies on correct side of the hyperplane, i.e., $y_n = 1$ and $y(\boldsymbol{x}_n) > 0$, or $y_n = -1$ and $y(\boldsymbol{x}_n) < 0$): $\dfrac{y_n y(\boldsymbol{x}_n)}{\| \boldsymbol{w} \|}$

("negative distance" for points that lie on the wrong side of the hyperplane)

(Assume that data is linearly separable!)

Find hyperplane defined by $\boldsymbol{w}, b$ as

$$\underset{\boldsymbol{w},b}{arg\,max}\left\{\frac{1}{\parallel \boldsymbol{w} \parallel}\min_n[y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)]\right\}$$

(b formerly known as w_0).

Simplification

The decision boundary (and distance of datapoints to decision boundary) is not changed by scaling $\boldsymbol{w}, b$ with a common factor κ:

$$\boldsymbol{w}\mapsto\kappa\boldsymbol{w},\ b\mapsto\kappa b$$

$\rightsquigarrow$ can calibrate $\boldsymbol{w}, b$ so that for support vectors $\boldsymbol{x}_n$:

$$y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)=1,$$

and for *all* datapoints $\boldsymbol{x}_n$:

$$y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\geq 1.$$

The original optimization problem

$$\underset{\boldsymbol{w},b}{arg\,max}\left\{\frac{1}{\parallel \boldsymbol{w} \parallel}\min_n[y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)]\right\}$$

then becomes: find

$$\underset{\boldsymbol{w},b}{arg\,max}\ \frac{1}{\parallel \boldsymbol{w} \parallel}$$

subject to the constraints

$$y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\geq 1 \qquad (n=1,\ldots,N).$$

SVM Learning Objective

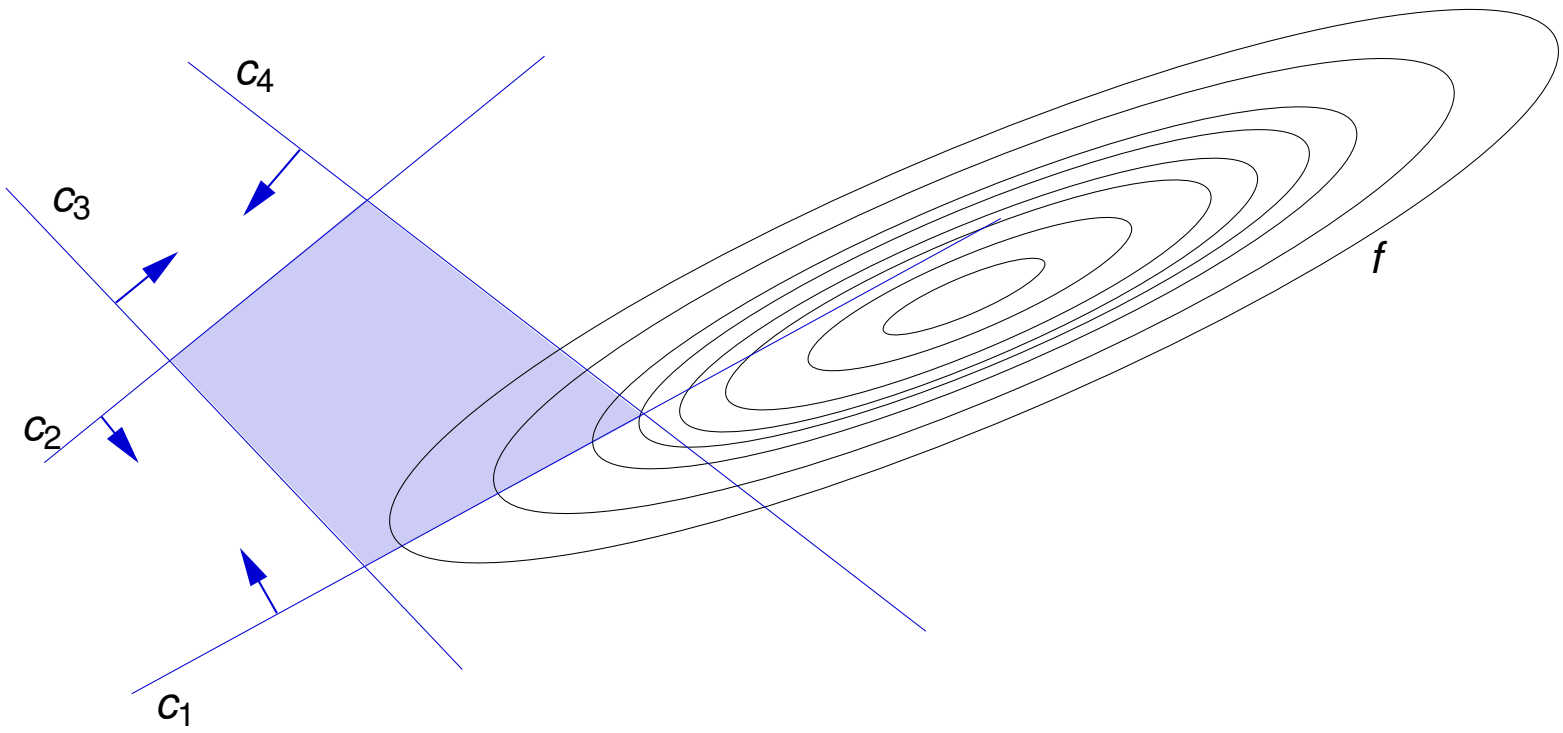
Replacing maximization of $1/\parallel \boldsymbol{w} \parallel$ by equivalent minimization of $\parallel \boldsymbol{w} \parallel^2/2$ leads to the formulation: find

$$\underset{\boldsymbol{w},b}{arg\,min}\ \frac{1}{2}\parallel \boldsymbol{w} \parallel^2$$

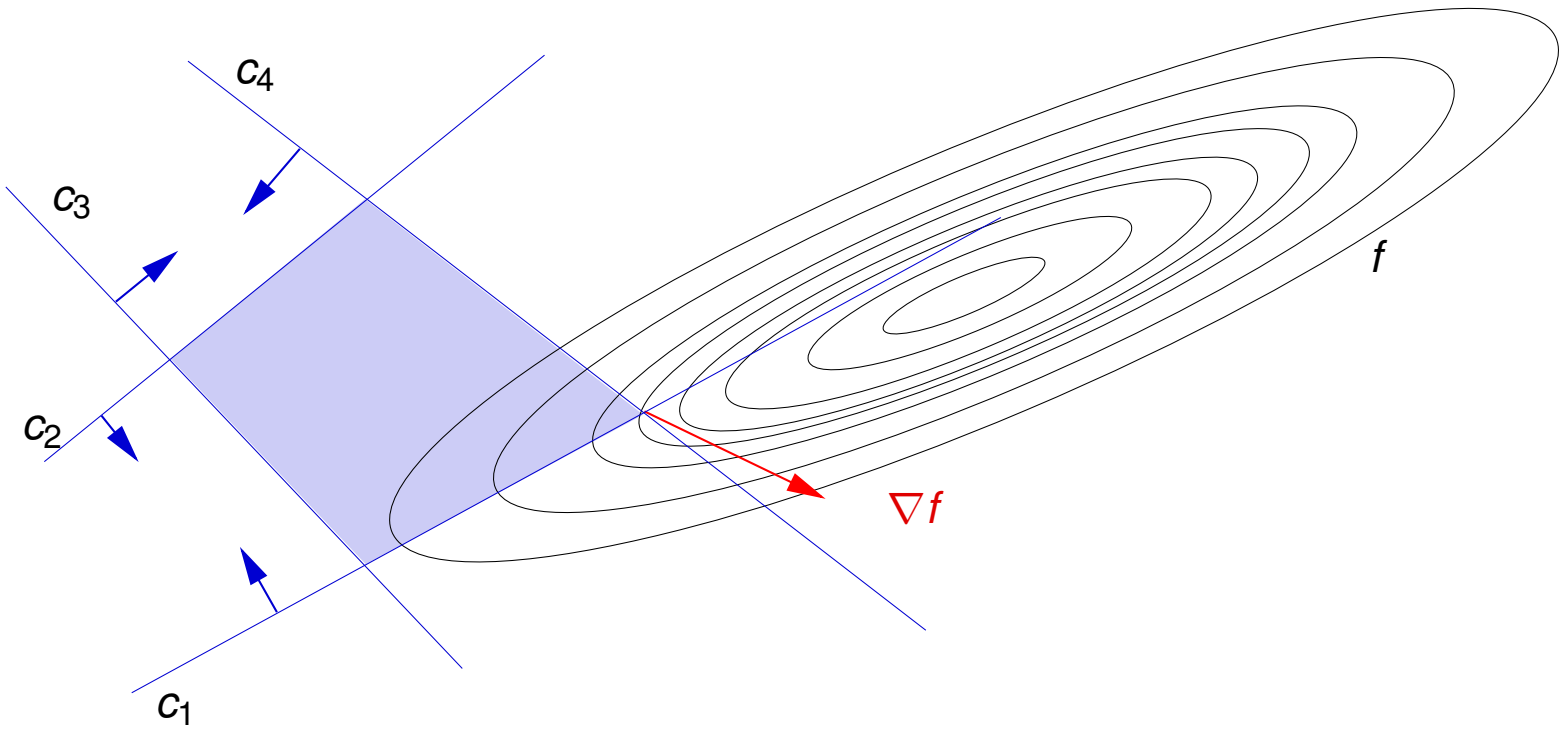
subject to the constraints

$$y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\geq 1 \qquad (n=1,\ldots,N).$$

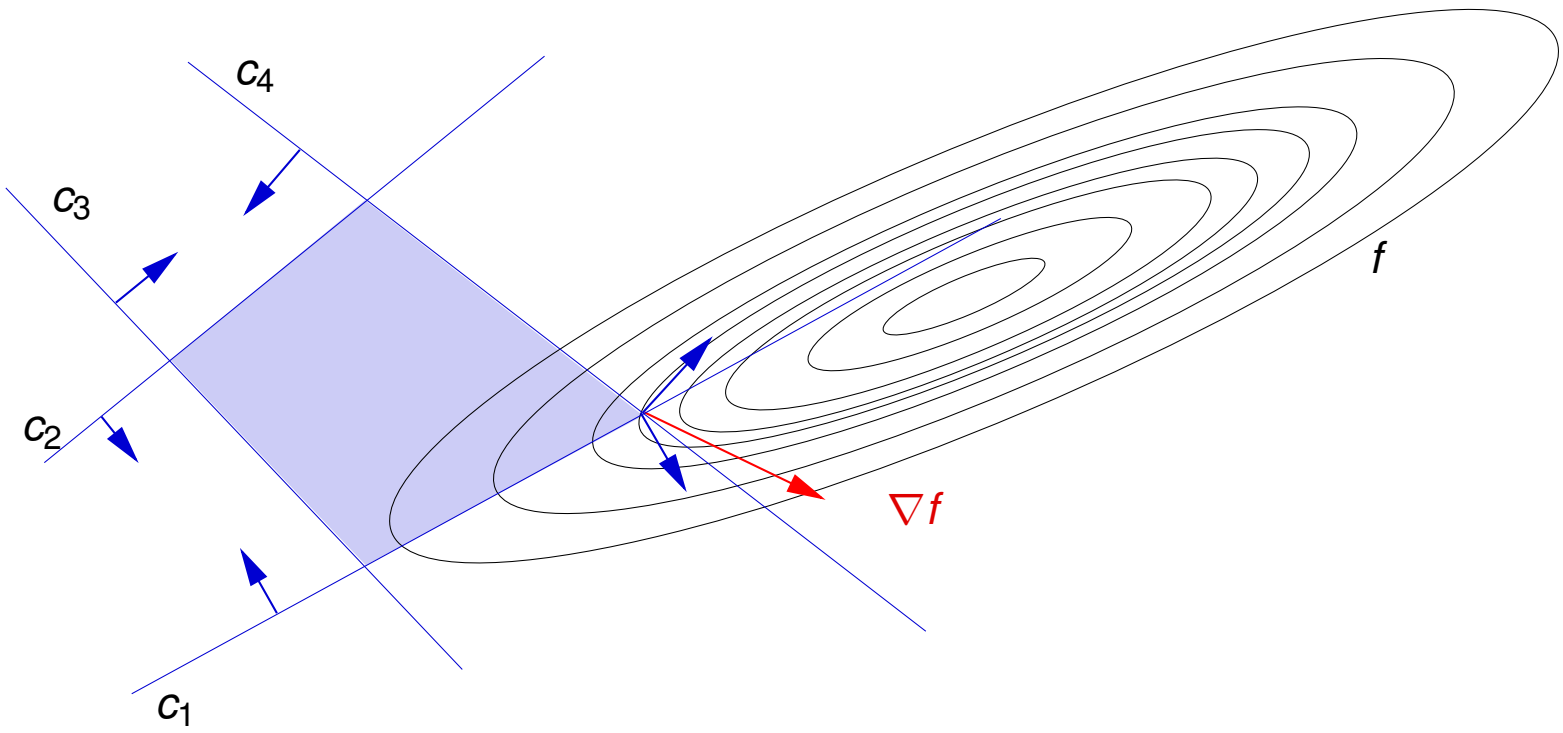
Constrained optimization problem: maximize $f(\boldsymbol{w})$ subject to *constraints* $c_i(\boldsymbol{w}) \geq 0$:



Constrained optimization problem: maximize $f(\boldsymbol{w})$ subject to *constraints* $c_i(\boldsymbol{w}) \geq 0$:



Constrained optimization problem: maximize $f(\boldsymbol{w})$ subject to *constraints* $c_i(\boldsymbol{w}) \geq 0$:



At the constrained optimum, we have the following relationship between the gradients of f and the gradients of the constraints:

$$\nabla f = \sum -\lambda_i \nabla c_i \quad \text{with} \quad \lambda_i \geq 0$$

$$\nabla f = \sum -\lambda_i \nabla c_i \;\Leftrightarrow\; \nabla(f + \sum \lambda_i c_i) = 0$$

where $f + \sum \lambda_i c_i$ is the **Lagrange function** of the problem, and the λ_i are the **Lagrange multipliers**.

- The technique of Lagrange multipliers transforms the constrained optimization problem for $\boldsymbol{w}$ into an unconstrained optimization problem for $\boldsymbol{\lambda}$ and $\boldsymbol{w}$.
- The solution for $\boldsymbol{\lambda}$ identifies the constraints c_i that are *active* at the solution: those with $\lambda_i > 0$.

The SVM learning problem is solved using the method of Lagrange multipliers.

- the solution identifies the **support vectors**: $\boldsymbol{x}_i$ that lie on the margin (= those $\boldsymbol{x}_i$ whose constraint is active at the optimal solution)
- the vector $\boldsymbol{w}$ is a linear combination of support vectors:

$$\boldsymbol{w} = \sum_{i:\lambda_i>0} \lambda_i y_i \boldsymbol{x}_i$$

- a test instance $\boldsymbol{z}$ is classified as

$$sign(\boldsymbol{w} \cdot \boldsymbol{z} + b) = sign(\sum_{i:\lambda_i>0} \lambda_i y_i \boldsymbol{x}_i \cdot \boldsymbol{z} + b)$$

- In the Lagrange optimization process: to determine the support vectors $\boldsymbol{x}_i$, the λ_i and b: the only operations required on data items is to compute dot products $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$.
- for classification: only need to compute dot products $\boldsymbol{x}_i \cdot \boldsymbol{z}$

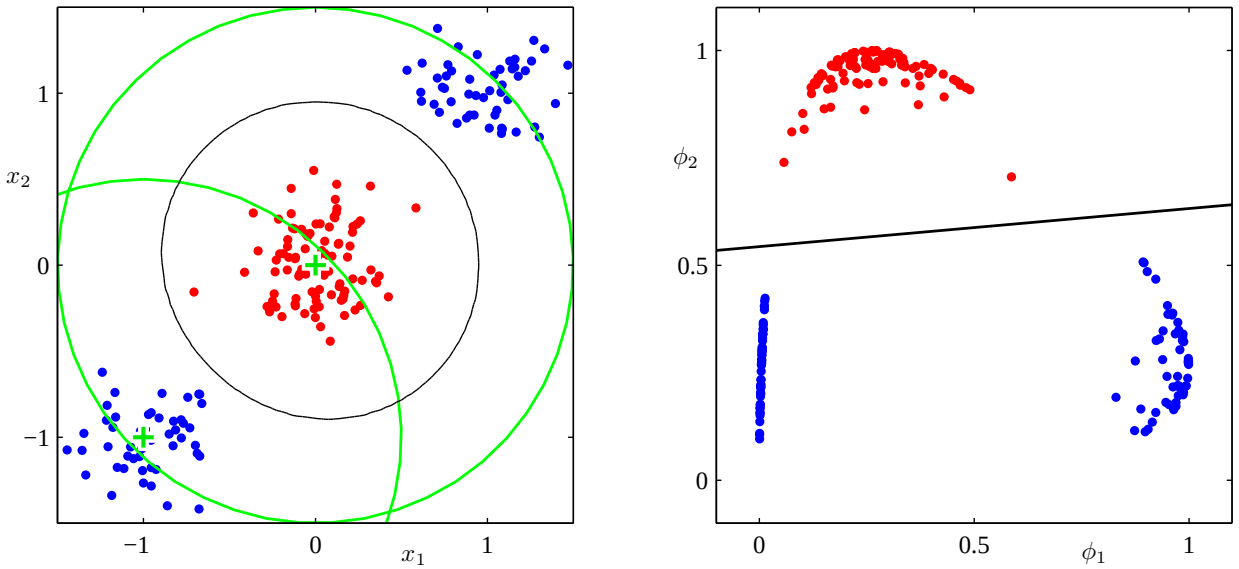
Data Transformations

Any mapping

$$\phi : \mathbb{R}^D \to \mathbb{R}^{D'}$$

defines a data transformation that transforms the original data instance $\boldsymbol{x}$ into a transformed data instance $\phi(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \ldots, \phi_{D'}(\boldsymbol{x}))$.

- The components $\phi_i(\boldsymbol{x})$ are also called **features** or **basis functions**, and $\mathbb{R}^{D'}$ the **feature space** of ϕ.
- Often: $D' > D$, and ϕ_i typically non-linear.



[Bishop Fig. 4.12]

Example: Original data (left), transformed data in feature space (right), decision boundaries (black lines). Here: $\phi_1(x_1, x_2) = (x_1 + 1)^2 + (x_2 + 1)^2$ (?) and $\phi_2(x_1, x_2) = 1 - (x_1^2 + x_2^2)$ (?)

- All our linear models can be applied to the transformed data
- The linear decision boundary in feature space then corresponds to a non-linear decision boundary in the original space