

Machine Learning

Winter 2011/12

Roland Memisevic

Lecture 3, Oct. 31, 2011

Linear classification

$$\mathbf{x} \rightarrow t$$

- ▶ Outputs, t , are discrete and can take on K possible values $\mathcal{C}_1, \dots, \mathcal{C}_K$.
- ▶ Task: *Learn to predict t from \mathbf{x} .*
- ▶ Like linear regression, this is a *supervised learning* problem.
- ▶ Like in linear regression, linearity makes the task simple.
- ▶ But “linear” obviously has to mean something different here than in the case of regression.

Outline

- ▶ Linear classification
- ▶ Discriminative models and logistic regression
- ▶ Generative models and Naive Bayes

Discriminative vs. generative

$$\mathbf{x} \rightarrow t$$

- ▶ There are two fundamentally different ways to approach classification tasks.
- ▶ **Discriminative methods** try to directly learn the mapping.
- ▶ **Generative methods** first learn a (probabilistic) model for each class \mathcal{C}_k , or equivalently a conditional distribution $p(\mathbf{x}|t)$ (note K is finite!), and then “invert” the model to get the mapping.

Discriminant functions

- ▶ In the case of $K = 2$ classes, a linear classifier can be defined as follows:

Linear classifier (two classes)

- ▶ Define the linear function

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

with parameters \mathbf{w}, w_0

- ▶ Assign \mathbf{x} to class \mathcal{C}_1 if $y(\mathbf{x}) \geq 0$, to class \mathcal{C}_2 otherwise.
- ▶ The function $y(\mathbf{x})$ is known as **discriminant function**.
- ▶ Learning a classifier can be achieved by adapting the parameters \mathbf{w} as we will soon see.

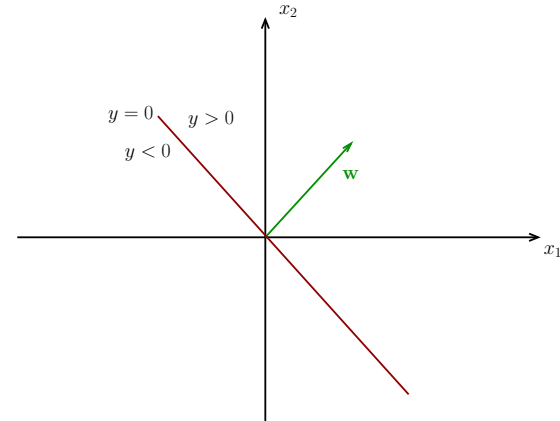
Two classes

- ▶ $w_0 \neq 0$ will move this boundary away from the origin.
- ▶ We can use the bias trick that we used for linear regression to eliminate w_0 from our model:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} \rightarrow \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix}$$

- ▶ Parameter w_0 , which multiplies the 1 in the input, can be thought of as a negative threshold.

Geometric interpretation



- ▶ The classification rule leads to a **linear boundary** separating the classes. In $D > 2$ dimensions, this will be a $(D - 1)$ -dimensional hyperplane.
- ▶ Here, $w_0 = 0$

Extension to more than two classes

- ▶ We can define a multi-class classifier for $K > 2$ classes as follows:

Multi-class linear classifier

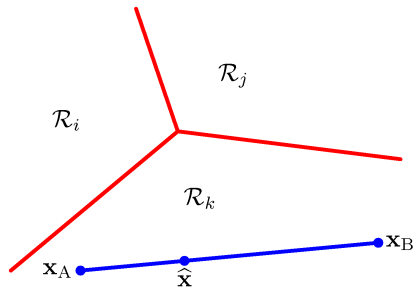
- ▶ Define K discriminant functions

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

with parameters \mathbf{w}_k, w_{k0}

- ▶ Assign \mathbf{x} to class \mathcal{C}_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$
- ▶ This amounts to choosing the maximum over all k discriminants.
- ▶ We can define a 2-class model in this way, too.

Extension to more than two classes



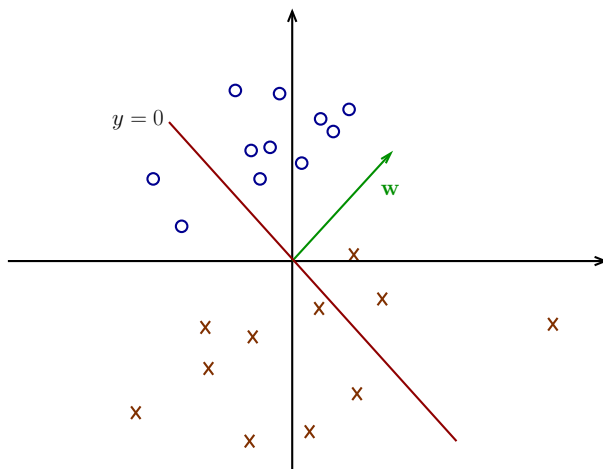
- ▶ A linear multi-class classifier partitions the input-space into K convex regions:
- ▶ Any convex combination of two points in the same class will also be in that class (see Bishop, page 183)

Learning linear classifiers

- ▶ We defined how to assign points to classes, given a model.
- ▶ The real question is, how to *learn* the parameters based on training data $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}$, such that we will make as few mis-classifications as possible on test data.
- ▶ It may, or may not, be possible to learn a linear classifier that makes no mistakes at all on a given training set.
- ▶ Data-sets for which this is possible are called **linearly separable**.

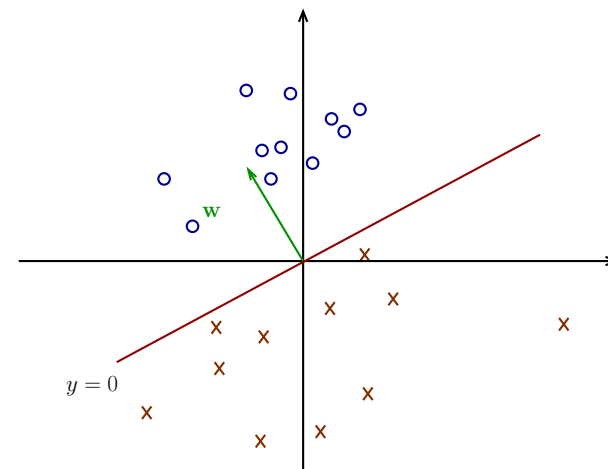
Learning linear classifiers

- ▶ Example with linearly separable data (before learning).



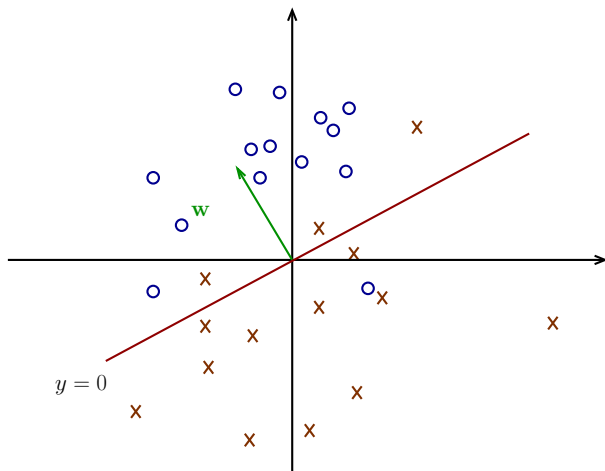
Learning linear classifiers

- ▶ Example with linearly separable data (after learning).



Learning linear classifiers

- ▶ Example with data that is not linearly separable.



Remark: How to represent discrete values

- ▶ There is a particularly useful way to represent one out of a set of K values:
- ▶ As a K -vector with $(K - 1)$ 0's, and one 1 at position k :

$$\mathbf{t} = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

- ▶ This is known as **one-of- K encoding**, **one-hot-encoding**, or as **orthogonal encoding**.
- ▶ Note that we can interpret \mathbf{t} as a probability distribution, because $t_k > 0 \forall k$ and $\sum_k t_k = 1$.
- ▶ We can now represent discrete class-labels row-wise in a matrix \mathbf{T} , like we did before for continuous vectors.

Learning linear classifiers

- ▶ In the following, we shall stack the parameter vectors \mathbf{w}_k column-wise in a matrix \mathbf{W} .
- ▶ Note that, for a given input \mathbf{x} ,

$$\mathbf{y} = \mathbf{W}^T \mathbf{x}$$

yields a vector of linear “scores” $\mathbf{w}_k^T \mathbf{x}$ for each class.

- ▶ A good learning criterion will adjust \mathbf{W} , such that on training cases with $t_n = \mathcal{C}_k$, the model will give large values $\mathbf{w}_k^T \mathbf{x}_n$ and small values $\mathbf{w}_j^T \mathbf{x}_n \forall j \neq k$.
- ▶ Note that least squares regression is *not* a good approach to training, because it would penalize large values even for the right class!

(Multi-class) logistic regression

- ▶ Logistic regression is probably the oldest and one of the most heavily used classifiers.
- ▶ Logistic regression defines a probabilistic model over classes, given inputs:

Multi-class logistic regression (“softmax regression”)

$$p(\mathcal{C}_k | \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)}$$

- ▶ Why this way?
Because the \exp -function ensures positivity, and the normalization that the outputs sum to one.
- ▶ Note that this amounts to “squashing” the components of $\mathbf{y} = \mathbf{W}^T \mathbf{x}$ to lie in the range between 0 and 1.

Multi-class logistic regression

- ▶ Since it is a probabilistic model, logistic regression allows us to use *maximum likelihood* for training.
- ▶ Like in the case of linear regression before, consider the negative log-likelihood cost:

$$\begin{aligned} E(\mathbf{W}; \mathcal{D}) &= -\log \prod_n p(\mathbf{t}_n | \mathbf{x}_n) \\ &= -\log \prod_n \prod_k p(C_k | \mathbf{x}_n)^{t_{nk}} \\ &= -\sum_n \sum_k t_{nk} \log p(C_k | \mathbf{x}_n) \\ &= -\sum_{nk} t_{nk} (\mathbf{w}_k^T \mathbf{x}_n - \log \sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)) \end{aligned}$$

The “logsumexp”-trick

- ▶ Expressions like

$$\frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x}_n)}$$

are *highly unstable* in practice, because the “exp” in the denominator can cause an under- or overflow.

- ▶ Never compute sums $\sum_i \exp(a_i)$ naively.
- ▶ Add a constant A to each argument in all exp’s, so that even the largest argument is small; then undo the operation after computing the sum!
- ▶ Many software packages supply a convenience function “logsumexp” for this purpose:

logsumexp

$$\text{logsumexp}(a_1, \dots, a_K) = \log \left(\sum_i \exp(a_i + A) \right) - A$$

with $A = -(\max_i a_i)$

Multi-class logistic regression

- ▶ In contrast to linear regression, there is no closed-form solution for \mathbf{W} .
- ▶ But we can use gradient-based optimization to minimize $E(\mathbf{W}; \mathcal{D})$ iteratively.
- ▶ The gradient with respect to each parameter-vector \mathbf{w}_k is

$$\begin{aligned} \frac{\partial E(\mathbf{W}; \mathcal{D})}{\partial \mathbf{w}_k} &= -\sum_n t_{nk} \mathbf{x}_n - \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x}_n)} \mathbf{x}_n \\ &= \sum_n (p(C_k | \mathbf{x}_n) - t_{nk}) \mathbf{x}_n \end{aligned}$$

- ▶ It can be shown that $E(\mathbf{W}; \mathcal{D})$ is convex, so there are no local minima!

Two-class Logistic regression

- ▶ The term “logistic regression” has been used traditionally to refer to the case of exactly two classes.
- ▶ Traditionally, the two-class case has been treated slightly differently, using one, rather than two, parameter vectors.
- ▶ Note that we can write

$$\begin{aligned} p(C_1 | \mathbf{x}) &= \frac{\exp(\mathbf{w}_1^T \mathbf{x})}{\exp(\mathbf{w}_1^T \mathbf{x}) + \exp(\mathbf{w}_2^T \mathbf{x})} \\ &= \frac{1}{1 + \exp(-(\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x})} \end{aligned}$$

- ▶ And likewise, for class 2, we can write

$$p(C_2 | \mathbf{x}) = \frac{1}{1 + \exp((\mathbf{w}_1 - \mathbf{w}_2)^T \mathbf{x})}$$

Two-class Logistic regression

- ▶ Thus, it is sufficient to use just *one* parameter vector $\mathbf{w} := (\mathbf{w}_1 - \mathbf{w}_2)$:

Two-class logistic regression

$$p(\mathcal{C}_1|\mathbf{x}) = 1 - p(\mathcal{C}_2|\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}$$

- ▶ This amounts to removing the “over-parameterization” inherent in the original softmax-regression definition, and it could in principle be done for the multi-class model, not just the 2-class model. But this is not common in practice.

The sigmoid function

- ▶ The function

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

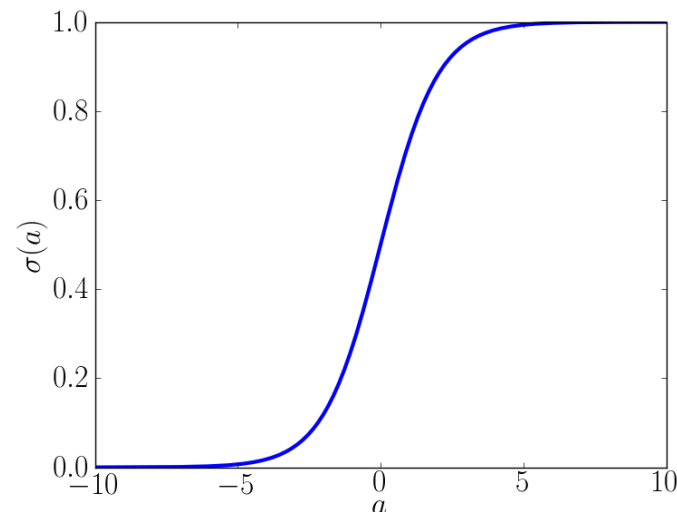
is an “S”-shaped 1-dimensional function, which known as “**sigmoid**” or as “logistic function”. (Hence the term “logistic regression”.)

- ▶ Its derivative can be written in a slightly unusual way, using the function itself, as:

$$\frac{d\sigma(a)}{da} = \sigma(1 - \sigma)$$

- ▶ Its inverse is given by $a = \log\left(\frac{\sigma}{1-\sigma}\right)$, which is known as the “logit-function” or as “log odds”.

The sigmoid function



Basis expansion and regularization

Basis expansions

- ▶ Like for linear regression, we can use a non-linear **basis-expansion** $\mathbf{x} \rightarrow \Phi(\mathbf{x})$ and fit a non-linear model using logistic regression.
- ▶ This will, in general, amount to fitting non-linear class boundaries.

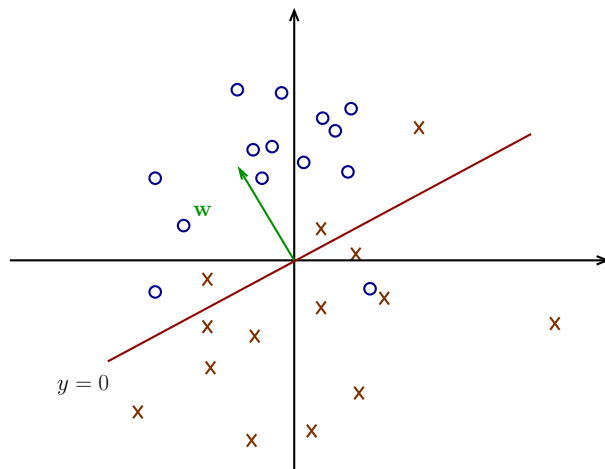
Regularization

- ▶ Like for linear regression, one should **regularize** the model by adding a weight penalty such as

$$E_{\mathbf{W}}(\mathbf{W}, \lambda) = \frac{\lambda}{2} \|\mathbf{W}\|^2$$

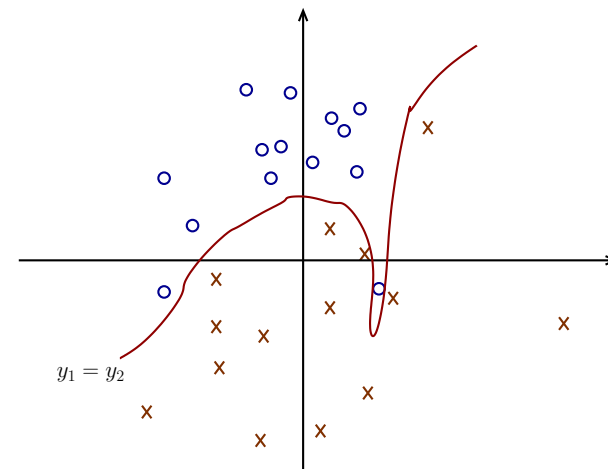
to the objective function to avoid overfitting.

Basis expansions and overfitting



- ▶ A linear model.

Basis expansions and overfitting



- ▶ A non-linear model.

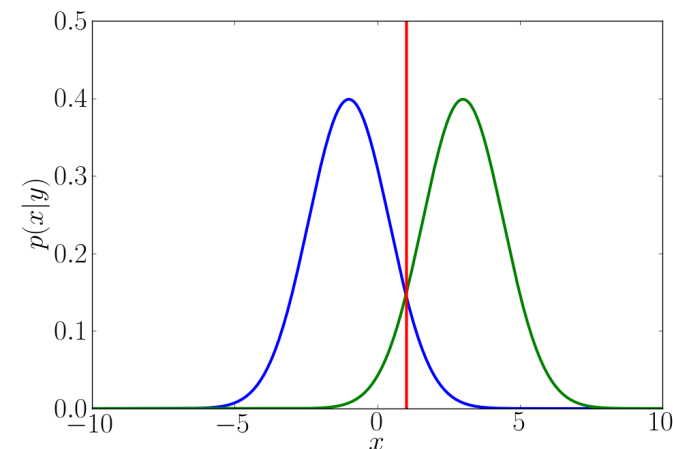
Generative methods

- ▶ An entirely different approach to classification is given by **generative methods**, which are based on learning, and then inverting, a model for each class-conditional distribution over inputs. This can be achieved by using Bayes' rule:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)}$$

- ▶ Under this rule, the winning class is class \mathcal{C}_k for which $p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k) > p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j) \forall j$
- ▶ Since they involve Bayes' rule, generative classifiers are sometimes called "Bayes classifier" or "Bayesian classifier". (This is a slight misnomer, as they are not really Bayesian models which integrate over parameters.)

Generative model example in 1-d



- ▶ Class boundaries between classes \mathcal{C}_1 and \mathcal{C}_2 are defined by $p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) = p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)$

Generative models

- ▶ A generative classifier requires two specifications:
 1. The prior probabilities $p(\mathcal{C}_k)$ over classes. In practice, these are usually set to $\frac{\sum_n t_{nk}}{N}$.
 2. The class-conditional distributions over inputs $p(\mathbf{x}|\mathcal{C}_k)$. There are a lot of possible choices. We shall discuss the most common ones in the following.
- ▶ Note that both, logistic regression, and generative models, define a classifier by modeling the conditional probability $p(\mathcal{C}_k|\mathbf{x})$. They differ only in *how* they define this probability.

Generative models with continuous inputs

- ▶ The maximum likelihood estimates are

$$\boldsymbol{\mu}_k = \frac{1}{\sum_n t_{nk}} \sum_{n=1}^N t_{nk} \mathbf{x}_n$$

and

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n t_{nk}} \sum_{n=1}^N t_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

- ▶ It is common, in practice, to **share** parameters between classes. In particular, if we let all classes have the same covariance-matrix $\boldsymbol{\Sigma}$, we can set:

$$\boldsymbol{\Sigma} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T$$

where $\boldsymbol{\mu}$ is the overall mean.

Generative models with continuous inputs

- ▶ For continuous (D -dimensional) inputs \mathbf{x} , the most common and simplest class-conditional model is the multivariate Gaussian:

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\boldsymbol{\Sigma}_k|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right)$$

- ▶ In the most general case, each class \mathcal{C}_k gets its own mean $\boldsymbol{\mu}_k$ and own covariance matrix $\boldsymbol{\Sigma}_k$
- ▶ Fitting a generative classifier then amounts to simply fitting K class-conditional Gaussians.

Class boundaries

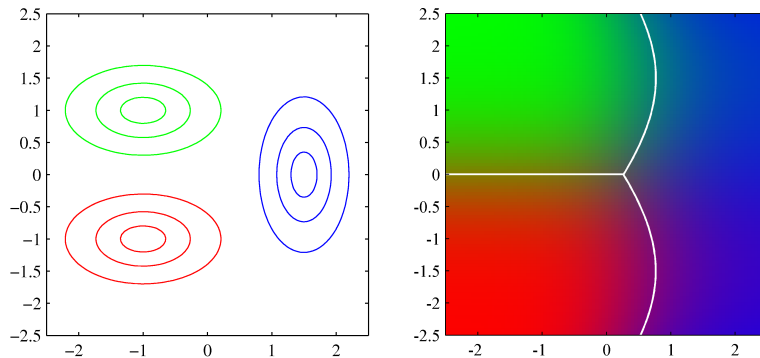
- ▶ We now look at the class boundaries between two classes $\mathcal{C}_1, \mathcal{C}_2$ in the case of a shared covariance matrix.
- ▶ For this end, set:

$$\begin{aligned} \log p(\mathbf{x}|\mathcal{C}_1) + \log p(\mathcal{C}_1) &= \log p(\mathbf{x}|\mathcal{C}_2) + \log p(\mathcal{C}_2) \\ \Leftrightarrow (\mathbf{x} - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_1) - \log p(\mathcal{C}_1) &= (\mathbf{x} - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_2) - \log p(\mathcal{C}_2) \\ \Leftrightarrow \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 - 2\boldsymbol{\mu}_1^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \log p(\mathcal{C}_1) &= \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 - 2\boldsymbol{\mu}_2^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \log p(\mathcal{C}_2) \\ \Leftrightarrow (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} \mathbf{x} &= \text{const} \end{aligned}$$

- ▶ This is a linear condition on the inputs \mathbf{x} . Thus when using class-conditional Gaussians with shared covariance matrix, we obtain linear class boundaries!
- ▶ Otherwise they will be quadratic.

Class boundaries

- ▶ Example with three classes, where two share a covariance matrix:



Class-conditional Gaussians and logistic regression

- ▶ Consider the conditional class-probabilities $p(\mathcal{C}_1|\mathbf{x}), p(\mathcal{C}_2|\mathbf{x})$ under the two-class model:
- ▶ We have:

$$\begin{aligned} p(\mathcal{C}_1|\mathbf{x}) &= \frac{p(\mathcal{C}_1|\mathbf{x})p(\mathcal{C}_1)}{p(\mathcal{C}_1|\mathbf{x})p(\mathcal{C}_1) + p(\mathcal{C}_2|\mathbf{x})p(\mathcal{C}_2)} \\ &= \frac{1}{1 + \exp(-a)} \end{aligned}$$

with $a = \log \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1} \mathbf{x} := \mathbf{w}^T \mathbf{x}$

- ▶ Thus, a Gaussian generative classifier with shared covariance matrix has the same functional form as logistic regression!

Discrete inputs and Naive Bayes

- ▶ **Discrete features** are easy to deal with, if we assume inputs to be *independent given classes*.
- ▶ A generative classifier that makes this assumption is known as a “**Naive Bayes** classifier”.
- ▶ Class-conditional independence seems like a severe restriction in practice, but it often works surprisingly well.

Bernoulli Naive Bayes classifier

- ▶ If we assume independent Bernoulli variables in each input dimension, we can write:

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^D \mu_{ki}^{x_i} (1 - \mu_{ki})^{1-x_i}$$

where μ_{ki} is the conditional probability for input i to be equal to 1, given class k .

- ▶ The maximum likelihood estimates for μ_{ki} are simply counts:

$$\mu_{ki} = \frac{\sum_n t_{nk} x_{ni}}{\sum_n t_{nk}}$$

Discrete (“multinomial”) Naive Bayes classifier

- ▶ If we assume independent discrete (“multinomial”) variables with M outcomes in each input dimension, and represent *each single input case* by a $D \times M$ -matrix of “stacked” one-hot encodings, we can write:

$$p(\mathbf{x}|\mathcal{C}_k) = \prod_{i=1}^D \prod_{j=1}^M \mu_{kij}^{x_{ij}}$$

with parameters μ_{kij} that represent the probability that for class k , input dimension i takes on the value j .

Discrete (“multinomial”) Naive Bayes classifier

- ▶ Stack training data in a 3d-“tensor” of dimensions $(N \times D \times M)$.
- ▶ The maximum likelihood estimates for the parameters μ_{kij} are again simply counts:

$$\mu_{kij} = \frac{\sum_n t_{nk} x_{nij}}{\sum_n t_{nk}}$$

- ▶ How do we know? Maximize the log-likelihood, enforce constraints using Lagrange multipliers.