Machine Learning I
Week 3: Linear Classification

Christian Osendorfer, Martin Felder

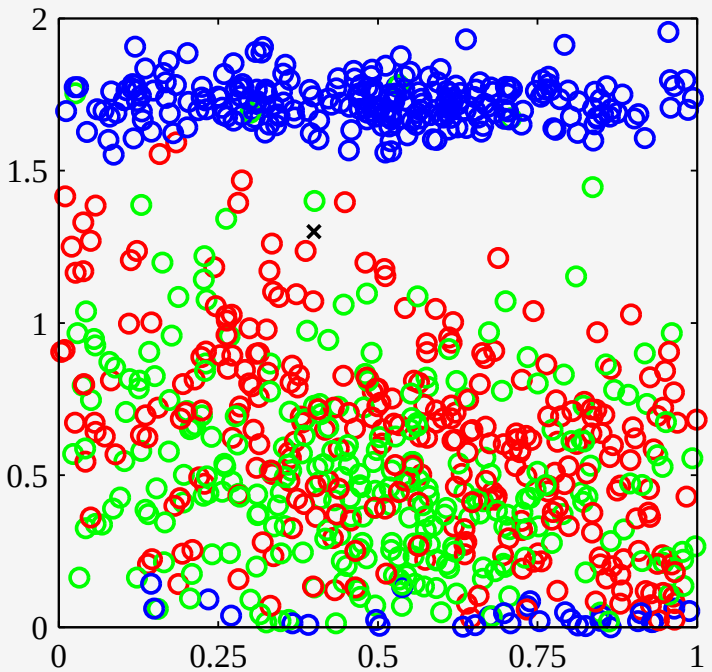
Technische Universität München

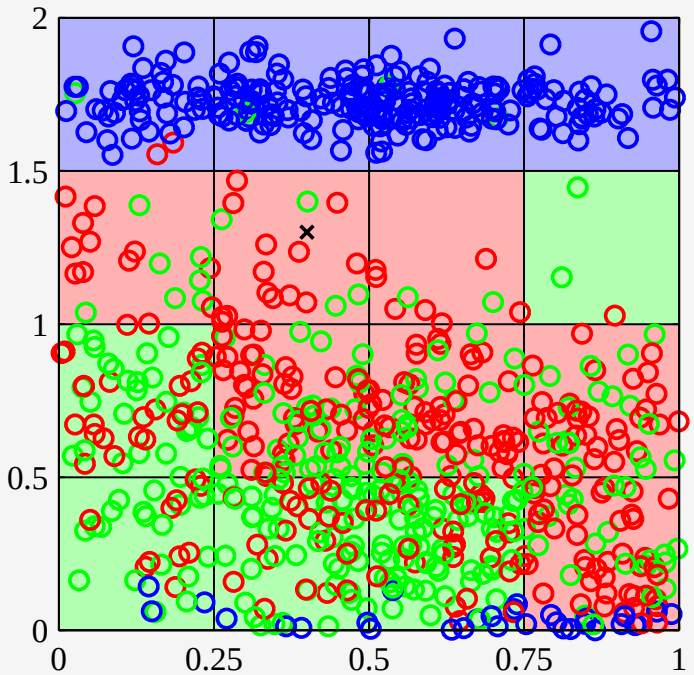
5 November 2009

Classification Problems

Goal: Assign unknown input vector $\mathbf{x}$ to one of K classes, denoted $\mathcal{C}_k$ $k = 1, \ldots, K$.

Example: $K = 3$, $\mathbf{x} \in \mathbb{R}^2$; Which class (=color) should the $\times$ get?

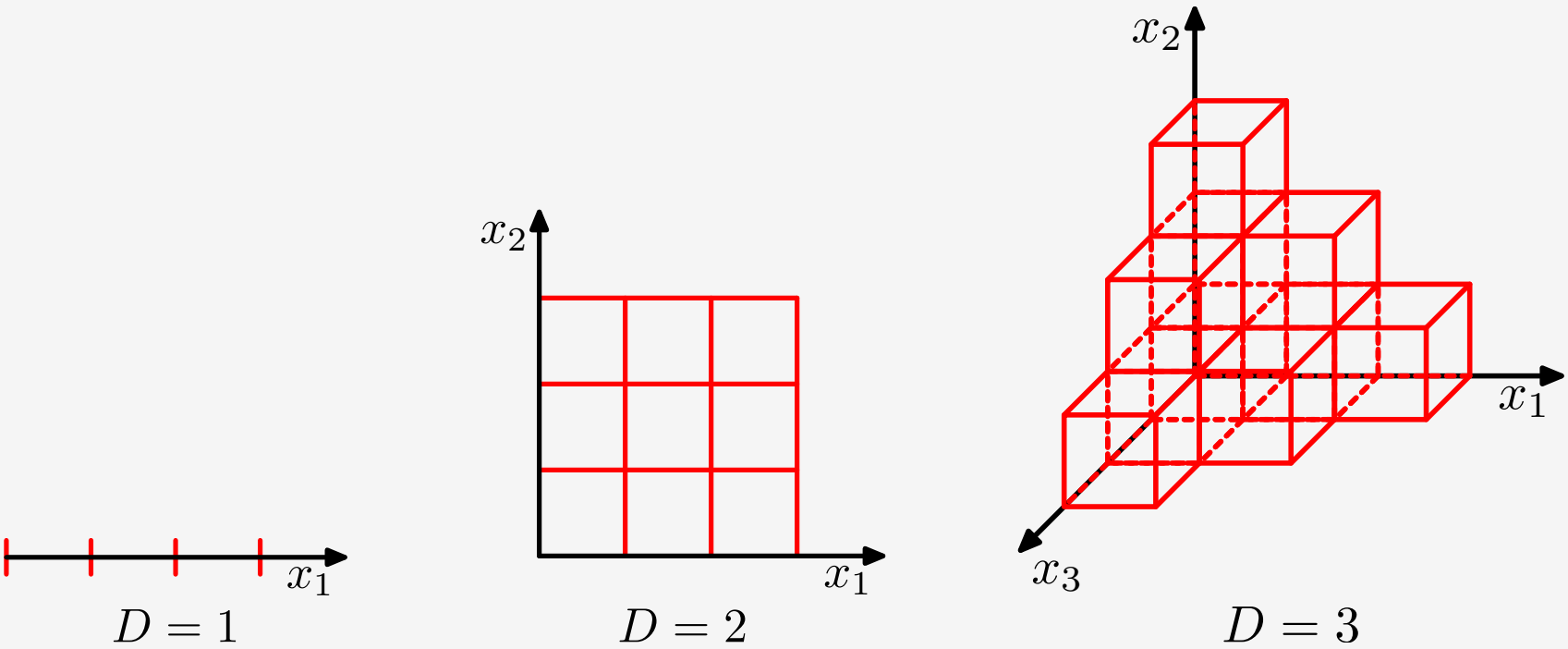




Naive solution: Divide $\mathbb{R}^2$ into boxes, color according to prevalent class.

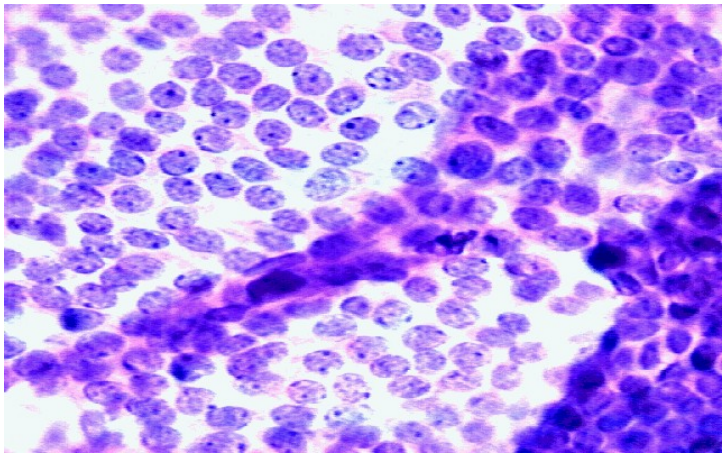
Curse of Dimensionality

Problem: Not scalable!

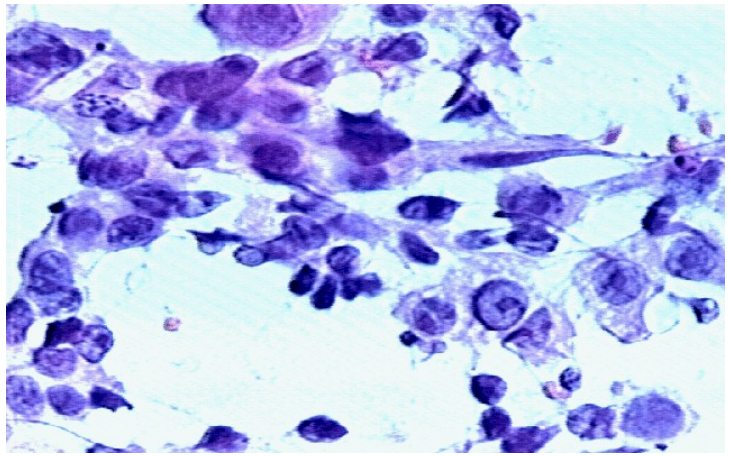


$\Longrightarrow$ For $\mathbf{x} \in \mathbb{R}^D$, scales with D^3.

Real-world Example: Breast cancer detection (UCI)



healthy tissue



cancerous tissue

$\Longrightarrow$ Use computer vision to automatically extract several features of the cell nuclei from the images:

$\mathbf{x} = [$ radius mean, r. variance, r. max, texture mean, t. variance, t. max, smoothness $\ldots$, symmetry $\ldots$, fractal dimension $\ldots, \ldots]^T$

$\longrightarrow$ 30 dimensions altogether!
(best classifier gets it right almost 100% of the time)

Three Approaches to Classification

In order of increasing complexity (cf. MLE, MAP, full Bayes):

Discriminant function: Find a function $f(\mathbf{x})$ that maps input directly to predicted class.

Discriminative model: Break the problem down into two stages:
Inference: Find the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$.
Decision: Use decision theory to find most likely $\mathcal{C}_k$ for given $\mathbf{x}$

Generative model: First infer class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ for each class $\mathcal{C}_k$. This allows us to "simulate" input data! Also infer $p(\mathcal{C}_k)$, then use Bayes:

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

Finally, use decision theory as above to find $\mathcal{C}_k$ for each $\mathbf{x}$.

Discriminant Functions

Recap: For linear regression, we used the model function

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \tag{1}$$

Goal: Find a discriminant function, i.e. a function that takes an input vector $\mathbf{x}$ and assigns it to class $\mathcal{C}_k$.

Idea: For 2 classes simply do a regression and wrap it into a step function:

$$y(\mathbf{x}) = f(w_0 + \mathbf{w}^\mathsf{T} \phi(\mathbf{x})) \tag{2}$$

where

$$f(a) = \begin{cases} 1 & \text{if } a > 0 \quad \Rightarrow \mathbf{x} \in \mathcal{C}_2 \\ 0 & \text{if } a \leq 0 \quad \Rightarrow \mathbf{x} \in \mathcal{C}_1 \end{cases}$$

Note:

- slight change of notation wrt. regression: w_0 is called bias, $\mathbf{w}$ is the weight vector
- in general, $f(\cdot)$ is called an activation function
- for properly designed $f(\cdot)$, $y(\mathbf{x})$ yields probability of $\mathbf{x} \in \mathcal{C}_2$

Discriminant Functions

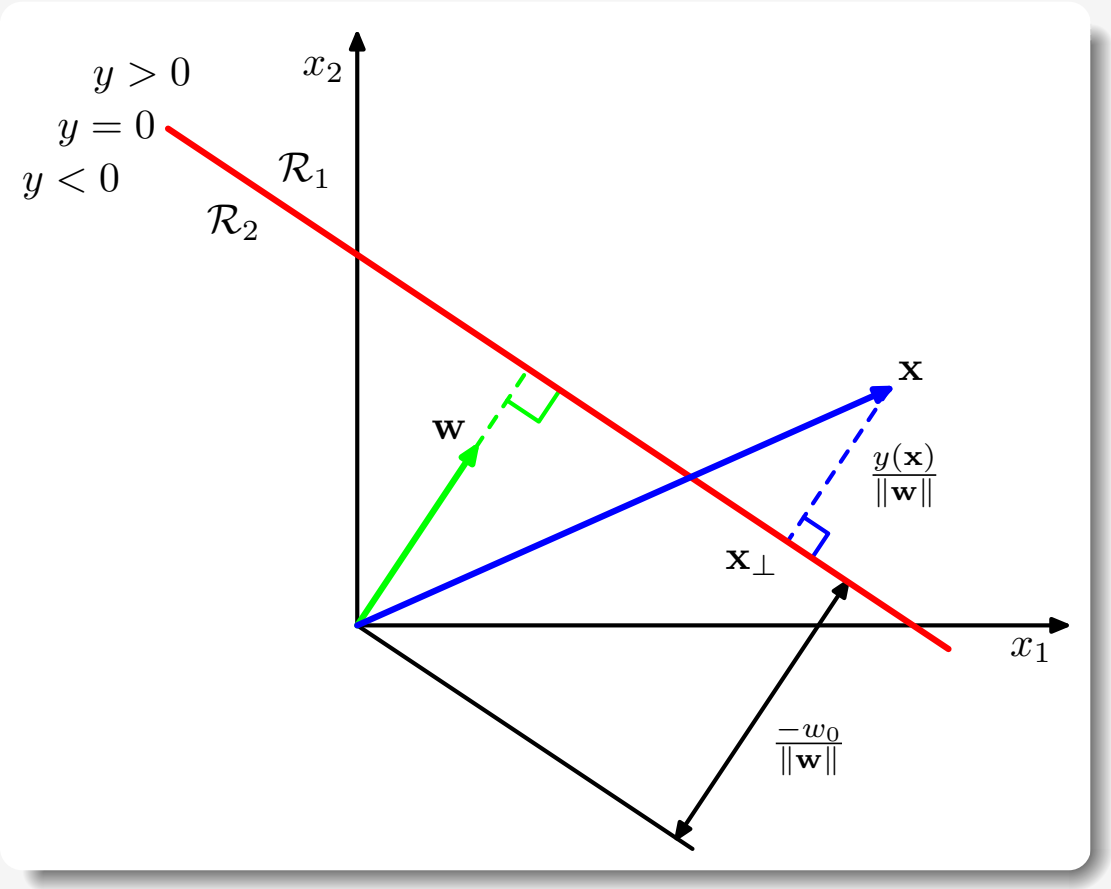
For now, consider linear discriminants, where the decision surfaces are hyperplanes. With only two classes, we get the simple representation:

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

y(**x**) gives the perpendicular signed distance of **x** from the decision surface, in units of $\|\mathbf{w}\|$:

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

$$\Longrightarrow r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}$$



Multiclass Problems

Idea: Simply introduce more decision levels for the activation function:

$$f(a) = \begin{cases} 3 & \text{if } a > 1 & \Rightarrow \mathbf{x} \in \mathcal{C}_4 \\ 2 & \text{if } 0 < a \leq 1 & \Rightarrow \mathbf{x} \in \mathcal{C}_3 \\ 1 & \text{if } -1 < a \leq 0 & \Rightarrow \mathbf{x} \in \mathcal{C}_2 \\ 0 & \text{if } a \leq -1 & \Rightarrow \mathbf{x} \in \mathcal{C}_1 \end{cases}$$

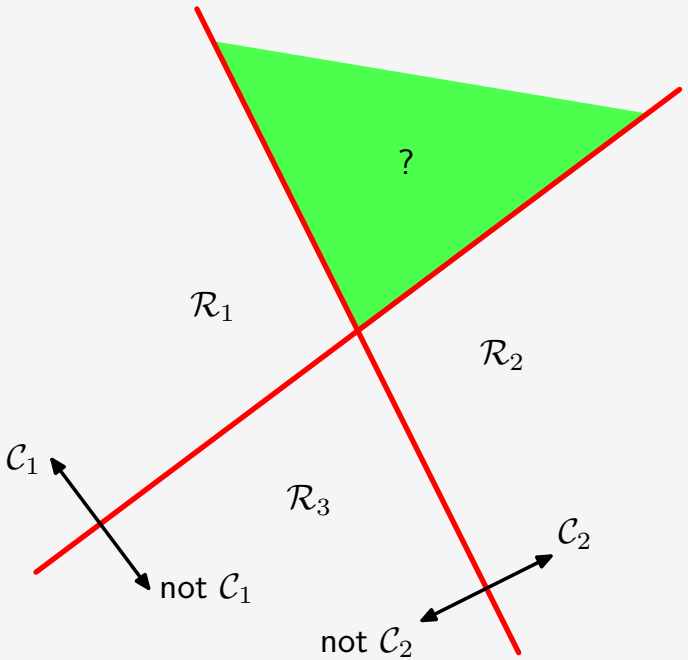
$\Longrightarrow$ Bad solution: Implies distance metric between classes:
$\mathcal{C}_1$ "neighbours" $\mathcal{C}_2$, but is "far away" from $\mathcal{C}_4$

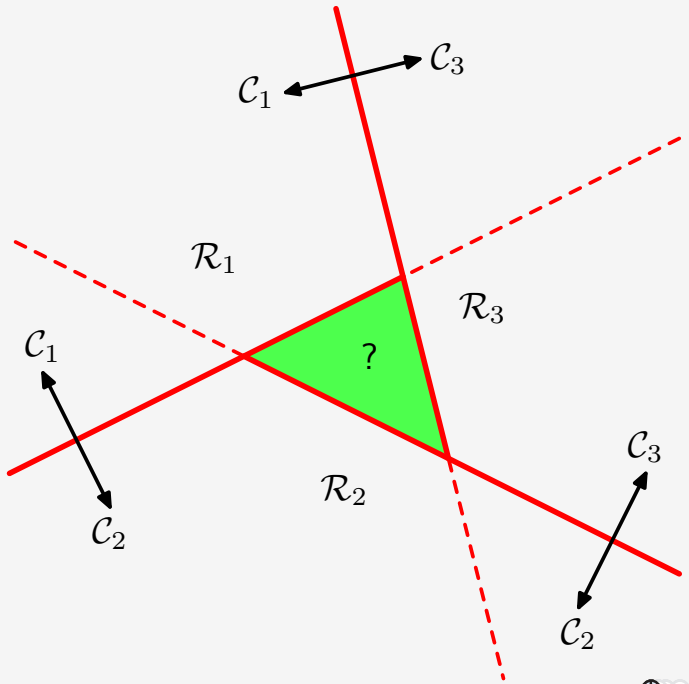
Multiclass Problems

Idea: Reformulate the problem to combine binary classifiers:

- 1-vs-rest needs $K - 1$ classifiers each "specialized" in picking out one class (last class is remainder)
- 1-vs-1 needs $K(K-1)/2$ classifiers each trained on a pair of classes. Use "voting" to pick predicted class.

$\Longrightarrow$ Suboptimal solution: Not all cases may be covered.





Multiclass Problems

Idea: For K classes use a 1-of-K coding scheme. E.g. if $K = 5$, the target vector for class $\mathcal{C}_2$ is

$$\mathbf{t} = (0, 1, 0, 0, 0)^{\mathsf{T}}$$

$\Longrightarrow$ Usually best solution:

- straightforward linear multi-class DF:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0} \tag{3}$$

- y_k can be interpreted as $p(\mathcal{C}_k|\mathbf{x})$ if DF properly designed
- otherwise just use largest y_k to determine class ("winner takes all")
- $K = 2$ is a special case that can (usually) be shown to be equivalent to the binary target

Least squares for classification

How do we find parameters $\mathbf{w}$?

Recap: For regression, we minimized the quadratic error function

$$E_{\mathcal{D}}(\mathbf{w}) = \frac{1}{2}\sum^{N}\left(t_n - \mathbf{w}^T\mathbf{x}\right)^2,$$

that resulted from log likelihood on Gaussians.

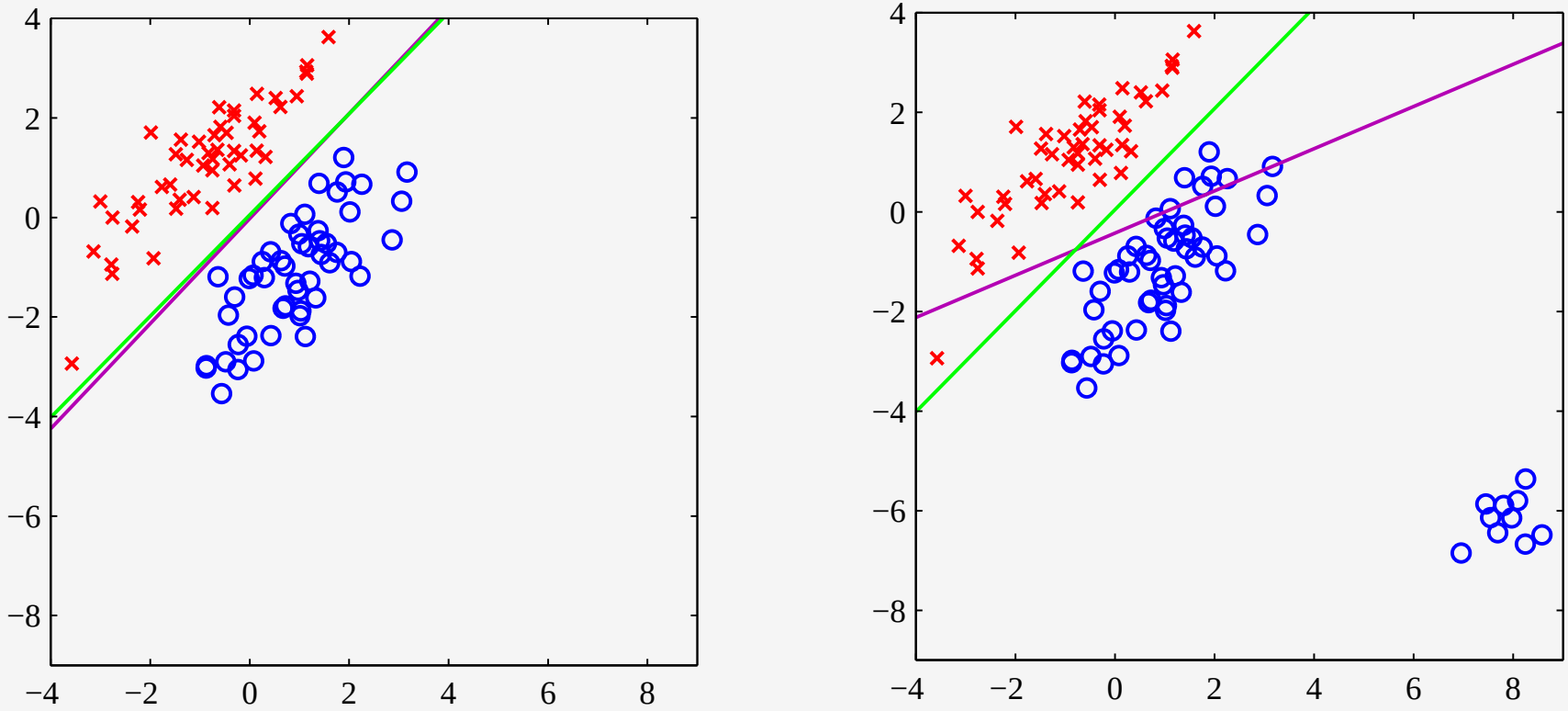
Idea: Good model for linear regression, use it for classification?

$$\longrightarrow \text{Combine} \quad y_k(\mathbf{x}) = \mathbf{w}_k^T\mathbf{x} + w_{k0} \quad \text{into} \quad (4)$$

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T\widetilde{\mathbf{x}} \quad (5)$$

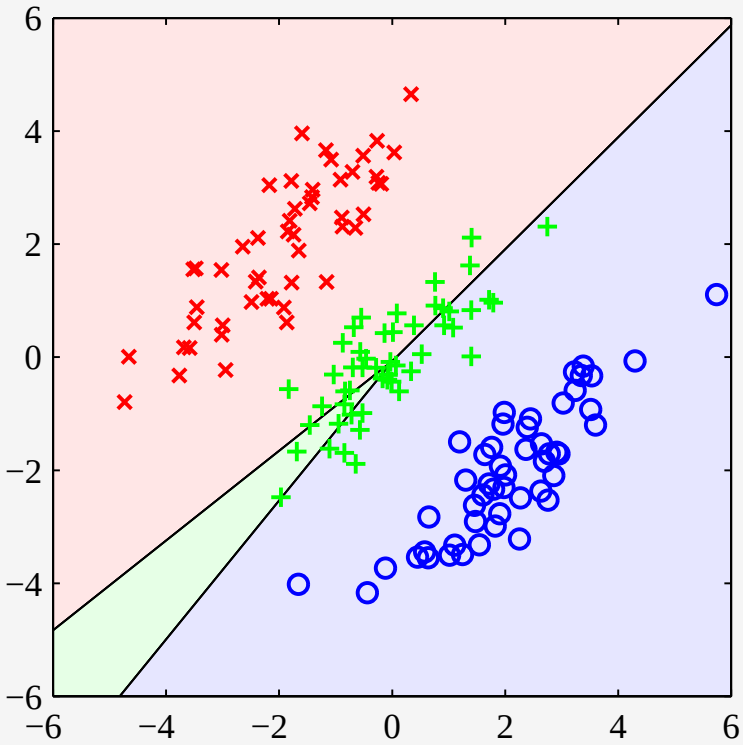
$\Longrightarrow$ Can find exact closed form solution for $\widetilde{\mathbf{W}}$ using pseudo-inverse, as before!

Least squares: Two classes

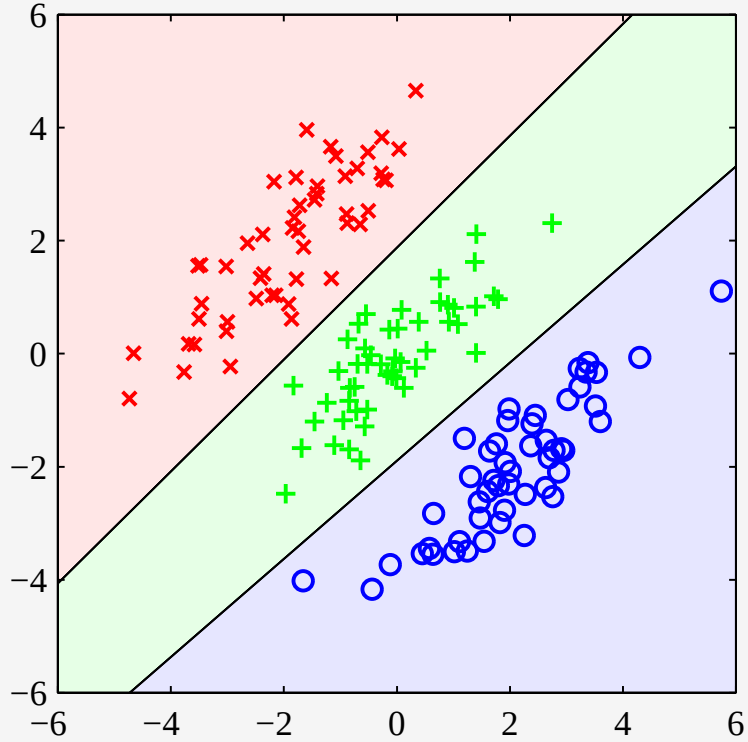


green: decision boundary found by logistic regression (later!)
purple: decision boundary found by least squares

Least squares: Three classes



least squares solution



logistic regression solution

Ok, closed form using least squares is out. Try iterative solution next!

The perceptron algorithm

Two classes: Look at generalized linear model again (Eqn. 2), this time with $\phi_0(\mathbf{x}) \equiv 1$:

$$y(\mathbf{x}) = f(\mathbf{w}^T \phi(\mathbf{x}))$$

Using a nonlinear transformation $\boldsymbol{\phi(\cdot)}$ and a nonlinear activation function $f(\cdot)$.

ToDo:

1. Find suitable error function.
2. Follow gradient $\nabla_{\mathbf{w}}$ iteratively to find minimum.

The perceptron algorithm

Idea: E = total number of misclassified patterns?
$\Longrightarrow$ not useful, this error is piecewise constant, so no gradient learning possible.
Idea: Instead, use perceptron criterion [Rosenblatt, 1962]:
Would like all patterns to satisfy

$$\mathbf{w}^T \phi(\mathbf{x}_n) t_n > 0$$

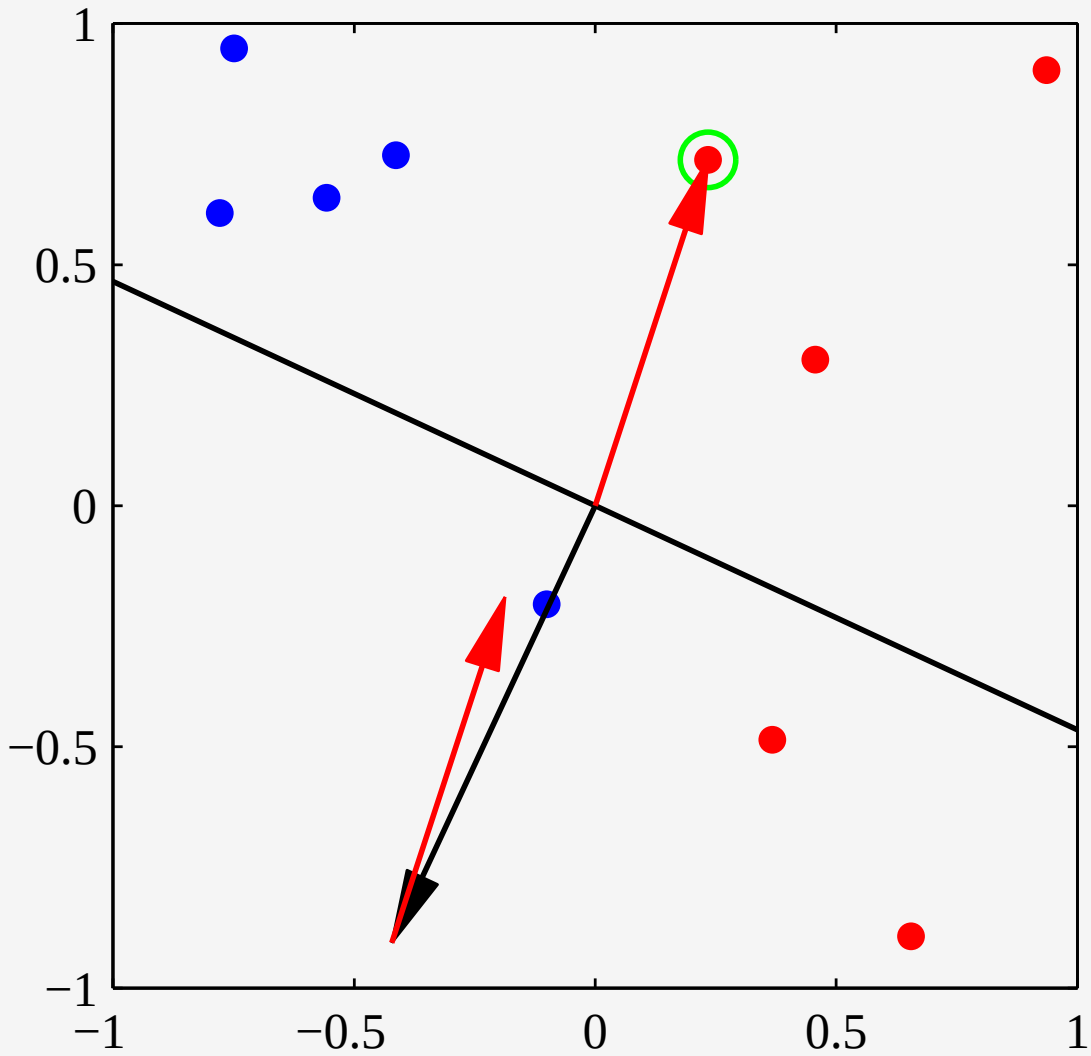
with $t_n \in \{-1, +1\}$ resembling the two possible classes.

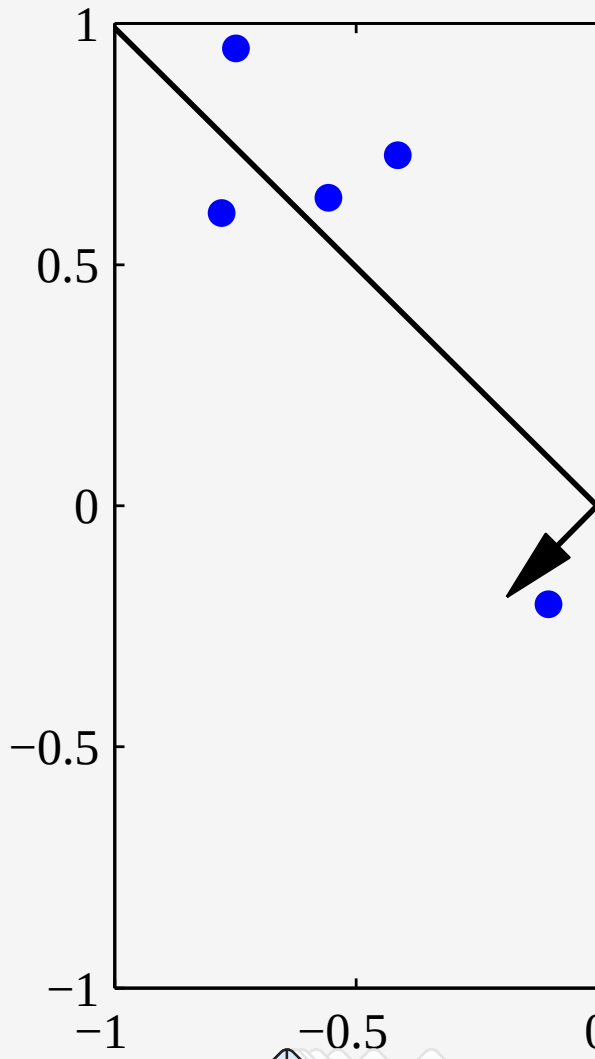
$$E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \phi_n t_n$$

$\mathcal{M}$ denotes the set of misclassified patterns. Applying stochastic gradient descent to this error function we get the perceptron learning rule

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \eta \phi_n t_n$$

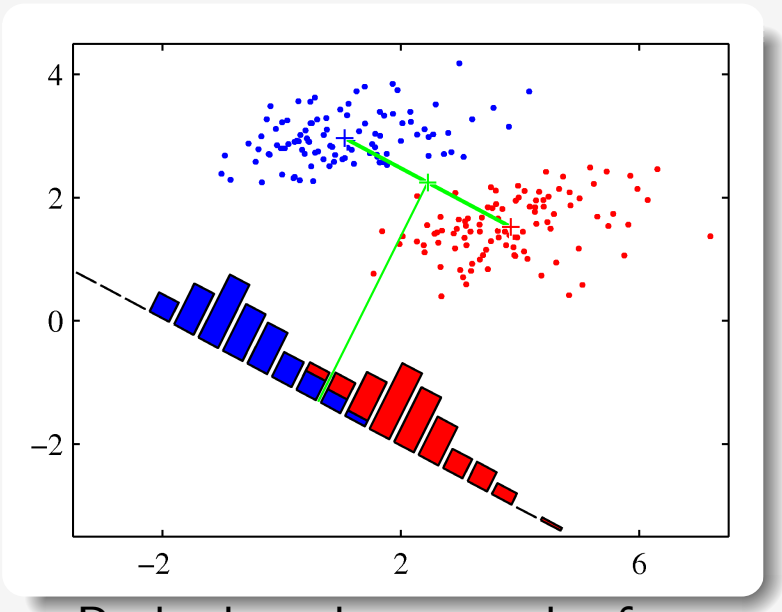
The perceptron algorithm

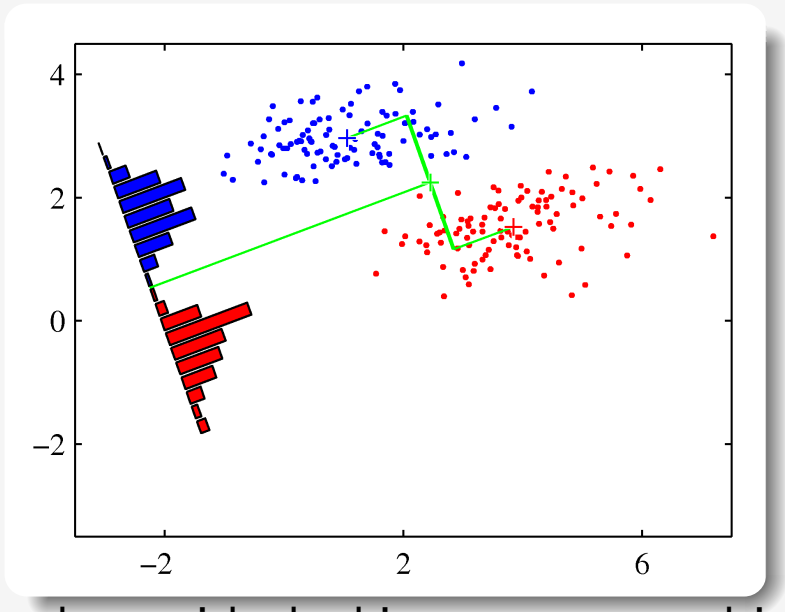




Fisher's linear discriminant

Maximize a function that will give a large separation between the projected class means while also giving small variance within each class and thereby minimizing class overlap.





Both plots show samples from two classes along with the histogramms resulting from projection onto a line. The line in the left plot joins the two class means, resulting in considerable class overlap in the projected space. On the right the corresponding projection based on the Fisher linear discriminant shows the greatly improved class separation.

Probabilistic Generative Models

For a given input $\mathbf{x}$ what we want are the posterior probabilities $p(\mathcal{C}_k|\mathbf{x})$. We get to these through Bayes' theorem by using the class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ and the class priors $p(\mathcal{C}_k)$. So for $K = 2$ classes we have

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

with

$$a(\mathbf{x}) = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

$\sigma(\cdot)$ is called the *logistic sigmoid* function.
It has the following nice (as we will see later on) symmetry property:

$$\sigma(-x) = 1 - \sigma(x)$$

For $K > 2$ classes we get

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

This is called the *normalized exponential* or also the *softmax function*. And we defined a_k as follows:

$$a_k(\mathbf{x}) = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

Note: It is a bit unclear, why one would choose the logistic sigmoid function or the softmax function to represent the posterior probabilities. Both forms are very useful if $a(\mathbf{x})$ or $a_k(\mathbf{x})$ have a certain kind of functional form.

Class conditionals – continuous inputs

We have to make assumptions about the class conditional densities. Let's choose a multivariate Gaussian:

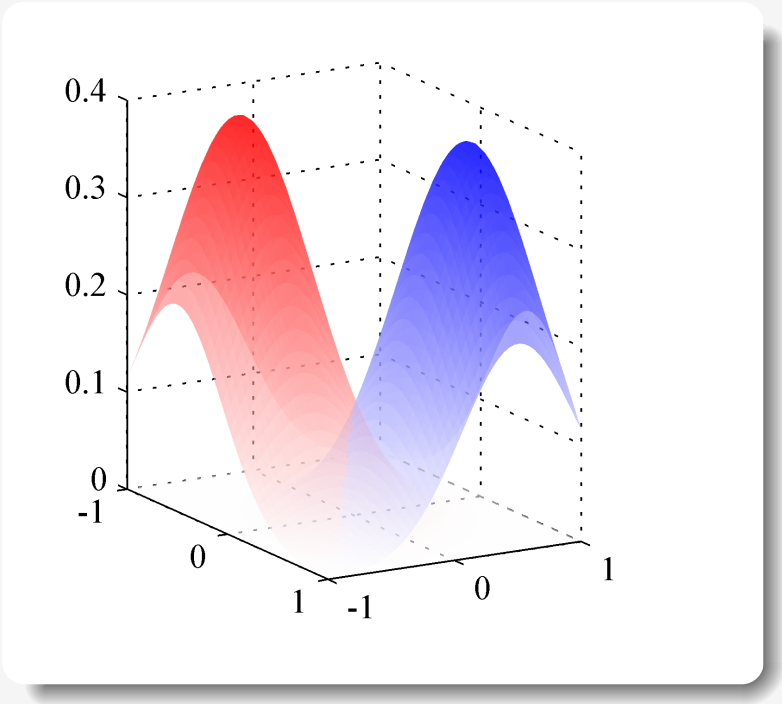
$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\}$$

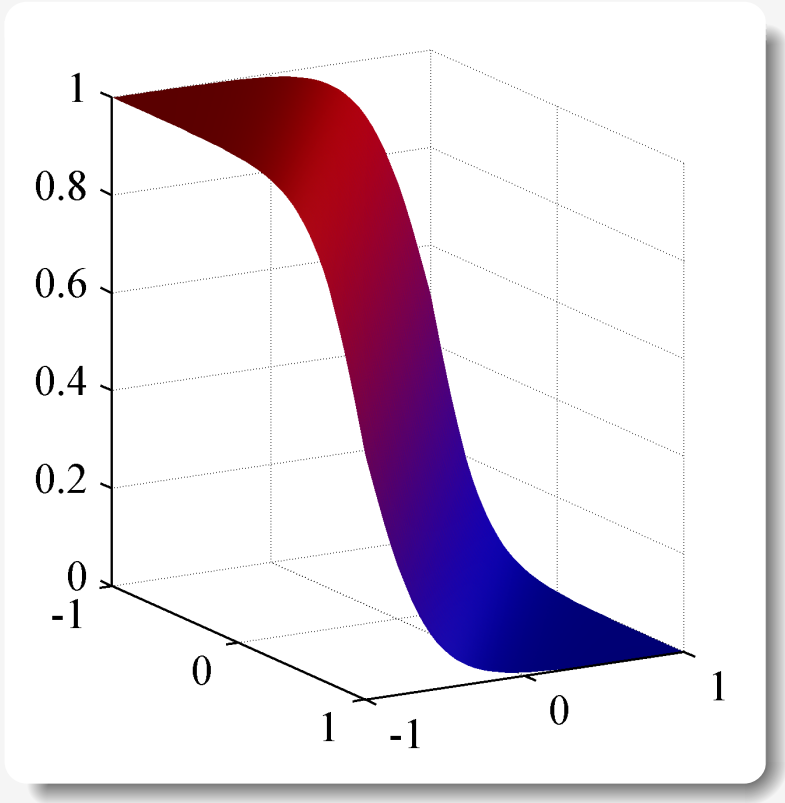
Note: each class has a different $\boldsymbol{\mu}_k$, but all share the same covariance matrix!
For the case with $K = 2$ we have

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$$

with

$$\begin{aligned} \mathbf{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ w_0 &= -\frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \end{aligned}$$





Similarly, in the general case with $K > 2$ classes we again get a linear function of $\mathbf{x}$ for $a_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_0$ with

$$\begin{aligned} \mathbf{w}_k &= \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k \\ w_{0k} &= -\frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln p(\mathcal{C}_k) \end{aligned}$$

In both cases, we see that the quardratic terms in $\mathbf{x}$ are cancelled, which is due to the assumption of a common covariance matrix. One gets linear decision boundaries.
If we allow each class conditional distribution to have its own covariance matrix $\boldsymbol{\Sigma}_k$ then we get quadratic decision boundaries.

Getting the parameters right – maximum likelihood

But we are not done yet .... We decided on the parametric functional form of the class conditinal densities $p(\mathbf{x}|\mathcal{C}_k)$ (and $p(\mathcal{C}_k)$), but these densities are governed by parameters and how do we determine the actual values of these?

We have a data set comprising of observations $\mathbf{x}$ along with their corresponding class labels.

So we can use maximum likelihood estimation! But we will not show the details in class for Gaussian class-conditionals ...

Discrete inputs – Naïve Bayes Classifier

Now we want to consider a classification task with discrete (categorial) features x_i. In this case, the simplest class conditional model is a joint multinomial (i.e. a table):

$$P(x_1 = a, x_2 = b, \ldots | y = c) = w_{cab\ldots}$$

So again, here we would use Baye's rule, as before, to determine the posterior class probabilities and, also again, use MLE to determine parameters. But now we have a big practical problem ...

$$w_{cab\ldots} = \frac{\sum_n [y_n = c][x_1 = a][x_2 = b][\ldots]}{\sum_n [y_n = c]}$$

Therefore we make the following assumption: Features x_i are independent given class y!! That is

$$P(\mathbf{x}|y) = \prod_i P(x_i|y)$$

Of course, our decision rule for the final assignment of the actual class stays the same: $y^* = \text{argmax}_y P(y) \prod_n P(x_n|y)$. So the only thing left to do is to determine $w_{kij} = P(x_i = j|y = k)$ from the training data.

Naïve Bayes Classifier - 2

Main challenge: Get likelihood fct. *formulated* in the right way. Consider the following abbreviations:

$$
\begin{aligned}
P(x_i = j | y = k) &= w_{kij} = \prod_j w_{kij}^{[x_i=j]} \\
P(\mathbf{x} | y = k, w) &= \prod_i \prod_j w_{kij}^{[x_i=j]} \\
P(\mathbf{x} | y, w) &= \prod_k \prod_i \prod_j w_{kij}^{[x_i=j][y=k]}
\end{aligned}
$$

$$
\begin{aligned}
\ell(\mathbf{w}) &= \ln P(y_1, \mathbf{x}_1, y_2, \mathbf{x}_2, \ldots | w) = \ln \prod_n P(y_n, \mathbf{x}_n | w) \\
&= \sum_n \ln P(\mathbf{x}_n | y_n, w) P(y_n | w) = \sum_n \ln P(y_n | w) + \sum_n \ln P(\mathbf{x}_n | y_n, w) \\
&= \sum_n \ln P(y_n | w) + \sum_n \sum_{kij} [x_i^n = j][y^n = k] \ln w_{kij}
\end{aligned}
$$

Naïve Bayes Classifier - 3

Optimize $\ell(\mathbf{w})$ by setting its derivative to zero (enforce normalization with Lagrange multipliers).
Results:

$$w_{kij} = \frac{\sum_n [x_i^n = j][y^n = k]}{\sum_n [y^n = k]}$$

$$w_k = P(y = k|w) = \frac{\sum_n [y^n = k]}{\sum_n \sum_j [y^n = j]}$$

Simple algorithm:

- Sort data cases into bins according to y_n.
- Compute marginal probabilities $P(y = k)$ using frequencies.
- Estimate $P(x_i|y = k)$ using frequencies.
- (Do not forget *smoothing*).

Probabilistic Discriminantive Models

So far we talked about an indirect approach to determine the posterior class probabilities (via Baye's rule and fitting class conditionals and class priors seperately). This represents an example of a *generative* model because given such a model, one can generate synthetic data by drawing values of $\mathbf{x}$ from the marginal distribution $p(\mathbf{x})$. Can you see any disadvantage with this method?

In the following we will consider another approach: If we (would) know the functional form of the posterior class probabilities (i.e. $p(\mathcal{C}_k|\mathbf{x})$), then we can determine its parameters directly using maximum likelihood. Because this approach is still embedded in a probabilistic framework the resulting models are called *probabilistic discriminative* models.

Logistic Regression

Again we will start with looking at two-class classification.

We model the posterior distribution directly

$$p(\mathcal{C}_1|\phi) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$

This is called *logistic regression*, though this is of course a model for classification.

Using Maximum Likelihood, we will determine the parameters of the logistic regression model. Given a data set $\{\phi_n, t_n\}$, with $t_n \in \{0, 1\}$ and $\phi_n = \phi(\mathbf{x}_n)$ we get for the likelihood function

$$p(\mathbf{T}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$$

with $\mathbf{T} = (t_1, \ldots, t_N)^T$ and $y_n = p(\mathcal{C}_1|\phi_n)$.

As usual we want to minimize the *negative log likelihood* function:

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$$

This kind of error function is called the (binary) *cross entropy error* function.

Taking the gradient of the error function, we get:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\phi_n = \mathbf{\Phi}^T(\mathbf{y} - \mathbf{T})$$

Note that this gradient has the same form as the gradient of the sum-of-squares error function for the linear regression model.

Iterative Reweighted Least Squares

But the logistic sigmoid function is nonlinear, so we no longer have a closed form solution.

However, minimization of the error function can be done via an efficient iterative optimization technique. And, even better, the error function is convex (i.e. it has a *unique* minimum)!

The Newton-Raphson algorithm:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1} \nabla E(\mathbf{w})$$

So just to have everything in one place:

$$
\begin{aligned}
E(\mathbf{w}) &= -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \\
\nabla_{\mathbf{w}} E(\mathbf{w}) &= \mathbf{\Phi}^T(\mathbf{y}-\mathbf{T}) \\
\nabla_{\mathbf{w}}\nabla_{\mathbf{w}} E(\mathbf{w}) &= \mathbf{H} = \mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}
\end{aligned}
\tag{6}
$$

$\mathbf{R}$ is a $N \times N$ diagonal matrix with elements $\mathbf{R}_{nn} = y_n(1-y_n)$. $\mathbf{H}$ is positive definite and thus the error function is a convex function of $\mathbf{w}$.

Using the Newton-Raphson update step, we get

$$\mathbf{w}^{t+1} = (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{z}$$

where we have defined $\mathbf{z}$ as follows

$$\mathbf{z} = \mathbf{\Phi}\mathbf{w}^t - \mathbf{R}^{-1}(\mathbf{y}-\mathbf{t})$$

The algorithm is called *iterative reweighted least squares*.

Multiclass logistic regression

For logistic regression with multiple classes we have to adapt the functional form of the posterior class probability. Considering the results from the domain of *generative* classification models, we choose the following (note that we have now y_k):

$$p(\mathcal{C}_k|\phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

with

$$a_k = \mathbf{w}_k^T \phi$$

Again we need the likelihood function and using the 1-of-K coding scheme it is given by

$$p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\phi_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

where $y_{nk} = y_k(\phi_n)$ and $\mathbf{T}$ is an $N \times K$ matrix of target variables with elements t_{nk}.

Taking the negative logarithm then gives the cross entropy error function:

$$E(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln(y_{nk})$$

We will need the derivatives of y_k with respect to the *activations* a_j:

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

with I_{kj} the elements of the identity matrix.
So we get

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_N) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\phi_n$$

$$\nabla_{\mathbf{w}_k}\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \ldots, \mathbf{w}_N) = -\sum_{n=1}^{N} y_{nk}(I_{kj} - y_{nj})\phi_n\phi_n^T$$