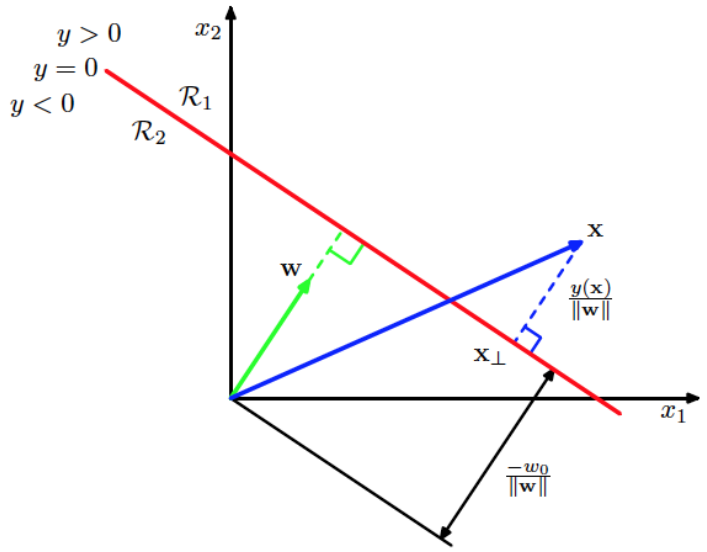
Linear models for classification (supervised learning)

The goal in classification is to take an input vector **x** and to assign it to one of K discrete classes C_k (k = 1, . . . , K)

The simplest models for classification are linear, i.e. the decision surfaces that separate the classes are linear functions of the input vector **x** – they are (D − 1)-dimensional hyperplanes within the D-dimensional input space **x**

Previously we've seen (linear) regression, where the goal is to estimate $y(\mathbf{x})=\mathbf{w}^*\mathbf{x}+w_0$, y being a real-valued function of **x**

In classification, instead, we want to predict a discrete variable (the class c_k), or a posterior probability $p(c_k|\mathbf{x})$ that **x** belongs to the class c_k.
To do this, we can model $y(\mathbf{x})= f(\mathbf{w}^*\mathbf{x}+w_0)$ (f is called the activation function, in general nonlinear) and then define e.g. that
x is in C_1 if $y(\mathbf{x})>0$, **x** is in C_2 if $y(\mathbf{x})<0$



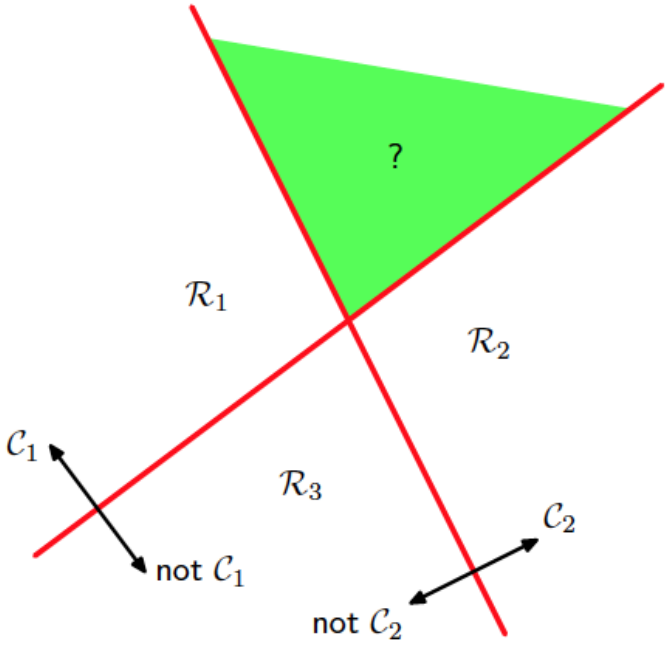
y=0 is the decision boundary

w determines orientation of surface (orthogonal to boundary)

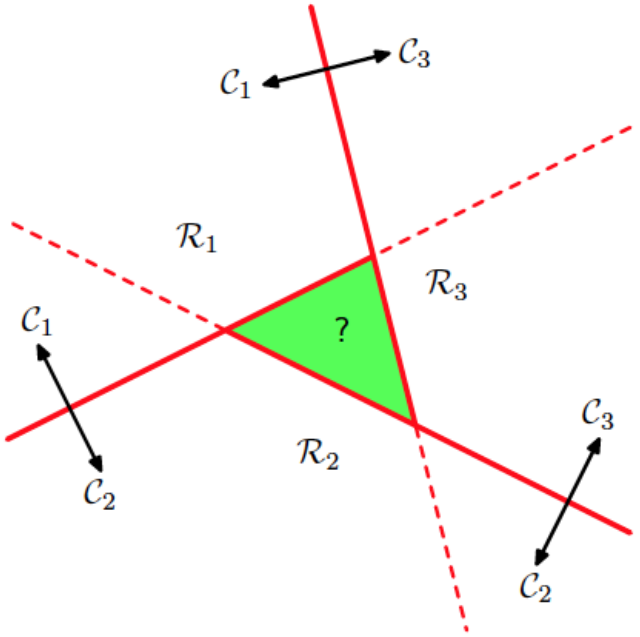
Generalized linear model (GLM)

After training (learning the weights **w**) on the training set, a new input (from the testing set) is assigned to a class according to the decision boundary

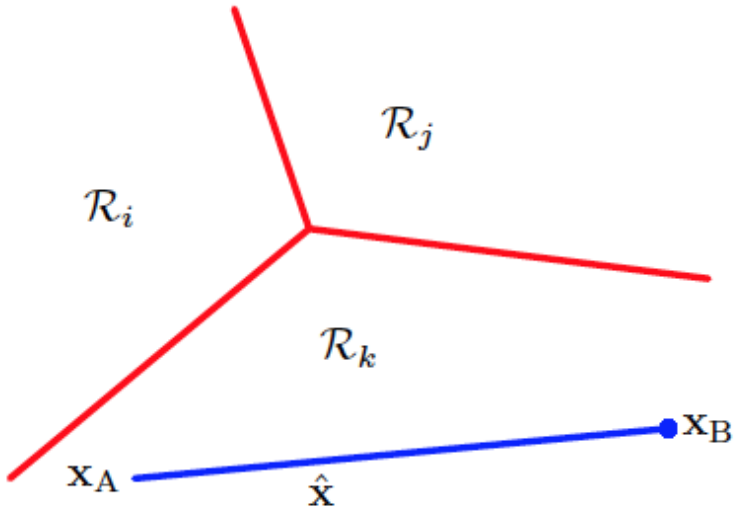
From two classes to multiple classes



One versus the rest



One versus one



General solution

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

x is assigned to c_k with the largest $y_k(\mathbf{x})$

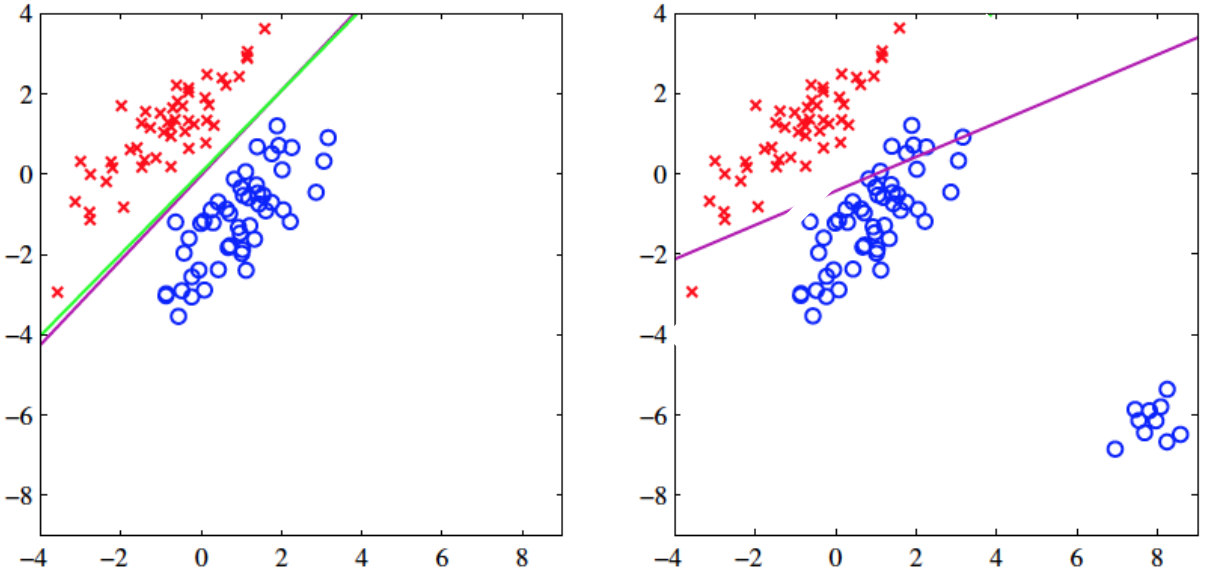
Learning the parameters of linear discriminant functions: least squares

As in regression, the first approach we can try to fit the (generalized linear) model is minimizing a sum-of-squares error function

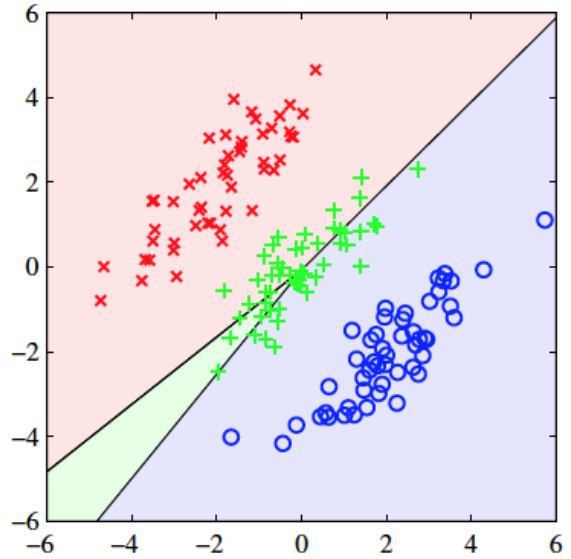
$$Tr[\ (XW-T)^T(XW-T)\ ] = \Sigma_{samples}\ (data_{sample}*weights - label_{sample})^2$$

Can be solved analytically, but

1) Least-squares discrimination boundary is not robust to outliers (too sensitive to “easy samples”)



2) Can fail with multiple classes

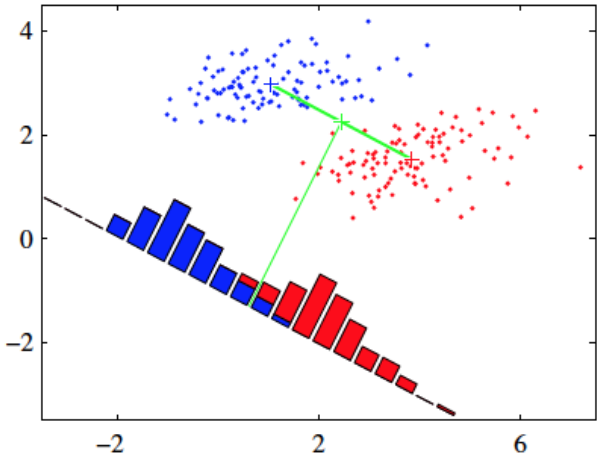


Indeed, least-squares corresponds to ML when assuming Gaussian distributed targets p(t|x), while discrete-valued targets (the classes) can be very non Gaussian!

Learning the parameters of linear discriminant functions: Fisher linear discriminant

Classification can be seen as dimensionality reduction via projection on optimal weights **w** (orthogonal to decision boundary).
One criterion to optimize classification can thus be finding the decision boundary that maximizes inter-class separation of the projections of data on **w**.

If you only focus on separating the class-means of the projections on **w**, the optimal boundary is: $\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$
But the projections can still overlap a lot due to anisotropic covariance matrices of data



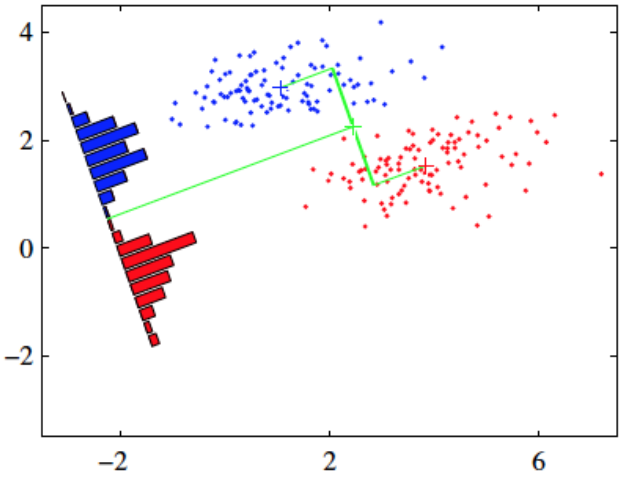
Fisher: maximize inter-class separation of the projections on **w** while also minimizing their variance, in order to minimize overlap.

Maximize ratio J(w) =

$$\frac{(\text{difference of means of projections})^2}{(\text{within class variance of projections})} = \frac{(\text{m2}-\text{m1})^2}{s_1^2+s_2^2} = \frac{\text{between-class variance}}{\text{total within-class variance}}$$

Optimal boundary: $\mathbf{w} \propto \mathbf{S}_\mathbf{W}^{-1}(\mathbf{m}_2 - \mathbf{m}_1).$

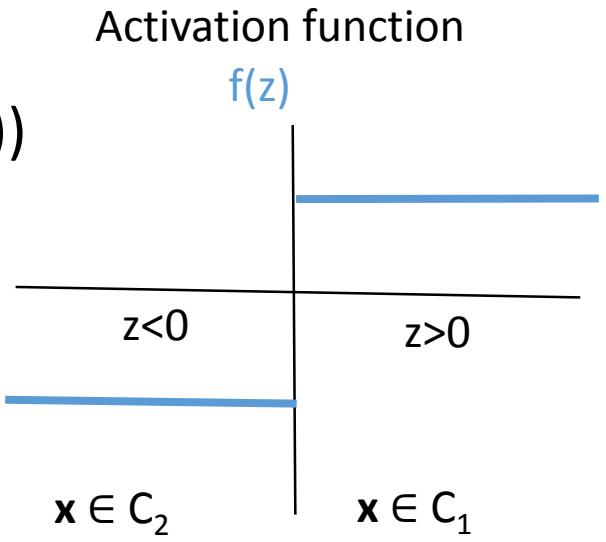
(not really a discriminant because it only gives direction of boundary, not threshold y_0 for classification



Learning the parameters of linear discriminant functions: perceptron algorithm

Rosenblatt 1962

Input observation data **x** ----> feature vector $\boldsymbol{\varphi}(\mathbf{x})$ ----> $y(\mathbf{x}) = f(\mathbf{w}^T * \boldsymbol{\varphi}(\mathbf{x}))$



To learn the weights **w**, you might minimize number of misclassified inputs – but this error function is piecewise constant (inputs are discrete) and thus its gradient is almost zero everywhere → hard to optimize **w** by iteratively decreasing the gradient of the error function.

Trick: if the labels are defined as $t(\mathbf{x})=1$ if $\mathbf{x} \in C_1$ and $t(\mathbf{x})=-1$ if $\mathbf{x} \in C_2$, the correct classification for the observations $\mathbf{x}_1,\ldots, \mathbf{x}_N$ can be written as the condition $\mathbf{w}^T * \boldsymbol{\varphi}(\mathbf{x}_i)\, t(\mathbf{x}_i) > 0$ for $i = \{1,\ldots,N\}$.
The corresponding error function is thus written as

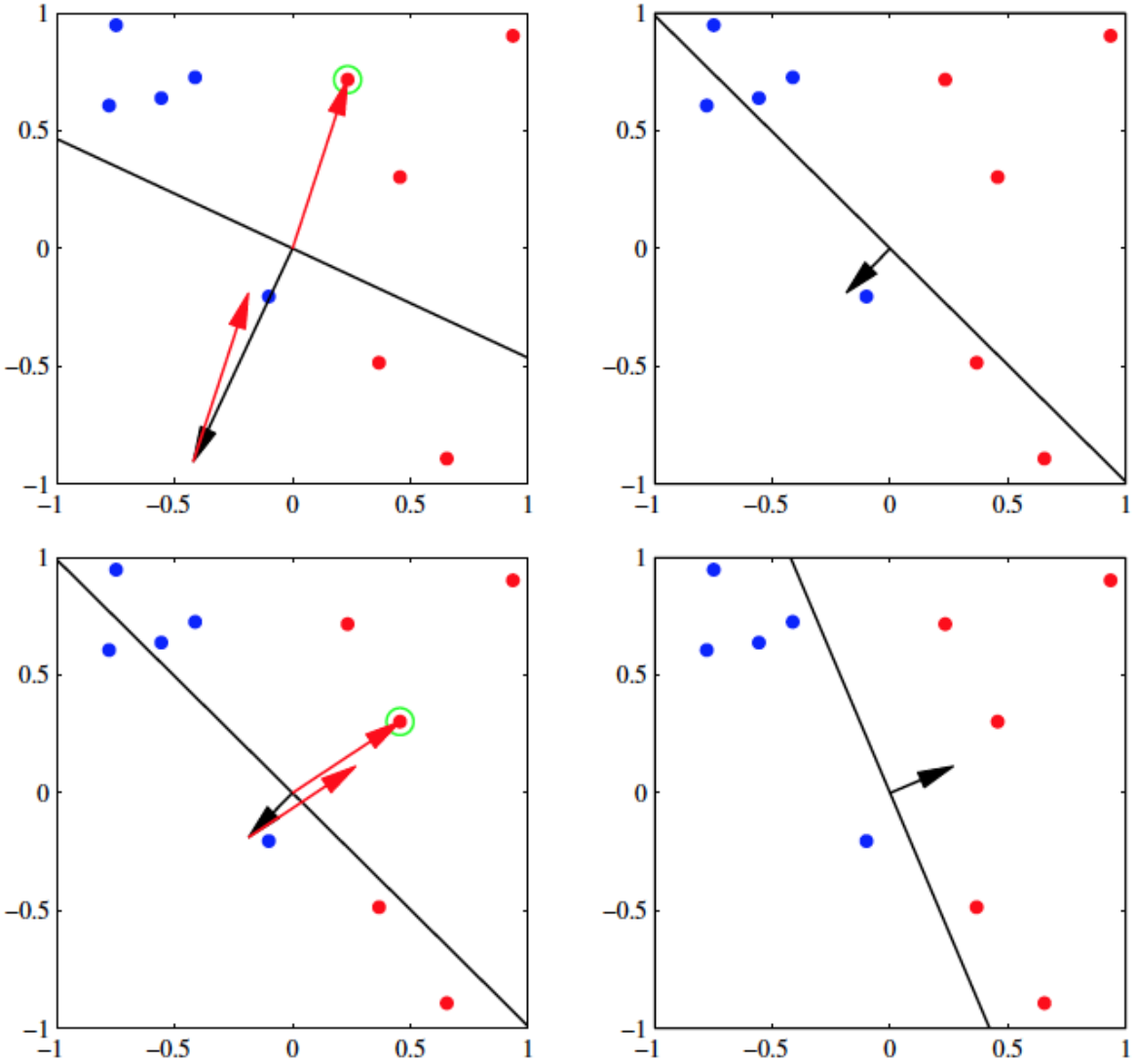
$$E(\mathbf{w}) = - \Sigma_{\text{misclassified observations}}\ \mathbf{w}^T * \boldsymbol{\varphi}(\mathbf{x}_i)\, t(\mathbf{x}_i) \qquad \text{linear in } \mathbf{w} !$$

Can be minimized with stochastic gradient descent: an iterative algorithm where, at iteration τ,

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\, \nabla E(\mathbf{w})$$

learning rate, can be set to 1

The perceptron algorithm illustrated



$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E(\mathbf{w}) \qquad 1)$$

The set of misclassified patterns can change from one iteration to the next

This update rule is not guaranteed to monotonically decrease the error function at each iteration.

Perceptron convergence algorithm: if the training data is linearly separable (if there exists an optimal **w**), then 1) will find it within a finite number of steps.

Limitations of the perceptron algorithm

$$\mathbf{w}^{\tau} - \eta \nabla E(\mathbf{w})$$

Practical problems:

- when running the algorithm on a dataset, you don't know in advance the number of iterations needed, and it's hard to distinguish between very-slowly-converging solution and no possible solution
- Even when solution exists, but there is more than one, solution found depends on initialization and training set
- No posterior probabilities, only discrimination
- Hard to generalize to more than two classes
- Limited to linear combinations of functions! (it's a fit to a generalized linear model and is thus linear in the weights **w**)

$$y(\mathbf{x}) = f(\mathbf{w}^{T} * \boldsymbol{\varphi}(\mathbf{x}))$$

This last limitation is solved with deep (more-than-one-layer) neural networks.

Check out chapter 3 for different choices of $\boldsymbol{\varphi}$!