Linear Models for Classification

Alberto Paccanaro
EMAp – FGV

www.paccanarolab.org

Material and images in these slides are from (or adapted from):
C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006

The Classification problem

Take an input vector **x** and to assign it to one of K discrete classes C_k where $k = 1, \dots, K$.

Classes are disjoint, input space is divided into *decision regions*

Boundaries: *decision boundaries* (*decision surfaces*).

Linear models: decision boundaries are linear functions

(D−1)-dimensional hyperplanes in the D dimensional input space.

Linearly separable: data sets whose classes can be separated exactly by linear decision surfaces

Representing target values

Two-classes (K=2)

- one variable, binary representation $t \in \{0, 1\}$

  $t = 1$ represents class C_1 and $t = 0$ represents class C_2.

- value of t is the probability that the class is C_1.

K > 2 classes

- it is convenient to use a 1-of-K coding scheme

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

The general model

$$\boxed{y(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0\right)}$$

To predict class labels we transform the linear function of w using a nonlinear *activation function* $f(\cdot)$

- The **model is not linear** in the parameters (remember that linear regression was linear in the parameters)

- **The decision surfaces are linear functions of x, even if the function $f(\cdot)$ is nonlinear** (correspond to $\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$ = constant)

Generalized linear models

As we discussed earlier, there are 3 approaches

1. Discriminant functions
2. Model $p(\mathbf{x}|C_k)$ and $P(C_k)$ and then use Bayes theorem to infer $p(C_k|\mathbf{x})$ (the *generative approach*)
3. Model $p(C_k|\mathbf{x})$ directly

1. Discriminant Functions

A discriminant is a function that takes an input vector x and assigns it to one of *K* classes, denoted C_k – *no probabilities here* ☺

… lets begin with some geometry…

A. Two Classes (K = 2)

Let us not consider the activation function for now.

The simplest linear discriminant:

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

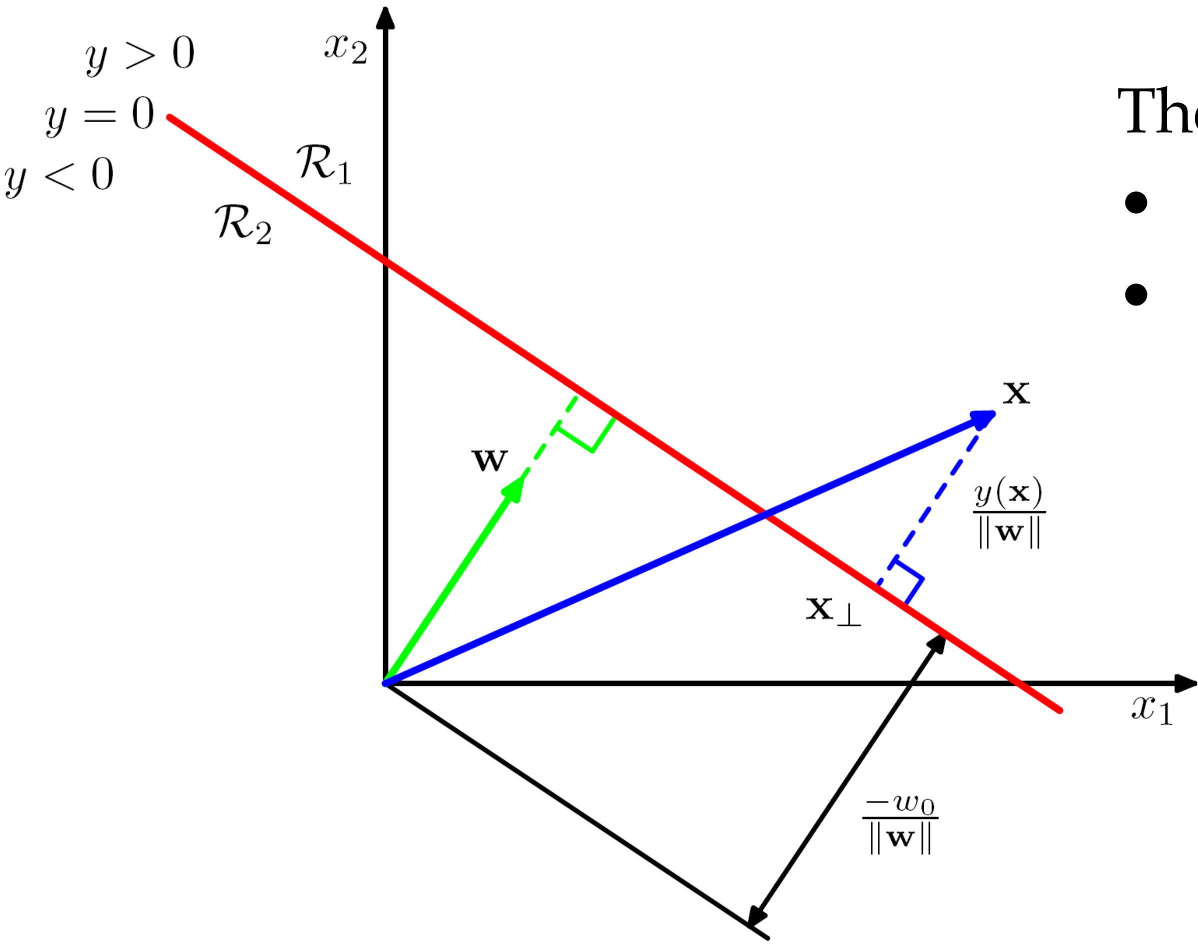
w: weight vector
w_0: bias

x assigned to C_1 if y(**x**) ≥ 0, C_2 otherwise.

$$y(\mathbf{x}) = \widetilde{\mathbf{w}}^{\mathrm{T}}\widetilde{\mathbf{x}}$$

including the bias

Decision boundary: $y(\mathbf{x}) = 0$, corresponds to a **(D–1)-dim. hyperplane** within the **D-dim. input space**.



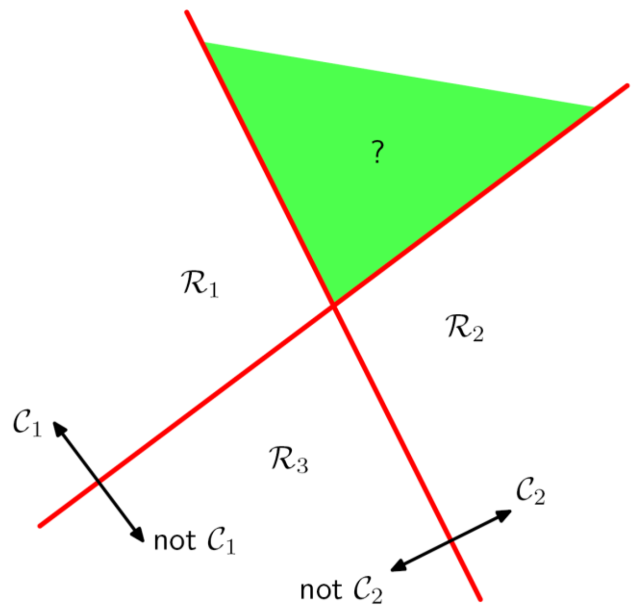
The decision surface is:

- perpendicular to **w**
- its displacement from the origin is controlled by the bias parameter w_0.

- The signed orthogonal distance of a general point **x** from the decision surface is given by $\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$

B. Multiple classes (K > 2)

Combinations of 2-class discriminants do not work !



one-versus-rest

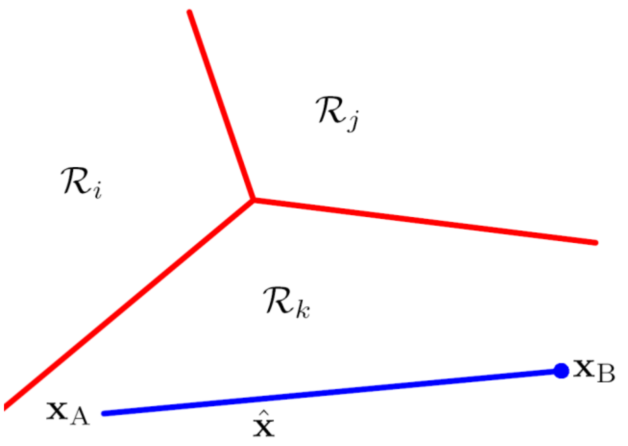
Possible solution: build a single K-class discriminant comprising K linear functions:

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

Assign x to class C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$.

- *There is no really good solution for this issue… (see Bishop 7.1.3 Multiclass SVMs, page 338)*

- The decision regions of such a discriminant are always connected and convex.



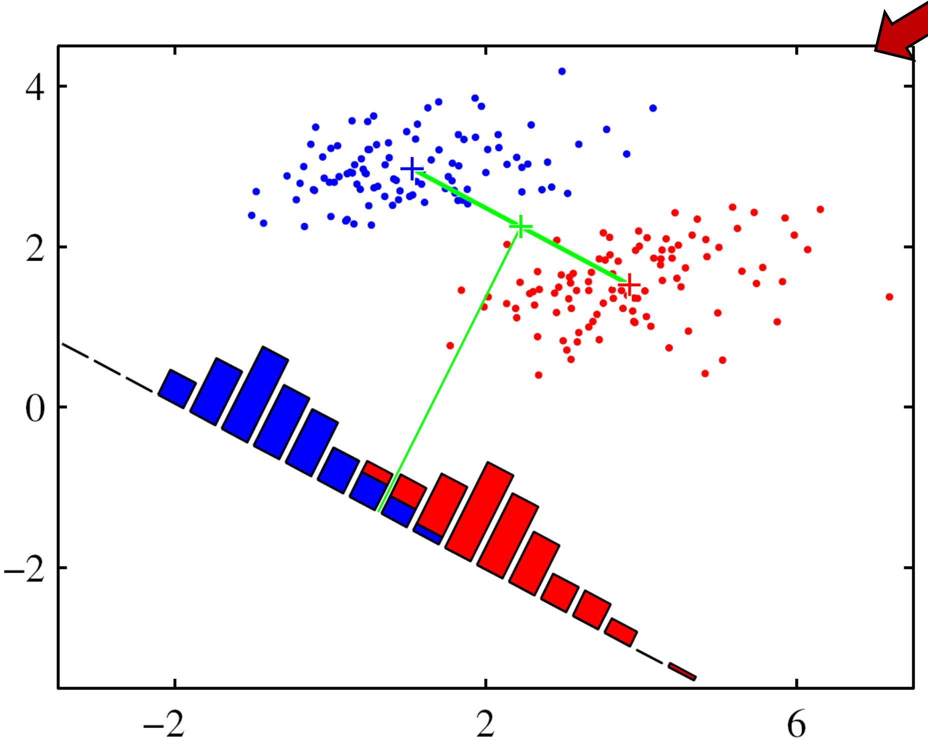
Fisher's Linear Discriminant (2 classes)

1) Project the points onto one dimension: $y = \mathbf{w}^{\mathrm{T}}\mathbf{x}$
2) If the value (scalar) is larger than w_0, then C_1, else C_2.

Note that this is equivalent to $y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0$

Question: can I "adjust" the projection so that I have the points that belong to the 2 different classes **as separated as possible**?

The simplest measure of the separation of the classes is the **separation of the projected class means**.



We need to maximize a large separation between the projected class means while also giving a **small variance within each class**

Two Classes, N_1 points ∈ C_1, N_2 points ∈ C_2

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n \qquad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n$$

$$m_k = \mathbf{w}^{\mathrm{T}} \mathbf{m}_k$$

To maximize the separation between the class means, maximize:

$$m_2 - m_1 = \mathbf{w}^{\mathrm{T}} (\mathbf{m}_2 - \mathbf{m}_1)$$

Within variance of the projected data, for class *k* (here $y_n = \mathbf{w}^{\mathbf{T}} \mathbf{x}_n$).

$$s_k^2 = \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2$$

The Fisher criterion:

Ratio of the between-class variance to the within-class variance

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

Making the dependence on **w** explicit:

$$J(\mathbf{w}) = \frac{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w}}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w}}$$

$$\mathbf{S}_{\mathrm{B}} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^{\mathrm{T}}$$ between-class covariance matrix

$$\mathbf{S}_{\mathrm{W}} = \sum_{n \in \mathcal{C}_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^{\mathrm{T}} + \sum_{n \in \mathcal{C}_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^{\mathrm{T}}$$

within-class covariance matrix

Taking the gradient of J(**w**), it is maximized when:

$$(\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{B}}\mathbf{w})\mathbf{S}_{\mathrm{W}}\mathbf{w} = (\mathbf{w}^{\mathrm{T}}\mathbf{S}_{\mathrm{W}}\mathbf{w})\mathbf{S}_{\mathrm{B}}\mathbf{w}$$

We only care about the direction, hence:

$$\boxed{\mathbf{w} \propto \mathbf{S}_{\mathrm{W}}^{-1}(\mathbf{m}_2 - \mathbf{m}_1)}$$ Fisher's Linear Discriminant

Note: when the data is "spherical", S_w is proportional to the identity, w is proportional to the difference between the means … ☺

… don’t forget …

Now we have the **<u>direction for the projection, but it is not a classification yet</u>**

We are left with determining a threshold w_0 for the classification…

One option:

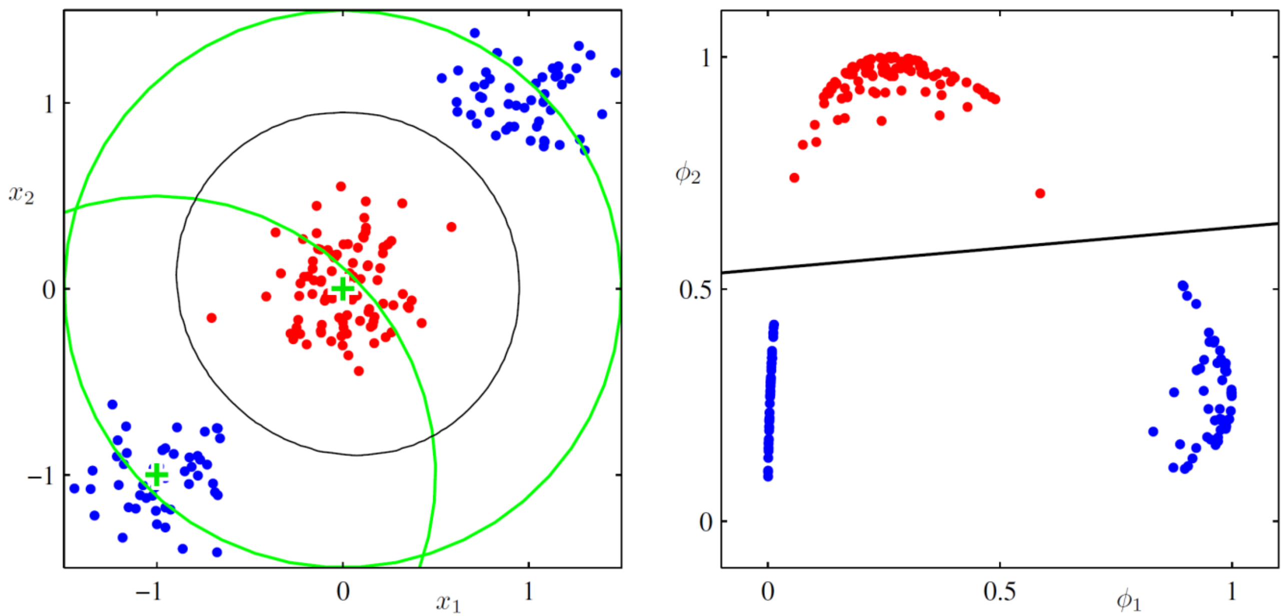
a) model the class conditional densities $p(y|C_k)$ (e.g. using Gaussian distributions, learn the parameters of the gaussians by maximum likelihood)
b) Calculate the priors $p(C_k)$ from the data
c) Use Bayes theorem to calculate w_0

This can be generalized to ***<u>multiple classes</u>*** *but we skip it because the maximization of the criteria is a bit involved.*

Note on fixed basis functions

We can first apply a nonlinear transformation of the inputs $\phi(x)$ – *just as we did in linear regression*.

The classification problem can become easier



The Perceptron

A two-class model:

$$y(\mathbf{x}) = f\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x})\right)$$

nonlinear transformation of $\mathbf{x}$ to get $\phi(\mathbf{x})$
$\phi(\mathbf{x})$ used for a generalized linear model
Note: $\phi_0(\mathbf{x}) = 1$ (bias)

Nonlinear activation function (step function):

$$f(a) = \begin{cases} +1, & a \geqslant 0 \\ -1, & a < 0 \end{cases}$$

Target values:

$t = +1$ for class C_1
$t = -1$ for class C_2

Goal of the algorithm: determine the parameters w that minimizes misclassification

We want $\mathbf{w}$ such that:

when $\mathbf{x}_n \in C_1$ $\quad \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) > 0$

when $\mathbf{x}_n \in C_2$ $\quad \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) < 0$

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

Perceptron criterion: no error for patterns that are correctly classified. For a misclassified pattern minimize $-\mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)t_n$

$$\boxed{E_{\mathrm{P}}(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^{\mathrm{T}}\phi_n t_n}$$

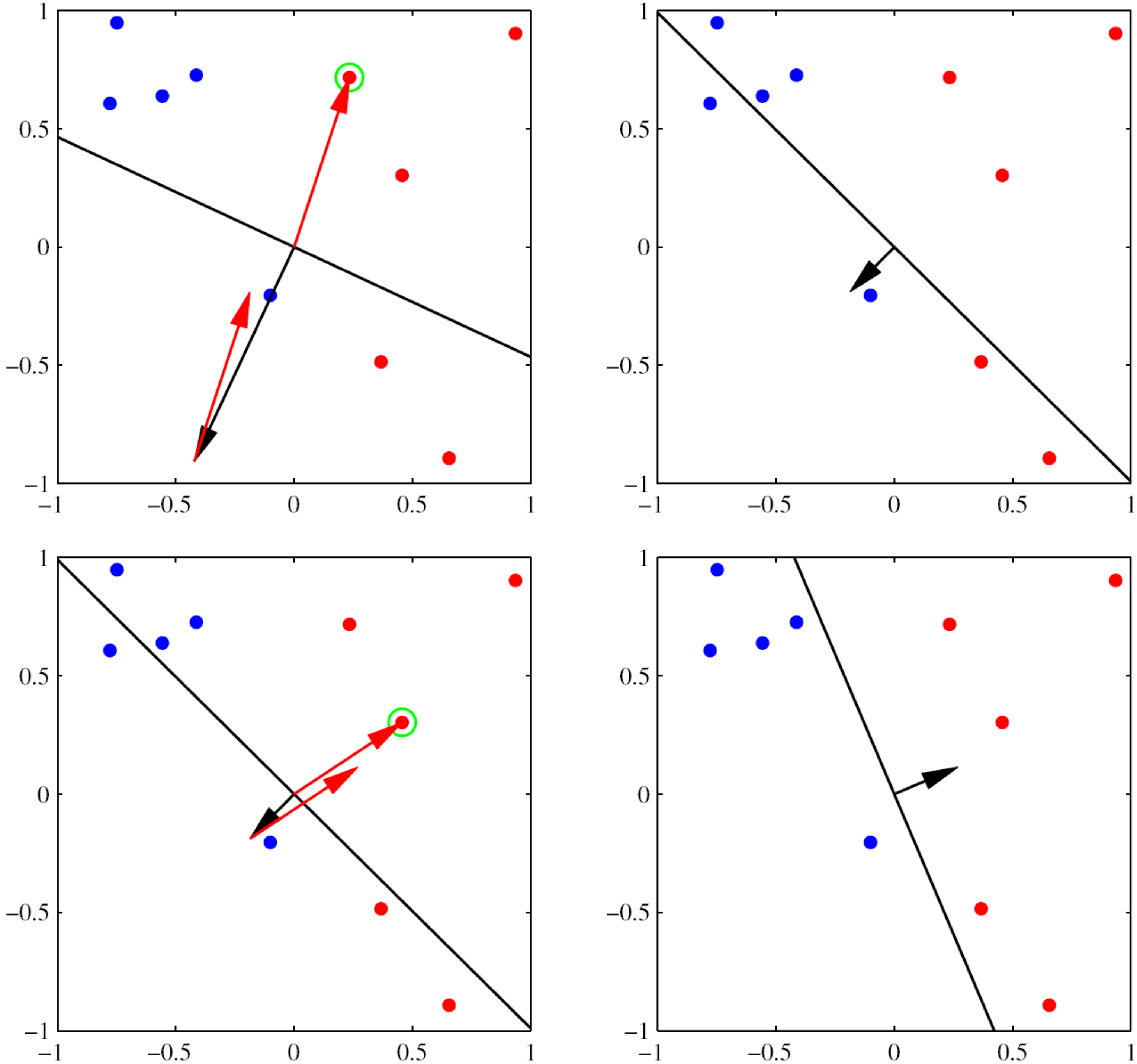
where $\mathcal{M}$ is the set of misclassified patterns

Applying stochastic gradient descent:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_{\mathrm{P}}(\mathbf{w}) = \mathbf{w}^{(\tau)} + \eta \boldsymbol{\phi}_n t_n$$

Interpretation: *we cycle through the training patterns in turn, and for each pattern x_n :*

- *If the pattern is correctly classified, do nothing*
- *If it is incorrectly classified:*
  - *for class C_1 add the vector $\boldsymbol{\phi}(\boldsymbol{x}_n)$ onto $\boldsymbol{w}$*
  - *for class C_2 we subtract the vector $\boldsymbol{\phi}(\boldsymbol{x}_n)$ from $\boldsymbol{w}$*


1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

Important

- learning rule not guaranteed to reduce the total error function at each stage
- **perceptron convergence theorem**: if the training data set is linearly separable the perceptron will find an exact solution in a finite number of steps
- Solution not unique, and will depend on the initialization of the parameters and on the order of presentation of the data points.
- For data sets that are not linearly separable, the algorithm will never converge.
- Does not generalize readily to $K > 2$ classes.
- Limited in what it can learn: it is based on linear combinations of fixed basis functions.

2. Probabilistic Generative Models

Models with linear decision boundaries arise from simple assumptions about the distribution of the data…

Two classes

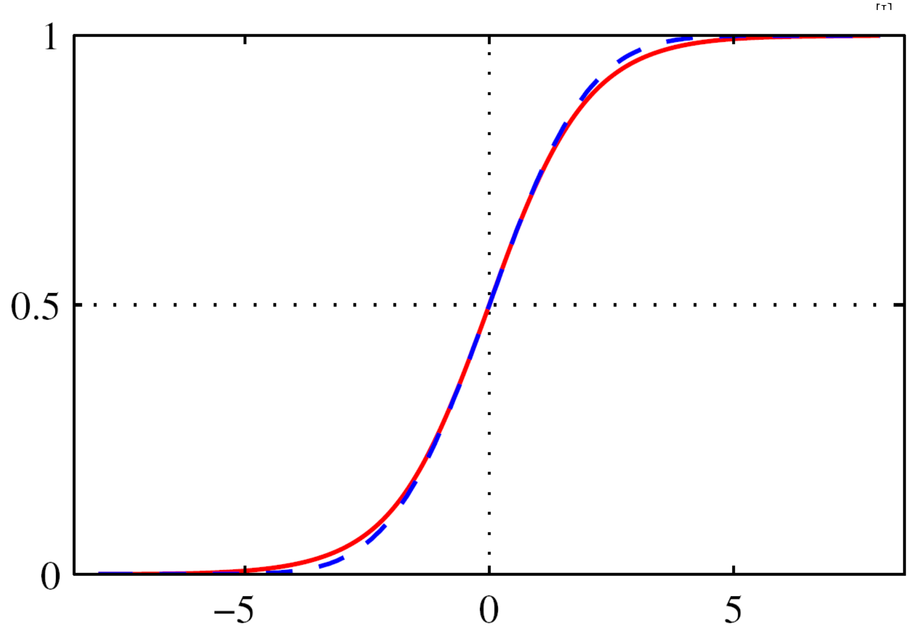
$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

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

sigmoid function

where we define:

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



K > 2 classes

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

softmax function

where we define:

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

IMPORTANT

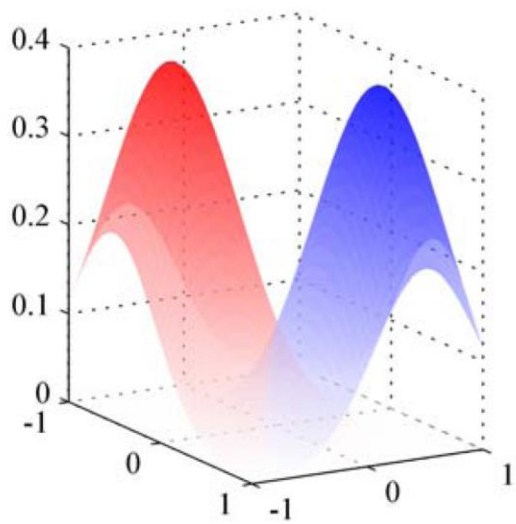
The use of the sigmoid/softmax activation functions allows the outputs to be interpreted as posterior probabilities of class membership.

QUESTION

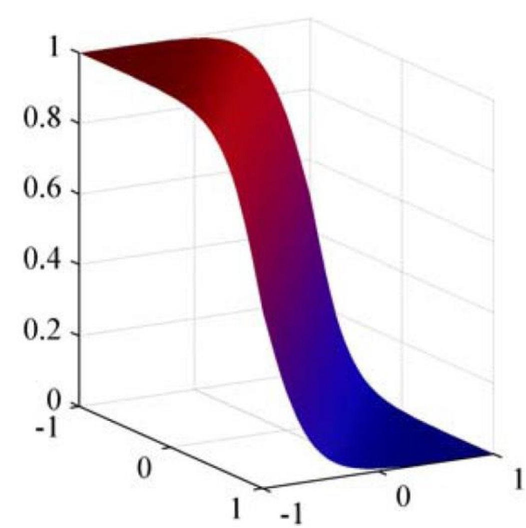
How do the form for the class conditional densities $p(\mathbf{x} \mid C_k)$ affects the form of $p(C_k \mid \mathbf{x})$?

2 classes, continuous inputs, Gaussian class-conditional densities, same covariance

We obtain a linear function of **x** in the argument of the logistic sigmoid.



$p(\mathbf{x}|C_k)$



$p(C_1|\mathbf{x})$

K classes, continuous inputs, Gaussian class-conditional densities, same covariance

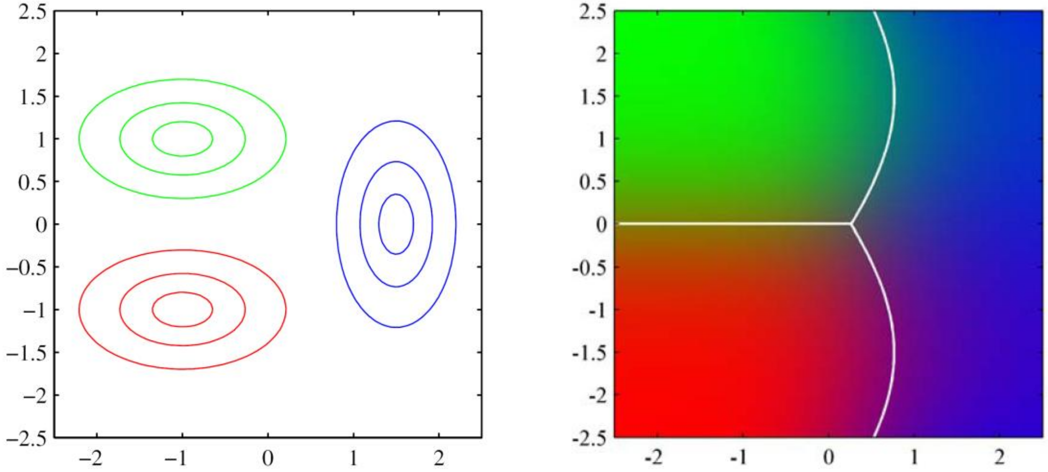
The $a_k(\mathbf{x})$ are linear functions of $\mathbf{x}$

As before, decision boundaries will be defined by linear functions of $\mathbf{x}$.

Again we have a generalized linear model.

K classes, continuous inputs, Gaussian class-conditional densities, different covariance

we obtain ***quadratic*** functions of **x**, giving rise to a quadratic discriminant.



These results can be extended to the class-conditional densities of the exponential family of distributions.

3. Model $p(C_k | \mathbf{x})$ directly – Probabilistic Discriminative Models

The posterior probability of class C_1 can be written as a logistic sigmoid (softmax for multiclass) acting on a linear function of **x.**

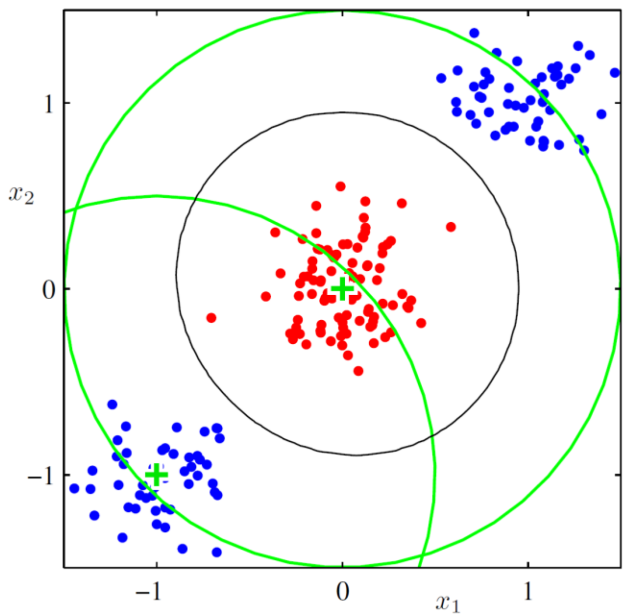
Idea: use the functional form of the generalized linear model <u>explicitly</u> and determine its parameters directly by using maximum likelihood. That is, we maximize a likelihood function defined through p(Ck|x) (discriminative training).

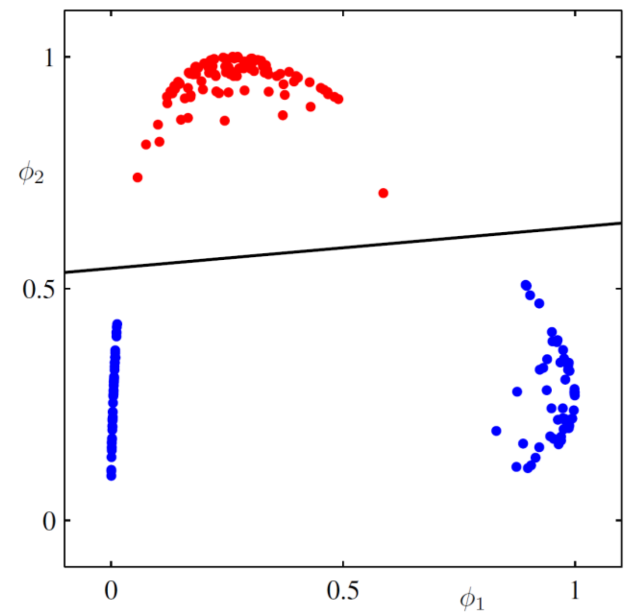
FGV INT

Again… on fixed basis functions

All these algorithms are equally applicable if we first apply a nonlinear transformation of the inputs $\phi(x)$

Decision boundaries will be linear in the feature space $\phi(x)$ and nonlinear in the original x space





Logistic Regression (2 classes)

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma\left(\mathbf{w}^{\mathrm{T}}\phi\right)$$
$$p(\mathcal{C}_2|\phi) = 1 - p(\mathcal{C}_1|\phi)$$

A logistic sigmoid acting on a linear function of the feature vector ϕ

In the terminology of statistics, this is known as logistic regression (a.k.a. Logistic Discriminant, Logistic Discrimination)

This is a model for classification not regression ☺

Only M parameters.

Let us use maximum likelihood to determine the parameters w of the model.

Dataset of N datapoints: $\{\boldsymbol{\phi}_n, t_n\}$

$$\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$$
$$t_n \in \{0, 1\}$$

Likelihood function:

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

$$\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$$
$$y_n = p(\mathcal{C}_1|\boldsymbol{\phi}_n)$$

Take the negative logarithm:

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

$$y_n = \sigma(a_n) \quad a_n = \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}_n$$

Cross-Entropy error function

$$\boxed{\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\phi_n}$$

$$y_n = \sigma(a_n)$$
$$a_n = \mathbf{w}^{\mathrm{T}}\phi_n$$

The contribution to the gradient from data point *n* is given by the 'error' $(y_n - t_n)$ between the target value and the prediction of the model, times the input for point n.

Remember… this is the same form as the gradient of the sum-of-squares error function for the linear regression model….

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2$$

$$\nabla \ln p(\mathbf{t}|\mathbf{w}, \beta) = \sum_{n=1}^{N}\left\{t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)\right\}\phi(\mathbf{x}_n)^{\mathrm{T}}$$

but there is no closed form this time… why ?

We use a sequential procedure where patterns are presented one at a time and weight vectors is updated by stochastic gradient descent

IMPORTANT

Cross-entropy error function arises from maximizing likelihood of the data in binary classification problems.

… in a similar way as…

Sum-of-squares error function arises from maximizing likelihood of the data under a Gaussian noise assumption.

Iterative Reweighted Least Squares (IRLS)

The cross-entropy error function is concave ➔ unique minimum.

The Newton-Raphson update:

$$\mathbf{w}^{(\text{new})} = \mathbf{w}^{(\text{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w})$$

Applying it to the cross-entropy error function for the logistic regression model:

$$\begin{aligned}
\nabla E(\mathbf{w}) &= \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^{\mathrm{T}}(\mathbf{y} - \mathbf{t}) \\
\mathbf{H} &= \nabla\nabla E(\mathbf{w}) = \sum_{n=1}^{N} y_n(1 - y_n)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^{\mathrm{T}} = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi}
\end{aligned}$$

where R is an N × N diagonal matrix R with elements

$$R_{nn} = y_n(1 - y_n)$$

Iterative algorithm, with update formula:

$$\boxed{\mathbf{w}^{(\text{new})} = (\mathbf{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{R}\mathbf{z}}$$

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

Multiclass Logistic Regression

$$p(\mathcal{C}_k|\boldsymbol{\phi}) = y_k(\boldsymbol{\phi}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \qquad a_k = \mathbf{w}_k^{\mathrm{T}}\boldsymbol{\phi}$$

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

Multiclass cross-entropy error function

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

Gradient for iterative update

Extension of IRLS to multiclass is also possible.

IMPORTANT

The derivative of the log likelihood function for a linear regression model takes the form of

'error' ($y_n - t_n$) times the feature vector $\boldsymbol{\phi_n}$

… similarly…

for logistic sigmoid activation function and cross-entropy error function &

for softmax activation function and multiclass cross-entropy error function

… we again obtain this same simple form.