CHAPTER 10:

LİNEAR DİSCRİMİNATİON

Some slides adapted from:

- *E. Alpaydin, cmpe.boun.edu.tr/~ethem/i2ml3e*
- *M.A Carreira-Perpinan, faculty.ucmerced.edu/mcarreira-perpinan/teaching/CSE176/lecturenotes.pdf*

Classification examples

House
price
size
location

Sold

Not Sold

X-ray
Pixel intensity

Cancer

Not cancer

Iris flower
Petal length
Petal width
Sepal width

I. setosa

I. versicolor

I. virginica

Classification

- In previous lectures we looked at classification:
  - Training time: learn a set of discriminant functions $\{g_i(\mathbf{x})\}$
  - Test time: given a new instance $\mathbf{x}$, choose class k with highest value of $\{g_i(\mathbf{x})\}$

Likelihood- vs. Discriminant-based Classification

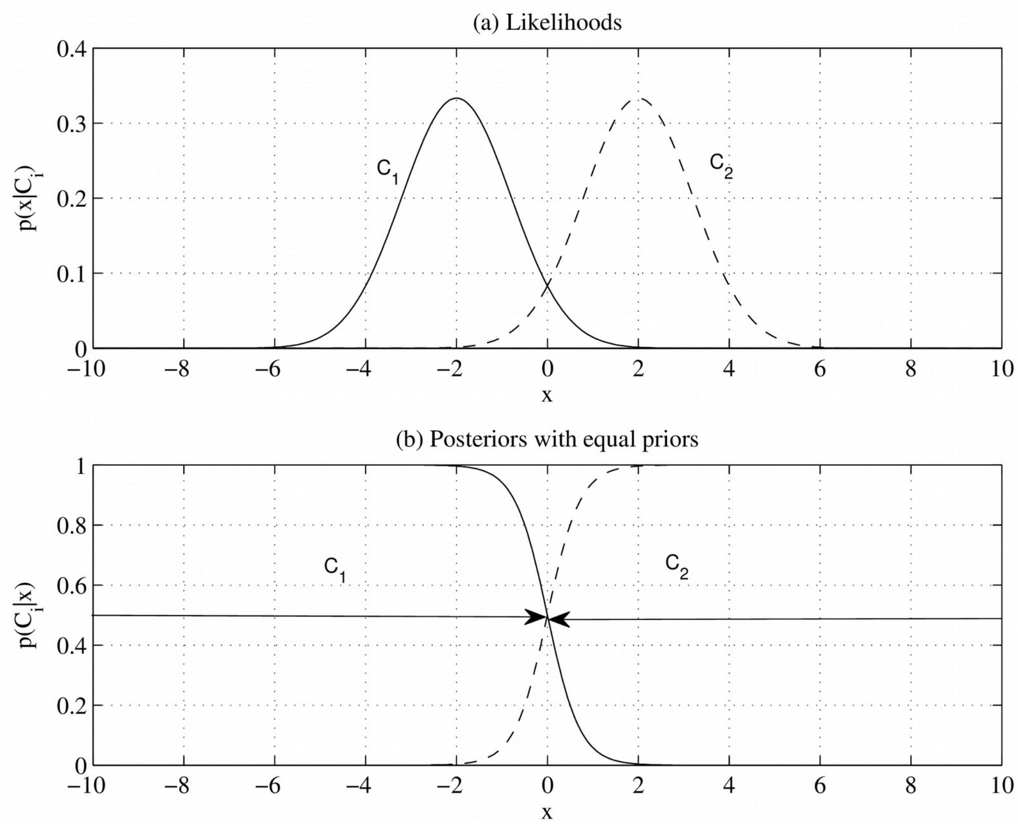
- There are two approaches to learning the discriminant functions $g_i(\boldsymbol{x})$:
  - Likelihood-based
  - Discriminant-based

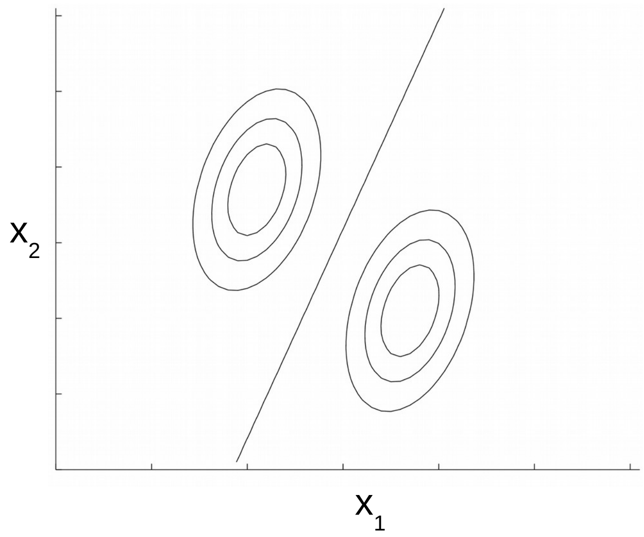
Likelihood-based Classification

- Assume a model for $p(\boldsymbol{x}|C_i)$
- Estimate $p(\boldsymbol{x}|C_i)$ and $p(C_i)$ from the data
- Use Bayes' rule to calculate $P(C_i|\boldsymbol{x})$

  e.g. $g_i(\boldsymbol{x}) = \log P(C_i|\boldsymbol{x})$

Likelihood-based Classification



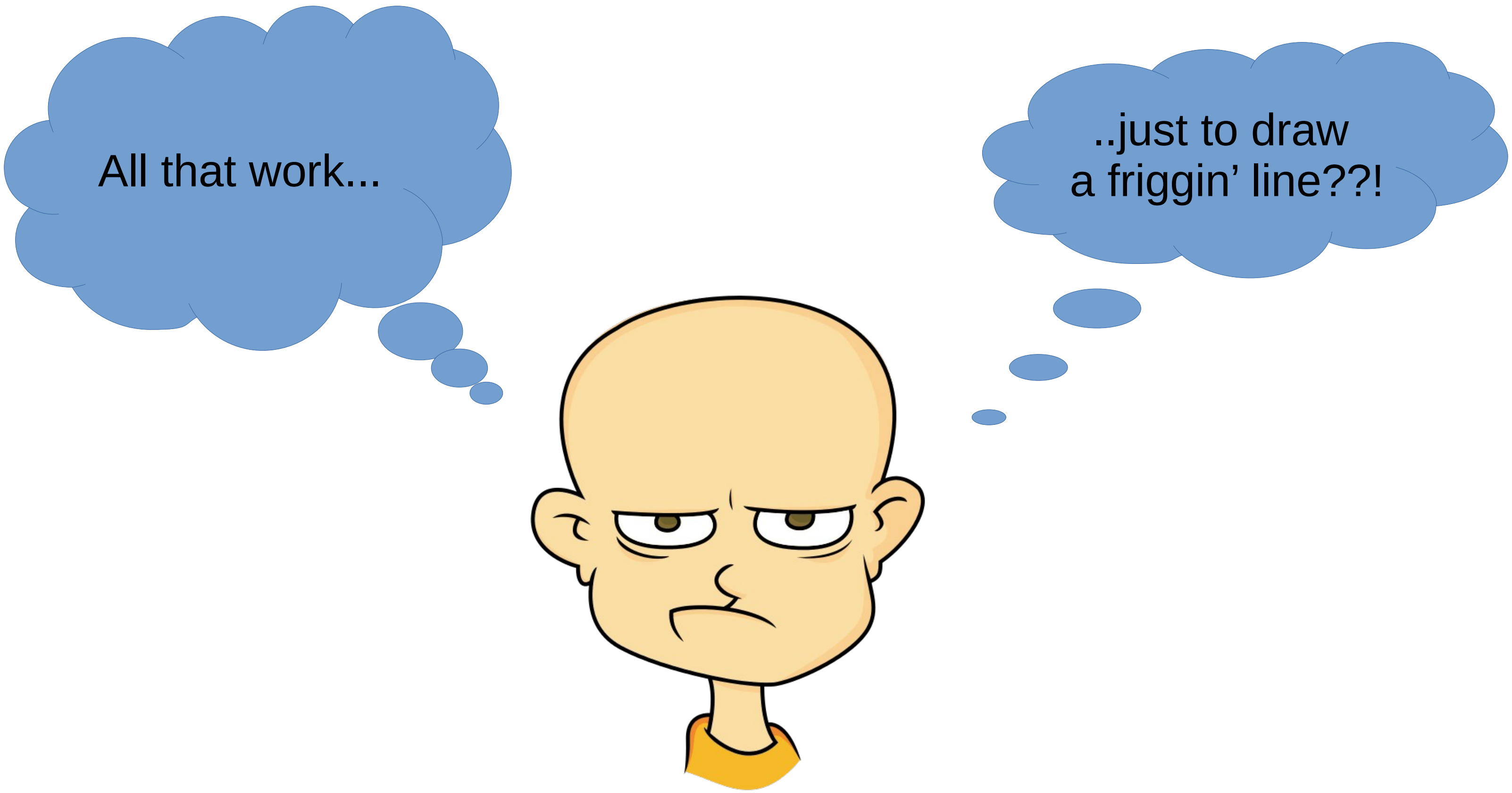


Likelihood-based Classification

- We learn both
  - Class boundaries
    - where $p(\boldsymbol{x}|C_i) = p(\boldsymbol{x}|C_j)$
  - Class densities

- *Generative* approach
- Previous chapters: parametric and non-parametric methods for finding $p(\boldsymbol{x}|C_i)$

Likelihood-based Classification



Clipart-library.com

Discriminant-based Classification

- Learn only class *boundaries*
- Assume a model for $g_i(\boldsymbol{x}|\Phi_i)$, learn this directly
- No density estimation

Discriminant-based Classification

- Estimating the boundaries is enough; no need to accurately estimate the densities inside the boundaries
- A simpler problem than density estimation
- This and following chapters: Discriminant functions

Linear Discriminant

- Define a model $g_i(\mathbf{x}|\Phi_i)$ for each class discriminant
- The *linear discriminant* has the form:

$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0} = \sum_{j=1}^{d} w_{ij} x_j + w_{i0}$$

- Assumption: instances from one class are linearly separable from instances of other classes
- Learning means finding the values of $\{\Phi_i\}$ that optimize separation

Linear Discriminant

- Advantages:
  - Faster to train
  - Simple at test time: O(*d*) space/computation
  - Simple interpretation:
  - Output is weighted sum of attributes
  - Sign of weights: are effects positive/negative
  - Magnitudes of weight: How important

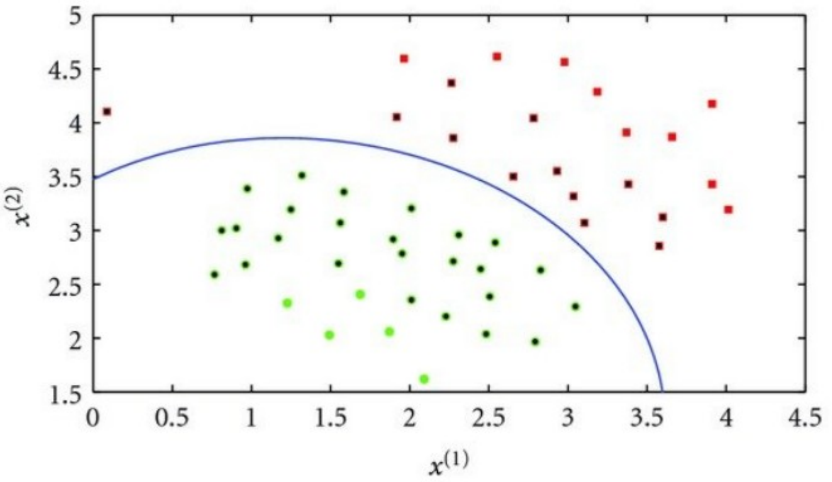
Linear Discriminant

- Optimal when $p(\boldsymbol{x}|C_i)$ are Gaussian with shared covariance matrix
- Useful when classes are (almost) linearly separable
- In many cases, good enough
  - Try linear discrimination before more complex models

Generalizing the Linear Model

- When a linear model (linear in **x**) is not good enough
  - We can add higher order terms
- Example: Quadratic discriminant:

$$g_i(\mathbf{x} \mid \mathbf{W}_i, \mathbf{w}_i, w_{i0}) = \mathbf{x}^T \mathbf{W}_i \mathbf{x} + \mathbf{w}_i^T \mathbf{x} + w_{i0}$$



Generalizing the Linear Model

- Alternative: preprocess input
- Higher-order (product) terms:

$$z_1 = x_1,\ z_2 = x_2,\ z_3 = x_1^2,\ z_4 = x_2^2,\ z_5 = x_1 x_2$$

Map from ***x*** to ***z*** using nonlinear basis functions and use a linear discriminant in ***z***-space

Generalizing the Linear Model

- Linear combination of non-linear functions of **x**
- Discriminant:

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \phi_{ij}(\boldsymbol{x})$$

where $\phi_{ij}(\boldsymbol{x})$ are *basis functions*

Generalizing the Linear Model

- Linear combination of non-linear functions of **x**
- Discriminant:

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \phi_{ij}(\boldsymbol{x})$$

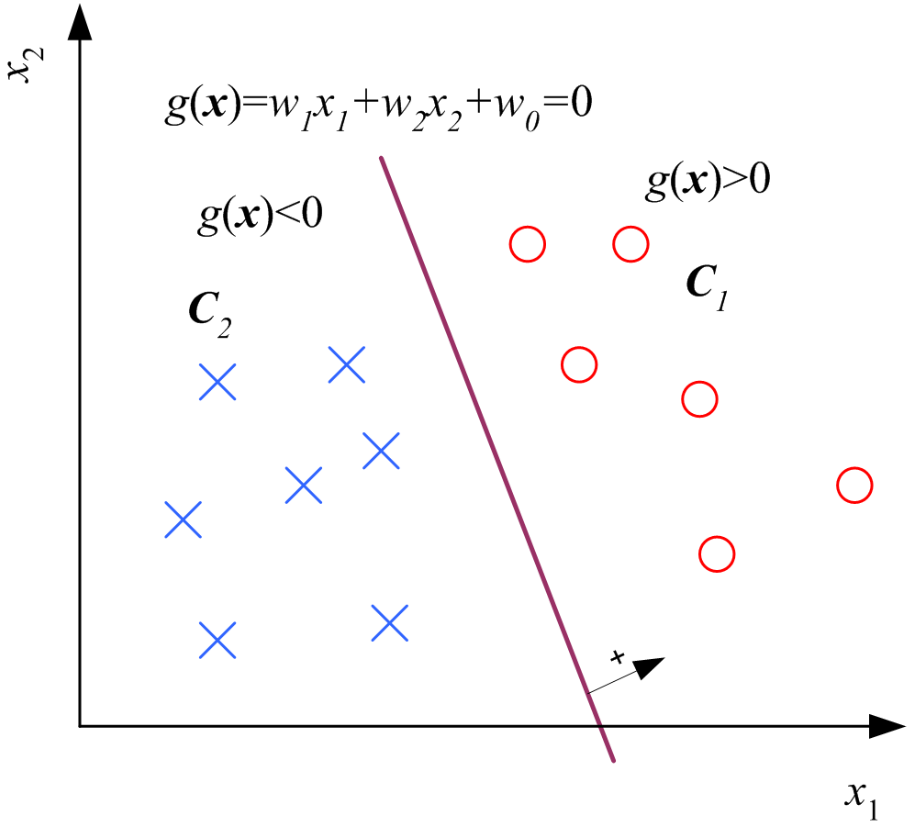
where $\phi_{ij}(\boldsymbol{x})$ are *basis functions*

Basis function examples:

(basis functions are used in later chapters; neural nets and SVMs)

- $\sin(x_1)$
- $\exp(-(x_1 - m)^2/c)$
- $\exp(-\|\boldsymbol{x} - \boldsymbol{m}\|^2/c)$
- $\log(x_2)$
- $1(x_1 > c)$
- $1(ax_1 + bx_2 > c)$

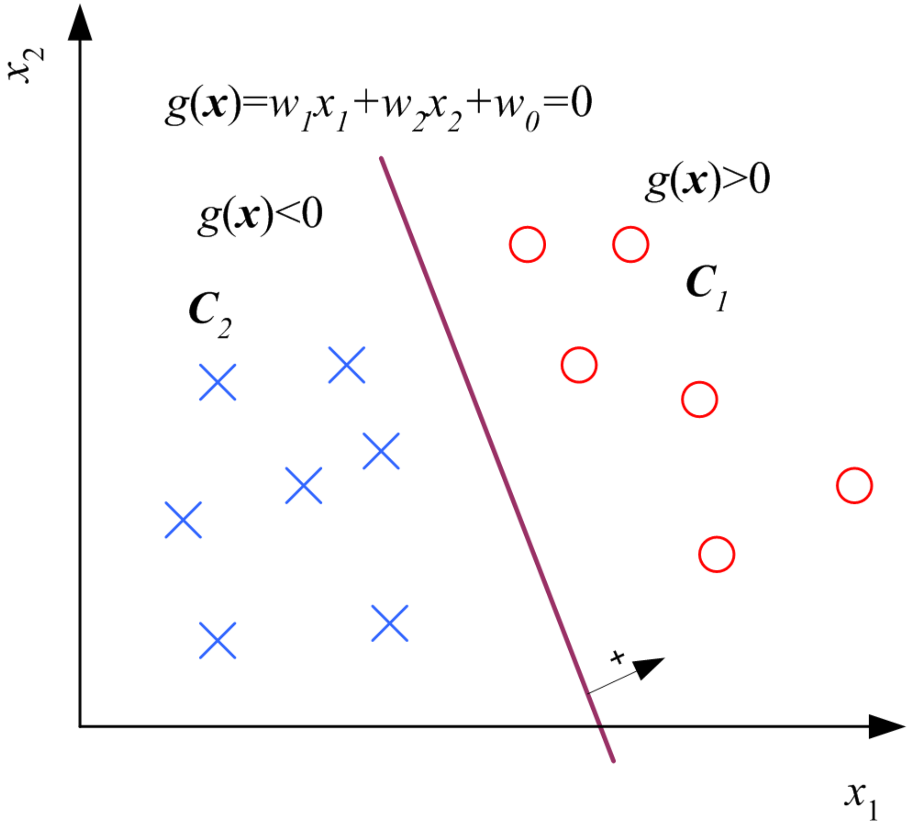
Two Classes



$$g(\boldsymbol{x}) \quad = \quad \boldsymbol{w}^T\boldsymbol{x} + w_0$$

$$\text{choose} \begin{cases} C_1 & \text{if } g(\boldsymbol{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$$

Two Classes

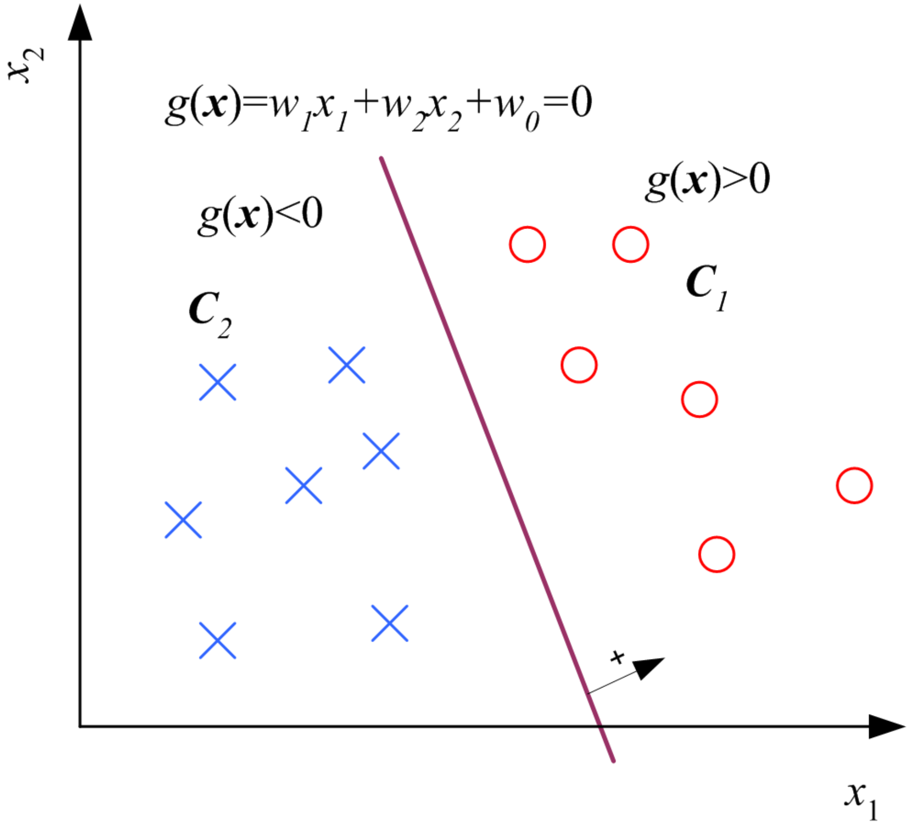


$$g(\mathbf{x}) \quad = \quad \mathbf{w}^T\mathbf{x} + w_0$$

w is orthogonal to the hyperplane

$$\text{The origin } \mathbf{x} = \mathbf{0} \text{ is on the } \begin{cases} \text{positive side} & \text{if } w_0 > 0 \\ \text{boundary} & \text{if } w_0 = 0 \\ \text{negative side} & \text{if } w_0 < 0 \end{cases}$$

Two Classes



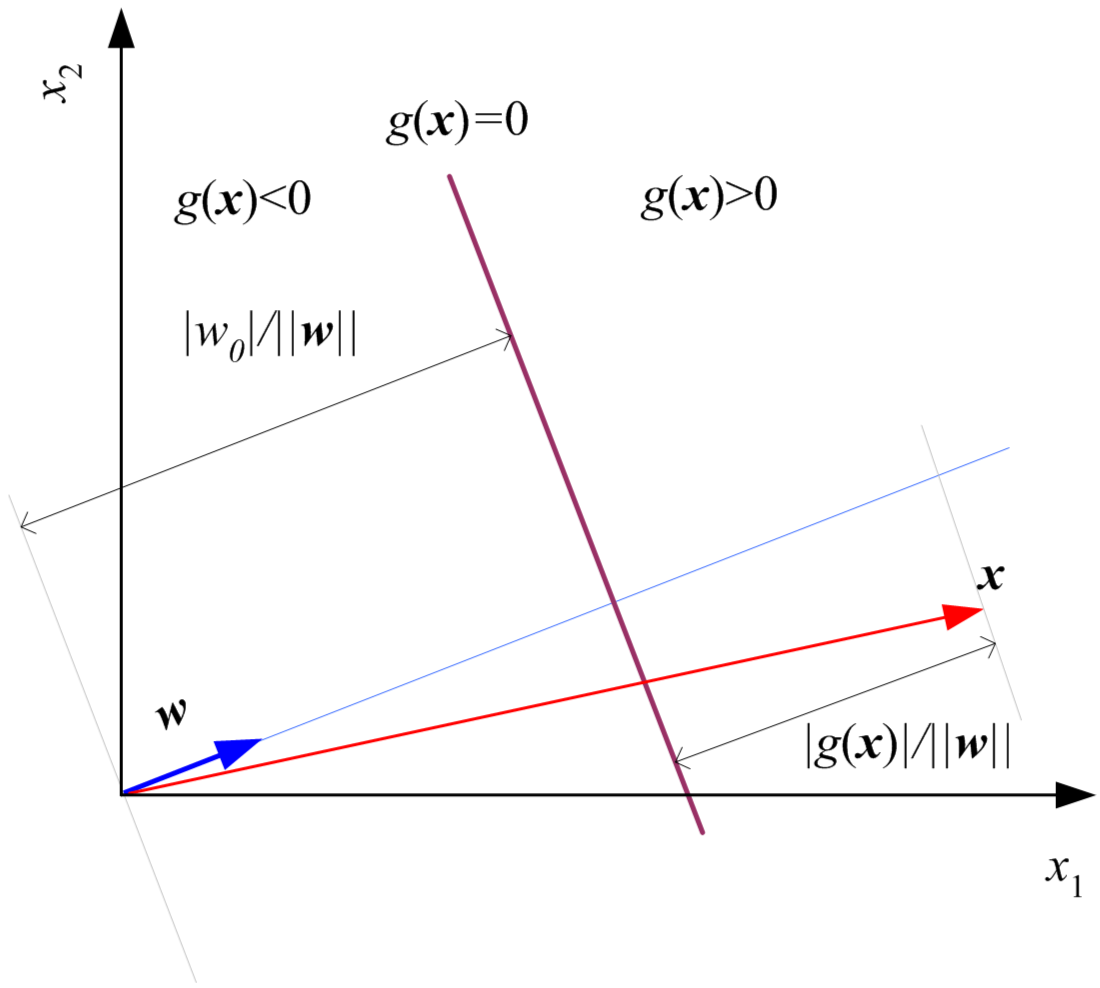
$$g(\boldsymbol{x}) \quad = \quad \boldsymbol{w}^T\boldsymbol{x} + w_0$$

w is orthogonal to the hyperplane
- determines orientation

Determines location wrt origin

$$\text{The origin } \mathbf{x} = \mathbf{0} \text{ is on the } \begin{cases} \text{positive side} & \text{if } w_0 > 0 \\ \text{boundary} & \text{if } w_0 = 0 \\ \text{negative side} & \text{if } w_0 < 0 \end{cases}$$

Geometry



Multiple Classes

- With K > 2 classes
- Two common ways to classify:
  - One-vs-all (linear separability)
  - One-vs-one (pairwise linear separability)

One-vs-all

- K discriminants:

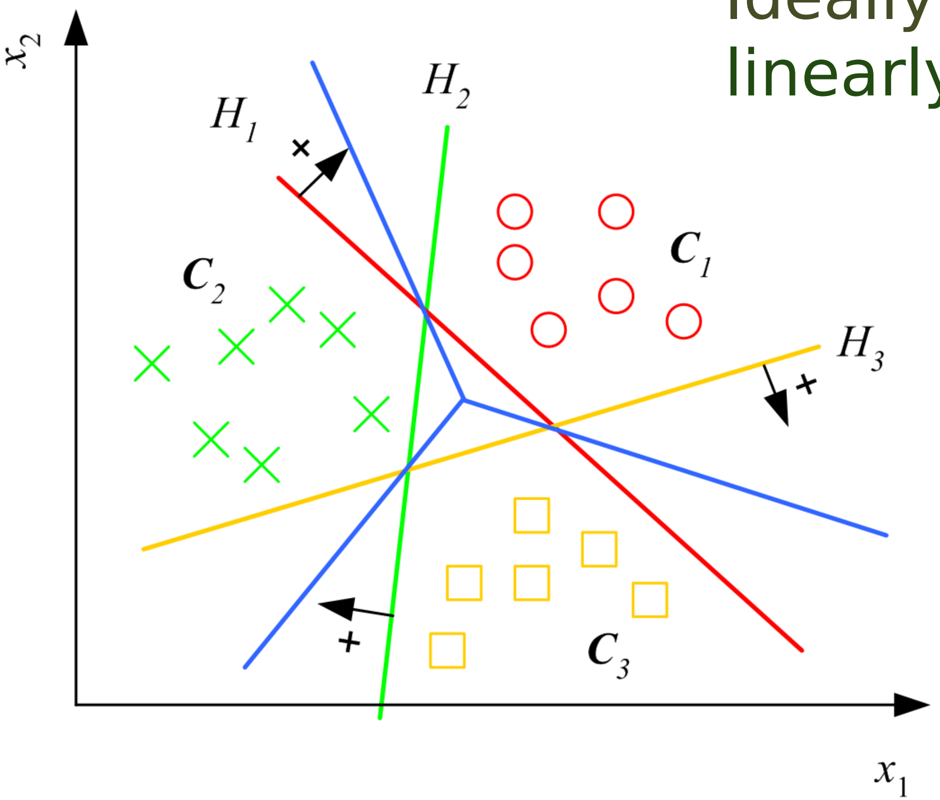
$$g_i(\mathbf{x} \mid \mathbf{w}_i, w_{i0}) = \mathbf{w}_i^T \mathbf{x} + w_{i0}$$

- Training:
  - train each g_i to classify instances of C_i vs all other instances
- Testing:
  - Choose with largest discriminant

Choose C_i if

$$g_i(\mathbf{x}) = \max_{j=1}^{K} g_j(\mathbf{x})$$

One-vs-all



Ideally, classes are completely linearly separable

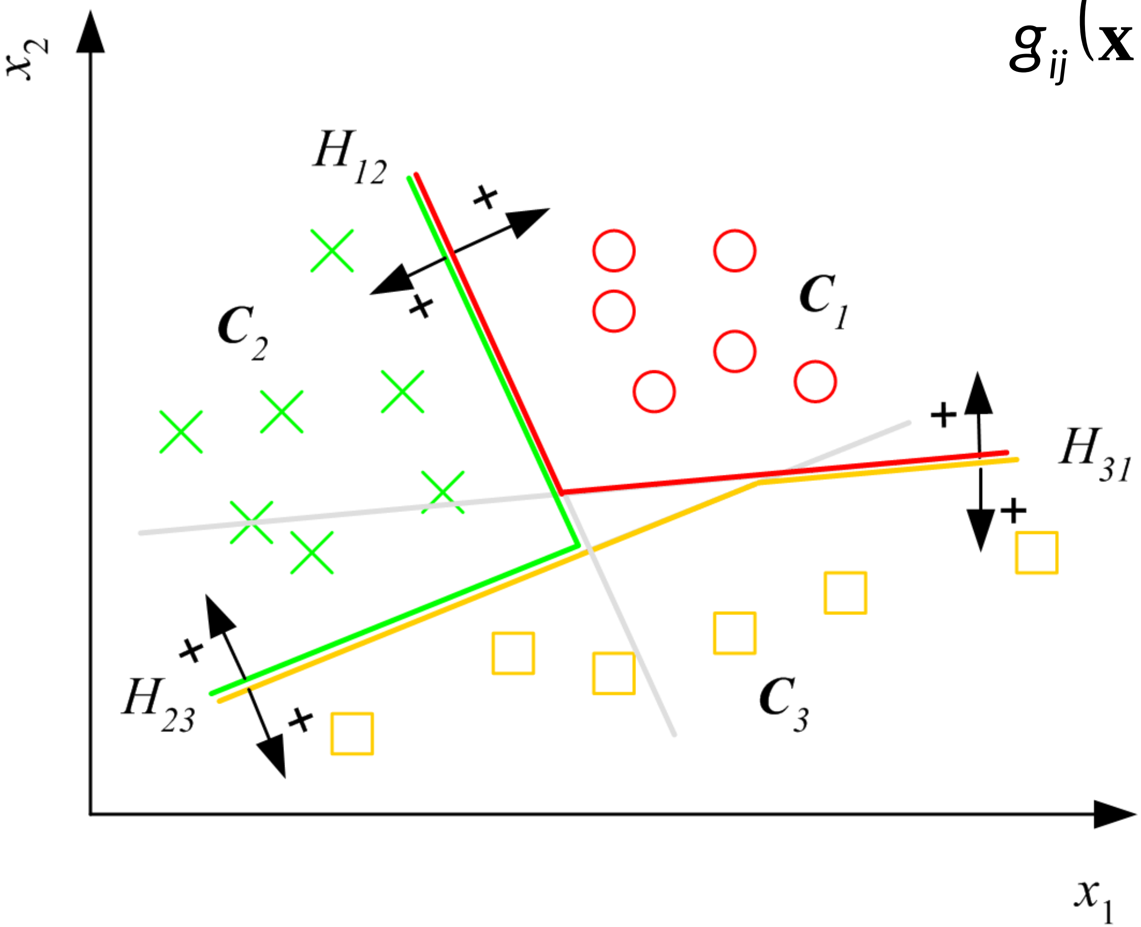
One-vs-one

- K(K-1)/2 discriminants $$g_{ij}\left(\mathbf{x}\,|\,\mathbf{w}_{ij}, w_{ij0}\right) = \mathbf{w}_{ij}^T\mathbf{x} + w_{ij0}$$
  - One for each pair of classes
- Training:
  - train g_{ij} to classify instances of C_i vs instances of C_j
  - ignore instances of other classes
- Testing:
  - choose C_i if $\forall j \neq i, g_{ij}(\boldsymbol{x}) > 0$
  - Or pick the class with largest summed discriminant

$$g_i(\boldsymbol{x}) = \sum_{j \neq i} g_{ij}(\boldsymbol{x})$$

One-vs-one

$$g_{ij}\left(\mathbf{x}\,|\,\mathbf{w}_{ij},w_{ij0}\right)=\mathbf{w}_{ij}^{T}\mathbf{x}+w_{ij0}$$



Both methods divide the input space into *K* convex decision regions

How do we find **w**?

- We know what we want!
  - To find **w**
- But how do we do that?
  - In some cases **w** can be found analytically
  - But often this is not possible: Iterative optimization instead

Gradient Descent

www.mec.ca

Gradient Descent

- There are many different iterative optimization methods
- Gradient Descent is a popular one
  - Simple
  - Reasonably effective
  - But can be very slow

Gradient Descent

- Error function: $E(\boldsymbol{w}|X)$ with parameters $\boldsymbol{w}$ on sample X

- Want to find the optimal **w** (with minimal error):

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w} \mid X)$$

Gradient Descent

- Error function: E(***w***|X) with parameters ***w*** on sample X
- Want to find the optimal **w** (with minimal error):

  w*=arg min$_{\boldsymbol{w}}$ *E*(***w*** | X)

- Begin: Set random **w**
- Iterate:
  - Calculate update term: $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$
  - Set new (updated) w:

$$w_i = w_i + \Delta w_i$$

Gradient Descent

- *Gradient* (vector of partial derivatives)

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d}\right]^T$$

- Learning factor (η):

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$$

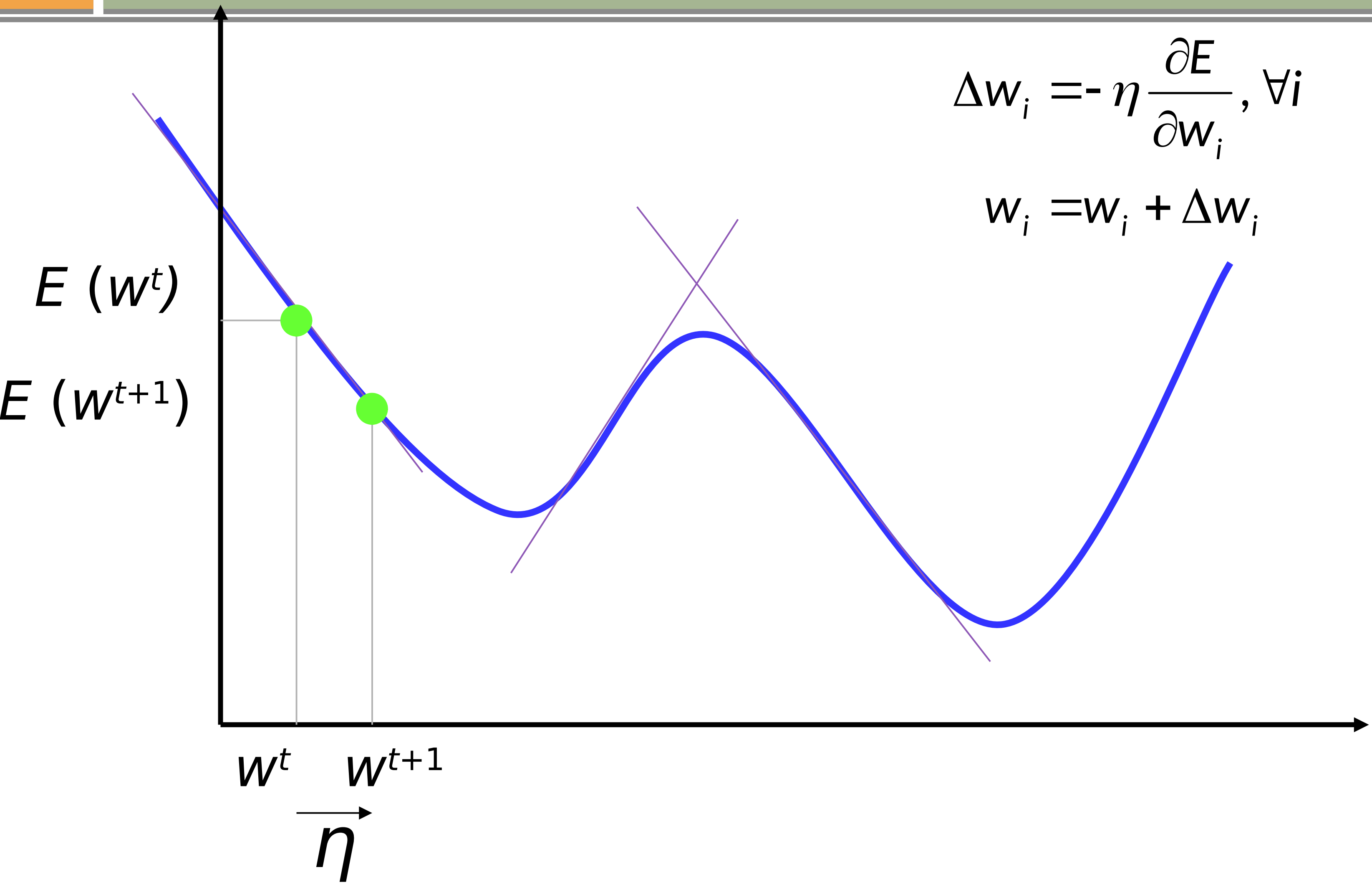
- Stop iterating when:

$$\nabla E(\mathbf{w}) \approx \mathbf{0}$$

Gradient-Descent

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i$$

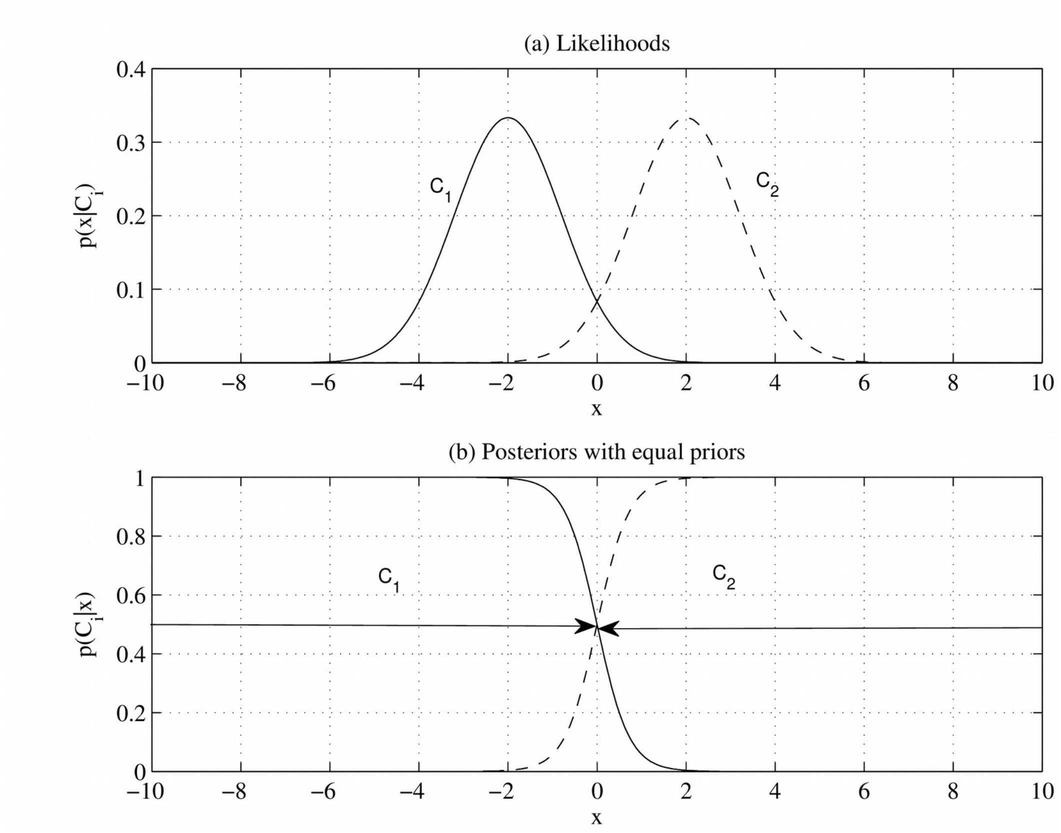
$$w_i = w_i + \Delta w_i$$



Logistic Regression

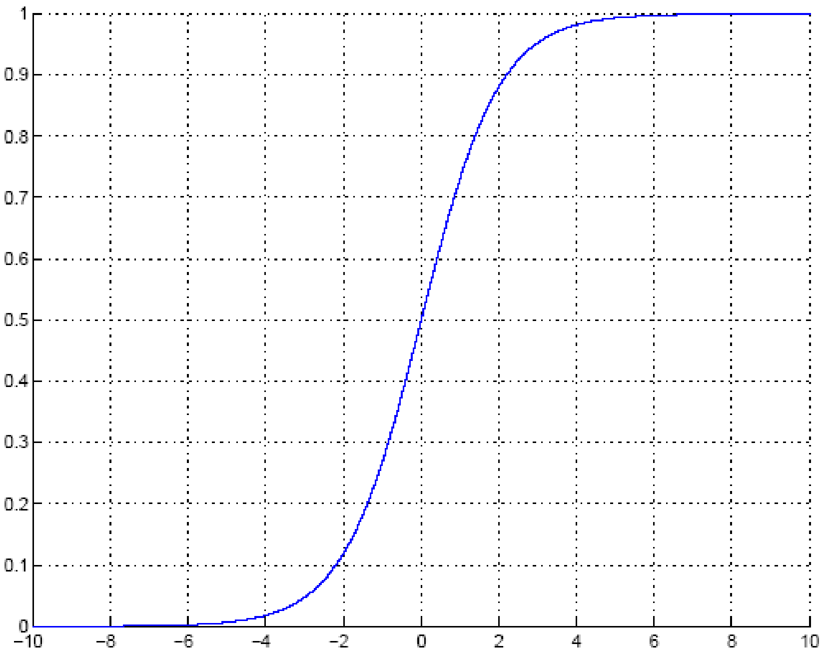
Image via flickr, CC

Previous chapters: Sigmoid posterior



Instead of first estimating densities, then calculating posteriors:
We want to model the posterior directly using the logistic function

Sigmoid (Logistic) Function



Transform a linear discriminant to a posterior probability:

Calculate $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ and choose C_1 if $g(\mathbf{x}) > 0$, or

Calculate $y = \text{sigmoid}(\mathbf{w}^T\mathbf{x} + w_0)$ and choose C_1 if $y > 0.5$

Logistic Discrimination

Why does the logistic approach work?

We know that:

- For Gaussian class densities with equal covariance: $$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0}$$
- Parameters can be calculated analytically:

$$\begin{aligned} \mathbf{w}_i &= \mathbf{\Sigma}^{-1}\boldsymbol{\mu}_i \\ w_{i0} &= -\frac{1}{2}\boldsymbol{\mu}_i^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_i + \log P(C_i) \end{aligned}$$

Logistic Discrimination

- Log odds is linear (again for Gaussian with shared cov):

$$
\begin{aligned}
\operatorname{logit}(P(C_1|\boldsymbol{x})) = \log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} &= \log \frac{P(C_1|\boldsymbol{x})}{P(C_2|\boldsymbol{x})} \\
&= \log \frac{p(\boldsymbol{x}|C_1)}{p(\boldsymbol{x}|C_2)} + \log \frac{P(C_1)}{P(C_2)} \\
&= \log \frac{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2}\exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_1)]}{(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2}\exp[-(1/2)(\boldsymbol{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_2)]} + \log \frac{P(C_1)}{P(C_2)} \\
&= \boldsymbol{w}^T\boldsymbol{x} + w_0
\end{aligned}
$$

where

$$
\begin{aligned}
\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\
w_0 &= -\frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \log \frac{P(C_1)}{P(C_2)}
\end{aligned}
$$

Logistic Discrimination

- So the Log-Odds (logit($P(C_1|\mathbf{x})$)) is linear:

$$\log \frac{P(C_1|\boldsymbol{x})}{1 - P(C_1|\boldsymbol{x})} = \boldsymbol{w}^T\boldsymbol{x} + w_0$$

- The *inverse* of the log-odds is the logistic function
- Which means that:

$$P(C_1|\boldsymbol{x}) = \mathrm{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1 + \exp\left[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)\right]}$$

Logistic Discrimination

- So the posterior can be described by a sigmoid:

$$P(C_1|\boldsymbol{x}) = \mathrm{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1 + \exp\left[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)\right]}$$

- We showed this in the case of two Gaussian class densities with shared cov
- It also holds in some other cases

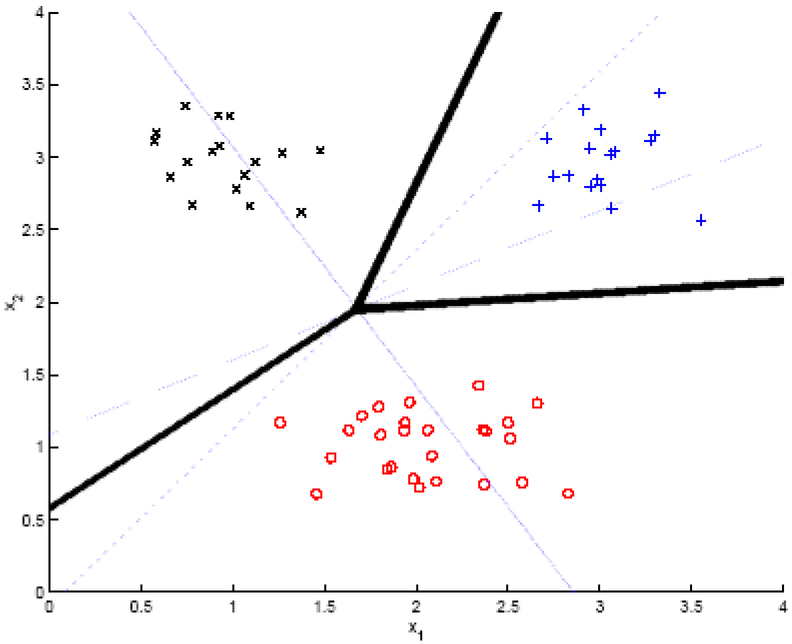
K>2 classes, softmax

- When there are more than two classes: *Softmax*

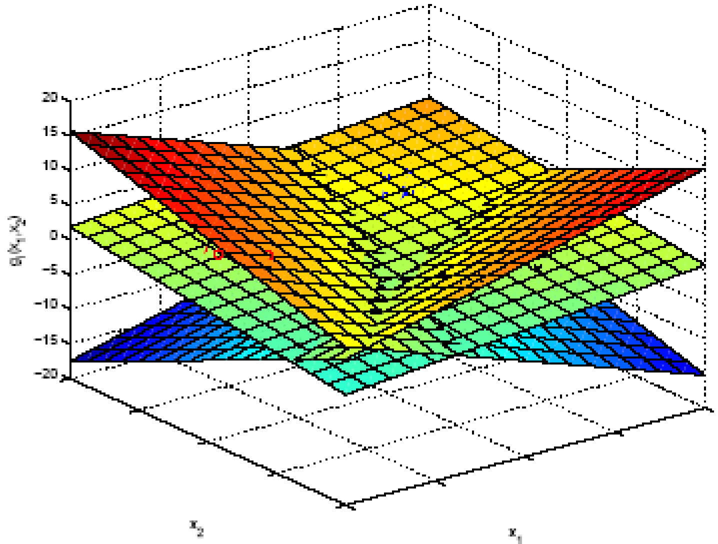
$$y_i = \hat{P}(C_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}]}{\sum_{j=1}^K \exp[\boldsymbol{w}_j^T\boldsymbol{x} + w_{j0}]}, \quad i = 1, \ldots, K$$

- If the 'linear output' for one class (C_k) is much larger than for the others:
  - $P(C_k|\mathbf{x})$ will be close to one
  - Close to 0 for other classes
  - Together they all sum to 1

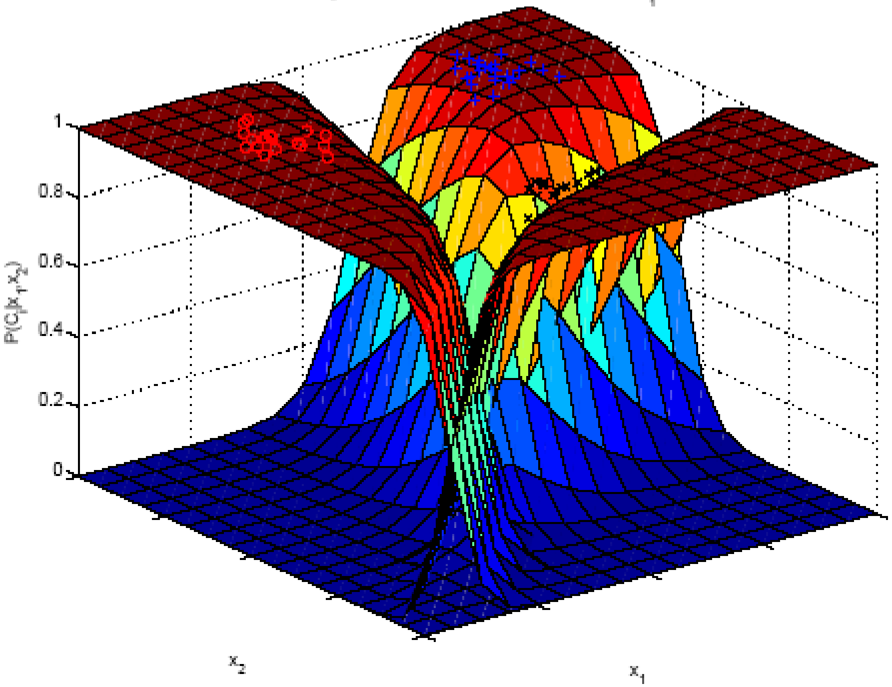
Examples



0
0.5
1
1.5
2
2.5
3
3.5
4
x_1
x_2



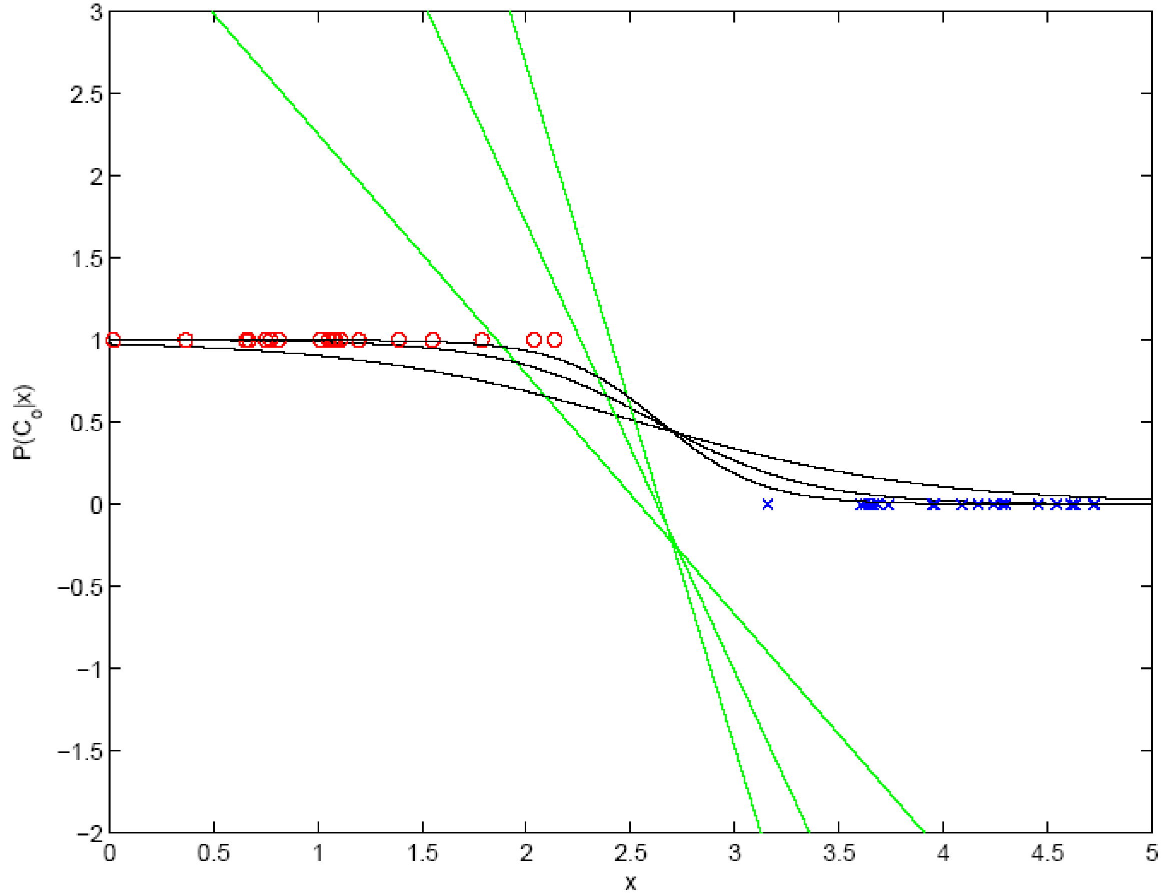
20
15
10
5
0
−5
−10
−15
−20
g_i(x_1,x_2)
x_2
x_1



1
0.8
0.6
0.4
0.2
0
P(C_i|x_1,x_2)
x_2
x_1

Training a logistic classifier

How do we find the correct parameters?



Univariate example

Training a logistic classifier

K= 2: We have a training set with input data and labels
e.g [house price, size, ….] → {sold, not sold}

We want to learn the parameters for

$$P(C_1|\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)]}$$

Training a logistic classifier

K= 2: We have a training set with input data and labels e.g [house price, size, ….] → {sold, not sold}

We want to learn the parameters for

$$P(C_1|\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1 + \exp[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)]}$$

There are two ways of doing that:

- Model the outcome as a Bernoulli distribution. Find the Maximum Likelihood parameter estimates.
- Use least-square regression, considering the labels {0,1} real values.

Training a logistic classifier

K= 2: We have a training set with input data and labels e.g [house price, size, ….] → {sold, not sold}

We want to learn the parameters for

$$P(C_1|\boldsymbol{x}) = \mathrm{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + w_0) = \frac{1}{1 + \exp\left[-(\boldsymbol{w}^T\boldsymbol{x} + w_0)\right]}$$

There are two ways of doing that:

- Model the outcome as a Bernoulli distribution. Find the Maximum Likelihood parameter estimates.
- Use least-square regression, considering the labels {0,1} real values.

In both cases: No closed-form solution. Need Gradient Descent

The ML & Bernoulli approach

- Labels are a Bernoulli distribution
- $$r^t|\mathbf{x}^t \sim \text{Bernoulli}(y^t)$$
- Sample likelihood is
- $$l(\mathbf{w}, w_0|\mathcal{X}) = \prod_t (y^t)^{(r^t)}(1-y^t)^{(1-r^t)}$$
- Turn this into an error function to minimize
- $$E(\mathbf{w}, w_0|\mathcal{X}) = -\sum_t r^t \log y^t + (1-r^t)\log(1-y^t)$$
- (this is *Cross Entropy*)

The ML & Bernoulli approach

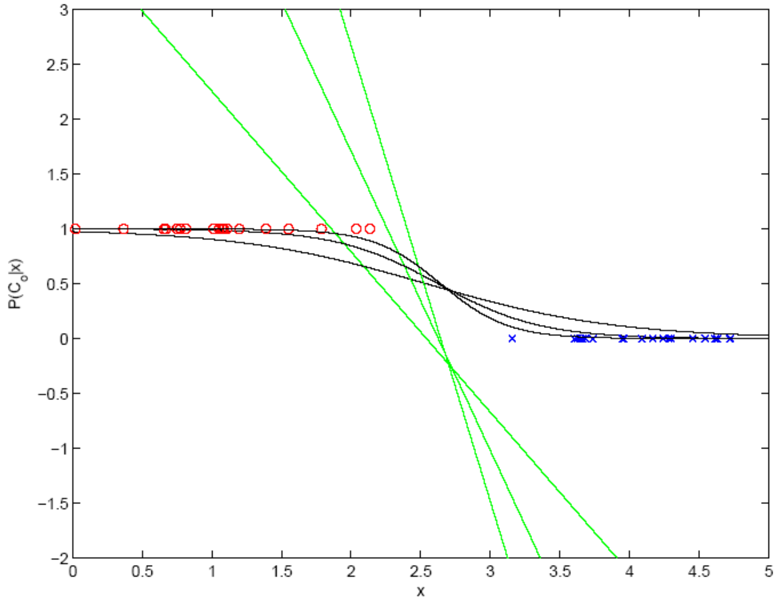
- To do Gradient Descent we need the derivative $$\frac{dy}{da} = y(1-y)$$

- This gives update equations for the descent:

$$\begin{aligned}
\Delta w_j &= -\eta \frac{\partial E}{\partial w_j} = \eta \sum_t \left(\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t} \right) y^t(1-y^t) x_j^t \\
&= \eta \sum_t (r^t - y^t) x_j^t, j = 1, \ldots, d \\
\Delta w_0 &= -\eta \frac{\partial E}{\partial w_0} = \eta \sum_t (r^t - y^t)
\end{aligned}$$

The ML & Bernoulli approach

- Initialize with weights near 0
- Stop when all samples are classified correctly
    - or even before (regularization)



K>2 Classes

- Similar, but labels are modelled as multinomial instead of Bernoulli
- We have the softmax instead of a single logistic function

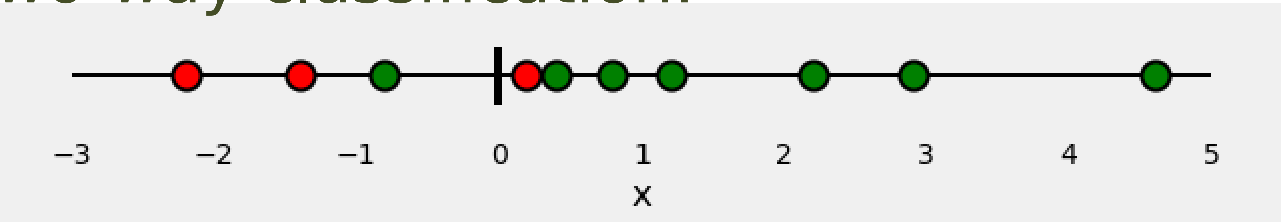
$$y_i = \hat{P}(C_i|\boldsymbol{x}) = \frac{\exp[\boldsymbol{w}_i^T\boldsymbol{x} + w_{i0}]}{\sum_{j=1}^{K}\exp[\boldsymbol{w}_j^T\boldsymbol{x} + w_{j0}]},\ i = 1,\ldots,K$$

Training by least-squares

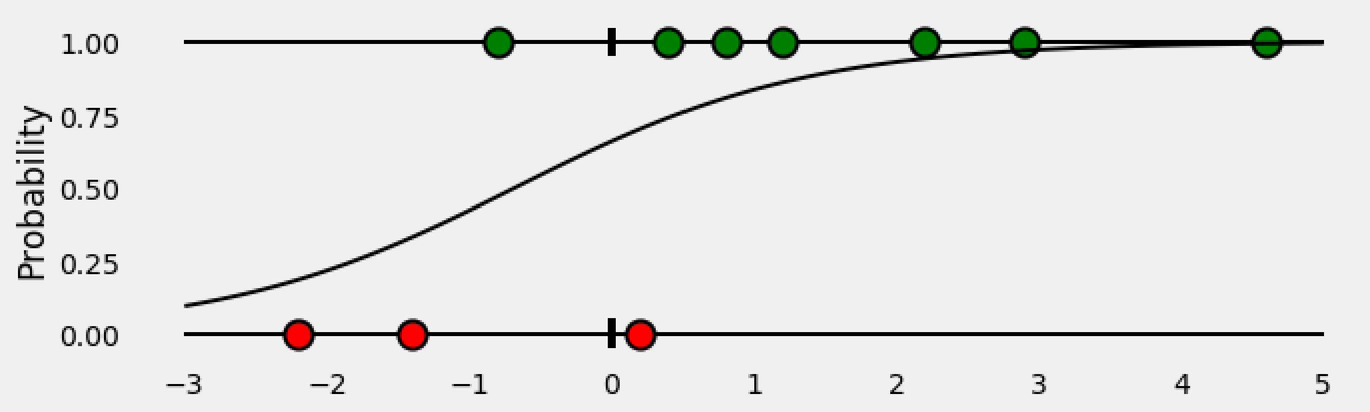
- Another way to train the logistic classifier: Least squares regression
- “Discrimination by regression”
- Can be useful when classes are not mutually exclusive and exhaustive

Training by least-squares

- Two-way classification:



- Use the class codes {0,1} as numeric target values for regression
- Fit to the logistic function by minimizing sum-of-squares



towardsdatascience.com

Training by least-squares

- Function to fit:

$$y^t = \text{sigmoid}(\mathbf{w}^T\mathbf{x}^t + w_0) = \frac{1}{1 + \exp[-(\mathbf{w}^T\mathbf{x}^t + w_0)]}$$

- Fit by minimizing sum of square Error:

$$E(\mathbf{w}, w_0|\mathcal{X}) = \frac{1}{2}\sum_t (r^t - y^t)^2$$

Training by least-squares

- Update equations for gradient descent

$$\Delta \boldsymbol{w} = \eta \sum_t (r^t - y^t) y^t (1 - y^t) \boldsymbol{x}^t$$

$$\Delta w_0 = \eta \sum_t (r^t - y^t) y^t (1 - y^t)$$

Training by least-squares

- Can also be used for K>2 classes
- A separate logistic function is fit for each class
- The resulting class probabilities P(C|x) don't necessarily sum to one!!
  - So x may be predicted to belong to more than one class, or none at all

Sources & resources

- Linear and GLM-intro:

  byclb.com/TR/Tutorials/neural_networks/ch9_1.htm
- Math animations (esp. linear algebra & calculus)

  youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw
- Various stats topics explained

  youtube.com/user/joshstarmer
- Machine learning lectures, Cornell

  youtube.com/playlist?list=PLl8OlHZGYOQ7bkVbuRthEsaLr7bONzbXS