University of Amsterdam
Informatics Institute

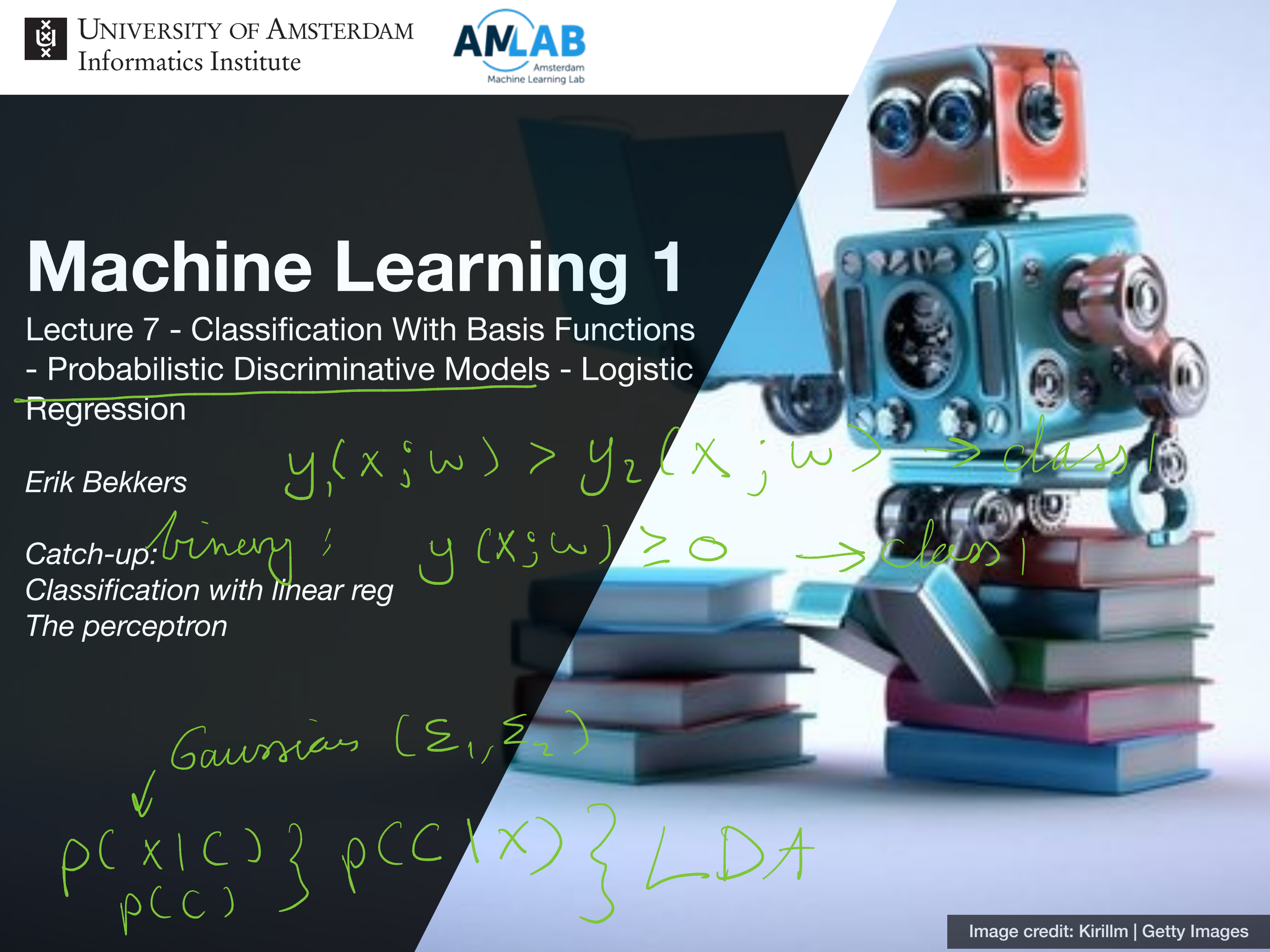
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 7 - Classification With Basis Functions - Probabilistic Discriminative Models - Logistic Regression

Erik Bekkers

Catch-up:
Classification with linear reg
The perceptron



University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 6.4 - Supervised Learning
Classification? - Discriminative Models - Least Squares Regression

Erik Bekkers

(Bishop 4.1.3)

Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

Least Squares for Classification

- Each class C_k has its own linear model:

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

- Shorter notation: $y(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$

  - Matrix $\tilde{\mathbf{W}}$, column k contains: $\tilde{w}_k = \text{Concat}(w_{k0}, \mathbf{w}_k) \in \mathbb{R}^M$

  - Input vector: $\tilde{\mathbf{x}} = \text{Concat}(1, \mathbf{x}) \in \mathbb{R}^M$

  - Output/prediction vector: $\mathbf{y}(\mathbf{x}) = \begin{pmatrix} y_1(\mathbf{x}) \\ y_2(\mathbf{x}) \\ \vdots \\ y_K(\mathbf{x}) \end{pmatrix} = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}} \in \mathbb{R}^K$

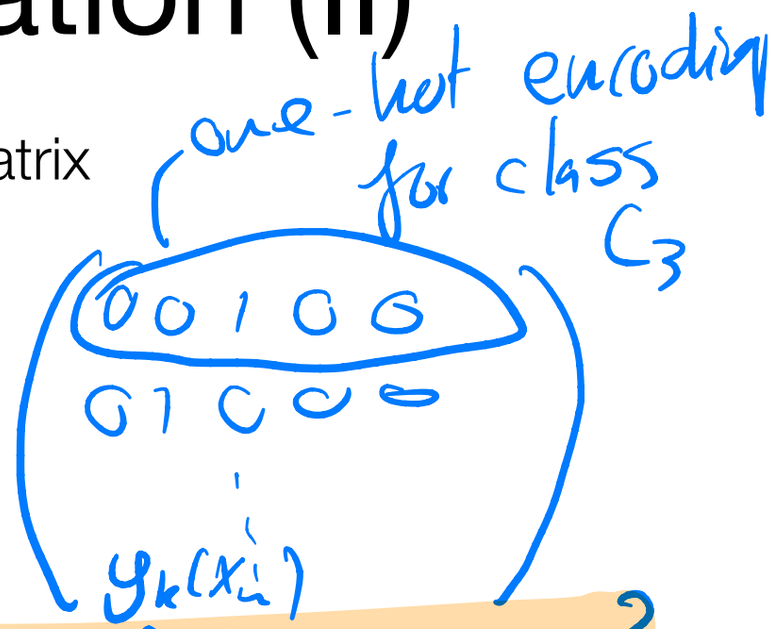
- **Classification**: Assign $\mathbf{x}$ to class C_K if $k = \underset{j}{\text{argmax}}\ y_j(\mathbf{x})$

Discriminative linear model

Least Squares for Classification (II)

- Data set: $N \times (D+1)$ data matrix $N \times K$ target matrix

$$\tilde{X} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{pmatrix} \qquad T = \begin{pmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{pmatrix}$$



- Use sum-of-squares regression error function

$$\sum_n \sum_k \left(\underline{w}_k^T \tilde{x}_n - t_{nk} \right)^2$$

$$E_D(\tilde{\mathbf{W}}) = \frac{1}{2} \mathrm{Tr}\left[(\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T})^T (\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T}) \right] = \frac{1}{2} \sum_n \sum_k \left(\sum_m \tilde{X}_{nm} \tilde{W}_{mk} - t_{nk} \right)^2$$

- Minimize $E_D(\tilde{\mathbf{W}})$ as a function of $\tilde{\mathbf{W}}$: $\frac{\partial E}{\partial W} = 0$ → solve
- Solution: $\tilde{\mathbf{W}}_{\mathrm{LS}} = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{T} = \tilde{\mathbf{X}}^\dagger \mathbf{T}$
- Discriminant function: $\mathbf{y}_{\mathrm{LS}}(\mathbf{x}) = \tilde{W}_{LS}^T \tilde{x} \in \mathbb{R}^K$

$\mathrm{error}_i \cdot \mathrm{error}_i = \mathrm{error}_i^2$

Matrix contains all products $\mathrm{error}_i \cdot \mathrm{error}_j$ but we are only interested in the trace (sum of diagonal) which is $\sum_k \mathrm{error}_k^2$

for a matrix M with elements m_{ij}, i.e., $M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & \cdots \\ m_{21} & m_{22} & m_{23} & \cdots \\ m_{31} & m_{32} & m_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$

the trace $\mathrm{Tr}[M]$ sums over the diagonal.

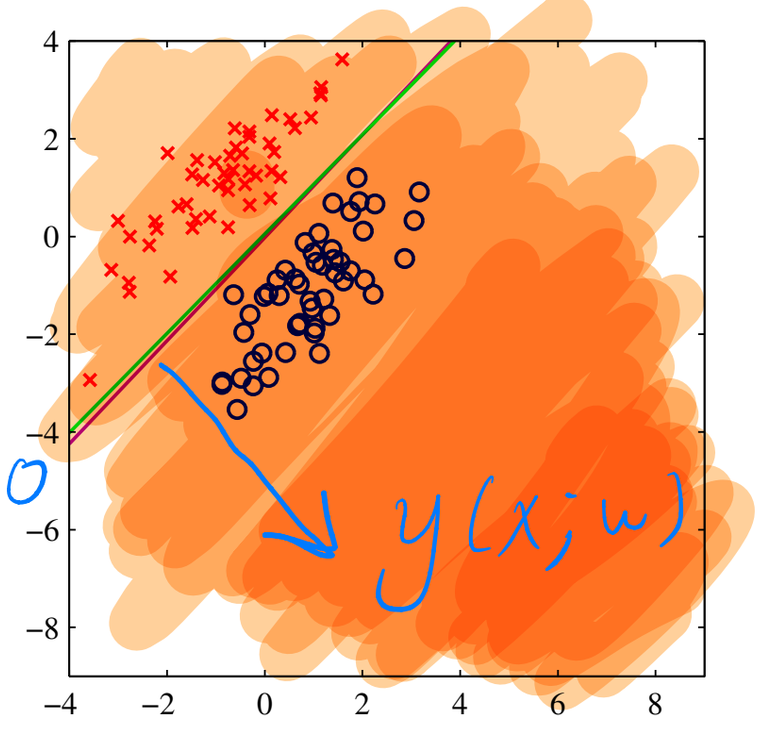
On the previous slide the matrix is $K \times K$ and contains the errors

$$m_{ij} = \sum_n (y_{ni} - t_{ni})(t_{nj} - t_{nj})$$

whereas we're only interested in the diagonal

$$m_{kk} = \sum_n (y_{nk} - t_{nk})^2$$

Least Squares for Classification: Problems



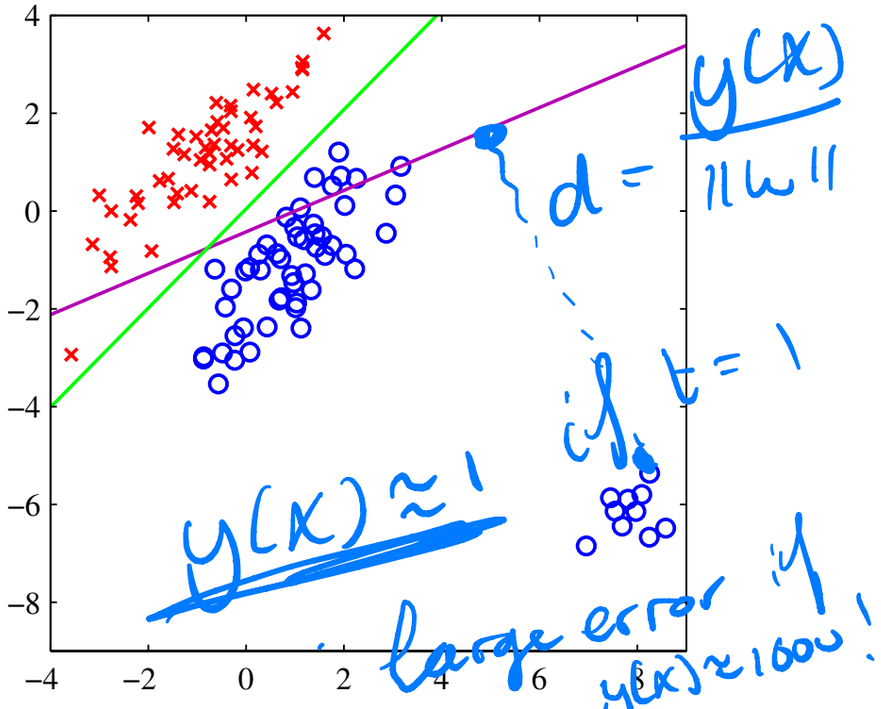


Figure: least squares is very sensitive to outliers (Bishop 4.4)

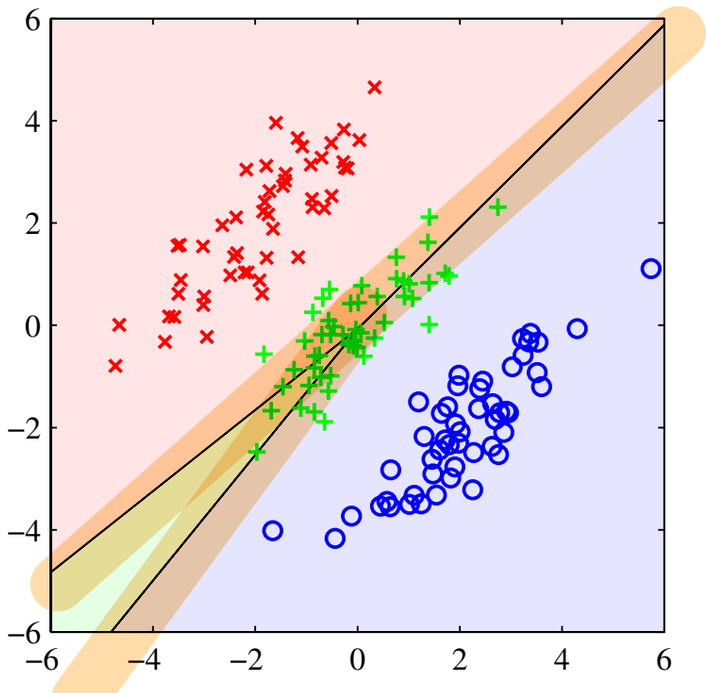


Figure: masking for least squares for K>2 (Bishop 4.5)

1. The decision boundaries are very sensitive to outliers
2. For $K > 2$ some decision regions can become very small or are even completely ignored
3. The components of $\mathbf{y}_{\mathrm{LS}}(\mathbf{x})$ are not probabilities!
   - $y_k(\mathbf{x})$
   - if $\sum_{k=1}^{K} t_k = 1$

University of Amsterdam
Informatics Institute

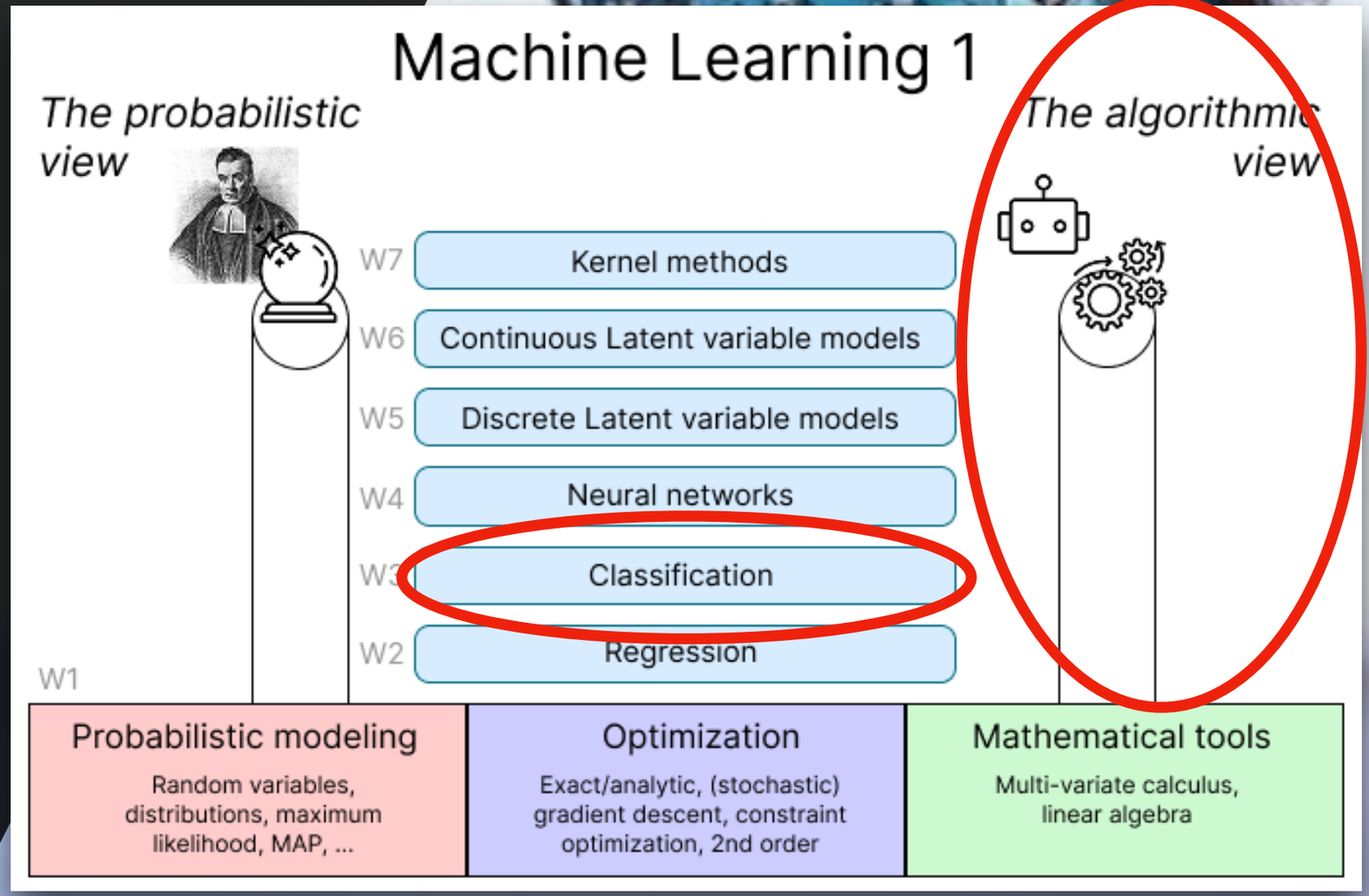
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 6.5 - Supervised Learning
Classification - Discriminative Models -
The Perceptron

Erik Bekkers

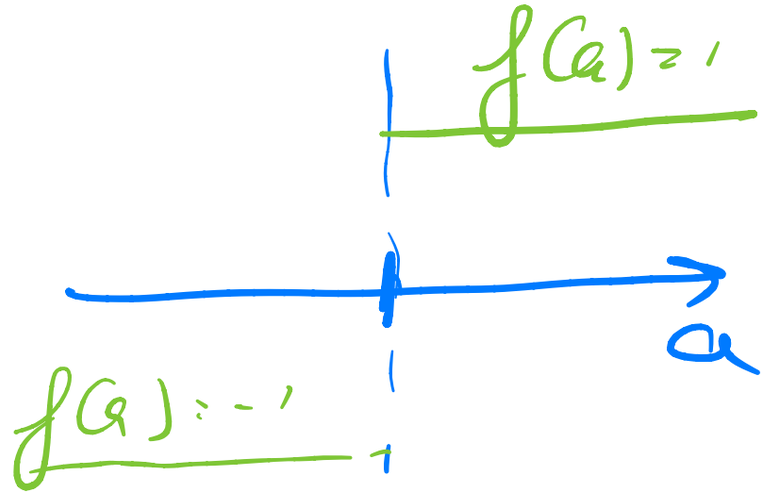
(Bishop 4.1.7)



Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

The Perceptron Algorithm



- Input: $\mathbf{x} \in \mathbb{R}^D$
- Targets: $t \in \{C_1, C_2\} = \{1, -1\}$
- Prediction: $y(\mathbf{x}) = f(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$, with $f(a) = \begin{cases} 1 & , a \geq 0 \\ -1 & , a < 0 \end{cases}$
- Class decisions: assign $\mathbf{x}$ to class C_1 if $\mathbf{w}^T \boldsymbol{\phi}_n \geq 0$

  (and to C_2 if $\mathbf{w}^T \boldsymbol{\phi} < 0$)

- For correct classification: find $\mathbf{w} \neq 0$ such that for all $(\mathbf{x}_n, t_n)$:

  $\mathbf{w}^T \boldsymbol{\phi}_n \cdot t_n \geq 0$

  both positive: +
  both negative: +
  otherwise: −

- Perceptron criterion: $E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) t_n$

  with $\mathcal{M} = \{n : \mathbf{w}^T \boldsymbol{\phi}_n t_n < 0\}$ ← set of incorrectly classified points

Perceptron: Stochastic Gradient Descent

- $E_P(\mathbf{w}) = -\sum_{n \in \mathcal{M}} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) t_n$

$$= -\sum_{n \in \mathcal{M}} E_n(\mathbf{w})$$

- Stochastic Gradient Descent (SGD). For each misclassified $\mathbf{x}_n$:

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

$$= \mathbf{w}^{(\tau)} + \eta \boldsymbol{\phi}_n t_n$$

(after each update we recompute the set of incorrect predictions and randomly select one point)

- If $\mathbf{X}$ is linearly separable, then perceptron SGD will converge

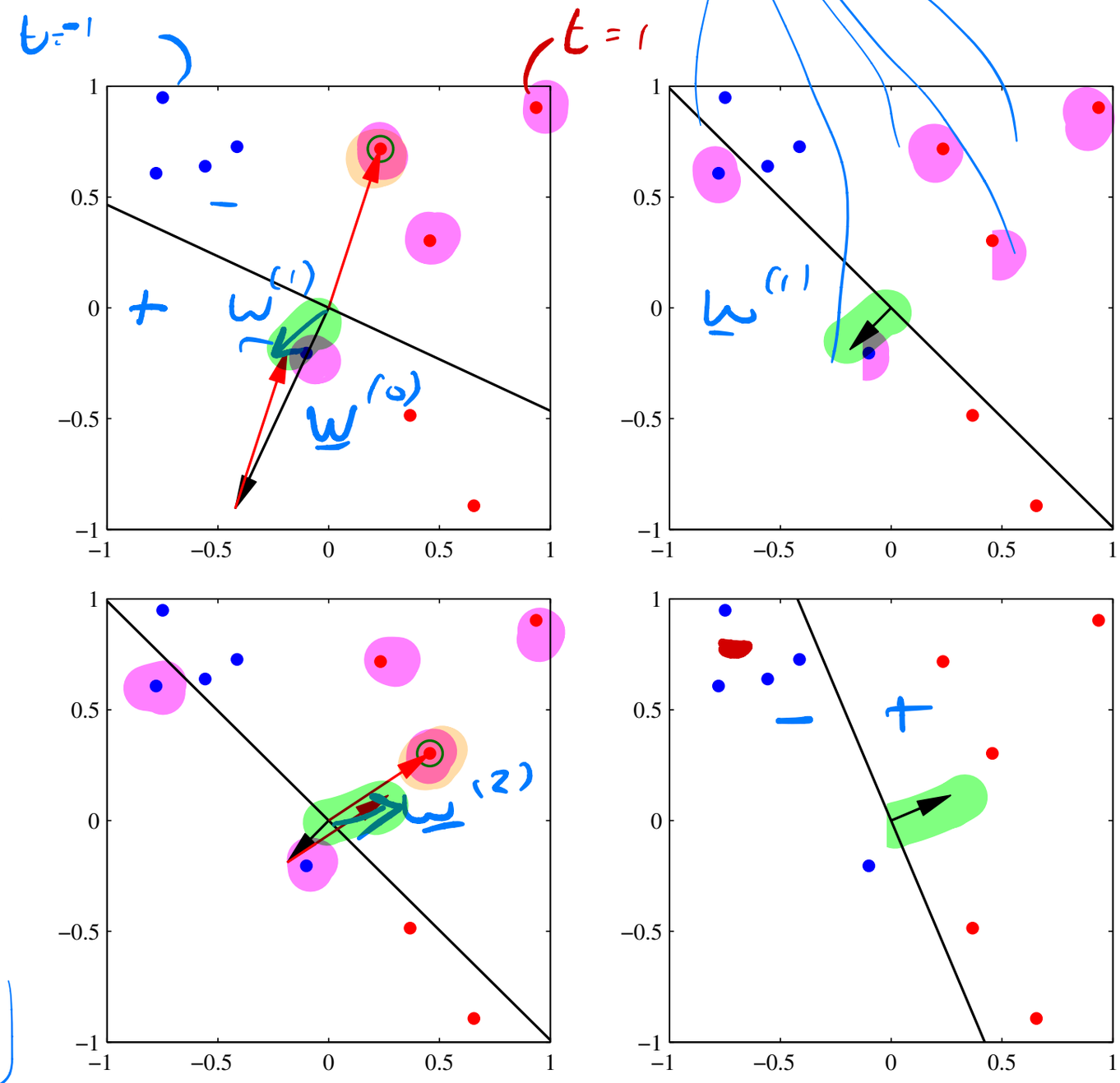


Figure: SGD for perceptron criterion (Bishop 4.7)
for $\mathbf{x}_n$ in C_1: add $\boldsymbol{\phi}(\mathbf{x}_n)$ to $\mathbf{w}$
for $\mathbf{x}_n$ in C_2: subtract $\boldsymbol{\phi}(\mathbf{x}_n)$ from $\mathbf{w}$.

Problems: Perceptron

- Perceptron only works for 2 classes

- There might be many solutions depending on the initialization of $\mathbf{w}$ and on the order in which data is presented in SGD

- If dataset is not linearly separable, the perceptron algorithm will not converge.

- Based on linear combination of fixed basis functions.

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 7.1 - Supervised Learning Classification - Classification With Basis Functions

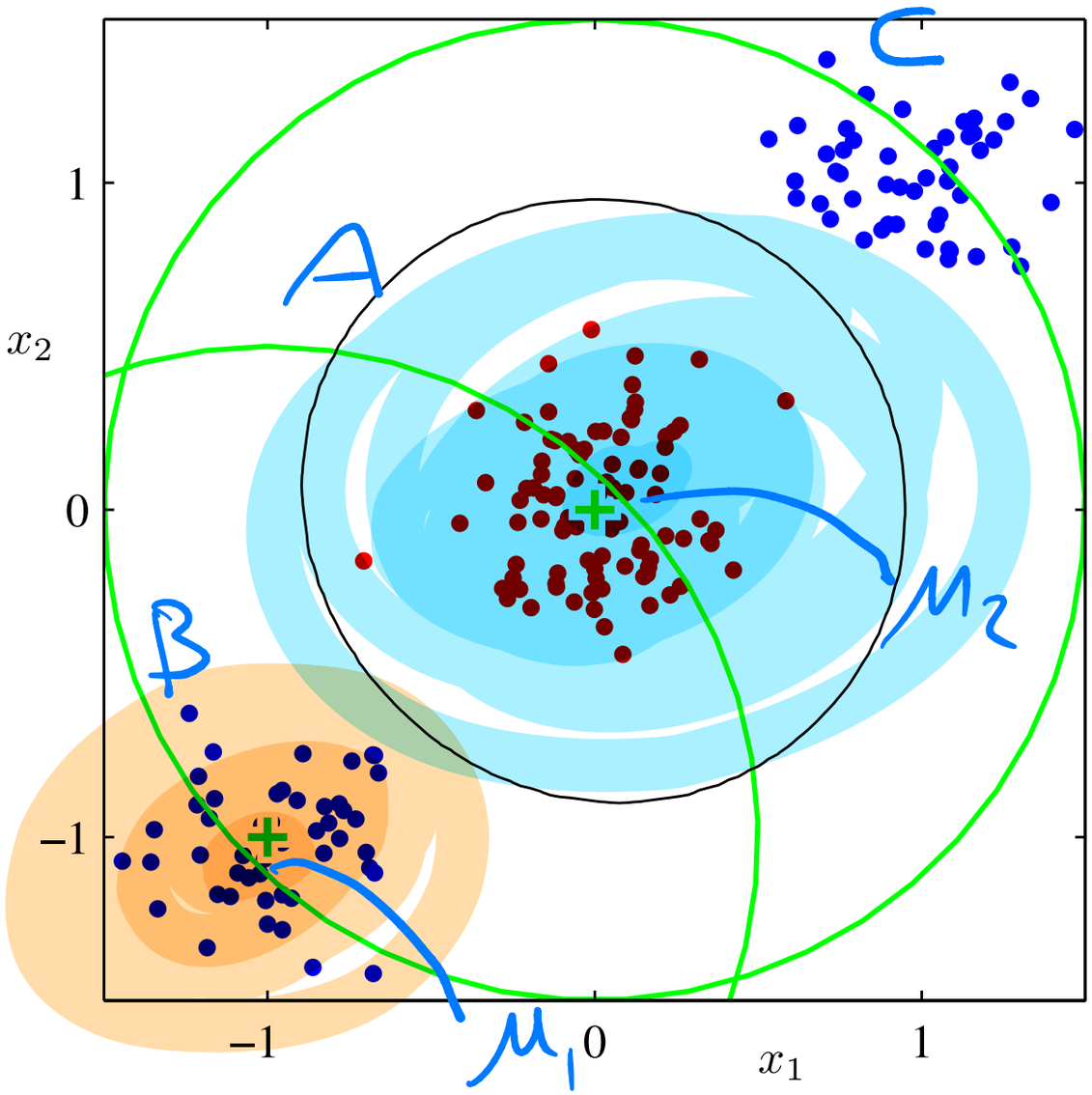
Erik Bekkers

(Bishop 4.3.1)

Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

Example: Use of Basis Functions



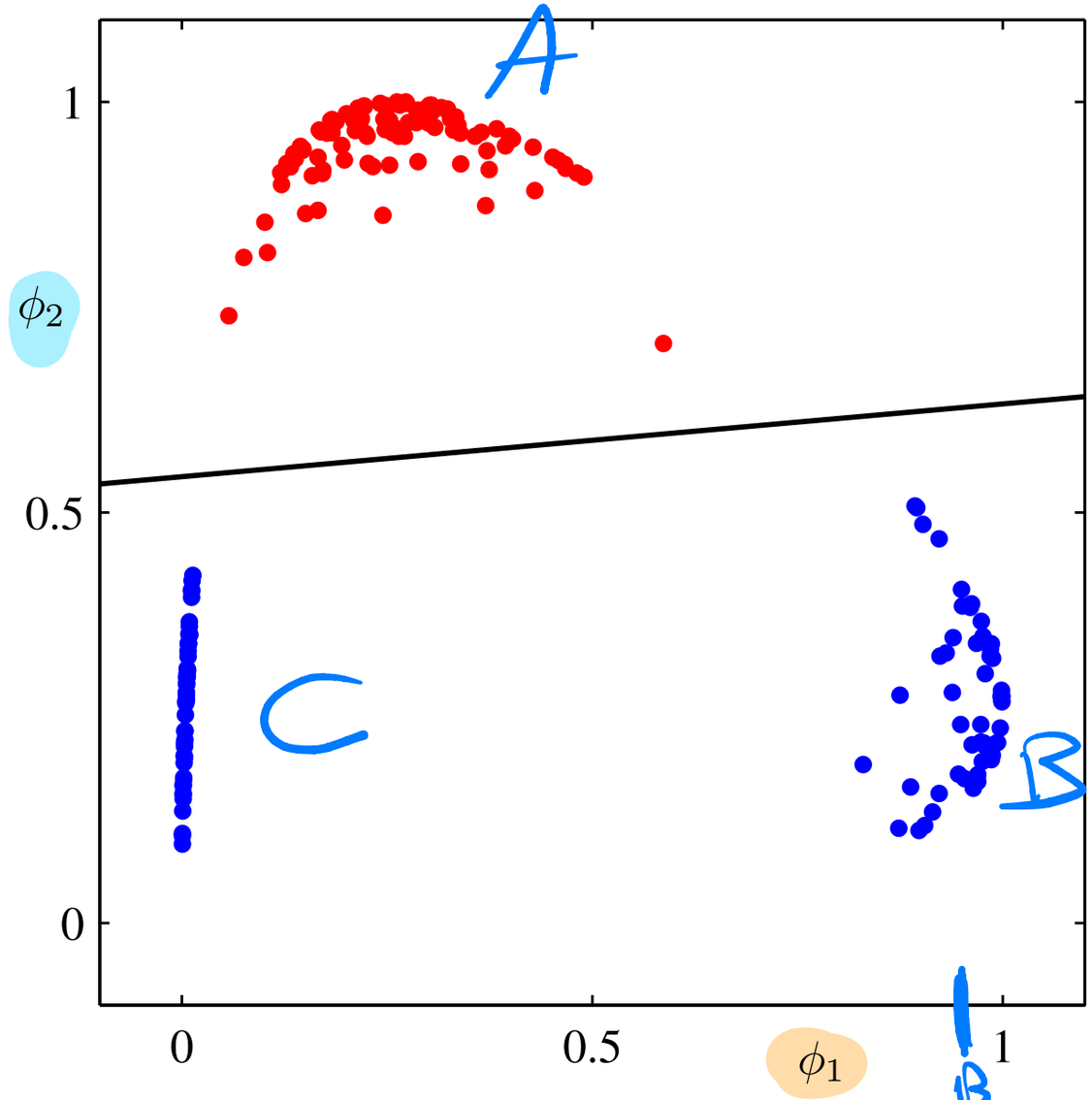


Figure: Left: original input space (x_1, x_2) , right: space of two gaussian basis functions with centres shown by the green crosses. (Bishop 4.12)

$$\phi_1(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T(\mathbf{x}-\boldsymbol{\mu}_1))$$

$$\phi_2(\mathbf{x}) = \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T(\mathbf{x}-\boldsymbol{\mu}_2))$$

Limitations of Fixed Basis Functions

Advantages:

- Closed form solution for least-squares problem
- ~~Tractable Bayesian treatment~~
- Nonlinear models mapping input variables to target variables through basis functions

neural networks

Limitations:

- Assumption: Basis functions $\phi_j(\mathbf{x})$ are fixed, not learned.
- Curse of dimensionality: to cover growing dimensions D of input vectors, the number of basis functions needs to grow rapidly / exponentially

University of Amsterdam
Informatics Institute

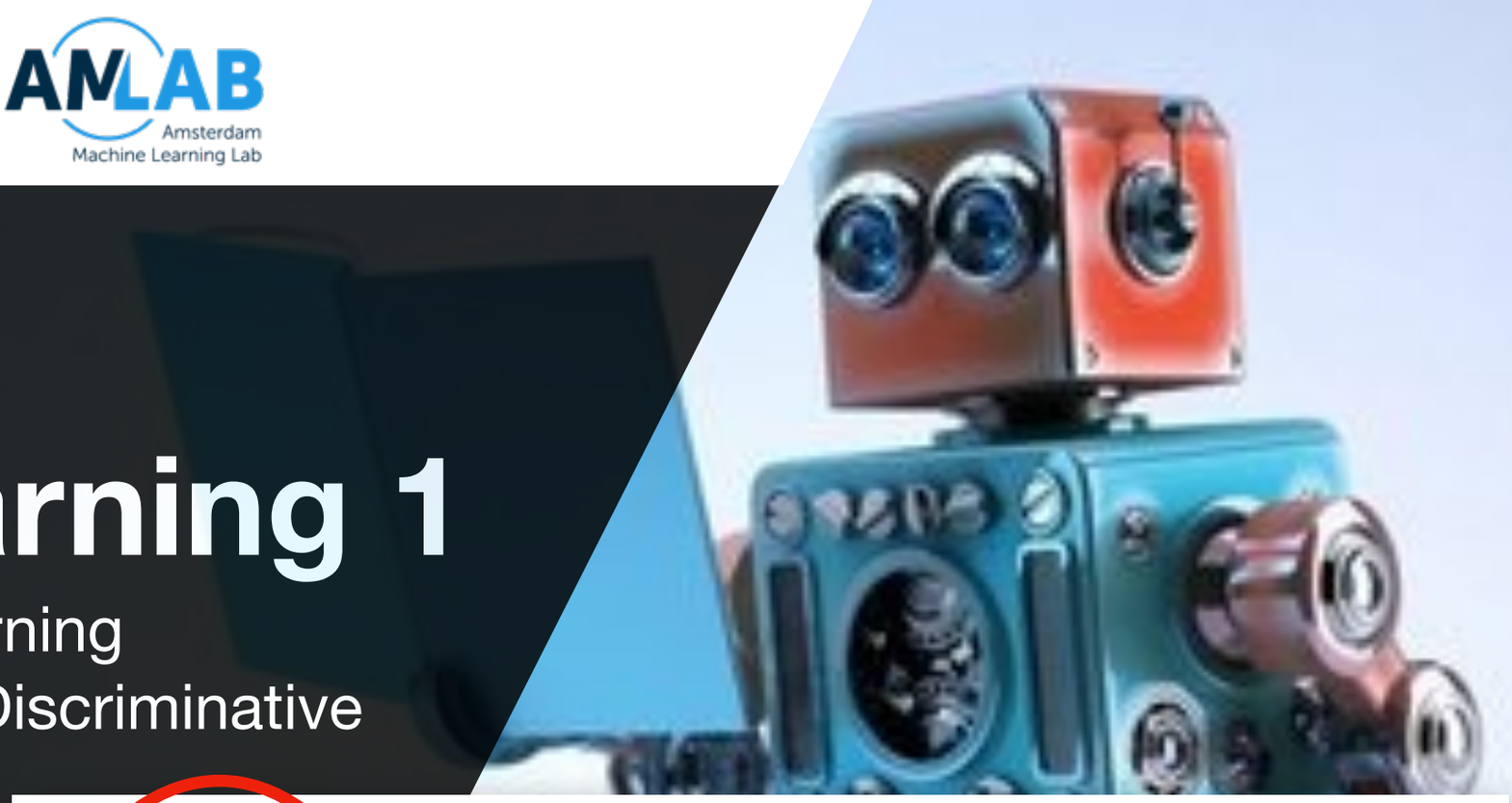
AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 7.2 - Supervised Learning
Classification - Probabilistic Discriminative Models - Logistic Regression

Erik Bekkers

(Bishop 4.3.2)



Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

Classification Strategies

- Discriminant functions

Direct mapping of input to target: $t = y(\mathbf{x}, \mathbf{w})$

e.g. perceptron, linear model

- Probabilistic discriminative models

Posterior class probabilities: $p(C_k \mid \mathbf{x}) = \sigma(\underline{w}^T \underline{x})$

- Probabilistic generative models

Class-conditional densities: $p(\mathbf{x} \mid C_k)$

Prior class probabilities: $p(C_k)$

Gaussian: LDA or QDA gives posterior of the form $p(C_k|x) = \sigma(\underline{w}^T x)$ in terms of $\{\mu_k\}, \Sigma$

Logistic Regression for Two Classes

- Given: Dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$ with binary targets

$$\mathbf{t} = (t_1, \ldots, t_N)^T \quad \text{with} \quad t_n \in \{C_1, C_2\} = \{1,0\}$$

From now on we write instead of $\mathbf{t}$

- Basis functions $\boldsymbol{\phi} := \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots)^T \in \mathbb{R}^M$

ϕ_i

- Probabilistic Discriminative Linear Models: Posteriors $p(C_k|\boldsymbol{\phi})$ are modeled with nonlinear functions f that take a linear function of $\boldsymbol{\phi}$ as input.

$$p(C_k|\boldsymbol{\phi}, \mathbf{w}) = f(\mathbf{w}^T\boldsymbol{\phi})$$

$C_1 \leftrightarrow t=1$, $C_2 \leftrightarrow t=0$

- Logistic regression (K=2)

Combined into a single expression

$$p(C_1|\boldsymbol{\phi}, \mathbf{w}) = y(\boldsymbol{\phi}) = \sigma(\mathbf{w}^T\boldsymbol{\phi})$$

$$p(C_2|\boldsymbol{\phi}, \mathbf{w}) = 1 - y(\boldsymbol{\phi}) = 1 - \sigma(\mathbf{w}^T\boldsymbol{\phi})$$

$$p(t|\boldsymbol{\phi}, \mathbf{w}) = y(\boldsymbol{\phi})^t \left(1 - y(\boldsymbol{\phi})\right)^{1-t}$$

$p(C_1|\phi, w)$ $\quad$ $p(C_2|\phi, w)$

Logistic Regression for Two Classes

- Let $\boldsymbol{\phi} \in \mathbb{R}^M$ and M-dimensional feature vector

- *Probabilistic Discriminative Modeling* with Logistic Regression:

$$p(C_1 \,|\, \boldsymbol{\phi}, \mathbf{w}) = \sigma(\mathbf{w}^T \boldsymbol{\phi}) \quad \text{and} \quad p(C_2 \,|\, \boldsymbol{\phi}, \mathbf{w}) = 1 - \sigma(\mathbf{w}^T \boldsymbol{\phi})$$

parameters: $\mathbf{w}$: M parameters → scales linearly

- *Generative Modeling* with Gaussian conditional densities:

$K = 2$

$$p(\mathbf{x} \,|\, C_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma_k|^{1/2}} \exp\{\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\}$$

$\mathbf{x} \in \mathbb{R}^D$

$\phi(x) \in \mathbb{R}^M$

and class priors $p(C_k)$ + K params

parameters:

Scales quadratically with M

$\boldsymbol{\mu}_k$: $2M$ parameters

Σ: $\sim M^2$ parameters

(# param could be further reduced with Naive Bayes)

Logistic Regression for Two Classes

- Given: Dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$ with binary targets

$$\mathbf{t} = (t_1, \ldots, t_N)^T \quad \text{with} \quad t_n \in \{C_1, C_2\} = \{0,1\}$$

position dependent Bernoulli

- Conditional likelihood function:

i.i.d.

$$p(\mathbf{t} \mid \mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} p(t_n \mid \mathbf{x}_n, \mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}$$

$p(x, t \mid y)$

$$y_n = p(C_1 \mid \boldsymbol{\phi}_n) = \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$$

$$\boldsymbol{\phi}_n := \boldsymbol{\phi}(\mathbf{x}_n)$$

- Maximizing the conditional likelihood/minimizing the cross-entropy

information theory

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

- $E(\mathbf{w})$: convex, but no closed form solution!

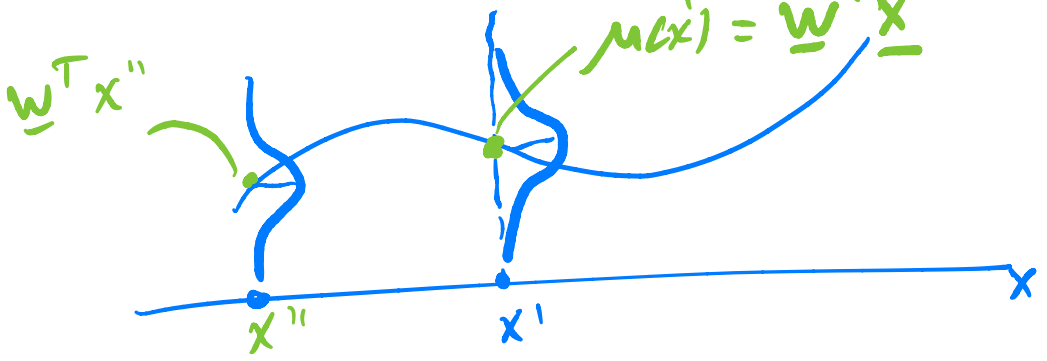
$y_n = \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$ is nonlinear in $\mathbf{w}$

$$H_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} E(\mathbf{w})$$

see lecture 7.4. $\Rightarrow \forall x : x^T H x \geq 0$

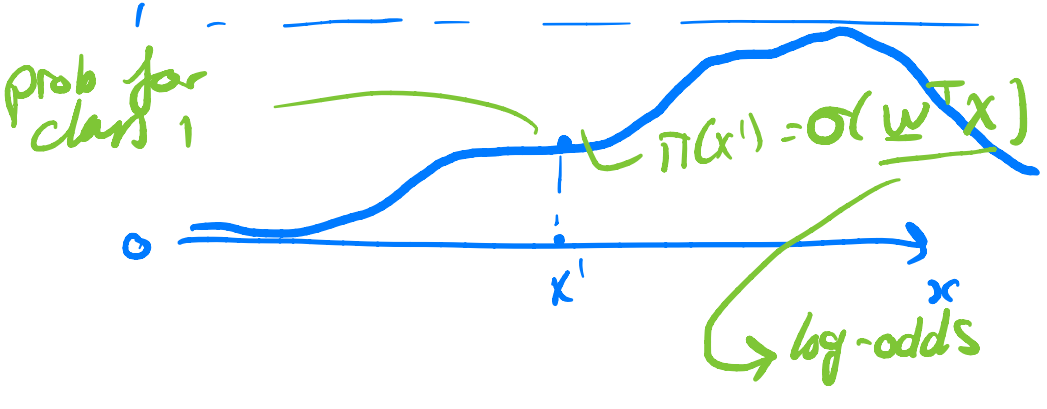
Regression w Gaussian predictive

$$p(t' \mid \underline{x}', \underline{w}) = \mathcal{N}(t' \mid \mu(x'), \sigma^2)$$



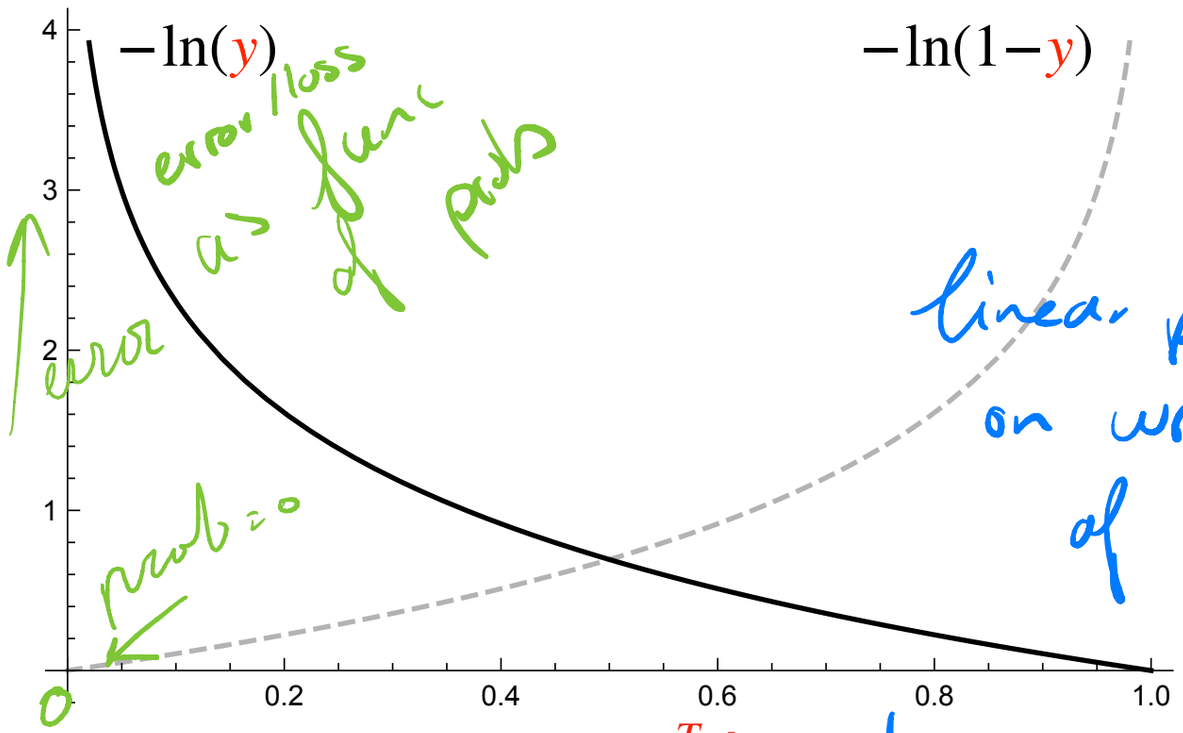
Classification w Bernoulli pred. distr.

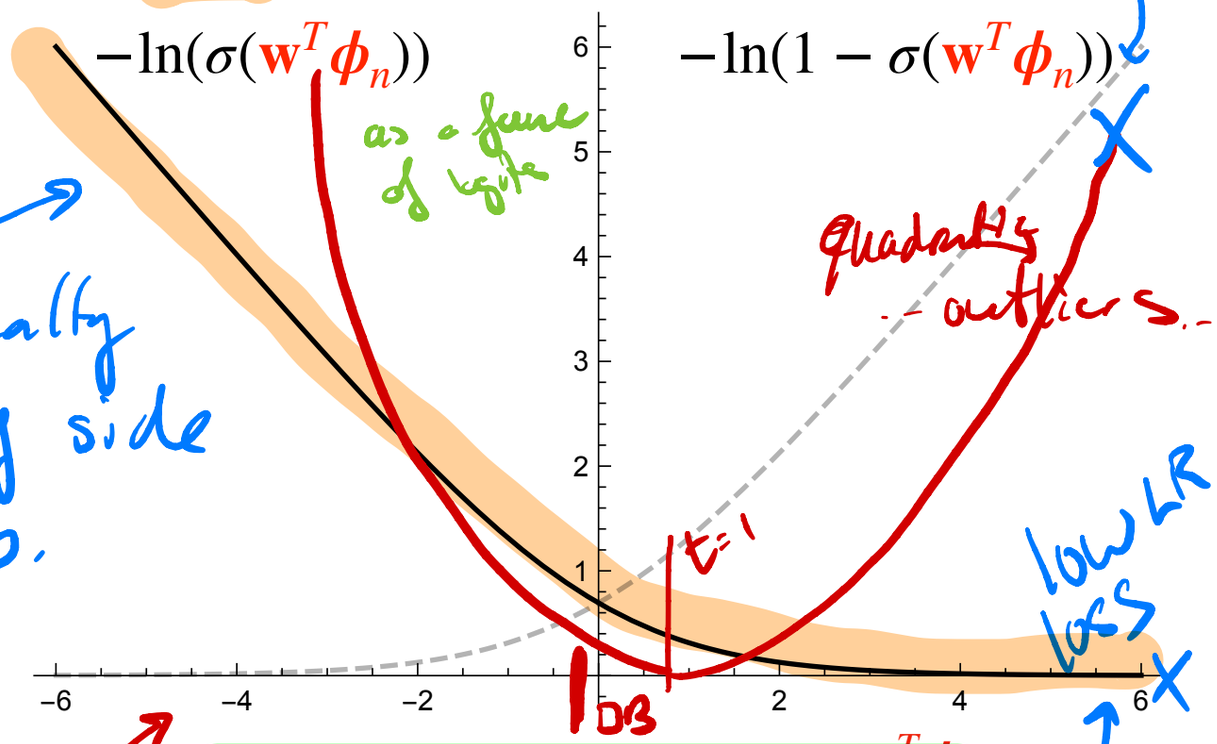
$$p(t' \mid x', \underline{w}) = \text{Bernoulli}(t' \mid \pi(x'))$$
$$= \pi(x')^{t'} (1 - \pi(x'))^{1-t'}$$



The cross-entropy loss

$$E_n(\mathbf{w}) = -\{t_n \ln(y_n) + (1 - t_n)\ln(1 - y_n)\}$$





Least squares outlier problem does not take place with logistic regression

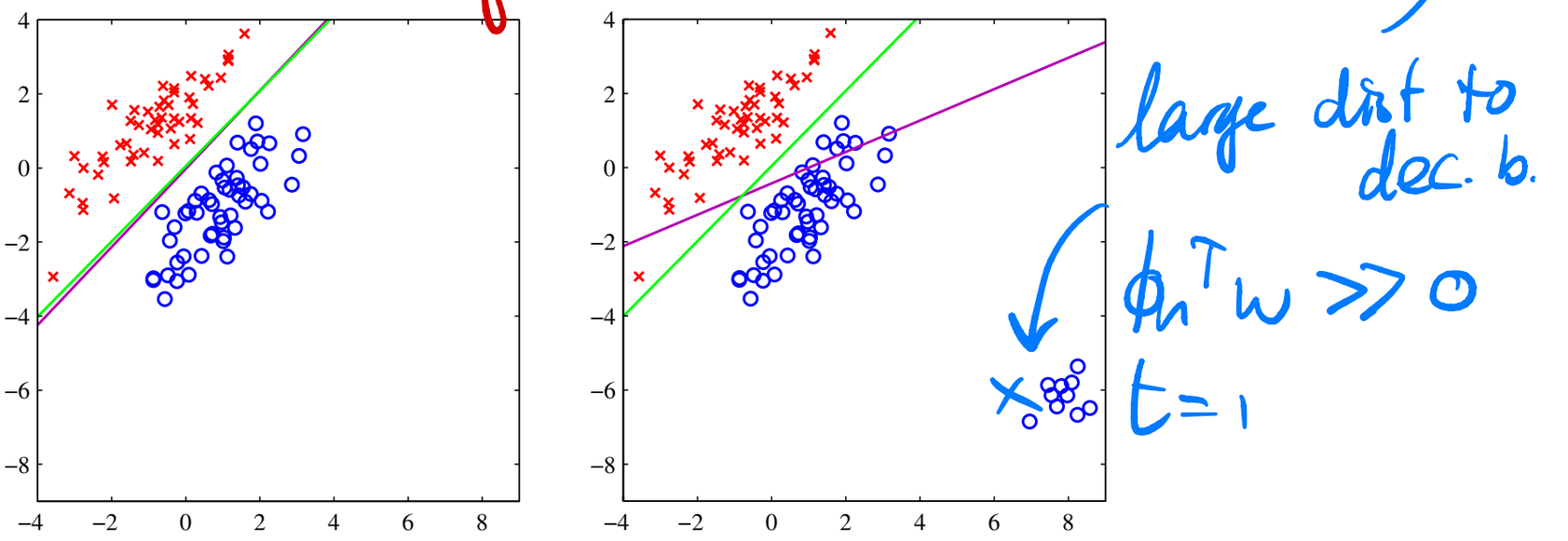


Figure: least squares is very sensitive to outliers (Bishop 4.4)

Classification with Logistic Regression

- Given: Dataset $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$ with binary targets

$$\mathbf{t} = (t_1, \ldots, t_N)^T \quad \text{with} \quad t_n \in \{C_1, C_2\} = \{0,1\}$$

- Basis functions: $\boldsymbol{\phi}(\mathbf{x}) = (\phi_0(\mathbf{x}), \ldots, \phi_{M-1}(\mathbf{x}))^T$

- Posterior distributions (Prob. Discr. Model): $p(C_1 \,|\, \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$

- Minimize

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

Convex so unique

with *stochastic gradient descent* or *iterative reweighted least squares*, to find $\mathbf{w}^\star$

- New datapoint $\mathbf{x}'$ is assigned to class C_1 if

$$p(C_1 \,|\, \mathbf{x}', \mathbf{w}^\star) = \sigma((\mathbf{w}^\star)^T \boldsymbol{\phi}(\mathbf{x}')) > \frac{1}{2}, \quad (\mathbf{w}^\star)^T \boldsymbol{\phi} > 0$$

- Decision boundaries:

linear

$$\mathbf{w}^{*T} \boldsymbol{\phi} = 0$$

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 7.3 - Supervised Learning
Classification - Logistic Regression:
Stochastic Gradient Descent

Erik Bekkers

(Bishop 4.3.2)

Image credit: Kirillm | Getty Images

Slide credits: Patrick Forré and Rianne van den Berg

Has very simple gradient!

Logistic Regression (K=2): SGD

- Stochastic Gradient Descent for cross-entropy:

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

$\hookrightarrow y_n = \sigma(\mathbf{w}^T \phi)$

- Update rule given a random data point $(\mathbf{x}_n, t_n)$

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

- Gradient: $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right)$

$$\frac{\partial E_n(\mathbf{w})}{\partial w_j} = \frac{\partial E_n(\mathbf{w})}{\partial y_n} \cdot \frac{\partial y_n}{\partial w_j} = \left(-\frac{t_n}{y_n} + \frac{1-t_n}{1-y_n} \right) \cdot \frac{\partial y_n}{\partial w_j}$$

$$\frac{\partial y_n}{\partial w_j} = \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$$

Logistic Regression (K=2): SGD

- Let's derive $\frac{\partial y_n}{\partial w_j} = \frac{\partial}{\partial w_j}\sigma(\mathbf{w}^T\boldsymbol{\phi}_n)$ using $\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$

$$\frac{\partial}{\partial w_j}\sigma(\mathbf{w}^T\boldsymbol{\phi}_n) = \sigma(\mathbf{w}^T\boldsymbol{\phi}_n)(1-\sigma(\mathbf{w}^T\boldsymbol{\phi}_n))\frac{\partial(\mathbf{w}^T\boldsymbol{\phi}_n)}{\partial w_j}$$

y_n

$\frac{\partial}{\partial w_j}\left(\sum_m w_m \phi_{nm}\right)$

$m \neq j : \frac{\partial}{\partial w_j} \neq 0$

$= y_n(1-y_n)\phi_{nj} = \phi_{nj}$

$$\frac{\partial E_n(\mathbf{w})}{\partial w_j} = -\frac{t_n}{y_n}\frac{\partial y_n}{\partial w_j} + \frac{1-t_n}{1-y_n}\frac{\partial y_n}{\partial w_j}$$

$$= -\frac{t_n}{y_n} y_n(1-y_n)\phi_{nj} + \frac{1-t_n}{1-y_n} y_n(1-y_n)\phi_{nj}$$

$$= -t_n\phi_{nj} + t_n y_n \phi_{nj} + y_n\phi_{nj} - t_n y_n \phi_{nj}$$

$$= (y_n - t_n)\phi_{nj}$$

Logistic Regression (K=2): SGD

- Stochastic Gradient Descent for cross-entropy:

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

- Update rule given a random data point $(\mathbf{x}_n, t_n)$

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

- Gradient: $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right) = (y_n - t_n)\boldsymbol{\phi}_n$

- Gradient Descent for LR has a perceptron-like update rule:

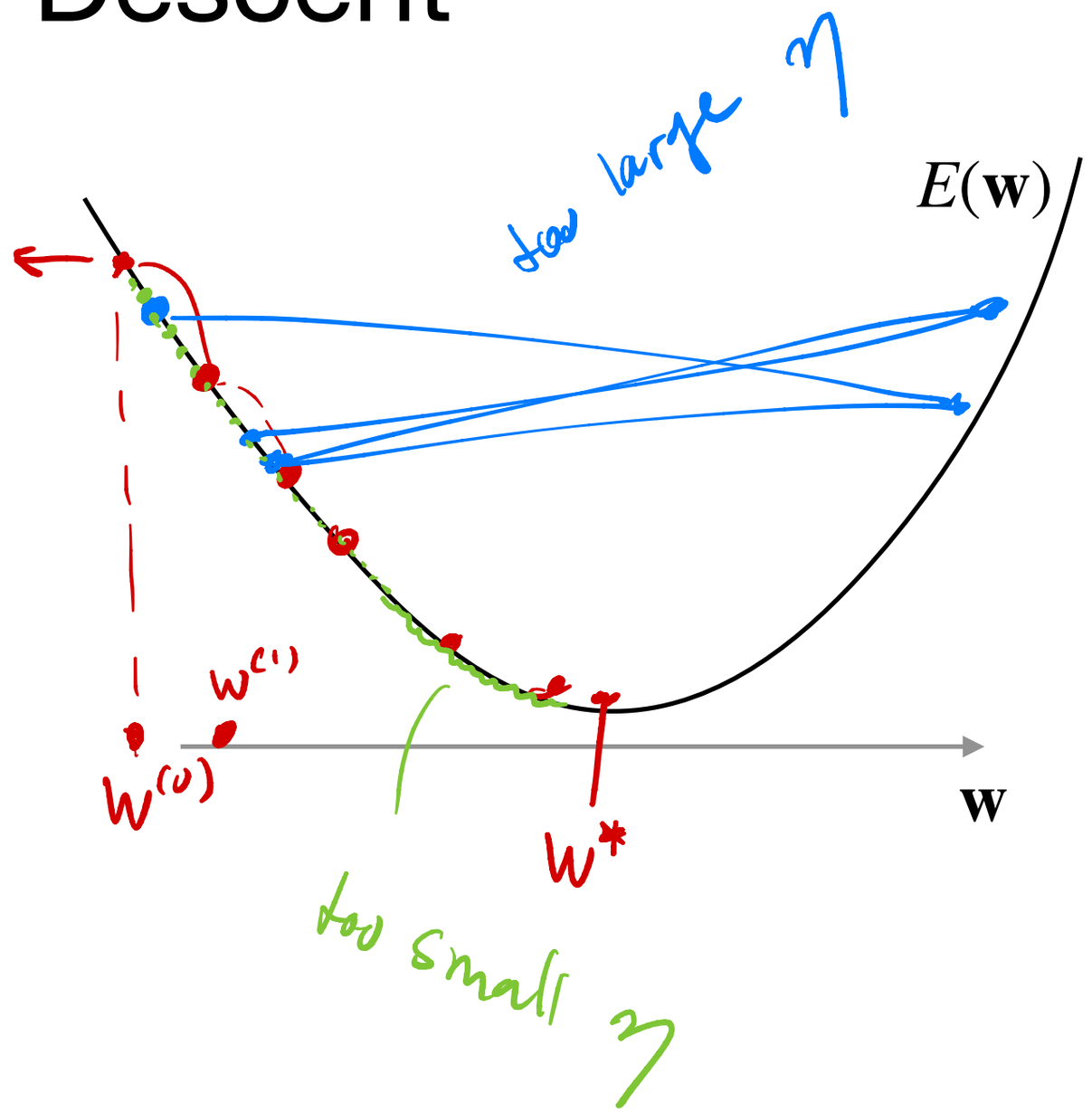
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta(y_n - t_n)\boldsymbol{\phi}_n$$

Simple form b/c activation function f is the link function! (Bishop 4.3.6)

Stochastic Gradient Descent

1. Initialize $\mathbf{w}^{(0)}$
2. Choose a learning rate η
3. While $\|\mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)}\| > \epsilon$
    I. Choose a random data point $(\mathbf{x}_n, t_n)$
    II. Update $\mathbf{w}$:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta(y_n - t_n)\boldsymbol{\phi}_n$$



- If η too large: no convergence
- If η too small: very slow convergence
- Converged $\mathbf{w}^{\star}$: estimate of minimizer of $E(\mathbf{w})$!

University of Amsterdam
Informatics Institute

AMLAB
Amsterdam
Machine Learning Lab

Machine Learning 1

Lecture 7.4 - Supervised Learning
Classification - Logistic Regression: Newton-Raphson Optimization

Erik Bekkers

(Bishop 4.3.3)

Not material for mid-term. But please watch video by the end of the course, or try to follow my annotations

Slide credits: Patrick Forré and Rianne van den Berg

Image credit: Kirillm | Getty Images

Newton-Raphson Iterative Optimization

- Goal: minimize

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

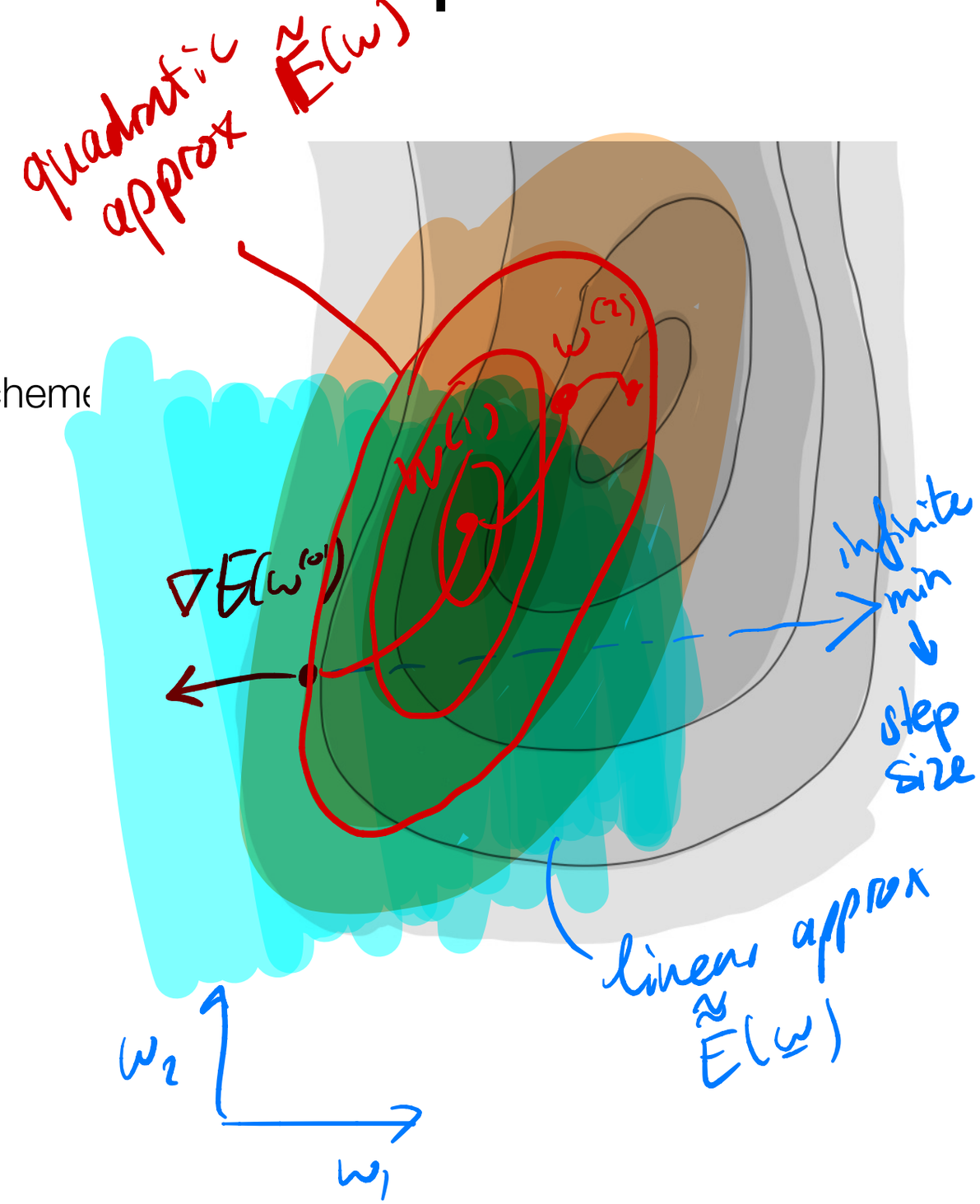
- Newton-Raphson iterative second order optimization scheme

1. Initial guess $\mathbf{w}^0$
2. For $\tau = 1,\ldots$:
   I. Approximate $E(\mathbf{w})$ with a quadratic function $\tilde{E}_{\tau-1}(\mathbf{w})$ around $\mathbf{w}^{(\tau-1)}$
   II. Set $\mathbf{w}^{(\tau)}$ such that it minimizes $\tilde{E}(\mathbf{w})$, i.e.,

$$\mathbf{w}^{(\tau)} = \underset{\mathbf{w}}{\operatorname{argmin}}\, \tilde{E}_{\tau-1}(\mathbf{w})$$

   III. Stop when $\|\mathbf{w}^{(\tau+1)} - \mathbf{w}^{(\tau)}\| = 0$
3. You have found $\mathbf{w}^\star$ such that $\frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}^\star) = 0$

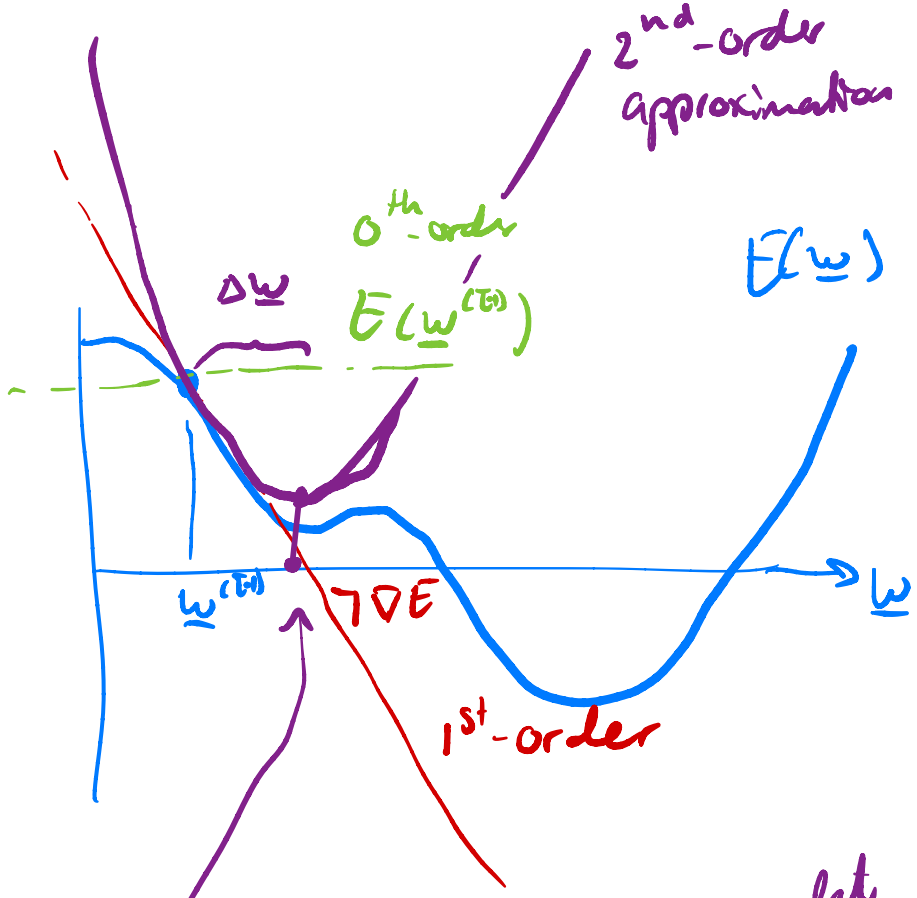
So no step size



Taylor expansion:

$$f(x_0 + \Delta x) \approx f(x_0) + \Delta x \left.\frac{\partial}{\partial x} f\right|_{x_0} + \frac{1}{2}\Delta x^2 \left.\frac{\partial}{\partial x^2} f\right|_{x_0} + \ldots$$

+ ..... higher order derivatives

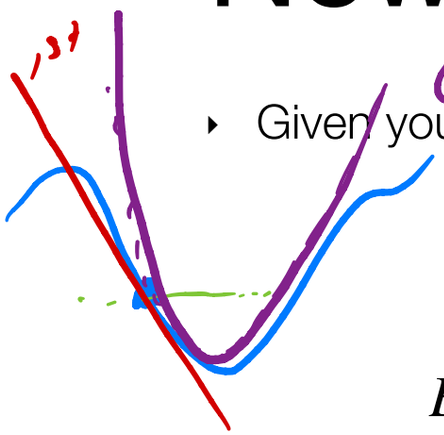


well-defined minimum so let

$$\underline{w}^{(\tau)} = \underset{\underline{w}}{\operatorname{argmin}}\, E^{2nd}_{\underline{w}^{(\tau-1)}}(\underline{w})$$

$$= \underline{w}^{(\tau-1)} + \Delta \underline{w}$$

Newton-Raphson Iterative Optimization



- Given your old estimate $\mathbf{w}^{(\tau-1)}$, approximate $E(\mathbf{w})$ with a second order Taylor expansion around $\mathbf{w}^{(\tau-1)}$

$$E(\mathbf{w}) \approx \tilde{E}(\mathbf{w}^{(\tau-1)} + \Delta\mathbf{w}) = E(\mathbf{w}^{(\tau-1)}) + (\Delta\mathbf{w})^T \nabla^T E(\mathbf{w}^{(\tau-1)}) + \frac{1}{2}(\Delta\mathbf{w})^T \mathbf{H} \Delta\mathbf{w}$$

- Gradient (first order derivatives):

$$\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right)$$

- Hessian Matrix (second order derivatives):

$$H_{ij} = \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}$$

- Choose $\Delta\mathbf{w}$ such that next estimate $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \Delta\mathbf{w}$ minimizes $\tilde{E}(\mathbf{w})$:

$$\frac{\partial}{\partial \Delta\mathbf{w}} \tilde{E}(\mathbf{w}^{(\tau-1)} + \Delta\mathbf{w}) =$$

- Update rule:

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1} \nabla^T E(\mathbf{w}^{(\tau-1)})$$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1}\nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}}\right) = (y_n - t_n)\boldsymbol{\phi}_n$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n =$$

$$\left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})\right)$$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1} \nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right) = (y_n - t_n)\boldsymbol{\phi}_n$

all data

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t})$$

feat vect error

$$\left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \right)$$

- Hessian $H_{ij} = \frac{\partial^2 E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j}$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1}\nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \cdots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}}\right) = (y_n - t_n)\boldsymbol{\phi}_n^T$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t})$$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

- Hessian $H_{ij} = \frac{\partial^2 E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i}\sum_{n=1}^{N}(y_n - t_n)\phi_j(\mathbf{x}_n)$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1} \nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right) = (y_n - t_n)\boldsymbol{\phi}_n$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t}) \qquad \left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \right)$$

- Hessian $H_{ij} = \frac{\partial^2 E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i} \sum_{n=1}^{N} (y_n - t_n)\phi_j(\mathbf{x}_n) = \sum_{n=1}^{N} \phi_j(\mathbf{x}_n) \frac{\partial y_n}{\partial w_i}$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1} \nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}} \right) = (y_n - t_n)\boldsymbol{\phi}_n$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t})$$

$$\left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \right)$$

- Hessian $H_{ij} = \frac{\partial^2 E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i} \sum_{n=1}^{N} (y_n - t_n)\phi_j(\mathbf{x}_n) = \sum_{n=1}^{N} \phi_j(\mathbf{x}_n) \frac{\partial y_n}{\partial w_i}$

$$= \sum_{n=1}^{N} y_n(1 - y_n)\phi_i\phi_j$$

$y_n = \sigma(\phi_n^T w)$

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

"weight per data point"

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1}\nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}}\right) = (y_n - t_n)\boldsymbol{\phi}_n$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t}) \qquad \left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})\right)$$

- Hessian $H_{ij} = \frac{\partial^2 E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i}\sum_{n=1}^{N}(y_n - t_n)\phi_j(\mathbf{x}_n) = \sum_{n=1}^{N}\phi_j(\mathbf{x}_n)\frac{\partial y_n}{\partial w_i}$

$\in \mathbb{R}^{M\times M}$

$R = \mathrm{diag}(\{y_n(1-y_n)\})$

$$= \sum_{n=1}^{N} y_n(1-y_n)\phi_i\phi_j$$

$$\mathbf{H} = \sum_{n=1}^{N} y_n(1-y_n)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T R\,\boldsymbol{\Phi}$$

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

$R_{nm} = 0$ if $n \neq m$

Newton-Raphson Iterative Optimization

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1}\nabla^T E(\mathbf{w}^{(\tau-1)})$
- Gradient $\nabla E_n(\mathbf{w}) = \left(\frac{\partial E_n(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial E_n(\mathbf{w})}{\partial w_{M-1}}\right) = (y_n - t_n)\boldsymbol{\phi}_n$

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N}(y_n - t_n)\boldsymbol{\phi}_n = \boldsymbol{\Phi}^T(\mathbf{y} - \mathbf{t})$$

$$\left(E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})\right)$$

- Hessian $H_{ij} = \frac{\partial E(\mathbf{w}^{(\tau-1)})}{\partial w_i \partial w_j} = \frac{\partial}{\partial w_i}\sum_{n=1}^{N}(y_n - t_n)\phi_j(\mathbf{x}_n) = \sum_{n=1}^{N}\phi_j(\mathbf{x}_n)\frac{\partial y_n}{\partial w_i}$

$$= \sum_{n=1}^{N} y_n(1 - y_n)\phi_i\phi_j$$

$$\mathbf{H} = \sum_{n=1}^{N} y_n(1 - y_n)\boldsymbol{\phi}_n\boldsymbol{\phi}_n^T = \boldsymbol{\Phi}^T\mathbf{R}\boldsymbol{\Phi}$$

$\Delta w = -H^{-1}\nabla E^T$

Iterative Reweighted Least Squares

- Update rule: $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \mathbf{H}^{-1}\nabla^T E(\mathbf{w}^{(\tau-1)})$

  optimal Δw give the approx.

  - With Gradient: $\nabla E(\mathbf{w}) = \mathbf{\Phi}^T(\mathbf{y} - \mathbf{t})$
  - With Hessian: $\mathbf{H} = \mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi}$, with $\mathbf{R} = \text{diag}_{N\times N}\{y_n(1-y_n)\}$

- Newton-Raphson updates solutions to a weighted least squres problem -> iterative reweighted least squares:

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

$$= (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\left\{(\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})\,\mathbf{w}^{(\tau-1)} - \mathbf{\Phi}^T(\mathbf{y} - \mathbf{t})\right\}$$

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

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

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

- Note similarity with ML solution to linear regression:

$\mathbf{w}_{ML} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{t}$ (solution to least squares regression $\mathbf{w}_{\text{ML}} = \underset{\mathbf{w}}{\text{argmin}} \sum_{n=1}^{N} (\mathbf{w}^T\boldsymbol{\phi}_n - t_n)^2$)

$\mathbf{w}^{(\tau)} = (\mathbf{\Phi}^T\mathbf{R}\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{R}\mathbf{z}$ (solution to a weighted least squares problem $\mathbf{w}^{(\tau)} = \underset{\mathbf{w}}{\text{argmin}} \sum_{n=1}^{N} r_i\,(\mathbf{w}^T\boldsymbol{\phi}_n - z_n)^2$

SGD vs Newton-Raphson

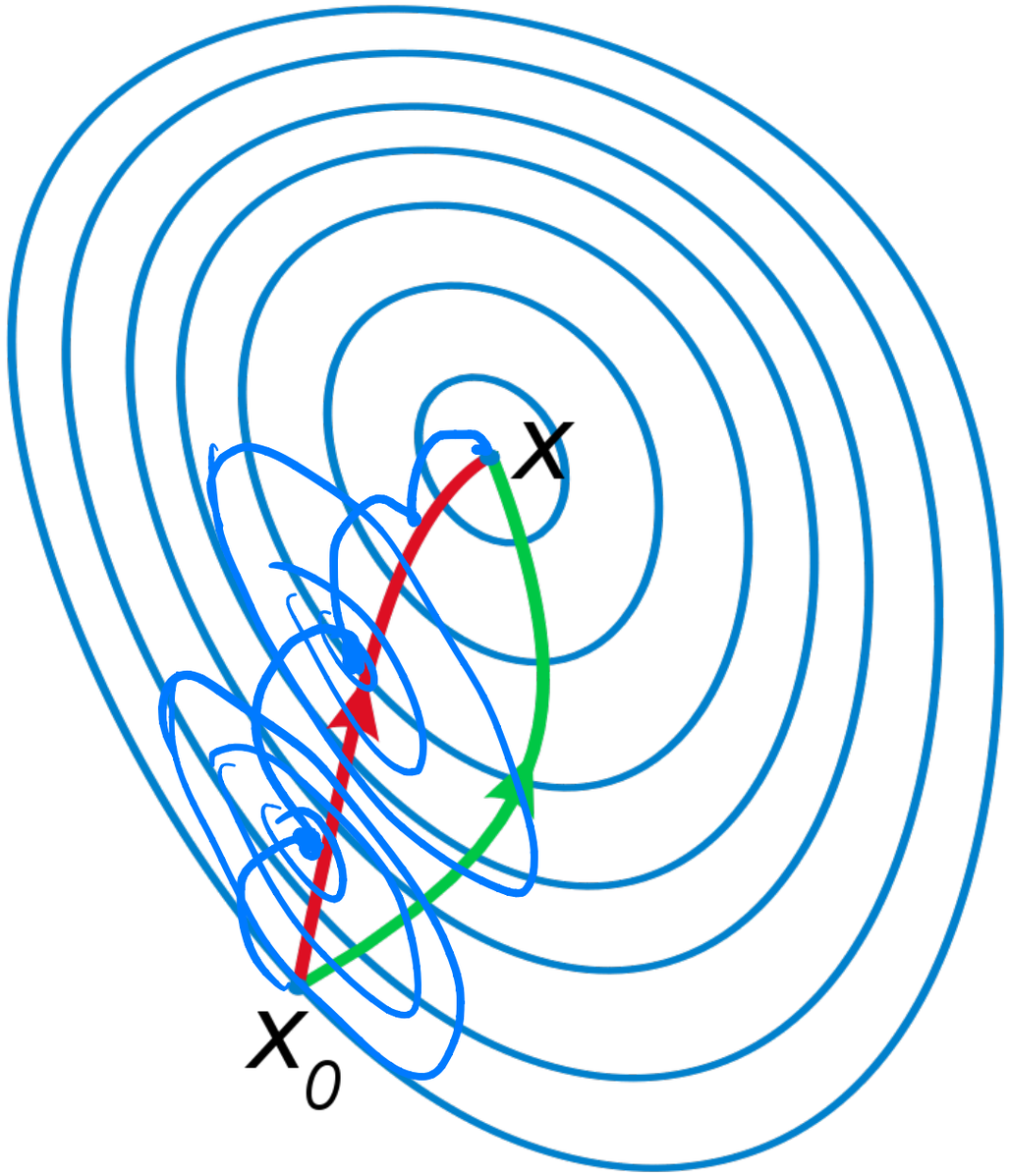


Figure: Green: gradient descent, which always goes in the direction of steepest descent. Red: Newton-Raphson's procedure, which takes into account curvature to take a more direct path. (Wikipedia)

The cross entropy loss is convex

- The error function $E(\mathbf{w})$ is convex since the Hessian is positive definite

- *Proof Hessian is convex:*

  The Hessian of $E(\mathbf{w})$ is given by $\mathbf{H} = \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi}$, with $\mathbf{R} = \text{diag}_{N \times N}\{y_n(1 - y_n)\}$.
  To show: $\forall_{\mathbf{w} \neq \mathbf{0} \in \mathbb{R}^M} : \mathbf{w}^T \mathbf{H} \mathbf{w} > 0$.

$$\mathbf{w}^T \mathbf{H} \mathbf{w} = \mathbf{w}^T \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi} \mathbf{w} \qquad \left(\mathbf{R} = \sqrt{\mathbf{R}}\sqrt{\mathbf{R}}\right)$$

$$= \mathbf{w}^T \mathbf{\Phi}^T \sqrt{\mathbf{R}}\sqrt{\mathbf{R}} \mathbf{\Phi} \mathbf{w} \qquad \left(\sqrt{\mathbf{R}} = \text{diag}(\sqrt{y_n(1 - y_n)})\right)$$

$$= (\sqrt{\mathbf{R}} \mathbf{\Phi} \mathbf{w})^T (\sqrt{\mathbf{R}} \mathbf{\Phi} \mathbf{w})$$

$$= \|\sqrt{\mathbf{R}} \mathbf{\Phi} \mathbf{w}\|^2 > 0 \qquad \left(\text{since } \mathbf{w} \neq \mathbf{0}\right)$$