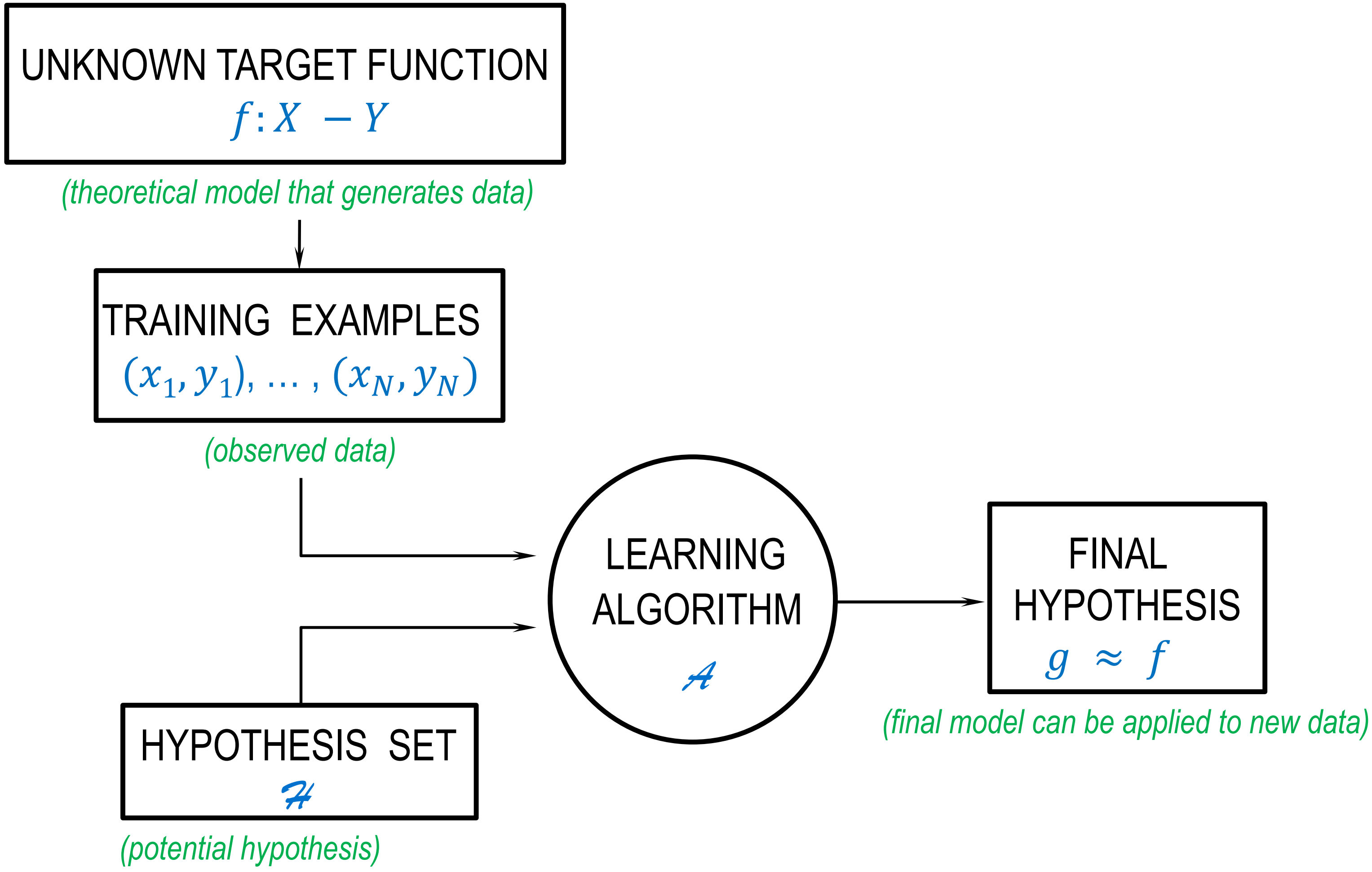
LEUPHANA
UNIVERSITÄT LÜNEBURG

INTRODUCTION TO MACHINE LEARNING

02. Linear models

Winter 2020/2021

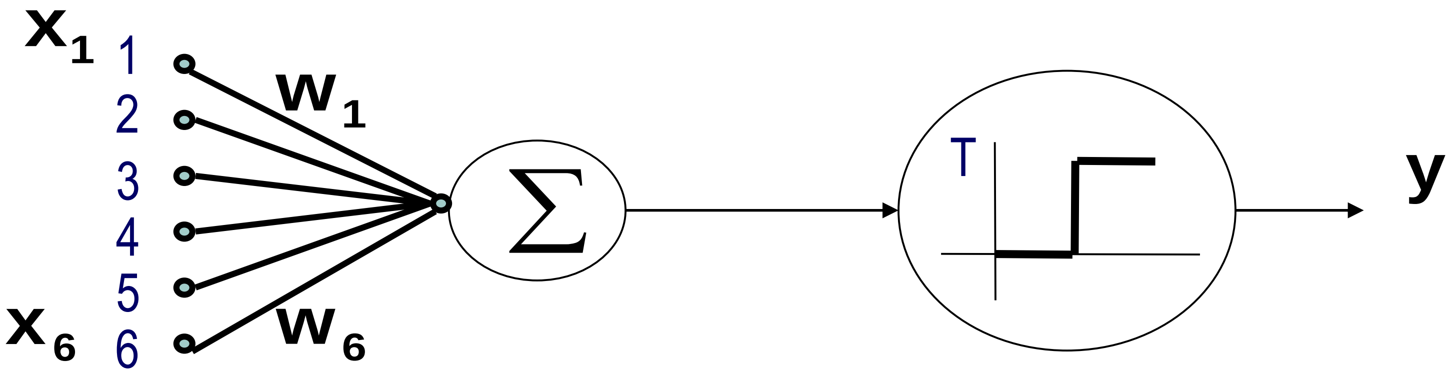
Review



UNKNOWN TARGET FUNCTION
$f: X \rightarrow Y$
(theoretical model that generates data)
TRAINING EXAMPLES
$(x_1, y_1), \dots, (x_N, y_N)$
(observed data)
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
(final model can be applied to new data)
HYPOTHESIS SET
$\mathcal{H}$
(potential hypothesis)

Perceptron algorithm

- Linearly separable classes can be learned
- Easy weight update rule $\mathrm{w}(t+1) = \mathrm{w}(t) + \eta y_i x_i$
- Quick learning is guaranteed (Novikoff Theorem)



Agenda

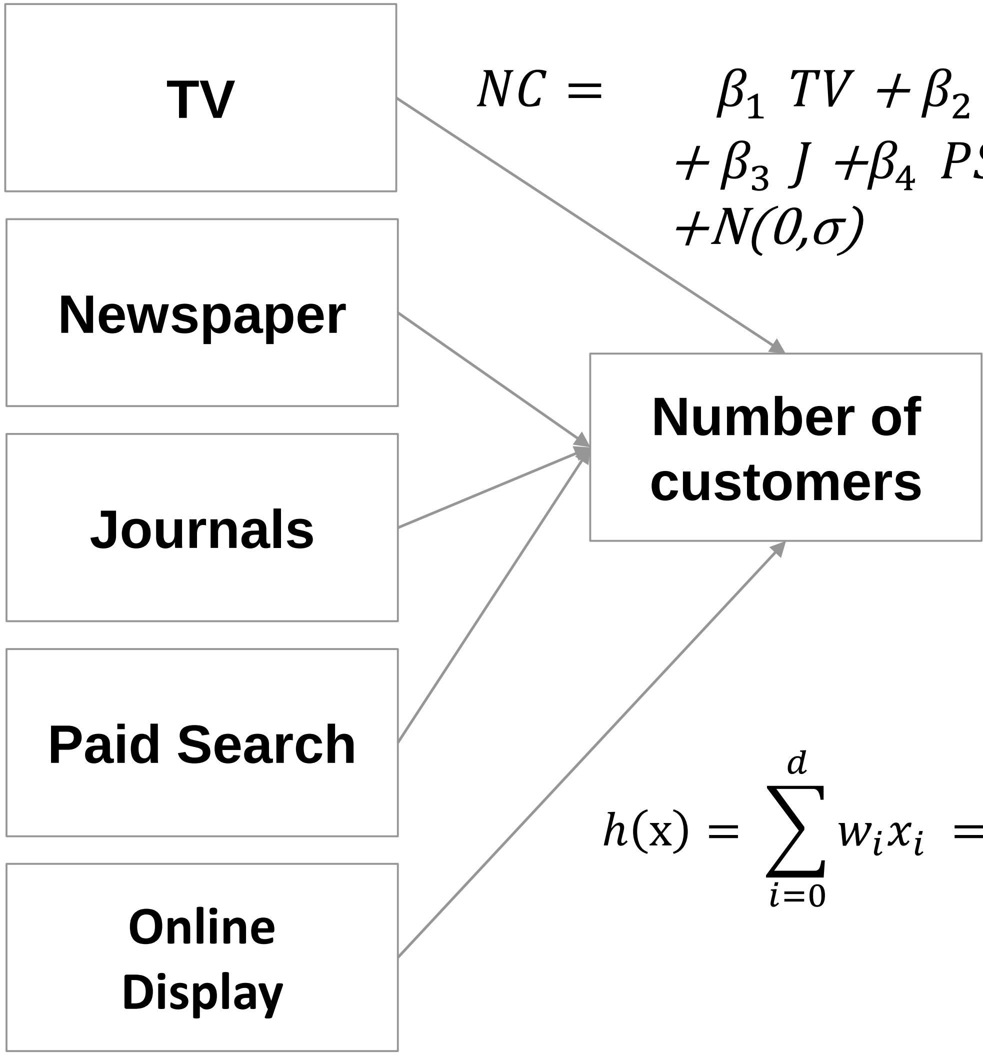
- Introduction
- Learning problem & linear classification
- **Linear models: regression & logistic regression**
- Non-linear transformation, overfitting & regularization
- Support Vector Machines and kernel learning
- Neural Networks: shallow [and deep]
- Theoretical foundation of supervised learning
- Unsupervised learning

Today's objectives

— Understand foundation of linear models
— Explore different approaches to (solving) linear models

Motivation

- Linear models are (i) widely used in practice, (ii) easy to solve (closed form solution), and (iii) easy to understand
- Linear models are the basis for more complex models (e.g. generalized linear models – GLM)



$$NC = \beta_1\ TV + \beta_2\ NP + \beta_3\ J + \beta_4\ PS + \beta_5\ OD + N(0,\sigma)$$

$$h(\mathrm{x}) = \sum_{i=0}^{d} w_i x_i = \mathrm{w}^\top \mathrm{x}$$

Graphical intuition

What problem should we solve? Optimizing the cost function

$$E(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}(x_n^\top w - \boldsymbol{y_n})^2$$

$$= \frac{1}{2}\|\mathrm{X}w - y\|^2$$

where $\mathrm{X} = \begin{bmatrix} - & x_1^\top & - \\ - & x_2^\top & - \\ & \vdots & \\ - & x_N^\top & - \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

Note: $x_i^\top = (1, \; x_{i1}, \dots, x_{id})$ and $w^\top = (w_0, w_1, \dots, w_d)$ where w_0 is the intercept

Dimensionality of *X* and *y*

Minimizing the cost function $\mathbf{E}(\mathbf{w})$ in closed form

$$E(\mathrm{w}) = \frac{1}{2}\, \|\mathrm{X}\mathrm{w} \; - \mathrm{y}\|^2$$

$$\nabla J(\mathrm{w}) = \frac{2}{2} \mathrm{X}^{\top}(\mathrm{X}\mathrm{w} - \mathrm{y}) = 0$$

$X^{\top}\mathrm{X}\mathrm{w} = X^{\top}\mathrm{y}$ **(normal equations)**

$\mathrm{w} = X^{\dagger}\mathrm{y}$ where $X^{\dagger} = (X^{\top}\mathrm{X})^{-1} X^{\top}$

$\mathrm{X}^{\dagger}$ is the 'pseudo-inverse' of X

The pseudo-inverse

$$X^{\dagger} = (X^{\intercal}X)^{-1}X^{\intercal}$$

$$\underbrace{\left(\underbrace{\begin{bmatrix} \quad \end{bmatrix}}_{d+1 \times d+1} \right)^{-1} \underbrace{\begin{bmatrix} \qquad \end{bmatrix}}_{d+1 \times N}}_{d+1 \times N}$$

„Learning *algorithm*” for linear regression

- Construct the matrix **X** and the vector **y** from the data set $(x_1, y_1), \cdots, (x_N, y_N)$ as follows

$$\underbrace{\mathrm{X} = \begin{bmatrix} - \, x_1^\intercal \, - \\ - x_2^\intercal \, - \\ \vdots \\ - x_N^\intercal \, - \end{bmatrix}}_{\text{input data matrix}}, \quad \underbrace{y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

- Compute the pseudo-inverse $\mathrm{X}^\dagger = (\mathrm{X}^\intercal \mathrm{X})^{-\mathbf{1}} \mathrm{X}^\intercal$.
- Return $\mathbf{w} = \mathrm{X}^\dagger \mathbf{y}$.

Minimizing the cost function $\mathbf{J}(\mathbf{w})$ – gradient descent

$$E(\mathrm{w}) = \frac{1}{2} \|\mathrm{Xw} - \mathrm{y}\|^2$$

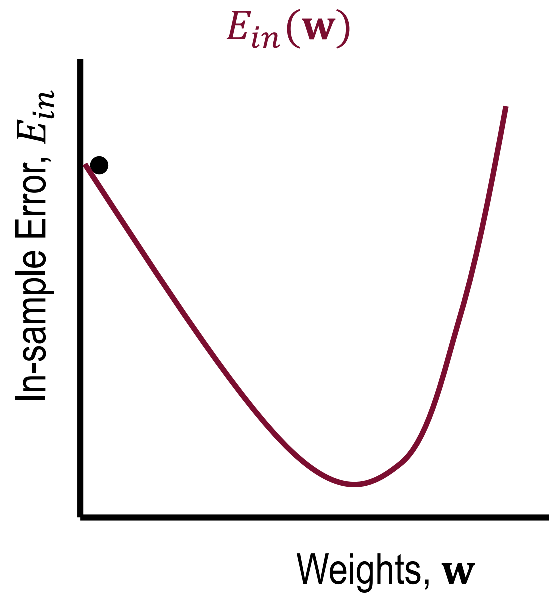
$$\nabla E(\mathrm{w}) = \frac{2}{2} \mathrm{X}^{\top}(\mathrm{Xw} - \mathrm{y})$$

Gradient descent iteratively finds the minimal cost:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\eta} \nabla \boldsymbol{E}(\boldsymbol{w}) = \boldsymbol{w} - \boldsymbol{\eta} \boldsymbol{X}^{\top}(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})$$

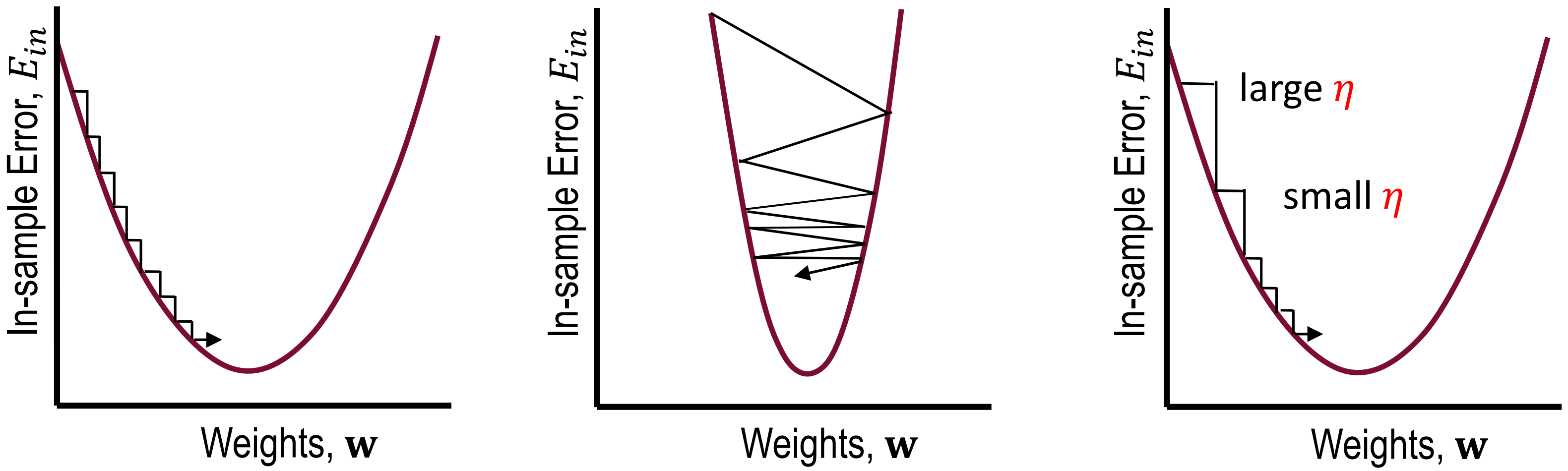
Iterative method: gradient descent

— General method for nonlinear optimization
— Start at $\mathrm{w}(0)$ and take a step along steepest slope
— Fixed step size: $\mathrm{w}(1) = \mathrm{w}(0) + \eta\,\hat{\mathrm{v}}$
— What is the direction $\hat{\mathrm{v}}$?



Fixed-size step?

How η affects the algorithm:



η too small

η too large

variable η — just right

Why gradient descent?

— We have seen, that there is a closed-form solution

$$\mathrm{w} = \mathrm{X}^{\dagger}\mathrm{y} \quad \text{where} \quad \mathrm{X}^{\dagger} = (\mathrm{X}^{\intercal}\mathrm{X})^{-1}\mathrm{X}^{\intercal}$$

— If the number of parameters/dimensions (d) becomes large the cost of calculating the inverse $(\mathrm{X}^{\intercal}\mathrm{X})^{-\mathbf{1}}$ increases with $O(nd^2)$

— The computational complexity of gradient descent is $O(nd)$

$$w \leftarrow w - \eta \nabla J(\mathrm{w}) = w - \eta \mathrm{X}^{\intercal}(\mathrm{X}\mathbf{w} - \mathbf{y})$$

Stochastic Gradient Descent (SGD)

— Instead of taking $X^{\top}(X\mathrm{w} - \mathrm{y})$ for gradient descent (batch mode), we can use individual instances (x_n, y_n)

$$w \leftarrow w - \eta x_n (w^T x_n - y_n)$$

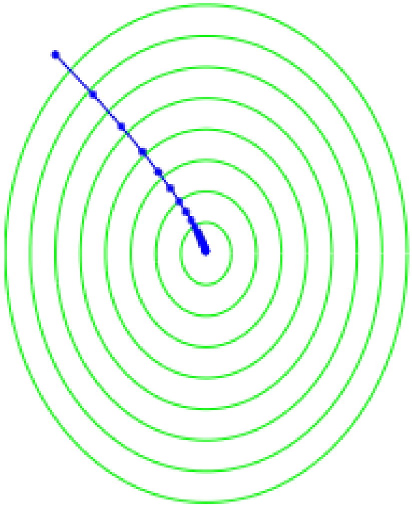
— This approach is called **stochastic** because we choose (x_n, y_n) randomly

— The algorithm can be applied to streaming data

— Computationally more efficient and can help to overcome local minima for more complex cost functions due to randomization

Gradient Descent vs Stochastic Gradient Descent (SGD)

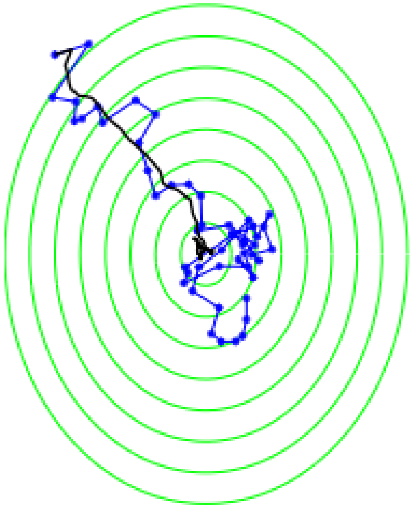
Batch Gradient Descent



$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$$

Play with the learning rate:
https://developers.google.com/machine-learning/crash-course/fitter/graph

Stochastic Gradient Descent (or mini batches)



choose $\mathbf{v}_t$ at random from a distribution such that $\mathbb{E}[\mathbf{v}_t \,|\, \mathbf{w}^{(t)}] \in \partial f(\mathbf{w}^{(t)})$
update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$

The black line denotes the averaged value of w

Note: we can also do GD for *k* < *N* points at a time - mini batch approach

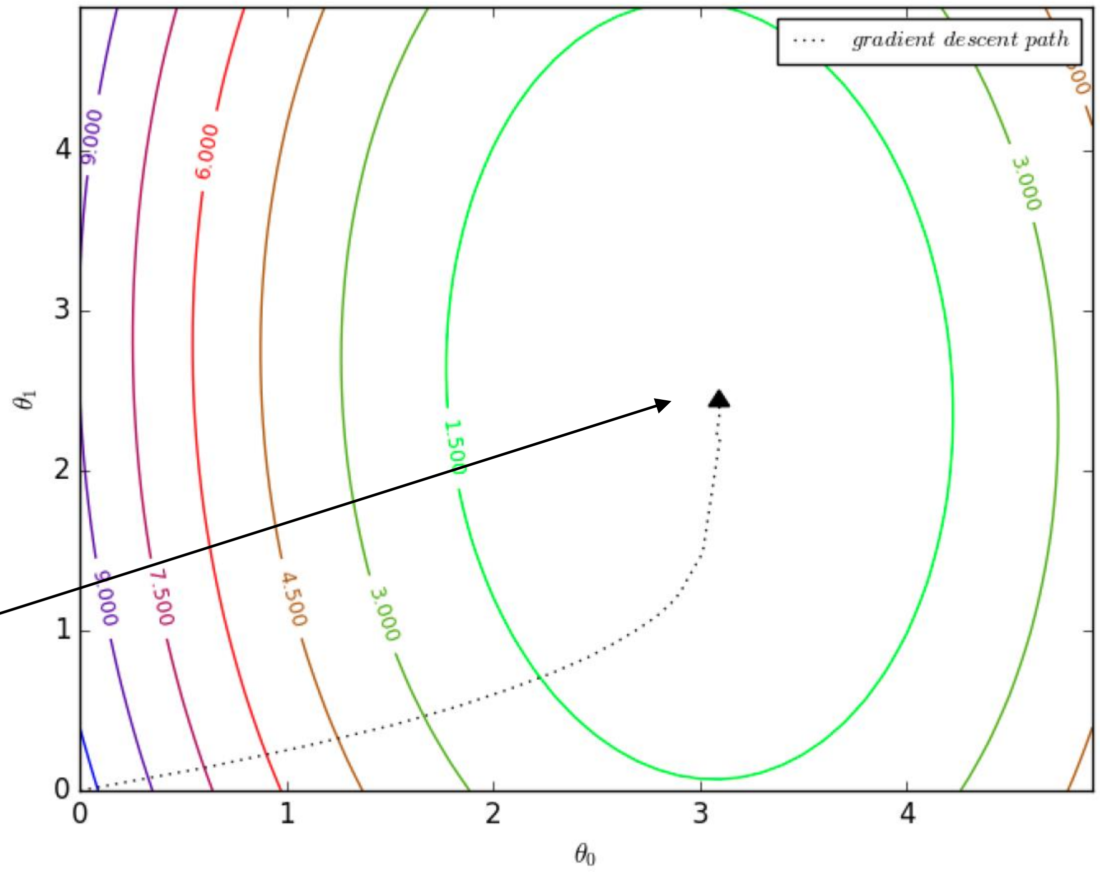
Source: Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: From theory to algorithms. Cambridge university press.

Widrow-Hoff Algorithm: a GD learning rule

Algorithm 1: Widrow-Hoff

initialize $\boldsymbol{w}_1 = \mathbf{0}$;
for $t = 1$ **to** T **do**
  get $\boldsymbol{x}_t \in \mathbb{R}^n$;
  predict $\hat{y}_t = \boldsymbol{w}_t \cdot \boldsymbol{x}_t$;
  observe y_t;
  incur loss of $(\hat{y}_t - y_t)^2$;
  update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta(\boldsymbol{w}_t \cdot \boldsymbol{x}_t - y_t)\boldsymbol{x}_t$;
end

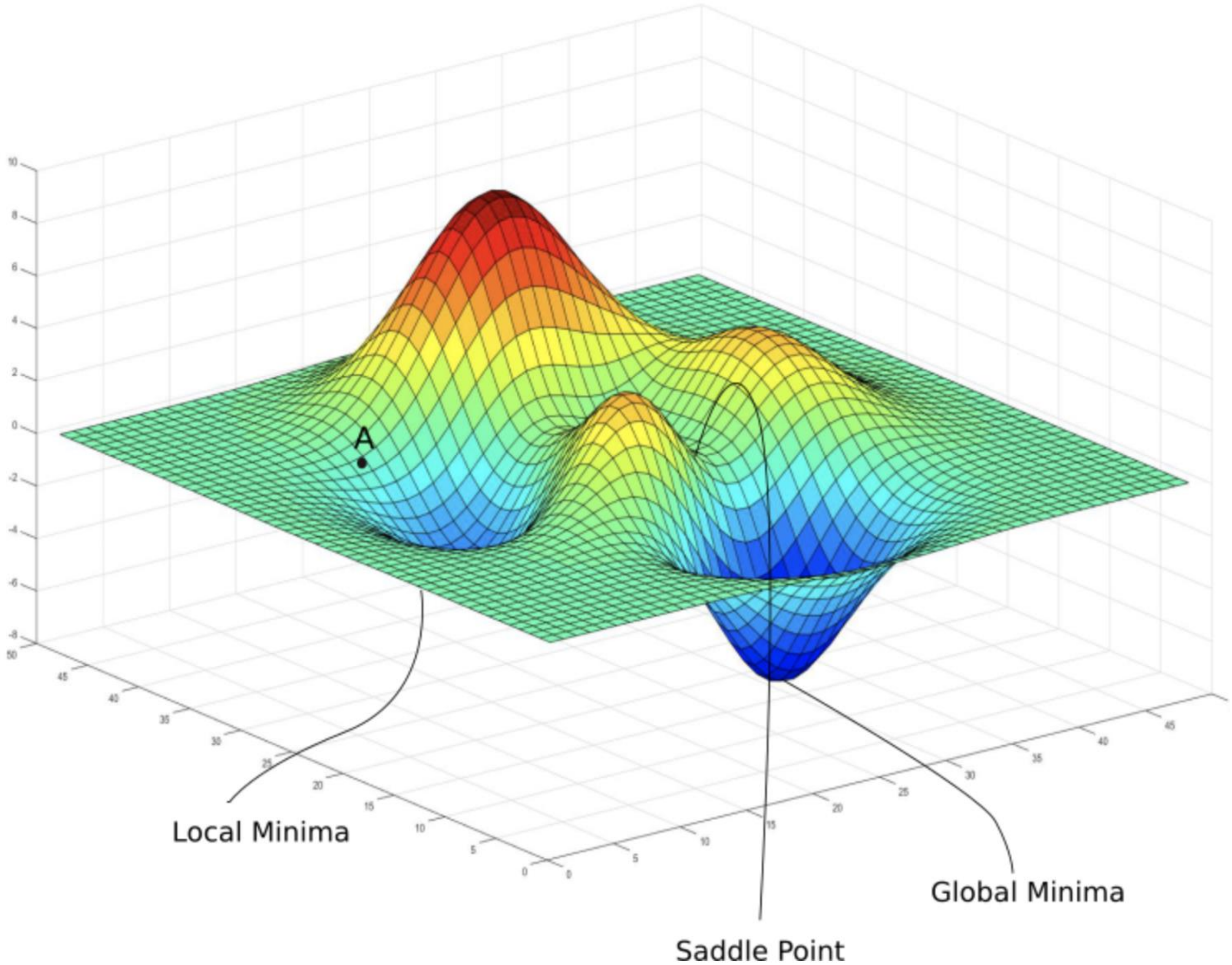
Least Squared Loss function (error)



Iteratively approaches the minimum value

Schapire, R. (2008). Cos 511: Theoretical machine learning. Princeton

Global vs local minima



Source: Tech Talks, gradient descent local minima

Agenda

— Introduction

— Learning problem & linear classification

— Linear models: regression & **logistic regression**

— Non-linear transformation, overfitting & regularization

— Support Vector Machines and kernel learning

— Neural Networks: shallow [and deep]

— Theoretical foundation of supervised learning

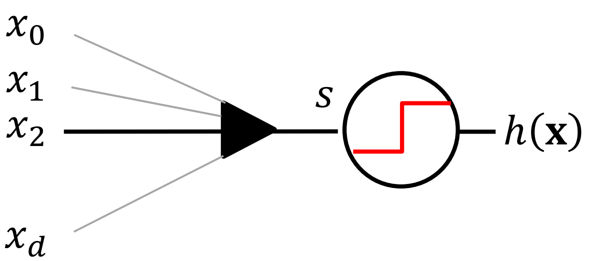
— Unsupervised learning

A third linear model

$$s = \sum_{i=0}^{d} w_i x_i$$

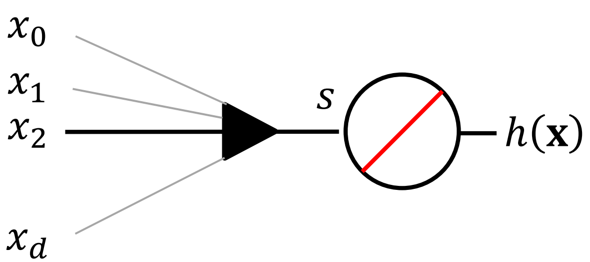
perceptron

$$h(\mathbf{x}) = sign(s)$$



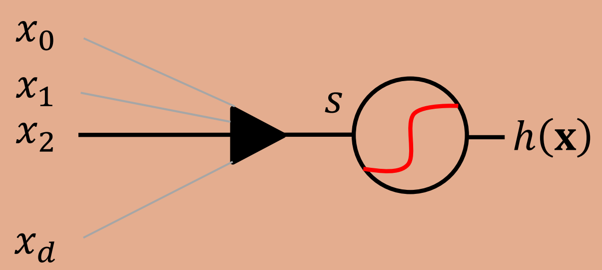
linear regression

$$h(\mathbf{x}) = s$$



logistic regression

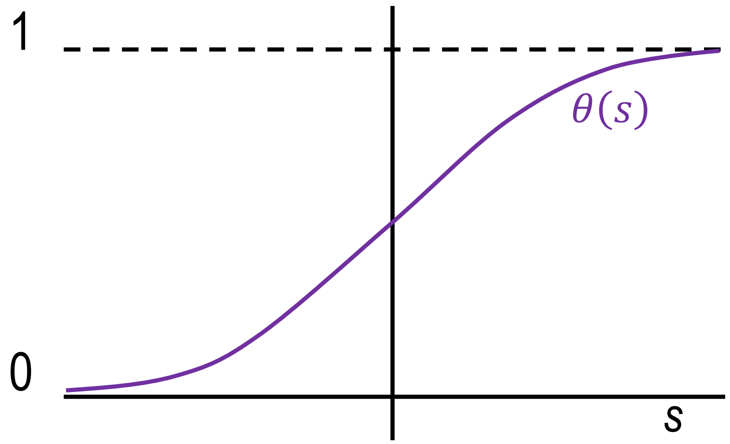
$$h(\mathbf{x}) = \theta(s)$$



The logistic function θ

$$s = \sum_{i=0}^{d} w_i x_i$$

$$\theta(s) = \frac{e^s}{1+e^s}$$ (aka sigmoid)



Probability interpretation

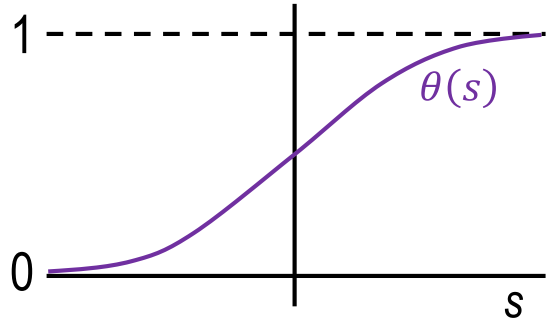
$\mathrm{h}(\mathbf{x}) = \theta(s)$ is interpreted as a probability

Example: Prediction of heart attacks

— Input $\mathbf{x}$ cholesterol level, age, weight, etc.

— $\theta(s)$ probability of a heart attack

— The signal $s = \mathbf{w}^\intercal \mathbf{x}$ "risk score"



Genuine probability

— Data $(\mathbf{x}, y)$ with binary y, generated by a noisy target:

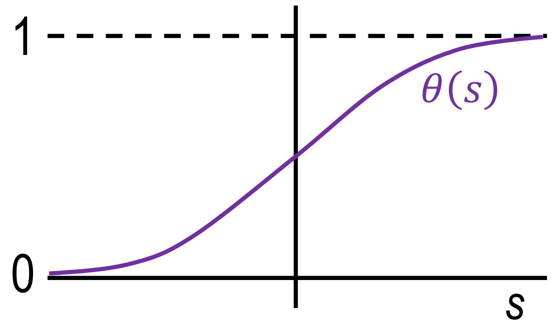
$$p(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & for\ y = +1 \\ 1 - f(\mathbf{x}) & for\ \ y = -1 \end{cases}$$

— The target $f\colon\ \mathbb{R}^d\ \longrightarrow\ [0,1]$ is the probability

— Learn $h(\mathbf{x}) =\ \theta(\mathrm{w}^\intercal\mathbf{x}) \approx f(\mathbf{x})$

Deriving the likelihood

$$p(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & for\; y = +1; \\ 1 - h(\mathbf{x}) & for\; y = -1. \end{cases}$$



Substitute $h(\mathbf{x}) = \theta(\mathrm{w}^\intercal \mathbf{x})$, note $\theta(-s) = 1 - \theta(s)$

$$p(y|\mathbf{x}) = \theta(y\, \mathrm{w}^\intercal \mathbf{x})$$

Likelihood of $\mathcal{D} = (x_1, y_1), \ldots, (x_N, y_N)$ is

$$\prod_{n=1}^{N} p(y_n|x_n) = \prod_{n=1}^{N} \theta(y_n\, \mathrm{w}^\intercal x_n)$$

Maximizing the likelihood defines an error measure

Minimize $$-\frac{1}{N}\ln\left(\prod_{n=1}^{N}\theta(y_n\,\mathrm{w}^\intercal x_n)\right)$$

$$=\frac{1}{N}\sum_{n=1}^{N}\ln\left(\frac{1}{\theta(y_n\,\mathrm{w}^\intercal x_n)}\right)$$

$$\left[\theta(s)=\frac{1}{1+e^{-s}}\right]$$

$$E_{in}(\mathbf{w})=\frac{1}{N}\sum_{n=1}^{N}\underbrace{\ln\left(1+e^{-y_n\,\mathrm{w}^\intercal x_n}\right)}_{e(h(x_n),\,y_n)}$$

How do we minimize E_{in}?

For logistic regression,

$$E_{in}(\mathrm{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \mathrm{w}^\intercal x_n}) \qquad \leftarrow \text{ iterative solution}$$

In general, there is no closed-form solution (for categorical predictors there is, see Lipovetsky, S. (2015). Analytical closed-form solution for binary logit regression by categorical predictors. *Journal of applied statistics*, *42*(1), 37-49.)

Gradient descent can be applied

$$\nabla E_{in} = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \mathrm{x}_n}{1 + e^{y_n \mathrm{w}^\intercal \mathrm{x}_n}}$$

Logistic regression algorithm

Initialize the weights at $t = 0$ to $\mathbf{w}(0)$

for $t = 0,1,2, \ldots$ do

Compute the gradient

$$\nabla E_{in} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathrm{x}_n}{1+ e^{y_n \mathrm{w}^{\intercal} \mathrm{x}_n}}$$

Update the weights $\mathrm{w}(t+1) = \mathrm{w}(t) - \eta \nabla E_{in}$

Iterate to the next step until it is time to stop

Return the final weights **w**

Note: criteria to stop the optimization can be set by a tolerance $\varepsilon = E_{in}^{(t+1)} - E_{in}^{(t)}$

Can we do better?

- Think of Newton's method to find the roots of a function. Assume we have $f: \mathbb{R} \to \mathbb{R}$ (in our case the gradient of the in-sample error) and want to find $f(x) = 0$
- Then Newton's method does the following (iterative) update

$$x \leftarrow x - \frac{f(x)}{f'(x)}$$

Newton-Raphson method

— Our starting point is

$$\mathrm{w}(\mathrm{t}+1) = \mathrm{w}(\mathrm{t}) - \eta \nabla E_{in}(\mathrm{w}(t))$$

— When applying Newton's method

$$\mathrm{w}(\mathrm{t}+1) = \mathrm{w}(\mathrm{t}) - \frac{\nabla E_{in}\big(\mathrm{w}(t)\big)}{\nabla^2 E_{in}\big(\mathrm{w}(t)\big)}$$

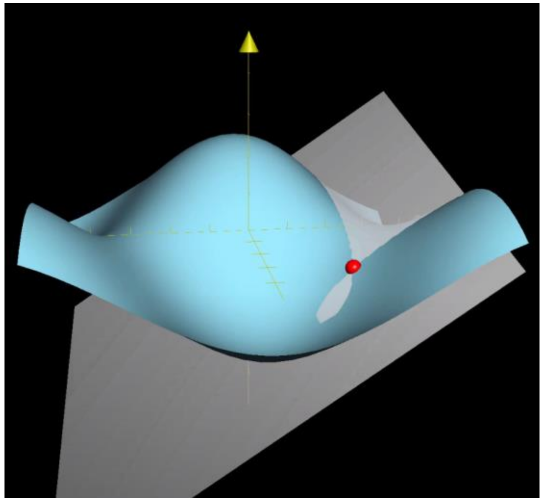
$$= \mathrm{w}(\mathrm{t}) - \Big(\nabla E_{in}\big(\mathrm{w}(t)\big)\Big)^T (\nabla^2 E_{in}\big(\mathrm{w}(t)\big)^{-1}$$

— where H is the Hesse (Hessian) matrix given by

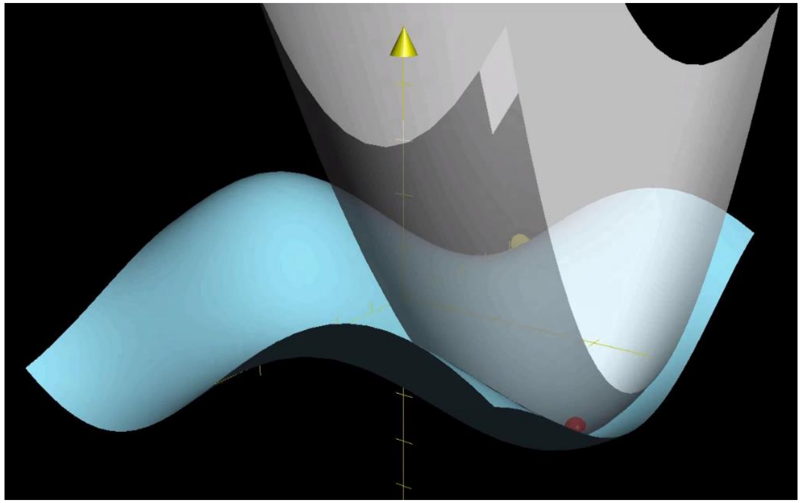
$$H_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} = \nabla^2$$

Gradient descent vs Newton's method

Gradient descent: first order approximation



Newton's method: second order approximation

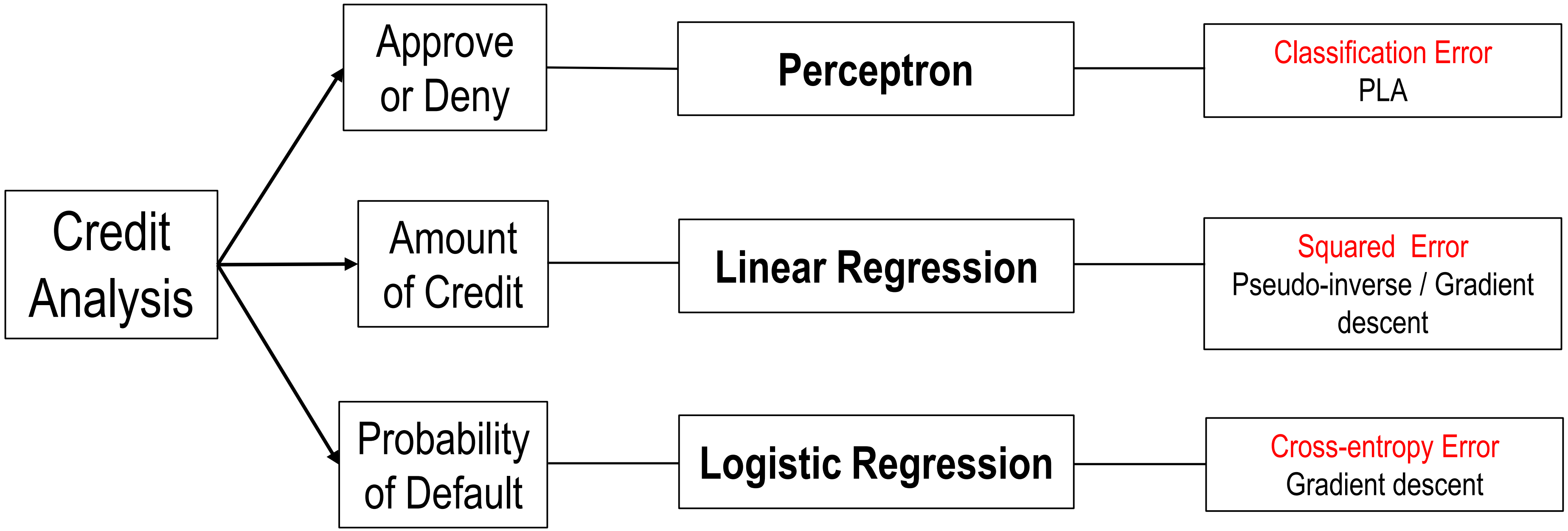


— Computing the Hessian for second-order methods is costly; update time $\boldsymbol{O(d^3)}$

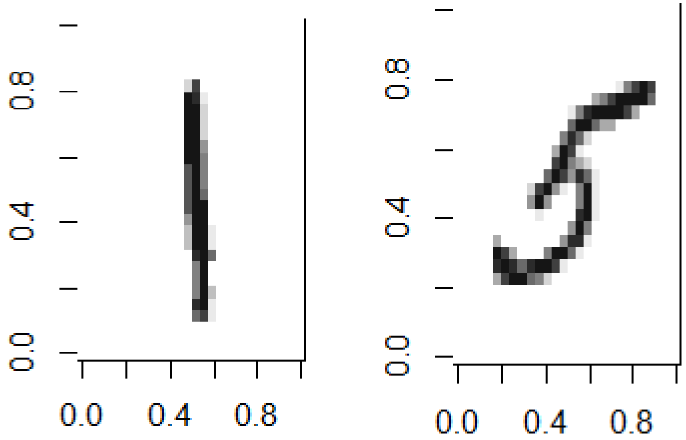
— Quasi-Newton methods exist for approximating the Hessian like BFGS and L-BFGS

Source: Khan Academy

Summary of Linear Models (so far)

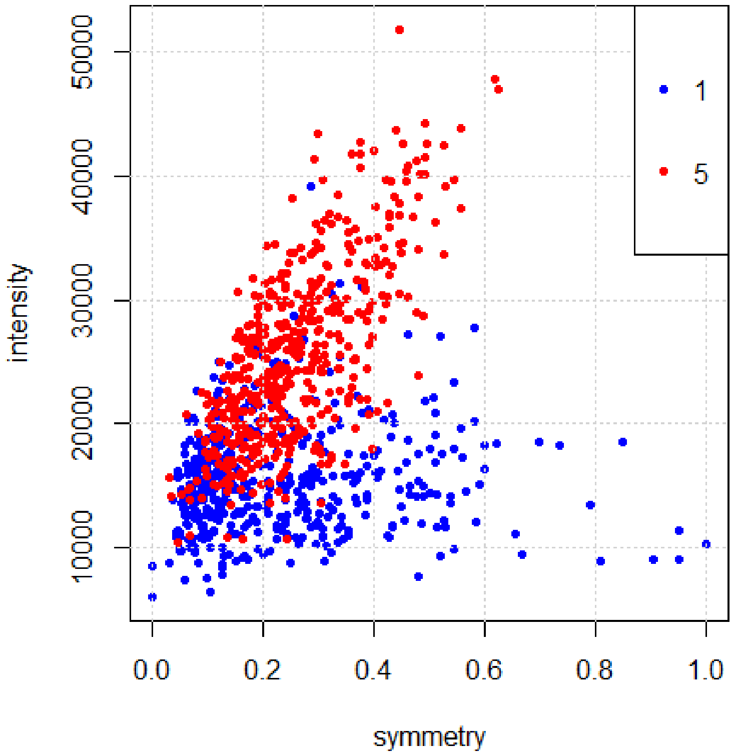


Applying logistic regression



— Task: use MNIST dataset and try to categorize the 1's and 5's against each other

— Use two features: symmetry and intensity



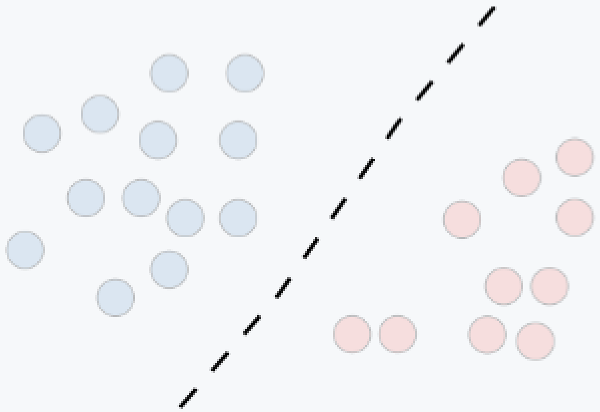
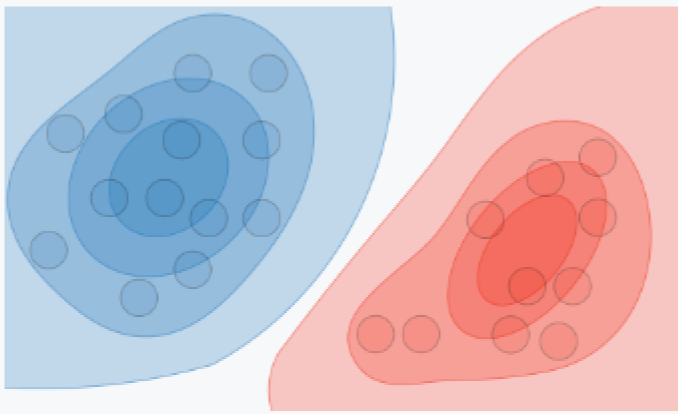
Confusion Matrix

	0	1
0	392	76
1	108	424

Andrew Ng's elephants and dog example

Is there another way to learn whether
an animal is an elephant or a dog?

Elephants and dogs – a bit more formal

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Source: https://i.stack.imgur.com/Xrmqg.png

Discriminative vs. generative classifiers

Discriminative classifiers

— Directly estimate the conditional distribution

$$\boldsymbol{p(y|x)}$$

— We do not attempt to estimate the underlying joint distribution

— Methods include logistic regression, Support Vector Machines

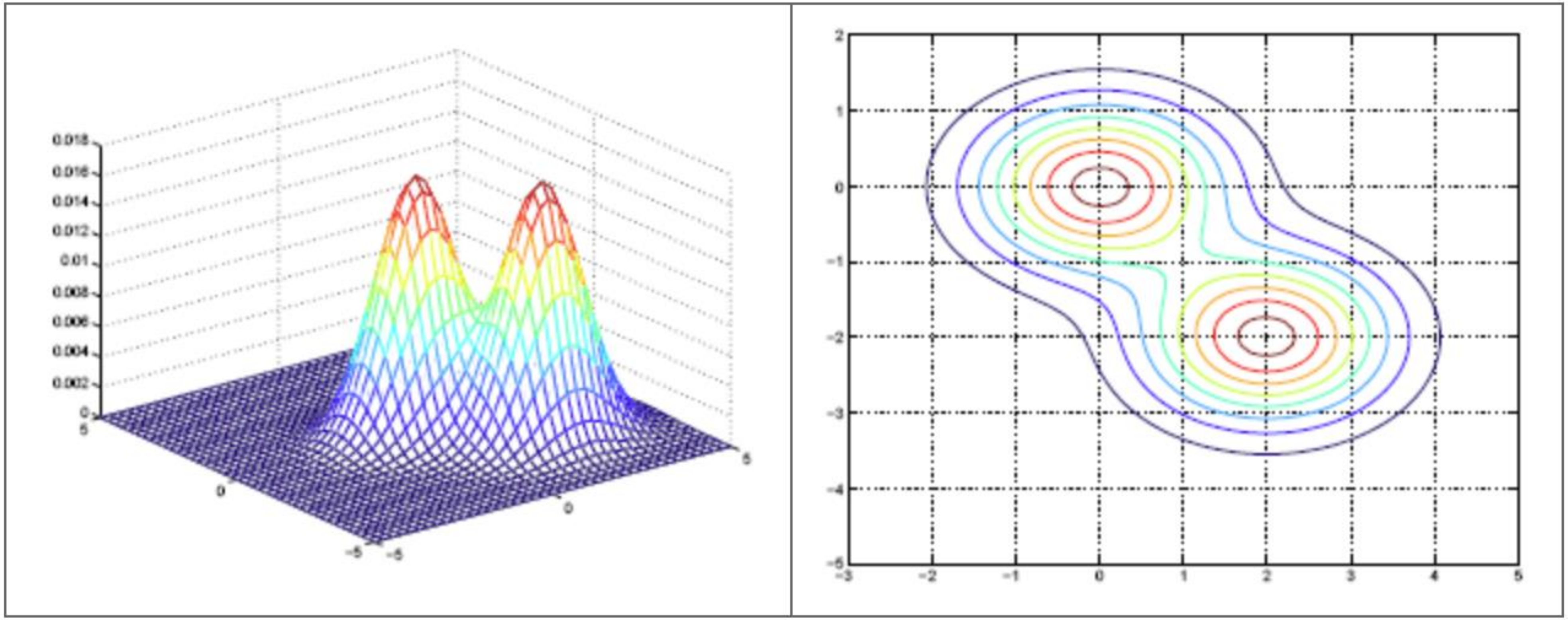
Generative classifiers

— Model joint probability distributions

$$\boldsymbol{p(y,x)}$$

— Methods include Naive Bayes, Discriminant Analysis

Gaussian Discriminant Analysis (GDA)



Source: https://onlinecourses.science.psu.edu/stat857/node/74/

Estimating the unknown parameters

Logistic regression vs. GDA

	LR	GDA
Number of parameters		
Link		
Assumptions		
Robustness		
Efficiency		

Many generative models exist … and can do fancy things

- **Gaussian Mixture (see GDA)**
- Naive Bayes
- Latent Dirichlet Allocation
- Hidden Markov Models
- Restricted Boltzmann Machines
- Variational Autoencoder
- Generative Adversarial Networks

Source: https://phillipi.github.io/pix2pix/

Backup material

— Why do we use the squared error loss?

Probabilistic interpretation – why square error?

— Given x, linear models (**linear in what?**) have the following form

$$y_i = f(x_i) = w^T x_i + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is random, zero-mean noise

— The target probability distribution is then given by

$$p(y_i | x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$$

Likelihood function

— The likelihood is the probability that a fixed set of parameters (often we use θ, a vector, for that) has generated the observed data set

$$\mathcal{L}(w|D) = \prod_{n=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_n - w^T x_n)^2}{2\sigma^2}\right)$$

— To find w of our hypothesis $h(x) = w^T x$ we look for the w that maximizes the likelihood function

$$\mathrm{w} = w_{MLE} = \arg\max_{w} \mathcal{L}(w|D)$$

Finding the best hypothesis (or $\mathbf{w}$)

— In order to find w we maximize $\mathcal{L}(w|D)$. Since log(x) is a monotone function we can take the logarithm and maximize it

$$log\mathcal{L}(w|D) = \quad \log\left(\left(\sigma\sqrt{2\pi}\right)^N\right) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - w^T x_n)^2$$

— After differentiating with respect to w, we get the following form implying that:

maximizing the likelihood => *minimizing the squared loss*

$$w = \arg\min_w \sum_{n=1}^{N}(y_n - w^T x_n)^2$$