

# Intro

...based on Toronto University ML class

# What is learning?

"The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."

Merriam Webster dictionary

L'attività di imparare, nel caso delle macchine queste dovrebbero migliorare con esperienze, valutando ciò in base ad indici di prestazioni

# What is learning?

"The activity or process of gaining knowledge or skill by studying, practicing, being taught, or experiencing something."

Merriam Webster dictionary

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Tom Mitchell

# What is machine learning?

ne sono esempi il riconoscimento dello spam, ma ciò sarebbe meglio se adattivo, ovvero non devo inserire io programmatore tutte le parole potenzialmente usate in un messaggio di spam.

- ▶ For many problems, it's difficult to program the correct behavior by hand
  - ▶ recognizing people and objects
  - ▶ understanding human speech
- ▶ Machine learning approach: program an algorithm to automatically learn from data, or from experience
- ▶ Why might you want to use a learning algorithm?
  - ▶ hard to code up a solution by hand (e.g. vision, speech)
  - ▶ system needs to adapt to a changing environment (e.g. spam detection)
  - ▶ want the system to perform better than the human programmers

# What is machine learning?

- ▶ It's similar to statistics...
  - ▶ Both fields try to uncover patterns in data
  - ▶ Both fields draw heavily on calculus, probability, and linear algebra, and share many of the same core algorithms
- ▶ But it's not statistics!
  - ▶ Stats is more concerned with helping scientists and policymakers draw good conclusions; ML is more concerned with building autonomous agents
  - ▶ Stats puts more emphasis on interpretability and mathematical rigor; ML puts more emphasis on predictive performance, scalability, and autonomy

# Relations to AI

ML è un "pezzo" di IA

- ▶ Nowadays, machine learning is often brought up with artificial intelligence (AI)
- ▶ AI does not always imply a learning based system
  - ▶ Symbolic reasoning
  - ▶ Rule based system
  - ▶ Tree search
  - ▶ etc.
- ▶ Learning based system → learned based on the data → more flexibility, good at solving pattern recognition problems.

## Relations to human learning

- ▶ Human learning is:
  - ▶ Very data efficient
  - ▶ An entire multitasking system (vision, language, motor control, etc.)
  - ▶ Takes at least a few years :)
- ▶ For serving specific purposes, machine learning doesn't have to look like human learning in the end.
- ▶ It may borrow ideas from biological systems, e.g., neural networks.
- ▶ It may perform better or worse than humans.

# What is machine learning

- ▶ Types of machine learning
  - ▶ **Supervised learning:** have labeled examples of the correct behavior
  - ▶ **Reinforcement learning:** learning system (agent) interacts with the world and learns to maximize a scalar reward signal
  - ▶ **Unsupervised learning:** no labeled examples instead, looking for interesting patterns in the data

Nel supervised ho esempi di comportamento corretto (etichette), nel reinforcement learning ho interazione con il mondo esterno ai fini di apprendere. Con unsupervised non ci sono etichette.

Posso migliorare perchè le componenti neurali non sono "lineari", in tal caso potrei fare ben poco.

# History of Machine Learning

- ▶ 1957 - Perceptron algorithm (implemented as a circuit!)
- ▶ 1959 - Arthur Samuel wrote a learning-based checkers program that could defeat him
- ▶ 1969 - Minsky and Papert's book *Perceptrons* (limitations of linear models)
- ▶ 1980s - Some foundational ideas
  - ▶ Connectionist psychologists explored neural models of cognition
  - ▶ 1984 - Leslie Valiant formalized the problem of learning as PAC (Probably Approximately Correct) learning
  - ▶ 1988 - Backpropagation (re-)discovered by Geoffrey Hinton and colleagues
  - ▶ 1988 - Judea Pearl's book *Probabilistic Reasoning in Intelligent Systems* introduced Bayesian networks

# History of Machine Learning

- ▶ 1990s - the “AI Winter”, a time of pessimism and low funding  
But looking back, the '90s were also sort of a golden age for ML research
  - ▶ Markov chain Monte Carlo
  - ▶ variational inference
  - ▶ kernels and support vector machines
  - ▶ boosting
  - ▶ convolutional networks
  - ▶ reinforcement learning
- ▶ 2000s - applied AI fields (vision, NLP, etc.) adopted ML
- ▶ 2010s - deep learning
  - ▶ 2010-2012 - neural nets smashed previous records in speech-to-text and object recognition
  - ▶ increasing adoption by the tech industry
  - ▶ 2016 - AlphaGo defeated the human Go champion
  - ▶ 2018-now - generating photorealistic images and videos
  - ▶ 2020 - GPT3 language model
- ▶ now - increasing attention to ethical and societal implications

# Applications

Computer vision: Object detection, semantic segmentation, pose estimation, and almost every other task is done with ML.



Figure 4. More results of Mask R-CNN on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).



Instance segmentation - [Link](#)



DAQUAR 1553  
What is there in front of the sofa?

Ground truth: table  
IMG+BOW: table (0.74)  
2-VIS+BLSTM: table (0.88)  
LSTM: chair (0.47)



COCOQA 5078  
How many leftover donuts is the red bicycle holding?  
Ground truth: three  
IMG+BOW: two (0.51)  
2-VIS+BLSTM: three (0.27)  
BOW: one (0.29)

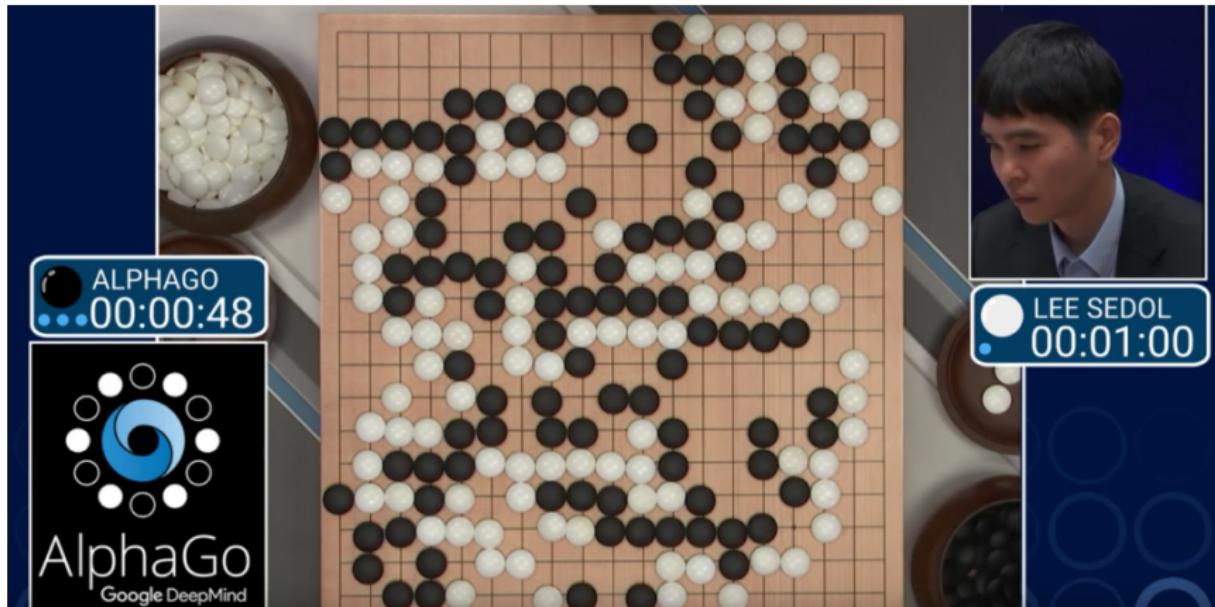
# Applications

Speech: Speech to text, personal assistants, speaker identification...



# Applications

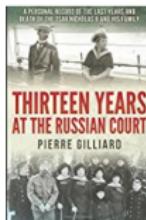
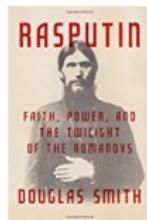
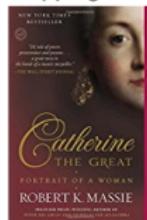
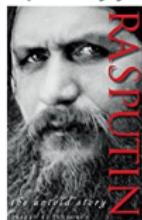
## Playing Games



# Applications

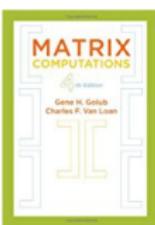
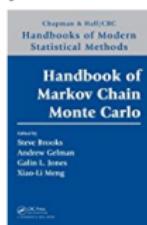
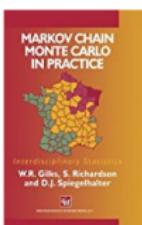
## E-commerce & Recommender Systems : Amazon, netflix, ...

Inspired by your shopping trends



Related to items you've viewed

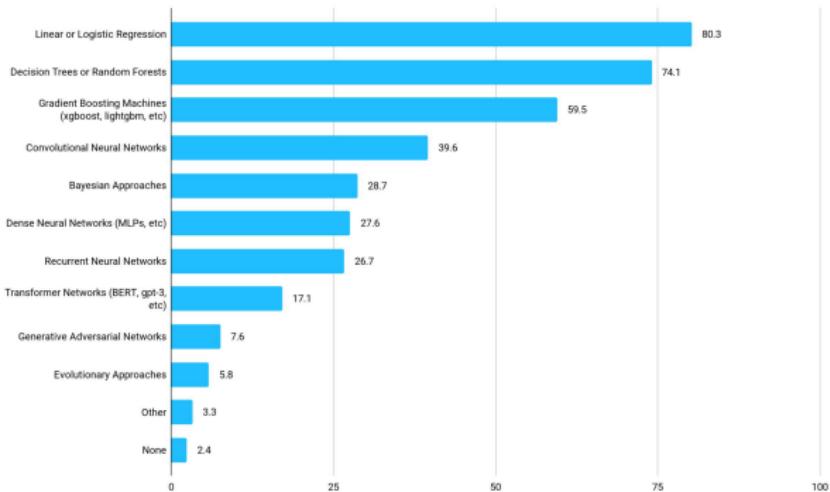
[See more](#)



# Why this class

2022 Kaggle survey of data science and ML practitioners: what data science methods do you use at work?

Methods and Algorithms Usage



# ML Workflow

- ▶ ML workflow sketch:
  1. Should I use ML on this problem?
    - ▶ Is there a pattern to detect?
    - ▶ Can I solve it analytically?
    - ▶ Do I have data?
  2. Gather and organize data.
    - ▶ Preprocessing, cleaning, visualizing.
  3. Establishing a baseline.
  4. Choosing a model, loss, regularization, ...
  5. Optimization
  6. Hyperparameter search.
  7. Analyze performance & mistakes, and iterate back to step 4 (or 2).

# Preliminaries and Nearest Neighbor Methods

...based on Toronto University ML class

# Introduction

- ▶ We start with **supervised learning**.
- ▶ We are given a **training set** consisting of **inputs** and corresponding **labels**, e.g.

Task	Inputs	Labels
object recognition	image	object category
image captioning	image	caption
document classification	text	document category
speech-to-text	audio waveform	text
:	:	:

# Input Vectors

What an image looks like to a computer



05 02 22 97 38 18 00 60 00 75 04 05 07 78 32 12 50 77 01 46
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 44 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 87 05 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 65 05 56 01 32 56 71 37 02 36 93
22 31 16 71 51 63 05 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 31 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 95 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 77 44 20 45 35 14 00 61 33 97 34 31 33 95
72 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 79 33 27 98 66
04 14 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 35 44 12 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 85 34 63 03 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 04 11 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 13 47 48

What the computer sees

→ 82% cat  
15% dog  
2% hat  
1% mug

image classification

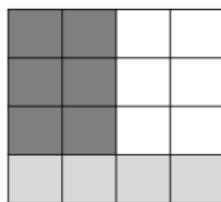
# Input Vectors

- ▶ Machine learning algorithms need to handle lots of types of data: images, text, audio waveforms, credit card transactions, etc.
- ▶ Common strategy: represent the input as an **input vector** in  $\mathbb{R}^d$ 
  - ▶ **Representation** = mapping to another space that's easy to manipulate
  - ▶ Vectors are a great representation since we can do linear algebra!

# Input Vectors

Can use raw pixels

Images  $\leftrightarrow$  Vectors



60	60	255	255
60	60	255	255
60	60	255	255
128	128	128	128



60
60
255
255
60
60
255
255
60
60
255
255
128
128
128
128

Can do much better if you compute a vector of meaningful features.

## Input Vectors

- ▶ Mathematically, our training set consists of a collection of pairs of an input vector  $\mathbf{x} \in \mathbb{R}^d$  and its corresponding target, or label,  $t$ 
  - ▶ Regression:  $t$  is a real number (e.g. stock price)
  - ▶ Classification:  $t$  is an element of a discrete set  $\{1, \dots, C\}$
- ▶ Denote the training set  $\{(\mathbf{x}^{(1)}; t^{(1)}), \dots, (\mathbf{x}^{(N)}; t^{(N)})\}$

# Nearest Neighbors

di tipo supervised, parto da set di dati (input,img,text che vedo come vettore) ed etichette.

- ▶ Suppose we're given a novel input vector  $\mathbf{x}$  we'd like to classify.
- ▶ The idea: find the nearest input vector to  $\mathbf{x}$  in the training set and copy its label.
- ▶ Can formalize nearest in terms of Euclidean distance

$$\|\mathbf{x}^{(a)} - \mathbf{x}^{(b)}\|_2 = \sqrt{\sum_{j=1}^d (x_j^{(a)} - x_j^{(b)})^2}$$

un esempio è la categorizzazione di animali partendo da una immagine

## Algorithm

- 1 Find example  $(\mathbf{x}^*, t^*)$  (from the stored training set) closest to  $\mathbf{x}$ . That is:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}^{(i)} \in \text{training set}} \text{distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

- 2 Output  $y = t^*$

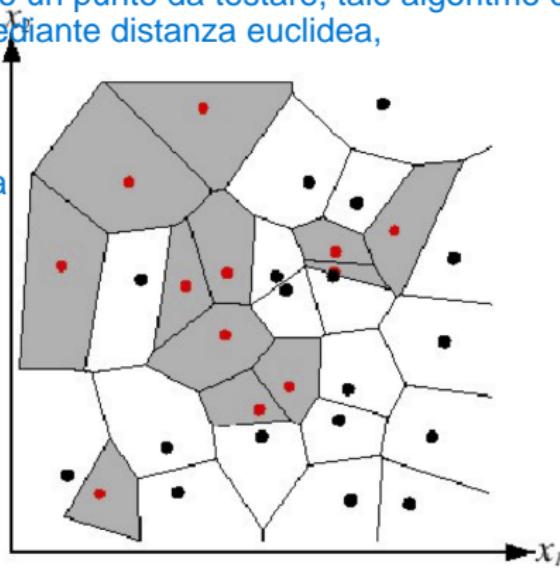
il set di dati training è come se fossero degli "esempi".

# Nearest Neighbors: Decision Boundaries

We can visualize the behavior in the classification setting using a Voronoi diagram.

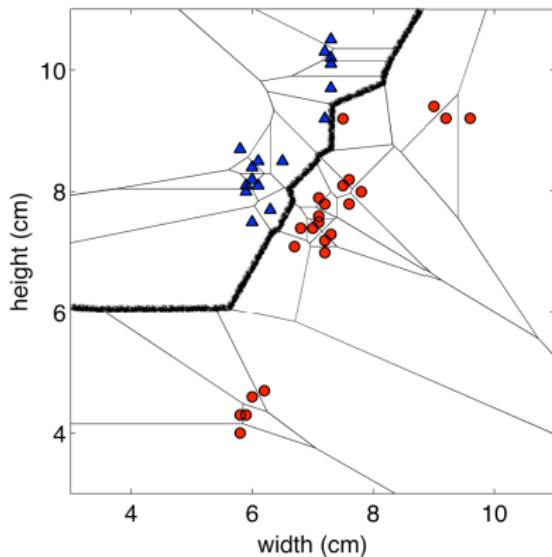
nel training abbiamo delle coppie di punti a cui associo una etichetta (ad esempio true/false). Se viene fornito un punto da testare, tale algoritmo cerca il punto più vicino al nuovo punto inserito mediante distanza euclidea, vede la sua etichetta, ed userà questa per il nuovo punto.

Devo calcolare la distanza da ogni punto, non so a priori quale sia il più vicino!



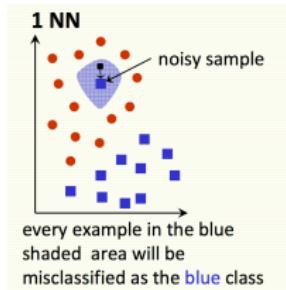
# Nearest Neighbors: Decision Boundaries

Decision boundary: the boundary between regions of input space assigned to different categories.



In questa variante non lavoro in aree, ma con una linea di delimitazione.

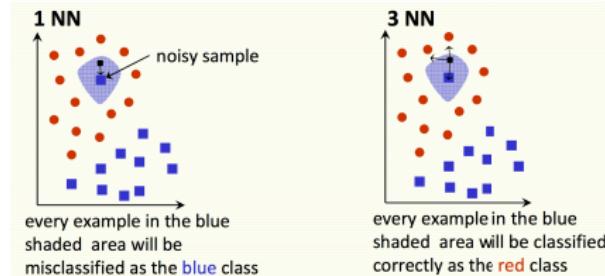
# Nearest Neighbors



- ▶ Nearest neighbors sensitive to noise or mis-labeled data ("class noise").

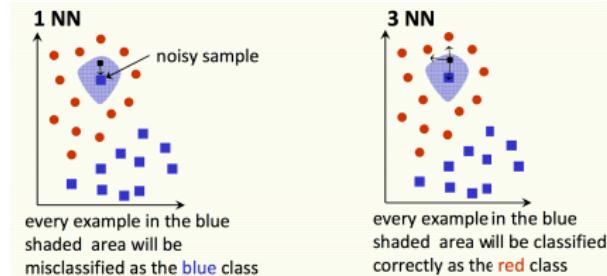
Nella pratica questo algoritmo va bene se non ci sono errori nel training (ad esempio ho sbagliato una label o una posizione). Se così non fosse, tale rumore si propaga molto velocemente.

# Nearest Neighbors



- ▶ Nearest neighbors sensitive to noise or mis-labeled data (“class noise”). Solution?

# Nearest Neighbors



- ▶ Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- ▶ Smooth by having  $k$ -nearest neighbors vote

## Algorithm

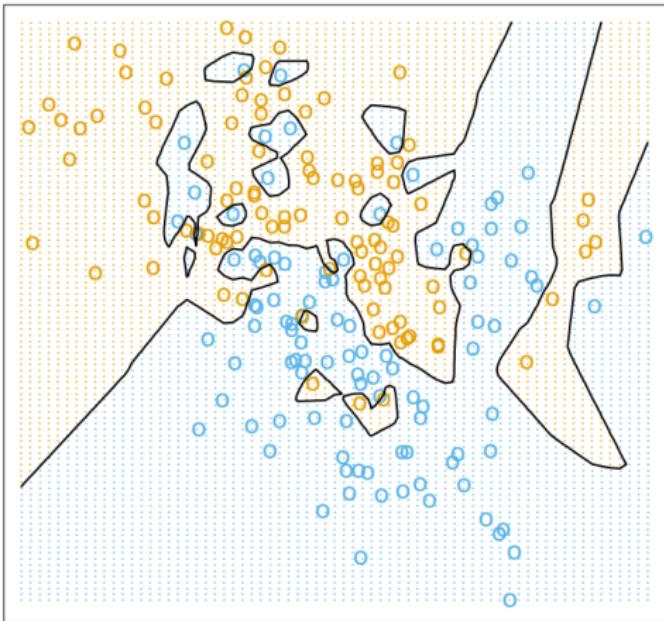
- 1 Find  $k$  examples  $(x^{(i)}, t^{(i)})$  closest to the test instance  $x$ .
- 2 Classification output is majority class

$$y = \arg \max_c \sum_{i=1}^k \mathbb{I}_{\{t^{(i)}=c\}}$$

*funzione indicatrice*

# K-Nearest Neighbors

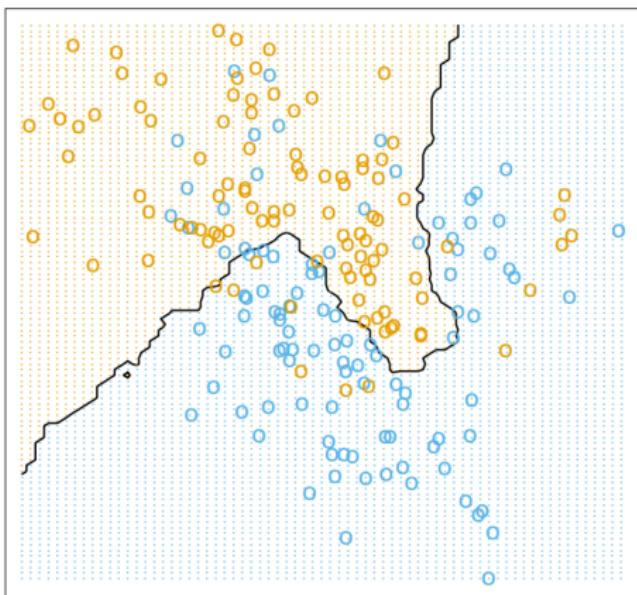
$k=1$



# K-Nearest Neighbors

in questa variante non vedo l'etichetta del nodo più vicino, ma l'etichetta maggiormente presente nei k nodi vicini al nodo. Più k cresce, più la demarcazione è rigida.

k=15



# K-Nearest Neighbors

Come scelgo il k migliore? Parliamo di "k migliore" perchè non possiamo parlare di k corretto.

Tradeoffs in choosing k? k non è "ottimizzabile" grazie al learning!

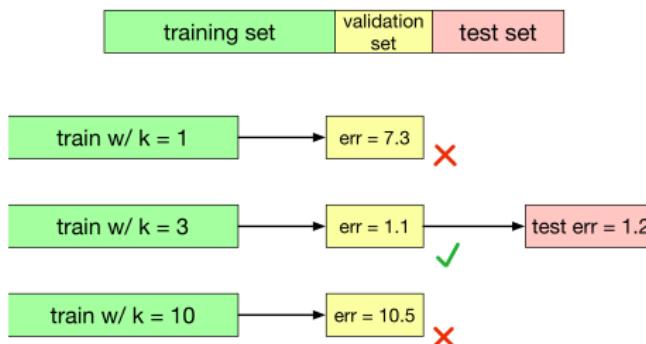
- ▶ Small  $k$ 
  - ▶ Good at capturing fine-grained patterns
  - ▶ May overfit, i.e. be sensitive to random idiosyncrasies in the training data (classico unicamente in base al nodo più vicino) (ipersensibilità)
- ▶ Large  $k$ 
  - ▶ Makes stable predictions by averaging over lots of examples
  - ▶ May underfit, i.e. fail to capture important regularities
- ▶ Balancing  $k$  predizioni stabili, ma non cattura bene condizioni particolari
  - ▶ Optimal choice of  $k$  depends on number of data points  $n$ .
  - ▶ Nice theoretical properties if  $k \rightarrow \infty$  and  $\frac{k}{n} \rightarrow 0$ .
  - ▶ Rule of thumb: choose  $k < \sqrt{n}$ . (metodo approssimativo)
  - ▶ We can choose  $k$  using validation set (next slides).

approccio migliore

# K-Nearest Neighbors

- ▶ We would like our algorithm to **generalize** to data it hasn't seen before. *k deve comportarsi bene anche con dati che non vede!*
- ▶ We can measure the **generalization error** (error rate on new examples) using a **test set**.
- ▶ ***k* is an example of a hyperparameter, something we can't fit as part of the learning algorithm itself**
- ▶ We can tune hyperparameters using a **validation set**:

la scelta del validation set può portare a risultati diversi!

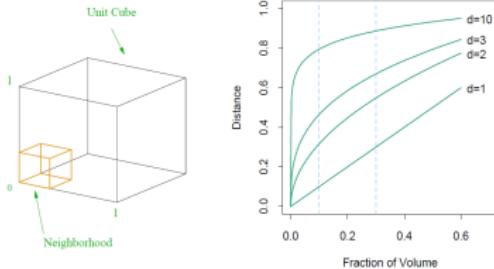


I'idea è questa:  
io non posso "vedere" il  
test set più di una volta.  
Quindi posso usarlo  
solo se so già quale "k"  
usare. Se non usassi il  
validation set, dovrei  
sperare che il k usato  
nell'allenamento vada  
bene!

- ▶ The **test set is used only at the very end**, to measure the generalization performance of the final configuration.

# Pitfalls: The Curse of Dimensionality

- ▶ Low-dimensional visualizations are misleading! In high dimensions, "most" points are **far apart.** se i punti sono distanti di loro, quanto sono affidabili?
- ▶ If we want the nearest neighbor to be closer than  $\epsilon$ , **how many points do we need to guarantee it?**
  - ▶ The volume of a single ball of radius  $\epsilon$  is  $\mathcal{O}(\epsilon^d)$
  - ▶ The total volume of  $[0, 1]^d$  is 1.
  - ▶ Therefore  $\mathcal{O}((\frac{1}{\epsilon})^d)$  balls are needed to cover the volume.



presa una pallina di raggio epsilon, essa ha volume pari a  $\epsilon^d$ .

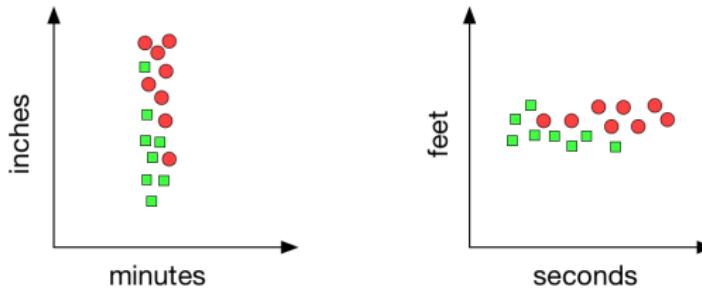
Il volume totale dello spazio (non della pallina) è 1.

Allora per coprire questo volume di 1 servono  $(1/\epsilon)^d$  palline, perchè  $n \cdot \text{volume palline} = 1$

# Pitfalls: Normalization

- ▶ Nearest neighbors can be sensitive to the ranges of different features.
- ▶ Often, the units are arbitrary:

devo normalizzare e avere valori adimensioni, lavorare in km/h è diverso rispetto a m/s!



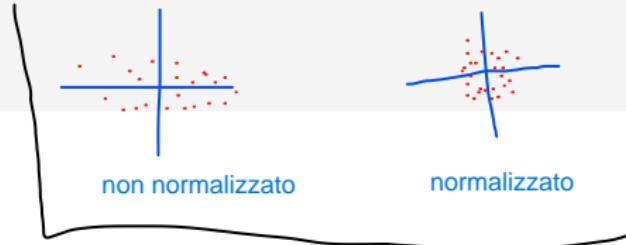
- ▶ Simple fix: **normalize** each dimension to be zero mean and unit variance, i.e., compute the mean  $\mu_j$  and standard deviation  $\sigma_j$ , and consider  $\tilde{x}_j$  in place of the original  $x_j$ , where

così standardizzato ho media 0 e varianza 1

$$\tilde{x}_j = \frac{x_j - \mu_j}{\sigma_j} \quad \text{in questa forma è adimensionale}$$

$\sigma_j \leftarrow \text{dev. std.}$

## Pitfalls: Computational Cost



- ▶ Number of computations at **training time**: 0 (aggiungo solo punti e labels)
- ▶ Number of computations at **test time**, per query (naive algorithm)
  - ▶ Calculate  $D$ -dimensional Euclidean distances with  $N$  data points:  
 $\mathcal{O}(ND)$  ho tale costo perchè devo calcolare la distanza per ogni punto  
e poi ordinare queste distanze
  - ▶ Sort the distances:  $\mathcal{O}(N \log N)$
- ▶ This must be done for *each* query, which is very expensive by the standards of a learning algorithm!
- ▶ Need to store the entire dataset in memory! Tons of work has gone into algorithms and data structures for efficient nearest neighbors with high dimensions and/or large datasets.

## Conclusions

- ▶ Simple algorithm that does all its work at test time - in a sense, no learning!
- ▶ Can control the complexity by varying  $k$
- ▶ Suffers from the Curse of Dimensionality
- ▶ Next: parametric models, which learn a compact summary of the data rather than referring back to it at test time.

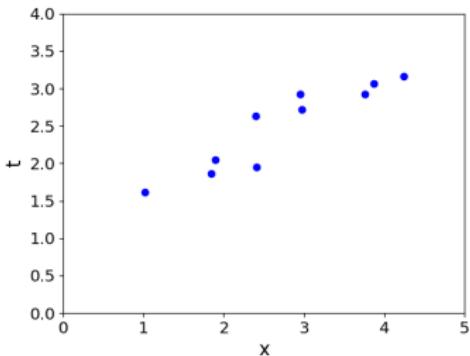
# Linear Regression

...based on Toronto University ML class

# Overview

- ▶ Second learning algorithm of the course: linear regression.
  - ▶ Task: predict scalar-valued targets (e.g. stock prices)
  - ▶ ...as linear function of the inputs
- ▶ While KNN was a complete algorithm, linear regression exemplifies a modular approach that will be used throughout this course:
  - ▶ choose a model describing the relationships between variables of interest
  - ▶ define a loss function quantifying how bad the fit to the data is
  - ▶ choose a regularizer saying how much we prefer different candidate models (or explanations of data)
  - ▶ fit a model that minimizes the loss function and satisfies the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- ▶ Mixing and matching these modular components give us a lot of new ML methods.

# Supervised Learning Setup



In supervised learning:

- ▶ There is input  $x \in \mathcal{X}$ , typically a vector of features (or covariates)
- ▶ There is target  $t \in \mathcal{T}$  (also called response, outcome, output, class)
- ▶ Objective is to learn a function  $f : \mathcal{X} \rightarrow \mathcal{T}$  such that  $t \approx y = f(x)$  based on some data  $\mathcal{D} = \{(x^{(i)}, t^{(i)})\}$  for  $i = 1, 2, \dots, N\}$

$t$  non deve essere uguale a  $t$ , ma "simile", questo perchè, come vedremo, se una funzione prende esattamente tutti i dati nel training, potrebbe essere troppo rigida e non matchare bene nel testing.

## Linear Regression - Model

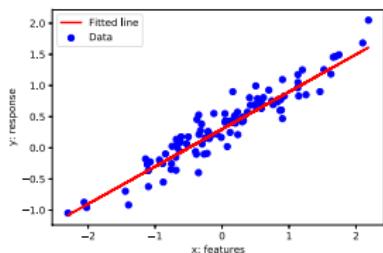
- ▶ Model: In linear regression, we use a linear function of the features  $\mathbf{x} = (x_1, \dots, x_D) \in \mathbb{R}^D$  to make predictions  $y$  of the target value  $t \in R$ :

D, la dimensione dello spazio,  
è anche il numero di features.

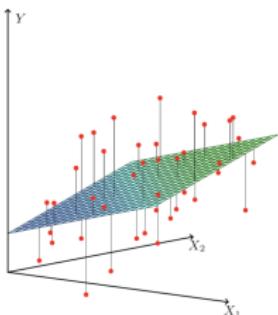
$$y = f(\mathbf{x}) = \sum_j w_j x_j + b$$

- ▶  $y$  is the prediction
- ▶  $w$  is a vector of weights
- ▶  $b$  is the bias (or intercept)
- ▶ **w and b together are the parameters**
- ▶ We hope that our prediction is close to the target:  $y \approx t$ .

# What is linear? 1 feature vs D features



- ▶ if we have only 1 feature  
 $y = wx + b$  where  $w, x$  and  $b \in \mathbb{R}$
- ▶  $y$  is linear in  $x$



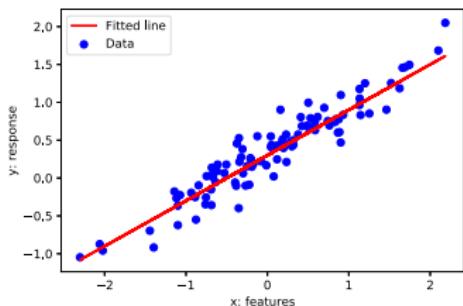
- ▶ if we have  $D$  features  
 $y = \mathbf{w}^\top \mathbf{x} + b$  where  $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D$   
 $b \in \mathbb{R}$
- ▶  $y$  is linear in  $\mathbf{x}$

Relation between the prediction  $y$  and inputs  $\mathbf{x}$  is linear in both cases.

# Linear Regression

We have a dataset  $\mathcal{D} = \{(x^{(i)}, t^{(i)}) \text{ for } i = 1, 2, \dots, N\}$  where

- ▶  $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_D^{(i)})^\top \in \mathbb{R}^D$  are the inputs (e.g. age, height)
- ▶  $t^{(i)} \in \mathbb{R}$  is the target or response (e.g. income)
- ▶ predict  $t^{(i)}$  with a linear function of  $x^{(i)}$ :



- ▶  $t^{(i)} \approx y^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$
- ▶ Different  $(\mathbf{w}, b)$  define different lines.
- ▶ We want the “best” line  $(\mathbf{w}, b)$ .
- ▶ How to quantify “best”?

# Linear Regression - Loss Function

Noi lavoriamo sempre con modello lineare, anche se i dati non lo sono. Quindi useremo sempre queste tecniche.

- ▶ A **loss function**  $\mathcal{L}(y, t)$  defines how bad it is if, for some example  $x$ , the algorithm predicts  $y$ , but the target is actually  $t$ .
- ▶ **Squared error loss function:**

Questa funzione è meglio di  $|y-t|$ , perché amplifica errori grandi.

La derivata di  $L(y,t)$  è lineare.

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

Con  $|y-t|$  ho 10 valori:

- tutti hanno  $y = 0.1$  e non  $t = 0$ : la funzione ritorna 1.

- tutti hanno  $y=0$  tranne uno con  $y=1$ , la funzione ritorna 1.

Con  $(y-t)^2$ :

nel primo caso ho 0.01, nel secondo 1

- ▶  $y - t$  is the **residual**, and we want to make this small in magnitude
- ▶ The  $\frac{1}{2}$  factor is just to make the calculations convenient. (se derivo si toglie)
- ▶ **Cost function:** loss function averaged over all training examples

nelle parentesi c'è una funzione convessa, avente unico punto di minimo.

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - t^{(i)})^2 \quad \text{media dei loss}$$

$$= \frac{1}{2N} \sum_{i=1}^N (\mathbf{w}^\top \mathbf{x}^{(i)} + b - t^{(i)})^2$$

- ▶ Terminology varies. Some call "cost" empirical or average loss.

# Vectorization

anche se il problema è semplice, ciò non vale per la scrittura, soprattutto in python, dove i loop risultano lenti ed è meglio evitarli.

- ▶ Notation-wise,  $\frac{1}{2N} \sum_{i=1}^N (y^{(i)} - t^{(i)})^2$  gets messy if we expand  $y^{(i)}$ :

$$\frac{1}{2N} \sum_{i=1}^N \left( \sum_{j=1}^D (w_j x_j^{(i)} + b) - t^{(i)} \right)^2$$

- ▶ The code equivalent is to compute the prediction using a for loop:

```
y = b  
for j in range(M):  
    y += w[j] * x[j]
```

- ▶ Excessive super/sub scripts are hard to work with, and Python loops are slow, so we **vectorize** algorithms by expressing them in terms of vectors and matrices.

$$\mathbf{w} = (w_1, \dots, w_D)^\top, \mathbf{x} = (x_1, \dots, x_D)^\top, y = \mathbf{w}^\top \mathbf{x} + b$$

- ▶ This is simpler and executes much faster:

tale funziona risulta molto più veloce rispetto al loop

```
y = np.dot(w, x) + b
```

questo perchè l'hardware può eseguire il prodotto riga per colonna in parallelo (basta avere una unità di calcolo per ogni riga). Col loop è in sequenza.

# Vectorization

## Why vectorize?

- ▶ The equations, and the code, will be simpler and more readable.  
Gets rid of dummy variables/indices!
- ▶ Vectorized code is much faster
  - ▶ Cut down on Python interpreter overhead
  - ▶ Use highly optimized linear algebra libraries (hardware support)
  - ▶ Matrix multiplication very fast on GPU (Graphics Processing Unit)

# Vectorization

- We can organize all the training examples into a **design matrix  $\mathbf{X}$**  with **one row per training example**, and all the targets into the **target vector  $\mathbf{t}$** .

one feature across  
all training examples

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \mathbf{x}^{(3)\top} \end{pmatrix} = \begin{pmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{pmatrix}$$

in una riga ho un singolo training.  
in una colonna ho, per una certa feature, tutti i risultati ottenuti dai vari training.

one training example (vector)

è come dire che se faccio dei test in cui ho label [sesso|età], per ogni riga ho le persone con sesso ed età, e nella colonna 'età' vedo tutte le età delle persone nel training.

- Computing the predictions for the whole dataset:

$$\mathbf{X}\mathbf{w} + b\mathbf{1} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \mathbf{y}$$

generalmente questa scrittura non piace,  
perchè può essere semplificata.

# Vectorization

- ▶ Computing the squared error cost across the whole dataset:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} \parallel \mathbf{y} - \mathbf{t} \parallel^2 \quad (\text{norma, cioè lunghezza vettore})$$

- ▶ We can also add a column of 1's to design matrix, combine the bias and the weights, and conveniently write

$$\mathbf{X} = \begin{bmatrix} 1 & [\mathbf{x}^{(1)}]^\top \\ \vdots & \vdots \\ 1 & [\mathbf{x}^{(N)}]^\top \end{bmatrix} \in \mathbb{R}^{N \times (D+1)} \text{ and } \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_N \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our prediction reduces to  $\mathbf{y} = \mathbf{X}\mathbf{w}$ . IN FORMA VETTORIALE

passo da 'D' a 'D+1', quindi lavoro in questo spazio di una dimensione in più. La differenza è che in X aggiungo una prima colonna di tutti '1', e in 'w' un peso pari a 'b', questo ci permette di mettere tutto insieme, mentre prima era separato.

# Solving the Minimization Problem

We defined a cost function. This is what we'd like to minimize.

Two commonly applied mathematical approaches:

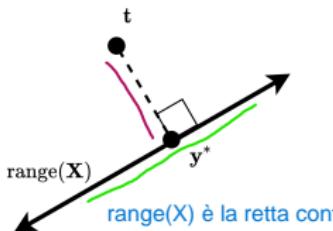
- ▶ Algebraic, e.g., using inequalities:
  - ▶ to show  $z^*$  minimizes  $f(z)$ , show that  $\forall z, f(z) \geq f(z^*)$
  - ▶ to show that  $a = b$ , show that  $a \geq b$  and  $b \geq a$
- ▶ Calculus: minimum of a smooth function (if it exists) occurs at a critical point, i.e. point where the derivative is zero (or equivalently the gradient).
  - ▶ multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).

Solutions may be direct or iterative

- ▶ Sometimes we can directly find provably optimal parameters (e.g. set the gradient to zero and solve in closed form). We call this a direct solution.
- ▶ We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

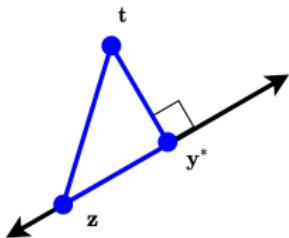
# Direct Solution I: Linear Algebra

- We seek  $w$  to minimize  $\| \mathbf{X}w - t \| ^2$
- $\text{range}(\mathbf{X}) = \{ \mathbf{X}w | w \in \mathbb{R}^D \}$  is a  $D$ -dimensional subspace of  $\mathbb{R}^N$ .
- Recall that the closest point  $y^* = \mathbf{X}w^*$  in subspace  $\text{range}(\mathbf{X})$  of  $\mathbb{R}^N$  to arbitrary point  $t \in \mathbb{R}^N$  is found by orthogonal projection.



al variare del vettore 'pesi' cosa succede?  
X si muove sulla retta, per un certo vettore 'w' troviamo  $y^* = Xw$ .  
Il punto 't' più vicino a tale punto forma un angolo retto, per il teorema di Pitagora.

- We have  $(\underline{y^*} - \underline{t}) \perp \underline{\mathbf{X}w}, \forall w \in \mathbb{R}^D$



- Why is  $y^*$  the closest point to  $t$ ?
  - Consider any  $z = \mathbf{X}w$
  - By Pythagorean theorem and the trivial inequality ( $x^2 \geq 0$ ):

$$\begin{aligned}\| z - t \| ^2 &= \| y^* - t \| ^2 + \| y^* - z \| ^2 \\ &\geq \| y^* - t \| ^2\end{aligned}$$

# Direct Solution I: Linear Algebra

due vettori ortogonali hanno il loro prodotto scalare = 0 per ogni colonna (componente)

- ▶ From the previous slide, we have  $(\mathbf{y}^* - \mathbf{t}) \perp \mathbf{Xw}, \forall \mathbf{w} \in \mathbb{R}^D$
- ▶ Equivalently, the columns of the design matrix  $\mathbf{X}$  are all orthogonal to  $(\mathbf{y}^* - \mathbf{t})$ , and we have that:

$(\mathbf{X}^\top \mathbf{T})$  va sempre prima!

qui sto dicendo che tutte LE COLONNE di  $\mathbf{X}$  sono ortogonali, per questo faccio la trasposta, per poter fare il prodotto. Il prodotto tra due matrici prevede di moltiplicare la riga della prima matrice per la colonna della seconda. Visto che noi vogliamo vedere che le colonne della prima siano ortogonali, faccio la trasposta per lavorare su di loro.

$$\mathbf{X}^\top (\mathbf{y}^* - \mathbf{t}) = 0$$

$$\mathbf{X}^\top \mathbf{X} \mathbf{w}^* - \mathbf{X}^\top \mathbf{t} = 0$$

$$(\mathbf{X}^\top \mathbf{X}) \mathbf{w}^* = \mathbf{X}^\top \mathbf{t}$$

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$$

qui stiamo operando in forma VETTORIALE, e la formula di  $y = \mathbf{Xw}$  usata qui è diversa da quella scalare.

Soluzione ottimale della funzione costo!

- ▶ While this solution is clean and the derivation easy to remember, like many algebraic solutions, it is somewhat ad hoc.
- ▶ On the hand, the tools of calculus are broadly applicable to differentiable loss functions...

il problema di questo approccio risiede nel fatto che l'operazione di inversa è soggetta ad errori di stabilità numerica, ovvero possiamo ottenere risultati sbagliati anche partendo da dati corretti. Inoltre, normalmente  $\mathbf{X}^\top \mathbf{T}$  non è quadrata, e solo in casi particolari vale la proprietà commutativa (vale solo se moltiplico per la sua inversa, per matrice identità, o matrice nulla).

## Direct Solution II: Calculus

altro approccio è con le derivate parziali. Il punto 'ottimale' lo si ha se tutte le derivate sono nulle.

- ▶ **Partial derivative:** derivative of a multivariate function with respect to one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \rightarrow 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

- ▶ To compute, take the single variable derivative, pretending the other arguments are constant.
- ▶ Example: partial derivatives of the prediction  $y$

$$\begin{aligned}\frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j} \left[ \sum_{j'} w_{j'} x_{j'} + b \right] \\ &= x_j\end{aligned}$$

$$\begin{aligned}\frac{\partial y}{\partial b} &= \frac{\partial}{\partial b} \left[ \sum_{j'} w_{j'} x_{j'} + b \right] \\ &= 1\end{aligned}$$

in questo caso, rimangono soltanto gli  $x(j)$ , perchè derivo rispetto  $w$  (la sua derivata è 1) e  $b$  è costante, quindi la sua derivata è 0.

qui deriviamo rispetto  $b$ , la derivata di  $b$  rispetto  $b$  è 1, il resto, non dipendendo da ' $b$ ', ha derivata nulla.

## Direct Solution II: Calculus

qui lavoriamo lungo le varie componenti, quindi non abbiamo  $y = Xw$  vettore, ma  $y = \text{sum}(X^*w) + b$   
quindi è come se operassimo con degli scalari!

- ▶ For loss derivatives, apply the chain rule:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_j} &= \frac{d\mathcal{L}}{dy} \frac{\partial y}{\partial w_j} \quad X_j w_j + b \\ &= \frac{d}{dy} \left[ \frac{1}{2}(y - t)^2 \right] \cdot x_j \\ &= (y - t)x_j \\ \frac{\partial \mathcal{L}}{\partial b} &= \frac{d\mathcal{L}}{dy} \frac{\partial y}{\partial b} \\ &= y - t\end{aligned}$$

- ▶ For cost derivatives, use linearity and average over data points:

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{1}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)} \\ \frac{\partial \mathcal{J}}{\partial b} &= \frac{1}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)})\end{aligned}$$

- ▶ Minimum must occur at a point where partial derivatives are zero.

$$\begin{aligned}\frac{\partial \mathcal{J}}{\partial w_j} &= 0 \quad (\forall j) \\ \frac{\partial \mathcal{J}}{\partial b} &= 0\end{aligned}$$

## Direct Solution II: Calculus

- ▶ The derivation on the previous slide gives a system of linear equations, which we can solve efficiently.
- ▶ As is often the case for models and code, however, the solution is easier to characterize if we vectorize our calculus.
- ▶ We call the vector of partial derivatives the gradient
- ▶ Thus, the gradient of  $f : \mathbb{R}^D \rightarrow \mathbb{R}$ , denoted  $\nabla f(\mathbf{w})$ , is:

$$\left( \frac{\partial}{\partial w_1} f(\mathbf{w}), \dots, \frac{\partial}{\partial w_D} f(\mathbf{w}) \right)$$

The gradient points in the direction of the greatest rate of increase.

## Direct Solution II: Calculus

- ▶ We seek  $\mathbf{w}$  to minimize  $\mathcal{J}(\mathbf{w}) = \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$
- ▶ Taking the gradient with respect to  $\mathbf{w}$  we get:

$$\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = \mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{t} = \mathbf{0}$$

(ortogonalità vista nelle slide precedenti)

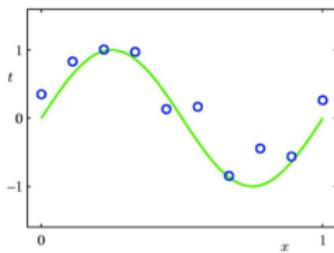
- ▶ We get the same optimal weights as before:

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$$

- ▶ Linear regression is one of only a handful of models in this course that permit direct solution.

# Feature Mapping (Basis Expansion)

The relation between the input and output may **not** be linear.

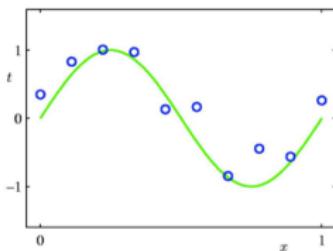


- ▶ We can still **use linear regression** by mapping the input features to another space using **feature mapping** (or **basis expansion**).  
 $\psi(x) : \mathbb{R}^D \rightarrow \mathbb{R}^d$  and treat the mapped feature (in  $\mathbb{R}^d$ ) as the input of a linear regression procedure.
- ▶ Let us see how it works when  $x \in \mathbb{R}$  and we use a polynomial feature mapping.

Posso trasformare i dati per essere in grado di usare nuovamente la regressione lineare. Stiamo usando la funzione  $\phi(x)$  che è un vettore di features, ovvero  $[\phi_0(x), \phi_1(x), \dots, \phi_D(x)]$ . Possiamo avere forma particolare in cui ad ogni  $\phi(x)$  corrisponde  $x_j$

# Polynomial Feature Mapping

If the relationship doesn't look linear, we can fit a polynomial.



Fit the data using a degree- $M$  polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \dots + w_M x^M = \sum_{i=0}^M w_i x^i$$

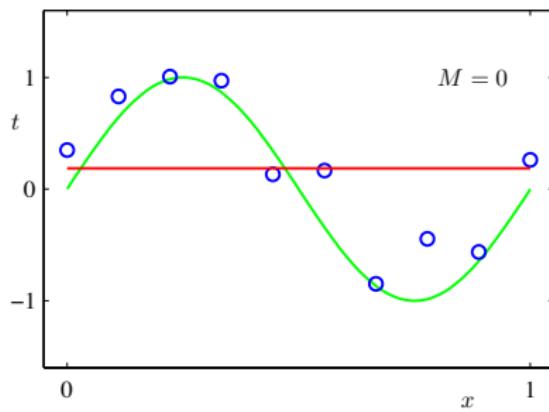
mentre prima avevamo  $\phi(x) = [1, x]$  perchè lineare, ora ragioniamo con più gradi.

- ▶ Here the feature mapping is  $\psi(x) = [1, x, x^2, \dots, x^M]^\top$ .
- ▶ We can still use linear regression to find  $w$  since  $y = \psi(x)^\top w$  is linear in  $w_0, w_1, \dots$
- ▶ In general  $\psi$  can be any function, i.e.,  
 $\psi(x) = [1, \sin(2\pi x), \cos(2\pi x), \sin(4\pi x), \dots]^\top$ .

# Polynomial Feature Mapping with $M = 0$

modello semplicissimo, ma fitta molto male coi dati.  
E' una costante.

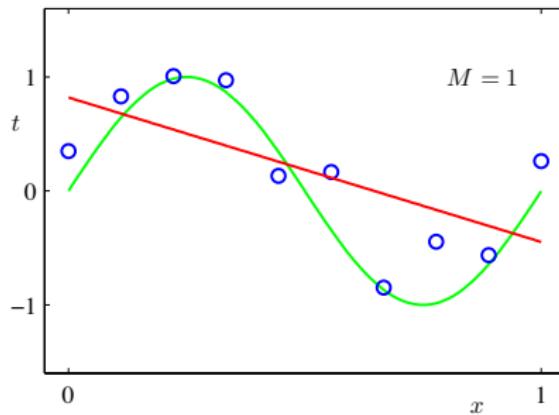
$$y = w_0$$



# Polynomial Feature Mapping with $M = 1$

modello già visto, poco più complesso del precedente, ma non matcha bene.

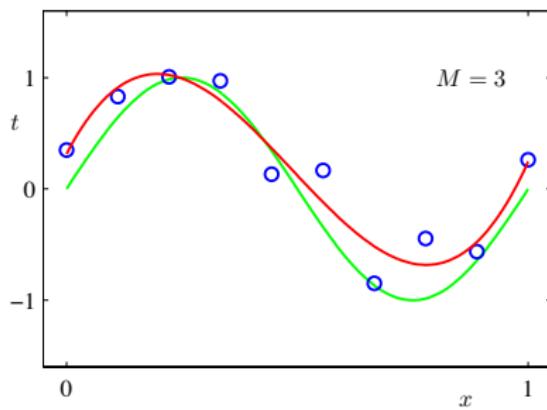
$$y = w_0 + w_1 x$$



# Polynomial Feature Mapping with $M = 3$

modello più interessante.

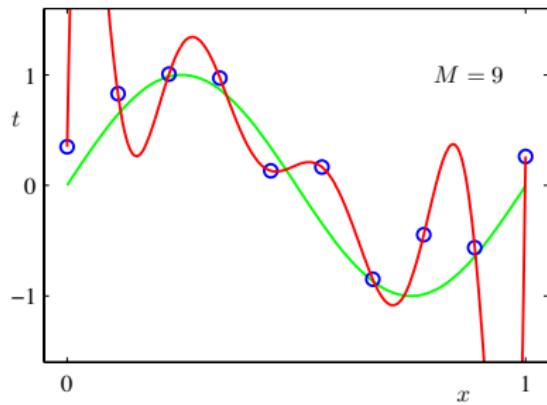
$$y = w_0 + w_1x + w_2x^2 + w_3x^3$$



# Polynomial Feature Mapping with $M = 9$

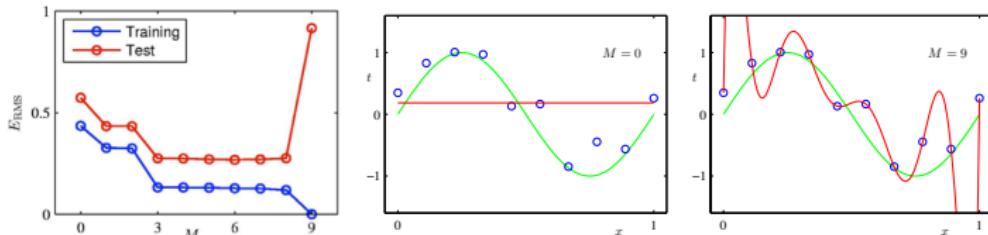
modello complesso, prende esattamente i punti nel training, ma è molto rigido, infatti il suo errore è 0 per i dati forniti. Noi non vogliamo un errore nullo, perché dobbiamo usarlo anche nel testing set, e non ci serve un qualcosa che va fortissimo sul training set e malissimo sul testing.

$$y = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_9x^9$$



# Model Complexity and Generalization

- ▶ Underfitting ( $M = 0$ ): model is too simple - does not fit the data.
- ▶ Overfitting ( $M = 9$ ): model is too complex - fits perfectly.



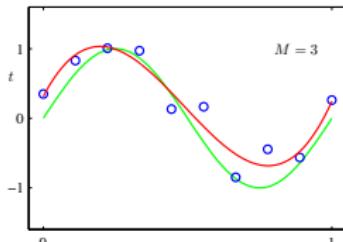
in questo modello sull'ordinata abbiamo l'errore.

vediamo come per il training, con  $M=9$ , esso sia 0, ma per il testing (rosso) è altissimo.

Molto meglio  $M=3$  che si mostra più "costante".

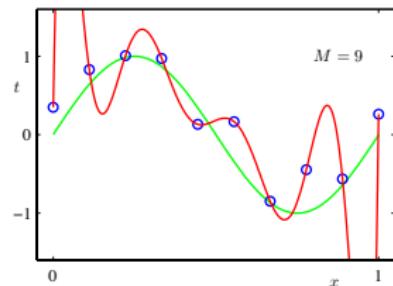
- ▶ Good model ( $M = 3$ ): Achieves small test error (generalizes well).

$$y = w_0 + w_1x + w_2x^2 + w_3x^3$$



# Model Complexity and Generalization

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^*$	0.19	0.82	0.31	0.35
$w_1^*$		-1.27	7.99	232.37
$w_2^*$			-25.43	-5321.83
$w_3^*$			17.37	48568.31
$w_4^*$				-231639.30
$w_5^*$				640042.26
$w_6^*$				-1061800.52
$w_7^*$				1042400.18
$w_8^*$				-557682.99
$w_9^*$				125201.43



la relazione tra  $x_6$  e  $w_6$  è di -1 milione, grande oscillazione

- ▶ As  $M$  increases, the magnitude of coefficients gets larger.
- ▶ For  $M = 9$ , the coefficients have become finely tuned to the data.
- ▶ Between data points, the function exhibits large oscillations.

Nessuno mi dice il grado giusto del polinomio, ma abbiamo visto una tecnica per ridurre le grandi oscillazioni.

# Regularization

- ▶ The degree  $M$  of the polynomial controls the model's complexity.
- ▶ The value of  $M$  is a hyperparameter for polynomial expansion, just like  $k$  in KNN. We can tune it using a validation set.
- ▶ Restricting the number of parameters/basis functions ( $M$ ) is a crude approach to controlling the model complexity.
- ▶ Another approach: keep the model large, but regularize it
  - ▶ Regularizer: a function that quantifies how much we prefer one hypothesis vs. another

M non può essere oggetto del learning, possiamo provarlo solo col validation set. Un buon modo per gestirlo è tramite la REGOLARIZZAZIONE, ovvero aggiungere qualcosa alla funzione obiettivo per avere PESI RAGIONEVOLI. Come? Usando delle "Penalità" da aggiungere alla funzione. Se lambda cresce, i pesi si riducono.  
Vediamo meglio:

## $L^2$ Regularization

- We can encourage the weights to be small by choosing as our regularizer the  $L^2$  penalty.

$$\mathcal{R}(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \sum_j w_j^2$$

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.

$$\mathcal{J}_{\text{reg}}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \lambda \mathcal{R}(\mathbf{w}) = \mathcal{J}(\mathbf{w}) + \frac{\lambda}{2} \sum_j w_j^2$$

Noi vogliamo che la funzione costo, ovvero la media delle varie funzioni Loss, sia il minimo possibile. Noi aggiungiamo una penalità POSITIVA da sommare, quindi l'unico modo per avere funzione costo regolarizzata "piccola" è cercare pesi piccoli per la funzione costo (non regolarizzata)

- If you fit training data poorly,  $\mathcal{J}$  is large. If your optimal weights have high values,  $\mathcal{R}$  is large.
- Large  $\lambda$  penalizes weight values more.
- Like  $M$ ,  $\lambda$  is a hyperparameter we can tune with a validation set.

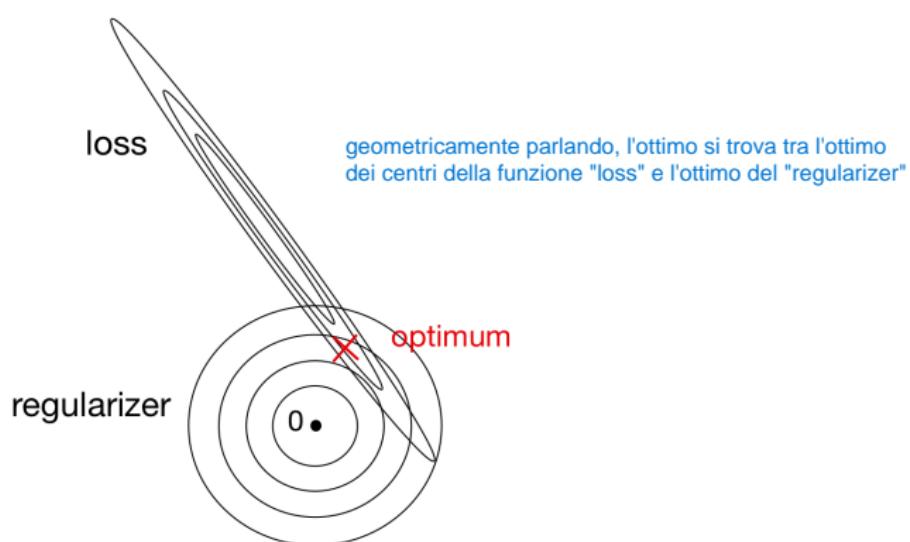
# $L^2$ Regularization

Ricordiamo che la funzione Costo (media delle funzione Loss) è una funzione convessa, avente unico minimo!

## ► The geometric picture



geometricamente sono due figure convesse con una intersezione.



# $L^2$ Regularized Least Squares

A seconda di lambda ottengo soluzioni diverse. Anche con la sua introduzione, le operazioni sono COMPUTAZIONALMENTE semplici.

For the least square problem, we have  $\mathcal{J}(\mathbf{w}) = \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$ .

- When  $\lambda > 0$  ((with regularization), regularized cost gives)

$$\begin{aligned}\mathbf{w}_\lambda &= \arg \min_{\mathbf{w}} \mathcal{J}_{\text{reg}}(\mathbf{w}) = \arg \min_{\mathbf{w}} \frac{1}{2N} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\ &\quad \text{w tale che minimizzi} \\ &\quad \text{la regularized cost} \\ &= (\mathbf{X}^\top \mathbf{X} + \underbrace{\lambda N \mathbf{I}}_{\lambda N \mathbf{I}})^{-1} \mathbf{X}^\top \mathbf{t}\end{aligned}$$

- The case  $\lambda = 0$  (no regularization) reduces to least squares solution!
- Note that it is also common to formulate this problem as  $\arg \min_{\mathbf{w}} \frac{1}{2} \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$  in which case the solution is  $\mathbf{w}_\lambda = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}$

In quest'altra formulazione ciò che cambia è semplicemente il fatto che non è più in funzione di N.

## Conclusions...so far

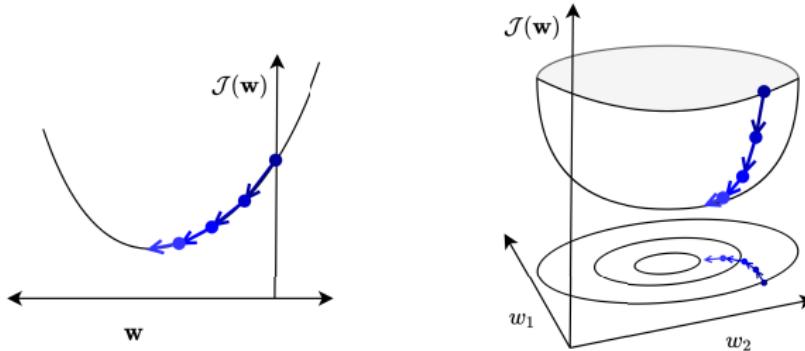
Linear regression exemplifies recurring themes of this course:

- ▶ choose a **model** and a **loss function**
- ▶ formulate an **optimization** problem
- ▶ solve the minimization problem using one of two strategies
  - ▶ **direct solution** (set derivatives to zero)
  - ▶ **gradient descent** (next topic)
- ▶ **vectorize** the algorithm, i.e. represent in terms of linear algebra
- ▶ make a linear model more powerful using **features**
- ▶ improve the generalization by adding a **regularizer**

# Gradient Descent

- ▶ Now let's see a second way to minimize the cost function which is more broadly applicable: **gradient descent**.
- ▶ Many times, we do not have a direct solution: Taking derivatives of  $\mathcal{J}$  w.r.t  $w$  and setting them to 0 doesn't have an explicit solution.
- ▶ Gradient descent is an **iterative algorithm**, which means we apply an update repeatedly until some criterion is met.
- ▶ We **initialize** the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the **direction of steepest descent**.

Spesso non si ha una soluzione diretta (derivo ma non riesco a procedere). Questo algoritmo, di tipo iterativo, propone questo approccio: prendere dei punti della funzione, e calcolare la derivata in quel punto, e vedere se è 0. Ciò è diverso dal calcolare la derivata generica e impostarla a 0. Anche se può sembrare un approccio "rozzo", in realtà come vedremo è molto diffuso.



# Gradient Descent

I calcolatori non hanno concezione di limiti o integrali. Calcolare l'area di una figura non convenzionale vuol dire prendere ad esempio un milione di punti, e vedere quanti cadono in tale figura, sfruttando il metodo Montecarlo.

- ▶ Observe comportamento dell'iterazione:
  - ▶ if  $\partial \mathcal{J} / \partial w_j > 0$ , then increasing  $w_j$  increases  $\mathcal{J}$ .
  - ▶ if  $\partial \mathcal{J} / \partial w_j < 0$ , then increasing  $w_j$  decreases  $\mathcal{J}$ .
- ▶ The following update always decreases the cost function for small enough  $\alpha$  (unless  $\partial \mathcal{J} / \partial w_j = 0$ ):

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

- ▶  $\alpha > 0$  is a learning rate (or step size). The larger it is, the faster  $w$  changes.
  - ▶ We'll see later how to tune the learning rate, but values are typically small, e.g. 0.01 or 0.0001.
  - ▶ If cost is the sum of  $N$  individual losses rather than their average, smaller learning rate will be needed ( $\alpha' = \alpha/N$ ).

vedremo poi pro e contro associati alla scelta di alpha.

# Gradient Descent

- ▶ This gets its name from the gradient:

Ho derivata nulla se tutte le componenti del vettore sono pari a 0.

$$\nabla_{\mathbf{w}} \mathcal{J} = \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

- ▶ This is the direction of fastest increase in  $\mathcal{J}$
- ▶ Update rule in vector form:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

And for linear regression we have:

può sembrare inutile tale

approccio nel caso lineare,

ma come vedremo ha un suo campo d'uso!

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- ▶ So gradient descent updates  $\mathbf{w}$  in the direction of fastest decrease.
- ▶ Observe that once it converges, we get a critical point, i.e.  $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \mathbf{0}$ .

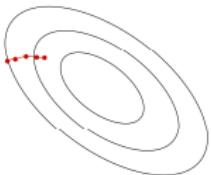
# Gradient Descent for Linear Regression

- ▶ The squared error loss of linear regression is a **convex function**.
- ▶ Even for linear regression, where there is a direct solution, we sometimes need to use GD.
- ▶ Why gradient descent, if we can find the optimum directly?
  - ▶ GD can be applied to a much broader set of models
  - ▶ GD can be easier to implement than direct solutions
  - ▶ For regression in high-dimensional space, GD is more efficient than direct solution
    - ▶ Linear regression solution:  $(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$
    - ▶ Matrix inversion is an  $\mathcal{O}(D^3)$  algorithm
    - ▶ Each GD update costs  $\mathcal{O}(ND)$
    - ▶ Even less with stochastic GD (SGD, in a few slides)
    - ▶ Huge difference if  $D \gg 1$

Tale algoritmo è facile da implementare, oltretutto se lavoriamo con  $D$  grandi, abbiamo costi computazioni minori. Con  $N$  molto grandi (che spesso avremo) può sembrare non così migliorativo, ma scopriremo che tale algoritmo può essere implementato ancora più

# Learning Rate (Step Size)

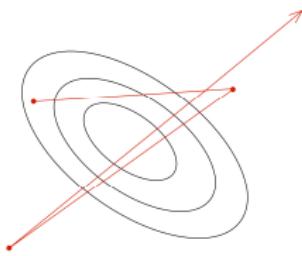
- In gradient descent, the learning rate  $\alpha$  is a hyperparameter we need to tune. Here are some things that can go wrong:



$\alpha$  too small:  
slow progress



$\alpha$  too large:  
oscillations



$\alpha$  much too large:  
instability

Se alpha è piccolo, mi sto muovendo piano ma "campiono" meglio.

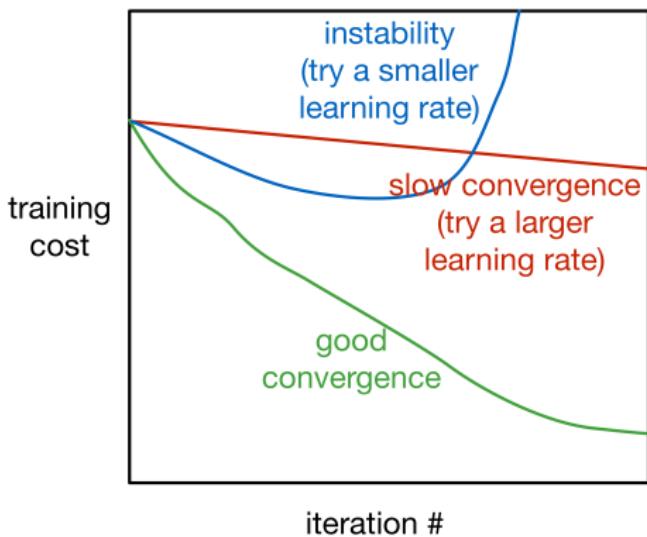
Se alpha è grande, mi muovo velocemente, a zig zag, e potrei perdere punti salienti.

- Good values are typically between 0.001 and 0.1. You should do a grid search if you want good performance (i.e. try 0.1, 0.03, 0.01, ...).

# Training Curves

Alpha dipende dalla derivata del problema. Tecnica utile è vedere come si evolve il comportamento dell'errore.

- To diagnose optimization problems, it's useful to look at **training curves**: plot the training cost as a function of iteration.  
Tipicamente si parte da alpha = 0.1 e lo si fa decrescere.



- Warning: in general, it's very hard to tell from the training curves whether an optimizer has converged. They can reveal major problems, but they can't guarantee convergence.

# Stochastic Gradient Descent

- ▶ So far, the cost function  $\mathcal{J}$  has been the average loss over the training examples:

$$\mathcal{J}(\theta) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}^{(i)} = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y(\mathbf{x}^{(i)}, \theta), t^{(i)})$$

( $\theta$  denotes the parameters; e.g., in linear regression,  $\theta = (\mathbf{w}, b)$ )

- ▶ By linearity,

mettiamo la derivata  
nella sommatoria

$$\frac{\partial \mathcal{J}}{\partial \theta} = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$$

- ▶ Computing the gradient requires summing over all of the training examples. This is known as **batch training**.
- ▶ Batch training is **impractical** if you have a large dataset  $N \gg 1$  (e.g. millions of training examples!)

Dovrei derivare ogni singolo componente... un pò esoso in termini computazionali.

# Stochastic Gradient Descent

- ▶ Stochastic gradient descent (SGD): update the parameters based on the gradient for a single training example,
  1. Choose  $i$  uniformly at random,
  2.  $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$
- ▶ Cost of each SGD update is independent of N!
- ▶ SGD can make significant progress before even seeing all the data!
- ▶ Mathematical justification: if you sample a training example uniformly at random, the stochastic gradient is an unbiased estimate of the batch gradient:

$$E \left[ \frac{\partial \mathcal{L}^{(i)}}{\partial \theta} \right] = \frac{1}{N} \sum_{i=1}^N \frac{\partial \mathcal{L}^{(i)}}{\partial \theta} = \frac{\partial \mathcal{J}}{\partial \theta}$$

sul singolo dato ragiono con la funzione Cost (che è la media delle loss), ma se è un dato solo non dovrei ragionare sulla loss (unica) invece che Cost (media di tutte)?

La risposta è che mediamente sono quasi uguali, quindi posso lasciare la funzione Cost.

Non devo avere tutti i dati, ma comunque posso aggiornare "caso per caso", non devo avere tutti i dati.

# Stochastic Gradient Descent

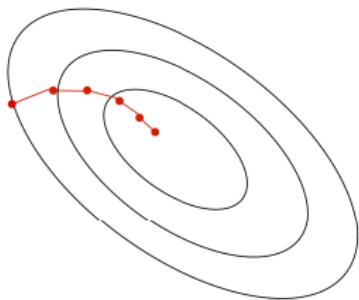
Concettualmente, ci stiamo chiedendo: è meglio poca benzina e spostarsi in modo strano, oppure consumare tanta benzina ma con spostamenti corretti? La risposta che preferiamo è la prima, perché alla fine convergiamo alla stessa soluzione IN MEDIA (ma più lentamente), e spesso i terminali usati non possono sopportare un tale sforzo. Ci sono tanti fattori influenzanti (hardware, tempi, temperature...)

- ▶ Problems with using single training example to estimate gradient:
  - ▶ Variance in the estimate may be high
  - ▶ We can't exploit efficient vectorized operations
- ▶ Compromise approach:
  - ▶ compute the gradients on a randomly chosen medium-sized set of training examples  $\mathcal{M} \subset \{1, \dots, N\}$  called a mini-batch.
- ▶ Stochastic gradients computed on larger mini-batches have smaller variance.
- ▶ The mini-batch size  $|\mathcal{M}|$  is a hyperparameter that needs to be set.
  - ▶ Too large: requires more compute; e.g., it takes more memory to store the activations, and longer to compute each gradient update
  - ▶ Too small: can't exploit vectorization, has high variance
  - ▶ A reasonable value might be  $|\mathcal{M}| = 100$ .

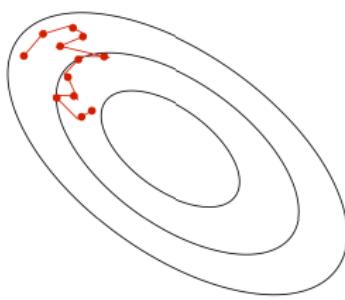
Ovviamente per ogni MINI BATCH abbiamo soluzioni di minimo locale diverso, ma poi noi dobbiamo ragionare su tutto.

# Stochastic Gradient Descent

- ▶ Batch gradient descent moves directly downhill (locally speaking).
- ▶ SGD takes steps in a noisy direction, but moves downhill on average.



batch gradient descent

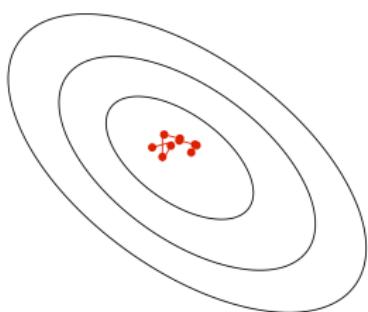


stochastic gradient descent

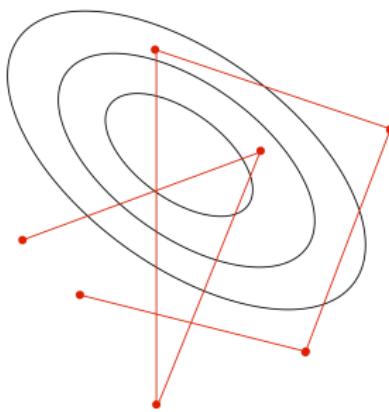
# Stochastic Gradient Descent Learning Rate

- In stochastic training, the learning rate also influences the fluctuations due to the stochasticity of the gradients.

small learning rate

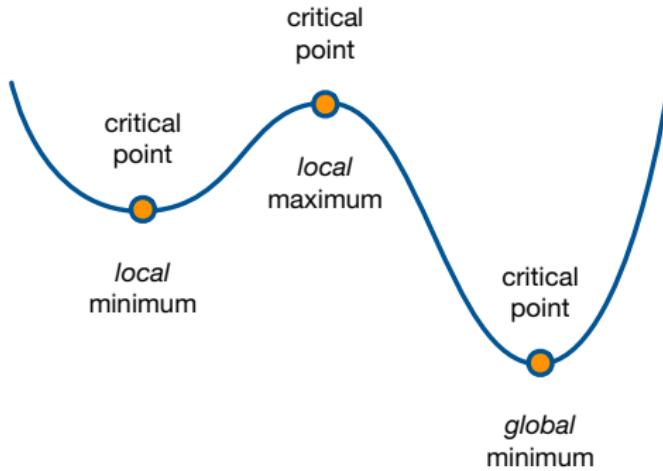


large learning rate



- Typical strategy:
  - Use a large learning rate early in training so you can get close to the optimum partiamo "velocemente" e poi rallentiamo.
  - Gradually decay the learning rate to reduce the fluctuations

# When are critical points optimal?



- ▶ Gradient descent finds a critical point, but it may be a **local optima**.
- ▶ **Convexity** is a property that guarantees that all critical points are **global minima**.

Con le funzioni convesse sono tranquillo che il minimo locale sia anche il minimo globale.

# Conclusions

$$w = \bar{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - \bar{y})x^{(i)}$$

Alcuni punti salienti (recap):

- inizializzo 'w' con tutti 0, ma anche da qualsiasi altro punto convergo sempre al minimo.
- nell'aggiornamento di 'w', mi fermo finché non converge, cioè derivata nulla, cioè la differenza tra il nuovo 'w' e il vecchio 'w' non è piccola. Aggiornando i pesi, le 'y' cambiano. Mi fermo se non ho errore. ('t' e 'x' sono costanti). Quello che vogliamo minimizzare è sempre la funzione costo, e graficamente è come se, dato un set di punti, passo dopo passo aggiustiamo la nostra retta di regressione. Il miglioramento non è per ogni iterazione, ma "in media". Nella versione stocastica, prendo un sottoinsieme.

- ▶ Linear regression exemplifies a modular approach that will be used throughout this course:
  - ▶ choose a **model** describing the relationships between variables of interest (**linear**)
  - ▶ define a **loss** function quantifying how bad the fit to the data is (**squared error**)
  - ▶ fit/optimize the model (**gradient descent, stochastic gradient descent, convexity**)
- ▶ By mixing and matching these modular components, we can obtain new ML methods.
- ▶ Next apply this framework to classification

# Logistic Regression

...based on Toronto University ML class

# Overview

Fino ad ora abbiamo ragionato su valori numerici, mentre ora passiamo ad una classificazione con target discreto (covid/no covid, bianco/nero/blu/rosso etc..). Un sottocaso è quello binario, in cui possiamo ricondurci al caso (0,1) o anche (-1,+1). Clò che abbiamo visto fino ad ora, può essere utilizzato di nuovo?

- ▶ **Classification:** predicting a discrete-valued target
  - ▶ **Binary classification:** predicting a binary-valued target
  - ▶ **Multiclass classification:** predicting a discrete(> 2)-valued target
- ▶ Examples of binary classification
  - ▶ predict whether a patient has a disease, given the presence or absence of various symptoms
  - ▶ classify e-mails as spam or non-spam
  - ▶ predict whether a financial transaction is fraudulent

# Binary linear classification

- ▶ **classification:** given a  $D$ -dimensional input  $\mathbf{x} \in \mathbb{R}^D$  predict a discrete-valued target
- ▶ **binary:** predict a binary target  $t \in \{0, 1\}$ 
  - ▶ Training examples with  $t = 1$  are called **positive examples**, and training examples with  $t = 0$  are called **negative examples**.
  - ▶  $t \in \{0, 1\}$  or  $t \in \{-1, +1\}$  is for computational convenience.
- ▶ **linear:** model prediction  $y$  is a linear function of  $\mathbf{x}$ , followed by a threshold  $r$ :

$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Viene introdotta una "soglia  $r$ ", che può essere anche 0 (ha molto senso se classichiamo in 0 e 1 ad esempio).

# Some Simplifications

## Eliminating the threshold

- ▶ We can assume without loss of generality that the threshold  $r = 0$ :

$$\mathbf{w}^\top \mathbf{x} + b \geq r \rightarrow \underbrace{\mathbf{w}^\top \mathbf{x} + b - r}_{w_0} \geq 0$$

sostanzialmente, portando ' $r$ ' a sinistra, invece di lavorare con ' $b$ ', lavoriamo con ' $b-r$ ', e la chiamiamo in un nuovo modo. Poniamo poi  $r=0$

## Eliminating the bias

- ▶ Add a dummy feature  $x_0$  which always takes the value 1. The weight  $w_0 = b$  is equivalent to a bias (same as linear regression)

## Simplified model

- ▶ Receive input  $\mathbf{x} \in \mathbb{R}^{D+1}$  with  $x_0 = 1$ :

Questa variabile fittizia è utilizzata per catturare il termine di bias nel modello, che è simile al termine noto nella regressione lineare.

Questa è una funzione gradino, discontinua e non derivabile in 0.

Vorremmo un qualcosa di "smooth", che curvasse in prossimità del gradino per non avere questi problemi.

$$z = \mathbf{w}^\top \mathbf{x}$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

# Logistic Regression

Come detto prima, la funzione gradino è troppo rigida e con problemi, ragionando in termini probabilistici, cerchiamo di lavorare su una funzione più gestibile.

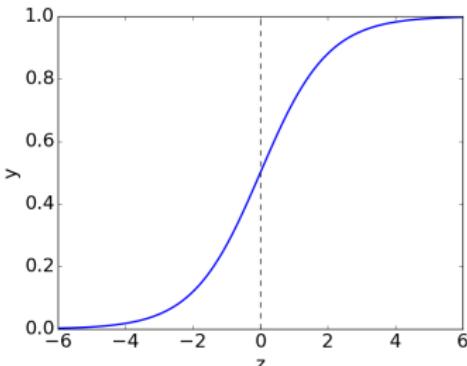
Take a probabilistic approach to learn a classifier

- ▶  $f(\mathbf{x}) = f_{\mathbf{w}}(\mathbf{x})$  should provide  $P(y = 1 | \mathbf{x}; \mathbf{w})$ 
  - ▶ Want  $0 \leq f_{\mathbf{w}}(\mathbf{x}) \leq 1$

Logistic regression model      mi dice quanto sono confidente

$$f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x})$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Se  $z=0$  sto al 50%.

Se  $z$  cresce e va a +infinito, la funzione tende a 1.

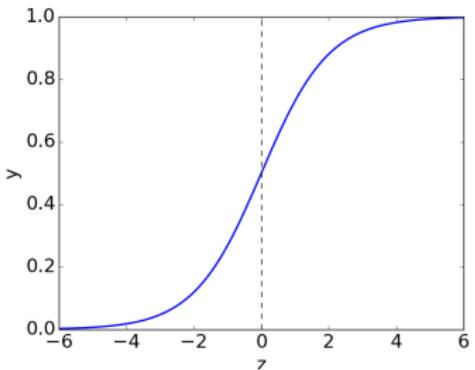
Se  $z$  decresce e va a -infinito, la funzione tende a 0.

# Logistic Regression

in  $\mathbb{R}^2$  la curva di separazione è una retta.

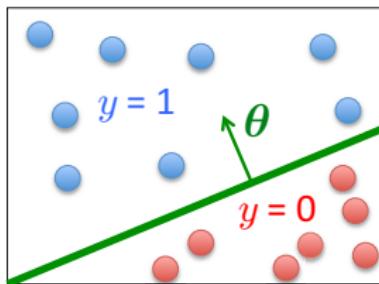
$$f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Assume a threshold, e.g., 0.5, and...

- ▶ Predict  $y = 1$  if  $f_{\mathbf{w}}(\mathbf{x}) > 0.5$
- ▶ Predict  $y = 0$  if  $f_{\mathbf{w}}(\mathbf{x}) \leq 0.5$



# Logistic Regression Objective Function

- ▶ Given  $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(N)}, t^{(N)})\}$  with  $x^{(i)} \in \mathbb{R}^d$ ,  $t^{(i)} \in \{0, 1\}$
- ▶ and the Logistic Regression Model:  $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$

Noi lavoriamo in spazi più grandi di  $\mathbb{R}^2$ , tendenzialmente  $D > 2$ , sono tutti vettori, e vorrei usare tale regressione.  
Posso?

# Logistic Regression Objective Function

- ▶ Given  $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(N)}, t^{(N)})\}$  with  $x^{(i)} \in \mathbb{R}^d$ ,  $t^{(i)} \in \{0, 1\}$
- ▶ and the Logistic Regression Model:  $f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$
- ▶ We can't just use the squared loss as in linear regression:

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^N (f_{\mathbf{w}}(\mathbf{x}^{(i)}) - t^{(i)})^2$$

Questo problema nasce dal fatto che la funzione sigma; che mettiamo nella funzione costo è NON convessa, e quindi non so se il minimo trovato sia locale o globale. Con tanti punti di minimo locale è un bel problema.

# Logistic Regression Objective Function

- ▶ Given  $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(N)}, t^{(N)})\}$  with  $x^{(i)} \in \mathbb{R}^d$ ,  $t^{(i)} \in \{0, 1\}$
- ▶ and the Logistic Regression Model:  $f_w(x) = \sigma(w^\top x)$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$
- ▶ We can't just use the **squared loss** as in linear regression:

$$\mathcal{J}(w) = \frac{1}{N} \sum_{i=1}^N (f_w(x^{(i)}) - t^{(i)})^2$$

- ▶ using the Logistic Regression Model

$$f_w(x) = \frac{1}{1 + e^{-w^\top x}}$$

**results in a non-convex optimization problem**

# Deriving the Cost Function via Maximum Likelihood Estimation

APPROCCIO: MASSIMA VEROSIMIGLIANZA

è la probabilità di osservare 't', dato x (vettore di variabili indipendenti) in funzione dei pesi 'w'.

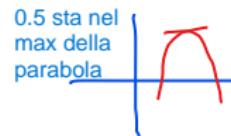
- ▶ Define Likelihood of observed data  $\mathbf{t} = (t^{(1)}, \dots, t^{(N)})$ , given  $\mathbf{x} = (x^{(1)}, \dots, x^{(N)})$  and parameter  $\mathbf{w}$  as

$$l(\mathbf{w}) = P(\mathbf{t}|\mathbf{x}; \mathbf{w})$$

that is the probability of observing  $\mathbf{t}$  given  $\mathbf{x}$

- ▶ which is function of the parameters  $\mathbf{w}$
- ▶ Given the independence between  $t^{(i)}$  and  $\mathbf{x}^{(j)}$ ,  $i \neq j$ , we can rewrite as:

$$l(\mathbf{w}) = P(\mathbf{t}|\mathbf{x}; \mathbf{w}) \equiv \prod_{i=1}^N P(t^{(i)}|x^{(i)}; \mathbf{w})$$



In Statistica, ci chiediamo "probabilità che si verifichi un evento?". Adesso è il contrario: "ho un evento, quale è la probabilità che mi realizza tale evento?". Ad esempio: lancio moneta: TCTC, sappiamo che la probabilità è  $T=C=1/2$ , allora tale osservazione la si ha con  $p^2(1-p)^4$ . Ovvero i "successi" ed i fallimenti. Non c'è il coefficiente binomiale perchè stiamo parlando di questa SPECIFICA osservazione, non di una generica. Se l'osservazione fosse TTTC dovrei credere che  $p(\text{testa})=0.5$ ? NO! Perchè tale osservazione la avrei solo con  $p(\text{testa})=0.75$

# Deriving the Cost Function via Maximum Likelihood Estimation

## Maximum Likelihood Estimation

- We look for the  $\mathbf{w}$  which maximizes the likelihood  $I(\mathbf{w})$

qui parliamo di massimizzare perchè cerco la probabilità più "reale" che realizzi l'osservazione

$$\mathbf{w}_{MLE} = \arg \max_{\mathbf{w}} I(\mathbf{w}) = \arg \max_{\mathbf{w}} \prod_{i=1}^N P(t^{(i)} | \mathbf{x}^{(i)} ; \mathbf{w})$$

- We can take the  $\log$  without changing the solution
  - log function is monotonically increasing

l'applicazione del log  
non influisce sui risultati.

$$\mathbf{w}_{MLE} = \arg \max_{\mathbf{w}} \log \prod_{i=1}^N P(t^{(i)} | \mathbf{x}^{(i)} ; \mathbf{w})$$

$$= \arg \max_{\mathbf{w}} \sum_{i=1}^N \log P(t^{(i)} | \mathbf{x}^{(i)} ; \mathbf{w})$$

Inoltre, ciò che minimizza  $f(x)$   
massimizza  $\log(f(x))$   
(quantomeno in casi "semplici").

I logaritmi trasformano i prodotti di probabilità in somme di contributi, più facili da gestire.

# Deriving the Cost Function via Maximum Likelihood Estimation

- ▶ ...which we can rewrite as follows:

$$\begin{aligned}\mathbf{w}_{MLE} &= \arg \max_{\mathbf{w}} \sum_{i=1}^N \log P(t^{(i)} | \mathbf{x}^{(i)} ; \mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \sum_{i=1}^N \left[ t^{(i)} \log P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w}) + (1 - t^{(i)}) \log (1 - P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w})) \right]\end{aligned}$$

abbiamo semplicemente esteso, dividendo i due casi in cui  $t=0$  e  $t=1$ , è come se venissero usati come funzioni indicatori.

# Deriving the Cost Function via Maximum Likelihood Estimation

- ▶ ...which we can rewrite as follows:

$$\begin{aligned}\mathbf{w}_{MLE} &= \arg \max_{\mathbf{w}} \sum_{i=1}^N \log P(t^{(i)} | \mathbf{x}^{(i)} ; \mathbf{w}) \\ &= \arg \max_{\mathbf{w}} \sum_{i=1}^N \left[ t^{(i)} \log P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w}) + (1 - t^{(i)}) \log (1 - P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w})) \right]\end{aligned}$$

- ▶ **Logistic Regression** semplicemente: prima massimizzavo, ma io devo minimizzare. allora ribalto!

- ▶ take the (negative of the) log-likelihood as cost function

$$\begin{aligned}\mathcal{J}(\mathbf{w}) &= - \sum_{i=1}^N \left[ t^{(i)} \log P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w}) + (1 - t^{(i)}) \log (1 - P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w})) \right] \\ &= - \sum_{i=1}^N \left[ t^{(i)} \log f_{\mathbf{w}}(\mathbf{x}) + (1 - t^{(i)}) \log (1 - f_{\mathbf{w}}(\mathbf{x})) \right]\end{aligned}$$

questa è una forma convessa!

# Deriving the Cost Function via Maximum Likelihood Estimation

## ► Logistic Regression

$$\begin{aligned}\mathcal{J}(\mathbf{w}) &= - \sum_{i=1}^N \left[ t^{(i)} \log P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w}) + (1 - t^{(i)}) \log (1 - P(t^{(i)} = 1 | \mathbf{x}^{(i)} ; \mathbf{w})) \right] \\ &= - \sum_{i=1}^N \left[ t^{(i)} \log f_{\mathbf{w}}(\mathbf{x}) + (1 - t^{(i)}) \log (1 - f_{\mathbf{w}}(\mathbf{x})) \right] \\ &= - \sum_{i=1}^N \left[ t^{(i)} \log \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} + (1 - t^{(i)}) \log \left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}\right) \right]\end{aligned}$$

► ...which is convex :-)

# Gradient Descent for Logistic Regression

- ▶ How do we minimize the cost  $\mathcal{J}$  for logistic regression? No direct solution.
  - ▶ Taking derivatives of  $\mathcal{J}$  w.r.t.  $\mathbf{w}$  and setting them to 0 doesn't have an explicit solution.
- ▶ However, the logistic loss is a convex function in  $\mathbf{w}$ , so let's consider the gradient descent.
  - ▶ Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - ▶ A standard initialization is  $\mathbf{w} = \mathbf{0}$ .

Il gradiente va verso la soluzione ottima!

# Gradient of Logistic Loss

Back to logistic regression:

CROSS ENTROPIA  $\mathcal{L}_{CE}(y, t) = -t \log(y) - (1 - t) \log(1 - y)$

$$y = \frac{1}{1 + e^{-z}} \text{ and } z = \mathbf{w}^\top \mathbf{x}$$

Therefore

$$\frac{\partial \mathcal{L}_{CE}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_j} = \left( -\frac{t}{y} + \frac{1-t}{1-y} \right) \cdot y(1-y) \cdot x_j$$

risolto nella cartella delle osservazioni  $= (y - t)x_j$  "singola" (funzione loss)

Gradient descent (coordinatewise) update to find the weights of logistic regression:

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$
$$= w_j - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)}$$

torna sempre questo risultato! "media" (funzione costo)

l'unica differenza è che t può essere solo 0 o 1.

# Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- ▶ Linear regression

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- ▶ Logistic regression

$$\mathbf{w} = \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

- ▶ Not a coincidence! These are both examples of generalized linear models. But we won't go in further detail.
- ▶ Notice  $\frac{1}{N}$  in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses ( $\alpha' = \alpha/N$ ).

# Multiclass Classification and Softmax Regression

...based on Toronto University ML class

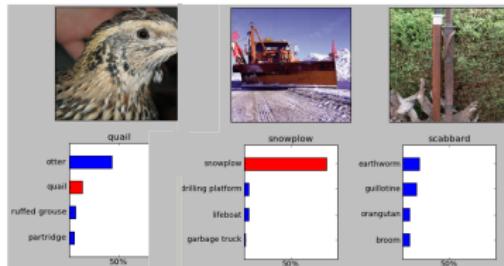
# Overview

- ▶ **Classification:** predicting a discrete-valued target
  - ▶ **Binary classification:** predicting a binary-valued target
  - ▶ **Multiclass classification:** predicting a discrete(> 2)-valued target
- ▶ Examples of multi-class classification
  - ▶ predict the value of a handwritten digit
  - ▶ classify e-mails as spam, travel, work, personal

# Multiclass Classification

- ▶ Classification tasks with more than two categories:

0 0 0 1 1 1 1 1 2  
2 2 2 2 2 2 2 3 3 3  
3 4 4 4 4 5 5 5 5  
6 6 7 7 7 7 8 8 8  
8 8 8 9 9 9 9 9 9



# Multiclass Classification

- ▶ Targets form a discrete set  $\{1, \dots, K\}$ .
- ▶ It's often more convenient to represent them as one-hot vectors, or a one-of-K encoding:

$$\mathbf{t} = (\underbrace{0, \dots, 0, 1, 0, \dots, 0}_{\text{entry } k \text{ is 1}}) \in \mathbb{R}^K$$

Per ricondursi ai casi precedenti, invece di dire che  $\mathbf{t}$  vada da 1 a  $K$ ; si preferisce porre ad 1 solo la  $k$ -esima entry.  
Questa ci informa della  $\text{prob}(t=k)$ .

# Multiclass Linear Classification

- ▶ We can start with a linear function of the inputs.
- ▶ Now there are  $D$  input dimensions and  $K$  output dimensions, so we need  $K \times D$  weights, which we arrange as a weight matrix  $\mathbf{W}$ .
- ▶ Also, we have a  $K$ -dimensional vector  $\mathbf{b}$  of biases.
- ▶ A linear function of the inputs:

$$z_k = \sum_{j=1}^D w_{kj}x_j + b_k \text{ for } k = 1, 2, \dots, K$$

ci dice "quanto il modello preferisce  $k$  come predizione"

- ▶ or in vectorized form

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{d}$$

- ▶ We can eliminate the bias  $\mathbf{b}$  by taking  $\mathbf{W} \in \mathbb{R}^{K \times (D+1)}$  and adding a dummy variable  $x_0 = 1$ , that is

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

Potenzialmente, sia  $\mathbf{W}$  sia ' $\mathbf{x}$ ' possono andare da "-infinito" a "+infinito", dovremmo correggere questo aspetto andando avanti.

# Multiclass Linear Classification

- ▶ How can we turn this linear prediction into a one-hot prediction?
- ▶ We can interpret the magnitude of  $z_k$  as a measure of how much the model prefers  $k$  as its prediction.
- ▶ ...that is

$$y_i = \begin{cases} 1 & i = \arg \max z_k \\ 0 & \text{otherwise} \end{cases}$$

esempio:

$z = (-2, 1, 3, 4.5, 6, 8, -2)$ . --->  $y = (0, 0, 0, 0, 0, 1, 0)$  perché 8 è il più grande.

Tuttavia lavoriamo su un concetto di "argmax", quindi ritorna il problema della funzione scalino.

Si usa altro:

# Softmax Regression

- ▶ We need to soften our predictions for the sake of optimization.
- ▶ We want soft predictions that are like probabilities, i.e.,  $0 \leq y_k \leq 1$  and  $\sum_k y_k = 1$ .
- ▶ A natural activation function to use is the softmax function, a multivariable generalization of the logistic function:

$$y_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

ho solo il k-esimo valore all'esponente  
sommo tutti

- ▶ Outputs can be interpreted as probabilities (positive and sum to 1)
- ▶ If  $z_k$  is much larger than the others, then  $\text{softmax}(\mathbf{z})_k \approx 1$  and it behaves like argmax.
- ▶ The inputs  $z_k$  are called the logits.

Softmax è inoltre differenziabile, e si comporta come argmax, enfatizzando le piccole differenze.

La formula nuda e cruda non è sempre quella che viene implementata, la formula del fattoriale, ad esempio, è semplice, ma farla al computer richiede molti calcoli, quindi si implementa in modo diverso!

# Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

$$\begin{aligned}\mathcal{L}_{CE}(\mathbf{y}, \mathbf{t}) &= - \sum_{k=1}^K t_k \log y_k \\ &= -\mathbf{t}^\top (\log \mathbf{y})\end{aligned}$$

where the log is applied elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.

# Softmax Regression

- ▶ Softmax regression (with dummy  $x_0 = 1$ ):

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

$$\mathbf{y} = \text{softmax}(\mathbf{z})$$

$$\mathcal{L}_{CE} = -\mathbf{t}^\top (\log \mathbf{y})$$

- ▶ Gradient descent updates can be derived for each row of  $\mathbf{W}$ :

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{w}_k} = \frac{\partial \mathcal{L}_{CE}}{\partial z_k} \cdot \frac{\partial z_k}{\partial \mathbf{w}_k} = (y_k - t_k) \cdot \mathbf{x}$$

$$\mathbf{w}_k \leftarrow \mathbf{w}_k - \alpha \frac{1}{N} \sum_{i=1}^N (y_k^{(i)} - t_k^{(i)}) \mathbf{x}^{(i)}$$

- ▶ Similar to linear/logistic regression (no coincidence)

## A Few Basic Concepts

- ▶ A hypothesis is a function  $f : \mathcal{X} \rightarrow \mathcal{T}$  that we might use to make predictions (recall  $\mathcal{X}$  is the input space and  $\mathcal{T}$  is the target space).
- ▶ The hypothesis space  $\mathcal{H}$  for a particular machine learning model or algorithm is set of hypotheses that it can represent.
  - ▶ E.g., in linear regression,  $\mathcal{H}$  is the set of functions that are linear in the data features
  - ▶ The job of a machine learning algorithm is to find a good hypothesis  $f \in \mathcal{H}$

## A Few Basic Concepts

- ▶ If an algorithm's hypothesis space  $\mathcal{H}$  can be defined using a finite set of parameters, denoted  $\theta$ , we say the algorithm is parametric.
  - ▶ In linear regression,  $\theta = (\mathbf{w}, b)$
  - ▶ Other examples: logistic regression, neural networks,  $k$ -means and Gaussian mixture models
- ▶ If the members of  $\mathcal{H}$  are defined in terms of the data, we say that the algorithm is non-parametric.
  - ▶ In  $k$ -nearest neighbors, the learned hypothesis is defined in terms of the training data
  - ▶ Other examples: decision trees, support vector machines, kernel density estimation
  - ▶ These models can sometimes be understood as having an infinite number of parameters